منصة حقيبتي التعليمية

منصـــة حقيبتي هو موقع تعليمي يعمـل على تســـهيـل العملية التعليمية بطريقة بســـيطة وســـهلة وتوفير كل ما يحتاجه المعلم والطالب لكافة الصـــفوف الدراســـية كما يحتوي الموقع على حلول جميع المواد مع الشروح المتنوعة للمعلمين.

المملكة العربية السعودية

وزارة التعليم
Ministry of Education

قررت وزارة التعليم تدريس
هذا الكتاب وطبعه على نفقتها

الذكاء الاصطناعي

التعليم الثانوي - نظام المسارات

السنة الثالثة

طبعة 1446-2024

فهرسة مكتبة الملك فهد الوطنية أثناء النشر
وزارة التعليم
الذكاء الاصطناعي - المرحلة الثانوية - نظام المسارات - السنة الثالثة . / وزارة التعليم . - الرياض، ١٤٤٤ هـ
٣٤١ ص ؛ ٢١ x ٢٥,٥ سم
ردمك : ٠-٤٩٥-٥١١-٦٠٣-٩٧٨
١ - التعليم - مناهج - السعودية أ . العنوان
ديوي ٣٧٥,٠٠٩٥٣١ ١١١٢٢ / ١٤٤٤

رقم الإيداع : ١١١٢٢ / ١٤٤٤
ردمك : ٠-٤٩٥-٥١١-٦٠٣-٩٧٨

مواد إثرائية وداعمة على "منصة عين الإثرائية"

ien.edu.sa

أعزاءنا المعلمين والمعلمات، والطلاب والطالبات، وأولياء الأمور، وكل مهتم بالتربية والتعليم:
يسعدنا تواصلكم؛ لتطوير الكتاب المدرسي، ومقترحاتكم محل اهتمامنا.

fb.ien.edu.sa

الناشر: شركة تطوير للخدمات التعليمية

تم النشر بموجب اتفاقية خاصة بين شركة Binary Logic SA وشركة تطوير للخدمات التعليمية
(عقد رقم 2022/0003) للاستخدام في المملكة العربية السعودية

binarylogic

مقدمة

إن تقدم الدول وتطورها يقاس بمدى قدرتها على الاستثمار في التعليم، ومدى استجابة نظامها التعليمي لمتطلبات العصر ومتغيراته. وحرصًا من وزارة التعليم على ديمومة تطوير أنظمتها التعليمية، واستجابة لرؤية المملكة العربية السعودية 2030 فقد بادرت الوزارة إلى اعتماد نظام «مسارات التعليم الثانوي» بهدف إحداث تغيير فاعل وشامل في المرحلة الثانوية.

إن نظام مسارات التعليم الثانوي يقدم أنموذجًا تعليميًا متميزًا وحديثًا للتعليم الثانوي بالمملكة العربية السعودية يسهم بكفاءة في:

- تعزيز قيم الانتماء لوطننا المملكة العربية السعودية، والولاء لقيادته الرشيدة حفظهم الله، انطلاقًا من عقيدة صافية مستندة على التعاليم الإسلامية السمحة.
- تعزيز قيم المواطنة من خلال التركيز عليها في المواد الدراسية والأنشطة، اتساقًا مع مطالب التنمية المستدامة، والخطط التنموية في المملكة العربية السعودية التي تؤكد على ترسيخ ثنائية القيم والهوية، والقائمة على تعاليم الإسلام والوسطية.
- تأهيل الطلبة بما يتوافق مع التخصصات المستقبلية في الجامعات والكليات أو المهن المطلوبة؛ لضمان اتساق مخرجات التعليم مع متطلبات سوق العمل.
- تمكين الطلبة من متابعة التعليم في المسار المفضل لديهم في مراحل مبكرة، وفق ميولهم وقدراتهم.
- تمكين الطلبة من الالتحاق بالتخصصات العلمية والإدارية النوعية المرتبطة بسوق العمل، ووظائف المستقبل.
- دمج الطلبة في بيئة تعليمية ممتعة ومحفزة داخل المدرسة قائمة على فلسفة بنائية، وممارسات تطبيقية ضمن مناخ تعليمي نشط.
- نقل الطلبة عبر رحلة تعليمية متكاملة بدءًا من المرحلة الابتدائية حتى نهاية المرحلة الثانوية، وتُسهّل عملية انتقالهم إلى مرحلة ما بعد التعليم العام.
- تزويد الطلبة بالمهارات التقنية والشخصية التي تساعدهم على التعامل مع الحياة، والتجاوب مع متطلبات المرحلة.
- توسيع الفرص أمام الطلبة الخريجين عبر خيارات متنوعة إضافة إلى الجامعات مثل: الحصول على شهادات مهنية، والالتحاق بالكليات التطبيقية، والحصول على دبلومات وظيفية.

ويتكون نظام المسارات من تسعة فصول دراسية تُدرّس في ثلاث سنوات، تتضمن سنة أولى مشتركة يتلقى فيها الطلبة الدروس في مجالات علمية وإنسانية متنوعة، تليها سنتان تخصصيتان، يُسكّن الطلبة بها في مسار عام وأربعة مسارات تخصصية تتسق مع ميولهم وقدراتهم، وهي: المسار الشرعي، مسار إدارة الأعمال، مسار علوم الحاسب والهندسة، مسار الصحة والحياة، وهو ما يجعل هذا النظام هو الأفضل للطلبة من حيث:

- وجود مواد دراسية جديدة تتوافق مع متطلبات الثورة الصناعية الرابعة والخطط التنموية، ورؤية المملكة 2030، تهدف لتنمية مهارات التفكير العليا وحل المشكلات، والمهارات البحثية.
- برامج المجال الاختياري التي تتسق مع احتياجات سوق العمل وميول الطلبة، حيث يُمكّن الطلبة من الالتحاق بمجال اختياري محدد وفق مصفوفة مهارات وظيفية محددة.
- مقياس ميول يضمن تحقيق كفاءة الطلبة وفاعليتهم، ويساعدهم في تحديد اتجاهاتهم وميولهم، وكشف مكامن القوة لديهم، مما يعزز من فرص نجاحهم في المستقبل.
- العمل التطوعي المصمم للطلبة خصيصًا بما يتسق مع فلسفة النشاط في المدارس، ويعد أحد متطلبات التخرج؛ مما يساعد على تعزيز القيم الإنسانية، وبناء المجتمع وتنميته وتماسكه.
- التجسير الذي يمكن الطلبة من الانتقال من مسار إلى آخر وفق آليات محددة.
- حصص الإتقان التي يتم من خلالها تطوير المهارات وتحسين المستوى التحصيلي، من خلال تقديم حصص إتقان إثرائية وعلاجية.

- خيارات التعليم المدمج، والتعلم عن بعد، والذي بُني في نظام المسارات على أسس من المرونة، والملاءمة والتفاعل والفعالية.
- مشروع التخرج الذي يساعد الطلبة على دمج الخبرات النظرية مع الممارسات التطبيقية.
- شهادات مهنية ومهارية تمنح للطلبة بعد إنجازهم مهامَّ محددة، واختبارات معينة بالشراكة مع جهات تخصصية.

وبالتالي فإن مسار علوم الحاسب والهندسة كأحد المسارات المستحدثة في المرحلة الثانوية يسهم في تحقيق أفضل الممارسات عبر الاستثمار في رأس المال البشري، وتحويل الطالب إلى فرد مشارك ومنتج للعلوم والمعارف، مع إكسابه المهارات والخبرات اللازمة لاستكمال دراسته في تخصصات تتناسب مع ميوله وقدراته أو الالتحاق بسوق العمل.

وتعد مادة الذكاء الاصطناعي أحد المواد الرئيسة في مسار علوم الحاسب والهندسة، حيث تسهم في توضيح مفاهيم الذكاء الاصطناعي والتقنيات المرتبطة بها بما يساعد على توظيف هذه التقنيات في عدة مجالات حياتية مثل المدن الذكية والتعليم والزراعة والطب وغيرها من المجالات الاقتصادية المتنوعة. وتهدف المادة إلى تعريف الطالب بأهمية الذكاء الاصطناعي ودوره في الجيل الرابع من الصناعة. وكذلك تركز على اللبنات الأساسية لتقنيات الذكاء الاصطناعي، ثم تتعرَّض بشكل تفصيلي للتطبيقات المتقدمة التي تتعلق بالأنظمة القائمة على القواعد وأنظمة معالجة اللغات الطبيعية. كما تشتمل هذه المادة على مشاريع وتمارين تطبيقية لما يتعلمه الطالب؛ لحل مشاكل واقعية تحاكي مستوياته المعرفية، بتوجيه وإشراف من المعلم.

ويتميز كتاب الذكاء الاصطناعي بأساليب حديثة، تتوافر فيه عناصر الجذب والتشويق، والتي تجعل الطلبة يقبلون على تعلمه والتفاعل معه، من خلال ما يقدمه من تدريبات وأنشطة متنوعة، كما يؤكد هذا الكتاب على جوانب مهمة في تعليم الذكاء الاصطناعي وتعلمه، تتمثل في:

- الترابط الوثيق بين المحتويات والمواقف والمشكلات الحياتية.
- تنوع طرائق عرض المحتوى بصورة جذابة ومشوقة.
- إبراز دور المتعلم في عمليات التعليم والتعلم.
- الاهتمام بترابط محتوياته مما يجعل منه كلًّا متكاملًا.
- الاهتمام بتوظيف التقنيات المناسبة في المواقف المختلفة.
- الاهتمام بتوظيف أساليب متنوعة في تقويم الطلبة بما يتناسب مع الفروق الفردية بينهم.

ولمواكبة التطورات العالمية في هذا المجال، فإن كتاب مادة الذكاء الاصطناعي سوف يوفر للمعلم مجموعة متكاملة من المواد التعليمية المتنوعة التي تراعي الفروق الفردية بين الطلبة، بالإضافة إلى البرمجيات والمواقع التعليمية، التي توفر للطلبة فرصة توظيف التقنيات الحديثة والتواصل المبني على الممارسة؛ مما يؤكد دوره في عملية التعليم والتعلم.

ونحن إذ نقدم هذا الكتاب لأعزائنا الطلبة، نأمل أن يستحوذ على اهتمامهم، ويُلبي متطلباتهم، ويجعل تعلّمهم لهذه المادة أكثر متعة وفائدة.

والله ولي التوفيق


بسم الله الرحمن الرحيم

الفهرس

الجزء الأول

الجزء الثاني

الجزء الأول

الوحدة الأولى
أساسيات الذكاء الاصطناعي

الوحدة الثانية
خوارزميات الذكاء الاصطناعي

الوحدة الثالثة
معالجة اللغات الطبيعية

1. أساسيات الذكاء الاصطناعي

سيتعرّف الطالب في هذه الوحدة على تاريخ الذكاء الاصطناعي (Artificial Intelligence - AI) وتطبيقاته. كما سيتعلّم المزيد حول هياكل البيانات المتقدمة، مثل الطوابير، والمُكدّسات، والقوائم المترابطة، والمُخطَّطات، والأشجار الثنائية، وسيَستخدِم هذه التراكيب لاحقًا لإنشاء مشاريع الذكاء الاصطناعي.

أهداف التعلُّم

بنهاية هذه الوحدة سيكون الطالب قادرًا على أن:

- يَذكُر معالم تاريخ الذكاء الاصطناعي (AI).
- يُعدِّد أمثلة لتطبيقات الذكاء الاصطناعي (AI).
- يَصِف عمليات هيكل بيانات المُكدَس.
- يَصِف عمليات هيكل بيانات الطابور.
- يُحدِّد الاختلافات بين هيكل بيانات المُكدَس وهيكل بيانات الطابور.
- يَصِف العمليات الرئيسة المُطبَّقة على البيانات في القائمة المترابطة.
- يَشرح استخدام هيكل بيانات الشجرة.
- يُحدِّد الاختلافات بين هيكل بيانات الشجرة وهيكل بيانات المُخطَّط.
- يَستخدِم لغة برمجة البايثون (Python) لاستكشاف هياكل البيانات المُعقَّدة.

الأدوات

- مفكَرة جوبيتر (Jupyter Notebook)

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الأول

مقدمة في الذكاء الاصطناعي

ما الذكاء الاصطناعي؟

What is Artificial Intelligence (AI)

الذكاء الاصطناعي (AI) هو أحد مجالات علوم الحاسب الآلي التي تُعنى بتصميم وتطبيق البرامج القادرة على محاكاة القدرات المعرفية البشرية. تُظهِر هذه البرامج الخصائص التي تَصِف السلوك البشري عادةً، مثل حل المشكلات، والتعلُّم، وصُنع القرارات، والاستدلال، والتخطيط، واتخاذ القرارات، إلخ.

وكلاء الذكاء الاصطناعي (AI Agents):

وكيل الذكاء الاصطناعي هو برنامج يعمل نيابةً عن المُستخدِم أو النظام في إدراك بيئته، وصنع القرارات، واتخاذ الإجراءات وفقًا لها، وقد يكون الوكيل بسيطًا أو مُعقدًا، ذاتي التحكّم أو شبه ذاتي التحكّم، أو يعمل في بيئات متنوعة، مثل المُستندة إلى الويب، أو الماديّة، أو الافتراضيّة.

الشبكات العصبية (Neural Networks):

الشبكات العصبية هي نوع من برامج الحاسب المُصمَّمة لمحاكاة طريقة عمل الدماغ البشري، وهي مكونة من خلايا وطبقات عصبية يمكنها معالجة المعلومات ونقلها.

شكل 1.1: بعض مجالات الذكاء الاصطناعي

الذكاء الاصطناعي والمجالات الأخرى AI and Other Fields

يرتبط **الذكاء الاصطناعي** (AI) ارتباطًا وثيق الصلة بعدة مجالات أخرى تشمل:

الفلسفة (Philosophy): هي أصل العلوم الحديثة، وتُعنى بدراسة المشكلات التي تمثّل أُسس الذكاء الاصطناعي، مثل أصل المعرِفة وتمثيلها، والاستدلال المُستنِد إلى القواعد والمنطق، والتحليل القائم على الأهداف، والصلة بين المعرِفة والتصرُّف.

الرياضيات (Mathematics): هي جوهر الذكاء الاصطناعي، حيث تُقدم له لبِنات البناء الأساسية مثل: المنطق، والحوسبة، ونظرية الاحتمالات.

نظرية القرار (Decision Theory): تُعنى بدراسة الخصائص المنطقية والرياضية لعملية صُنع القرار، حيث تحلّل عمليّة اتخاذ القرارات في نظام تكون فيه بيئة القرار غير واضحة، وتُطبَّق الأُطُر والأساليب النظرية في هذا المجال باستمرار لحلّ مشكلات الذكاء الاصطناعي.

علم الأعصاب (Neuroscience): يُعنى بدراسة الجهاز العصبي البشري، وقد توصل علم الأعصاب إلى نتيجة رئيسة عملت كمبدأ إرشادي للذكاء الاصطناعي، وهي أن مجموعة من الخلايا البسيطة يمكن أن تؤدي إلى نتائج مُعقدة مثل: الفكر، والعمل، والوعي. كما أن الشبكات العصبية الاصطناعية تُحاكي البُنيات العصبية الموجودة في الدماغ البشري.

علم النفس المعرفي (Cognitive Psychology): هو أحد فروع علم النفس، ويُعنى بدراسة طريقة تفكير البشر. ولطالما كان الفضل في تحقيق الانجازات والتقدّم في مجال الذكاء الصناعي راجعًا إلى الاكتشافات التي تمّ تحقيقها في هذا المجال، والتي ساعدت على توفير الرؤى التي تساعد أجهزة الحاسب على محاكاة التفكير البشري.

علوم الحاسب والهندسة (Computer Science and Engineering): تُعدُّ علوم الحاسب والهندسة حجر الأساس لتوفير البرمجيات والأجهزة اللازمة للذكاء الاصطناعي للانتقال من المبادئ النظرية إلى التطبيقات العملية. وقد واكَب التقدم في الذكاء الاصطناعي باستمرار التطورات في أنظمة التشغيل، والبرمجة، واللغات، والسعة التخزينية، والذاكرة، وقوة مُعالجة البيانات.

علم التحكّم الآلي (Cybernetics): يُعنى بدراسة الأنظمة التي تحقق الحالة المرجوة باستلام المعلومات من بيئتها وتعديل سلوكها وفقًا لذلك. الفرق الرئيس بين علم التحكّم الآلي وبين الذكاء الاصطناعيّ هو أن الأول يستخدِم الرياضيات لنمذجة الأنظمة المُغلقة التي يمكن وصفها بالكامل باستخدام متغيرات مُحدَّدة، بينما يستخدِم الذكاء الاصطناعي الاستدلال المنطقي والحوسبة للتغلب على هذه القيود ودراسة المشكلات المُعقَّدة مثل: فهم اللغة والمعلومات المرئية وتوليدهما.

علم اللُغويات (Linguistics): هو الدراسة العلمية للغة البشرية، فلطالما كان فهم اللغة البشريّة وتوليدها مجالًا رئيسًا في تطبيقات الذكاء الاصطناعي، كما أدى إلى نشوء حقول فرعية مثل: **معالجة اللغات الطبيعية** (Natural Language Processing - NLP) و**اللغويات الحاسوبية** (Computational Linguistics).

علم الرؤية (Vision Science): هو الدراسة العلمية للإدراك البصري. ويُعدّ تعليم أجهزة الحاسب كيفية فهم الصور، والرسوم المتحركة، ومقاطع الفيديو وتوليدها أحد أكثرِ تطبيقات الذكاء الاصطناعي إثارة، وتحديدًا في المجالات الفرعية للتعلُّم العميق ورؤية الحاسب.

معلومة

استُخدِم مصطلح الذكاء الاصطناعي رسميًا للمرة الأولى في عام 1956، مما يجعله أحد أحدث المجالات العلمية نسبيًا.

اختبار تورنغ Turing Test

قد يكون اختبار تورنغ هو الطريقة الأكثر شهرة لتعريف الذكاء الاصطناعي، ويعود تاريخ اقتراحه إلى عام 1950، حيث أجرى العالم تورنغ تجربة لمعرِفة ما إذا كان الحاسب ذكيًا أم لا.

وأثناء الاختبار، يتوجب على الحاسب أن يجيب عن بعض الأسئلة المكتوبة التي يقدمها **المُوجّه البشري** (Human Respondent). يُعدُّ الاختبار ناجحًا إذا لم يتمكن المُوجِّه من معرفة ما إذا كانت الإجابة مكتوبة بواسطة إنسان أم بواسطة الحاسب.

لاجتياز الاختبار بنجاح، يجب أن يتمتع الحاسب بالإمكانات الموضحة في الجدول التالي:

اختبار تورنغ (Turing Test):
يقيس اختبار تورنغ قدرة الآلة على إظهار سلوك ذكي مكافئ لسلوك الإنسان أو غير قابل للتمييز عنه.

شكل 1.2: تمثيل اختبار تورنغ

جدول 1.1: إمكانات الحاسب لاجتياز اختبار تورنغ

1	معالجة اللغات الطبيعية؛ لتمكين الحاسب من فهم الأسئلة والرد عليها.
2	تمثيل المعرفة لتنظيم المعلومات وتخزينها واسترجاعها خلال أداء الاختبار.
3	الاستدلال المُؤتمت؛ لاستخدام المعلومات المُخزَّنة للإجابة عن الأسئلة.
4	تعلُّم الآلة للتكيّف مع هياكل اللغات الجديدة مثل: بناء جُمل مختلفة، أو إيجاد مفردات لغوية مختلفة، لم يرها من قبل، أو ليست مخزّنة ضمن المعلومات.
5	رؤية الحاسب؛ حتى يتمكن من الاستجابة للإشارات البصرية التي يتلقّاها من المُوجِّه عبر وسائط نقل الصور والفيديو.
6	الروبوتية؛ حتّى يتمكّن من استقبال الأشياء التي يتلقّاها من المُوجِّه عبر المنفذ ويعالجها.

تغطي الإمكانات الموضحة بالأعلى جزءًا كبيرًا من مجال الذكاء الاصطناعي الواسع. سنستعرض هذه الإمكانات فيما يلي:

معالجة اللغات الطبيعية (NLP) هو أحد فروع الذكاء الاصطناعي الذي يَمنح أجهزة الحاسب القدرة على فهم الإنسان واللغة الطبيعية.

تمثيل المعرفة (Knowledge Representation) في الذكاء الاصطناعي يشير إلى عملية ترميز المعرفة البشرية في شكل مقروء آليا لتتمكن الأنظمة المستنِدة إلى الذكاء الاصطناعي من معالجتها واستخدامها. تأتي هذه المعرِفة في صورٍ عدة تشمل: الحقائق، والقواعد، والمفاهيم، والعلاقات، والعمليات.

الاستدلال المُؤتمت (Automated Reasoning) يُشير إلى قدرة الأنظمة المُستنِدة إلى الذكاء الاصطناعي على استنتاج المعرفة الجديدة وتقديم الاستنتاجات المنطقية وفقًا لمجموعة من القواعد والفرضيات المُقدَمة.

رؤية الحاسب (Computer Vision) هي مجال الذكاء الاصطناعي الذي يُمكِّن الحاسب من تفسير وفهم المعلومات المرئية من العالم الحقيقي، مثل الصور ومقاطع الفيديو.

الروبوتية (Robotics) هي فرع الذكاء الاصطناعي الذي يُعنى بتصميم الروبوت، وبنائه، واستخدامه. ويتضمن الجمع بين التقنيات المتنوعة مثل: تعلُّم الآلة، ورؤية الحاسب، وأنظمة التحكّم لابتكار آلات ذكية ذاتية التحكُّم أو تتطلب الحد الأدنى من التوجيه البشري.

الذكاء الاصطناعي: تاريخ مُمتَّد لتسعة عقود

Artificial Intelligence: 9 Decades of History

بالرغم من أن عمر الذكاء الاصطناعي لا يتجاوز 100 عام، إلا أنه يتمتع بتاريخ غني يَمتد منذ الأربعينيات من القرن الماضي حتى اليوم. وفيما يلي استعراض للإنجازات البارزة في مجال الذكاء الاصطناعي في كل عِقد.

الأربعينيات: البداية وأول خلية عصبية اصطناعية

1943: أُقترح النموذج الأول المبني على الخلايا العصبية الاصطناعية بحيث يمكن لكل خلية عصبية أن تكون في حالة نشطة (تشغيل) أو غير نشطة (إيقاف) وذلك وفق المحاكاة التي تتلقاها من الخلايا العصبية الأخرى المجاورة والمتصلة بها.

1948: في هذا العام ظهر روبوتان: **إلمر وإلسي** (Elmer and Elsie) وهما روبوتان ذاتيا التحكُّم، يمكنهما التنقل حول العقبات باستخدام الضوء واللمس.

خمسينات القرن الماضي: نشأة الذكاء الاصطناعي

1950: ظهر اختبار تورنغ وهو اختبار يحدِّد قدرة الآلة على إظهار سلوك ذكي مكافئ لسلوك الإنسان أو يَصعُب تمييزه عنه. إلى جانب ظهور العديد من مفاهيم الذكاء الاصطناعي الرئيسة مثل: تعلُّم الآلة، والخوارزميات الجينية، والتعلُّم المعزَّز.

1951: صُمِّم **حاسب التعزيز التناظري العصبي العشوائي** (Stochastic Neural Analog Reinforcement Computer-SNARC) كأول حاسب يعمل بالشبكات العصبية.

1958: طُوِّرت لغة **ليسب** (Lisp)، وهي لغة برمجة مُصمَّمة خصيصًا للذكاء الاصطناعي. وفي العام نفسه، نُشرت ورقة بحثية حول **متلقي المشورة الافتراضي** (Hypothetical Advice Taker)، وهو نظام الذكاء الاصطناعي القادر على التعلُّم من التجربة تمامًا مثل البشر.

الستينيات والسبعينيات من القرن الماضي: أول شتاء للذكاء الاصطناعي

1964: ظهر برنامج **إليزا** (ELIZA) وهو أول برنامج لمعالجة اللغات الطبيعية وهي الأصل الذي تفرّع منه جميع روبوتات الدردشة اليوم.

1974-1980: تُعرف هذه الفترة باسم أول شتاء للذكاء الاصطناعي. حيث انخفض تمويل مشروعات الذكاء الاصطناعي في هذه الفترة نظرًا لقلة التقدم المُحرز في هذا المجال، وانخفاض تأثيره في تطبيقات الحياة اليومية. أحد الانتقادات الرئيسة كانت عدم قدرة تقنيات الذكاء الاصطناعي على معالجة مشكلة الانفجار التوافقي التي جعلت قابلية تطبيقها محدودة على بعض المشكلات ومجموعات البيانات الصغيرة للغاية.

الثمانينيات والتسعينيات من القرن الماضي وثاني شتاء للذكاء الاصطناعي

1980: أُطلق أول نظام خبير تجاري ناجح مُصمَّم لمحاكاة القدرة على صُنع القرار مثل الإنسان.

1987-1993: تُعرف هذه الفترة باسم ثاني شتاء للذكاء الاصطناعي. فطبيعة أنظمة الذكاء الاصطناعي في المراحل المُبكرة كانت مستندة على القواعد، والتي بدورها قيدت من قابليتها للتطبيق وجعلتها غير قادرة على حل مشاكل الحياة الواقعية الرئيسة.

1997: تحقق الفوز الأول لبرنامج الذكاء الاصطناعي على بطل العالم في الشطرنج، حيث نجح الحاسب العملاق **ديب بلو** (Deep Blue) في هزيمة بطل العالم في الشطرنج **جاري كاسبارو** (Gary Kasparov).

الألفينيات: فترة الانتشار واسع النَطاق، والدعم الكبير للمكوّنات المادية والبرمجية، وتطورها

2005: طوَّرت **جامعة ستانفورد** (Stanford University) السيارة ذاتية القيادة **ستانلي** (STANLEY) التي فازت في تحدي السيارات ذاتية القيادة. كما بدأ الجيش الأمريكي الاستثمار في الروبوتات ذاتية التحكّم.

2009: استُخدمت **وحدات معالجة الرسومات** (Graphics Processing Units – GPUs) لتدريب الشبكات العصبية للتعلُّم العميق للمرة الأولى. أدى استخدام المكونات المادية المتخصصة إلى تسارع وتيرة تدريب الشبكات المُعقَّدة على مجموعات كبيرة جدًا من البيانات، مما أدى بدوره إلى عصرٍ جديد من التعلُّم العميق والذكاء الاصطناعي.

العقدين الثاني والثالث من القرن الحادي والعشرين: العصر الذَهبي

2011: هزم نظام الإجابة على الأسئلة المعروف باسم **واتسون** (Watson) أفضل لاعبين في العالم في برنامج المسابقات الأميركي **جيوباردي** (Jeopardy)، حيث تمكّن واتسون من فهم الأسئلة والإجابة عليها بنجاح، مما شكّل طفرة في استخدام الذكاء الاصطناعي لِفهم اللغة الطبيعية.

2012: ظهر نظام الذكاء الاصطناعي الذي يُترجِم فوريًا اللغة الإنجليزية المنطوقة إلى اللغة الصينية المنطوقة.

2021: ظهر نظام القيادة الذاتية الكامل الذي يَستخدِم الشبكات العصبية المُدرَّبة على سلوك مئات الآلاف من السائقين.

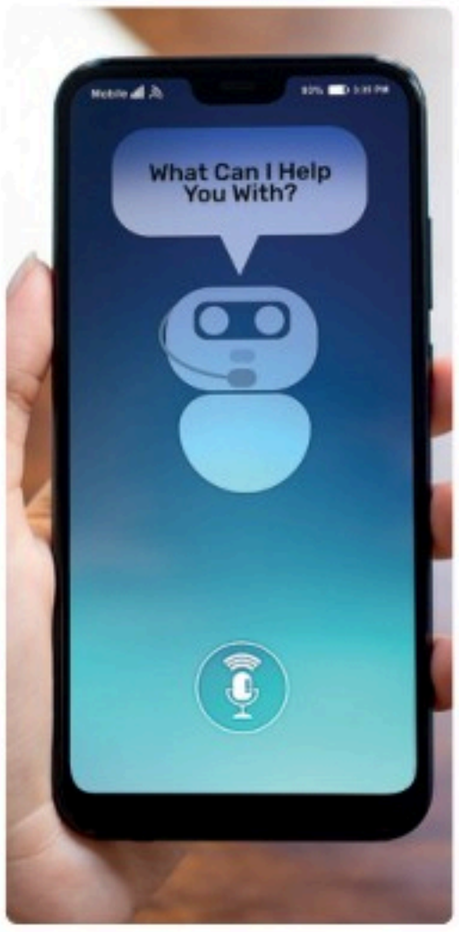
2022: ظهر **روبوت دردشة المُحوِّل التوليدي مُسبق التدريب** (Generative Pre-trained Transformer - ChatGPT) وهو روبوت الدردشة المبني على مجموعة كبيرة من النماذج اللغوية. هذه النماذج مُهيئَّة بدقة باستخدام كلٍ من تقنيات التعلُّم المُوجّه والمُعزّز لمحاكاة المحادثات البشرية.

تطبيقات الذكاء الاصطناعي Applications of AI

الذكاء الاصطناعي هو تقنية سريعة التطور لديها القدرة على تحوُّل مجموعة واسعة من القطاعات والصناعات. في هذه الوحدة ستستكشف تطبيقات الذكاء الاصطناعي المتنوعة، وكيفية استخدامها في إجراء تحسينات وابتكارات في مجموعة متنوعة من القطاعات والصناعات.

المساعدون الافتراضيون Virtual Assistants

واحدة من أشهر تطبيقات الذكاء الاصطناعي هي تطبيقات المساعدين الافتراضيين الذين يمكنهم التواصل مع المُستخدِمين عبر التفاعلات النصيّة أو الصوتية، ويمكن الوصول إليهم عبر الأجهزة المادية مثل: الهواتف الذكية، والأجهزة اللوحية، أو مكبرات الصوت الذكية، ويمكن استخدامهم لأداء مجموعة واسعة من المهام مثل: إعداد التذكيرات، والإجابة على الأسئلة، وتشغيل الوسائط الصوتية، وطلب المنتجات أو الخدمات. أحد الأمثلة الأكثر شهرة على تطبيقات الذكاء الاصطناعي في هذا المجال هو **سيري** (Siri) من شركة **آبل** (Apple). وهناك شركات أخرى طوّرت مساعدين افتراضيين: مثل **أليكسا** (Alexa) التابع لشركة **أمازون** (Amazon)، و**المساعد الافتراضي لقوقل** (Google's Assistant)، و**كورتانا** (Cortana) التابع لشركة **مايكروسوفت** (Microsoft). وبمرور الوقت تطوَّرت قدرة هذه التطبيقات على الفهم والاستجابة لعدد متزايد من الأوامر والاستفسارات والرد عليها. على سبيل المثال، يمكن استخدام المساعد الافتراضي للتحكم في الأجهزة المنزلية الذكية مثل: التحكم في درجة الحرارة، والإضاءة، والأجهزة الكهربائية. وقد يتمثل المساعد الافتراضي في صورة روبوتات الدردشة المتخصصة المُصمَّمة عادةً لتقديم المعلومات والإجابة على الأسئلة في مجال محدَّد، على سبيل المثال، في تطبيقات خدمة العملاء تُستخدَم روبوتات الدردشة المبنية على تقنية الذكاء الاصطناعي في الإجابة على أسئلة العملاء حول المنتجات أو الخدمات، وتحديد المشكلات وعلاجها، وتقديم المعلومات حول طلباتهم وحساباتهم. يمكن الوصول إلى روبوتات الدردشة عبر مجموعة واسعة من القنوات مثل: مواقع الويب، وتطبيقات المراسلة، ووسائل التواصل الاجتماعي، ويمكنها تقديم خدمات المساعدة على مدار الساعة طوال أيام الأسبوع. يمكنك الاطلاع على مثال لأحد تطبيقات روبوت الدردشة في الشكل 1.3.



شكل 1.3: المحادثة مع روبوت الدردشة

الروبوتية Robotics

ارتبط الذكاء الاصطناعي منذ بداياته بالروبوتية، فإذا كان الروبوت هو التصوير المادي للكائن الاصطناعي، فإنّ الذّكاء الاصطناعي يمثل دماغ الروبوت، ويَمنحه القدرة على الشعور بالبيئة من حوله، واتخاذ القرارات، والتكيف مع الظروف المتغيرة. كما يمكن للروبوتات الذكية تطبيق هذه الإمكانات والقدرات لأداء مجموعة واسعة من المهام دون التدخل البشري، مثل: مهام التصنيع، والاستكشاف، والبحث والإنقاذ، والعديد من المهام الأخرى. الشكل 1.4 يوضِّح خط تجميع روبوتي في مصنع سيارات.

شكل 1.4: خط تجميع روبوتي في مصنع سيارات

إنّ أحد أقدم الأمثلة على تطبيق الذكاء الاصطناعي في الروبوتية هو تطوير روبوتات المصانع المُستخدَمة في أداء المهام مثل: اللحام، والدهانات، والتجميع. منذ ذلك الحين، تطوَّر استخدام الذكاء الاصطناعي في الروبوتية إلى حد كبير، مع تطور الخوارزميات المتقدمة واستخدام تعلُّم الآلة لتحسين أداء الروبوت. وكانت إحدى الإنجازات البارزة في استخدام الذكاء الاصطناعي في الروبوتية تطوير الروبوتات البشرية، مثل: روبوت **هوندا أسيمو** (Honda's ASIMO) وقد سُمّي بذلك اختصارًا لمفهوم **الخطوة المتقدمة في النقل الإبداعي** (Advanced Step in Innovative Mobility) والذي قُدِّم للمرة الأولى في عام 2000 وكان قادرًا على السير وأداء المهام الأساسية.

الروبوتات الشبيهة بالبشر Humanlike Robots

طوّرت شركة **الدبران روبوتكس** (Aldebaran Robotics) الروبوتان الشبيهان بالبشر **بيبر** (Pepper) و**ناو** (Nao)، اللّذانِ صُمِّما لأغراض البحث والتطوير في مجال التفاعل بين الإنسان والروبوت، وقد استُخدما على نطاق واسع في مجالات البحث، والتعليم، والترفيه. أمّا **بيبر** (Pepper) فهو روبوت اجتماعي مُصمَّم للتفاعل مع الأشخاص بصورة طبيعية باستخدام كاميرا، وميكروفونات، ومُستشعرات اللمس لإدراك البيئة من حوله، والاستجابة لتصرفات وعواطف الأشخاص من حوله. يتمتع هذا الروبوت بالعديد من الخصائص التي تسمح له بالتعرّف على الوجوه، وفهم الكلام، والاستجابة للإيماءات. الشكل 1.5 يعرض صورة للروبوت بيبر. أمّا **ناو** (Nao) فهو روبوت مُدمج أصغر حجمًا مُصمَّم للتفاعل مع البشر، ويحتوي هذا الروبوت مثل السابق على مجموعة من المُستشعرات التي تسمح له بإدراك البيئة من حوله، إلى جانب الكاميرات، والميكروفونات للتعرّف على الكلام والوجوه. ويمتاز هذا الروبوت بأنّه قابل للتخصيص والبرمجة بدرجة توافقية عالية، مما يجعله الخيار الأمثل للباحثين والدارسين الذين يرغبون في دراسة وتطوير تطبيقات جديدة للروبوتات الشبيهة بالبشر.

شكل 1.5: الروبوت بيبر

في عام 2017 كانت الروبوت صوفيا (Sophia) أول روبوت يحصل على الجنسية السعودية، وفي عام 2023 طورت المملكة العربية السعودية سارة (Sarah)، وهي الروبوت التفاعلي الأول من نوعه.

السيارات ذاتية القيادة Self-Driving Cars

كان الإنجاز المهم الآخر هو تطوير السيارات ذاتية القيادة كما في الشكل 1.6 وهي سيارات تَستخدِم الذكاء الاصطناعي للانتقال عبر الطرق واتخاذ القرارات حول كيفية التفاعل الآمن مع المركبات الأخرى ومع المُشاة. أحد المتطلبات الرئيسة لهذه التطبيقات هو القدرة على معالجة البيانات المرئية مثل الصور ومقاطع الفيديو وفهمها، ويشار إلى ذلك عادة باسم **رؤية الحاسب** (Computer Vision)، ويمكن استخدام خوارزميات رؤية الحاسب للتعرُّف على الكائنات، والأشخاص، والخصائص الأخرى في الصور ومقاطع الفيديو، إلى جانب فهم سياق المحتوى ومعناه. ولهذا المجال العديد من التطبيقات غير الروبوتية مثل: التعرُّف على الوجه، وإدارة المحتوى، وتحليل الوسائط. وكان أحد الإنجازات البارزة في استخدام الذكاء الاصطناعي في تحليل الصور ومقاطع الفيديو تطوير خوارزميات التعلُّم العميق، التي يُمكنها تحليل كميات كبيرة من البيانات وتحديد الأنماط المُعقَّدة في الصور ومقاطع الفيديو.

شكل 1.6: سيارة ذاتية القيادة

المجالات التي تأثرّت بالذكاء الاصطناعي Industries Affected by AI

التعليم Education

على مدى العقود القليلة الماضية، كانت هناك العديد من الإنجازات الرئيسة لاستخدام الذكاء الاصطناعي في التعليم. بما في ذلك تطوير أنظمة التدريس القائمة على الذكاء الاصطناعي التي تَستخدِم تقنيات معالجة اللغات الطبيعية للتفاعل مع الطلبة وتقديم الملاحظات حول أعمالهم. ثم ظهرت منصّات التعلُّم التكيُّفي التي تَستخدِم خوارزميات تعلُّم الآلة لتخصيص العملية التعليمية لكل طالب استنادًا إلى نقاط قوته وضعفه. بعدها، طُوِّرت أنظمة التصحيح القائمة على الذكاء الاصطناعي التي تَستخدِم خوارزميات معالجة اللغات الطبيعية وتعلُّم الآلة لتصحيح الواجبات المكتوبة وتقديم الملاحظات. وفي الآونة الأخيرة، حدث دمج بين المساعدين الافتراضيين وروبوتات الدردشة في مجال التعليم لتقديم الدعم المخصص للطلبة والإجابة على أسئلتهم بشكل فوري. يمكن استخدام الذكاء الاصطناعي لتحليل البيانات حول أداء الطلبة، وخياراتهم المفضلة في التعليم، وغيرها من العوامل الأخرى اللازمة لوضع خطط تعليمية مخصصة للطلبة، وتقديم التوصيات بشأن المواد أو الأنشطة التي من المرجح أن تفيدهم بفعالية.

مزايا الذكاء الاصطناعي في التعليم

AI benefits in education

- يوفّر وقت المُعلِّمين والأساتذة الجامعيين.
- يُمكن **لمُعلّمي الذكاء الاصطناعي** (AI Tutors) مساعدة الطلبة.
- يساعد المُعلِّم على أن يصبح معلّمًا محفزا.
- تُقدَّم الوظائف المُستنِدة على الذكاء الاصطناعي الملاحظات لكل من الطلبة والمُعلِّمين.

الرعاية الصحية Healthcare

الرعاية الصحية هي مجال آخر حقَّق تقدمًا كبيرًا بفضل الذكاء الاصطناعي. كانت الابتكارات الأولى في صورة الأنظمة التشخيصية القائمة على الذكاء الاصطناعي واستخدامه في اكتشاف الأدوية. ثم دمجه مع السجلات الصحية الإلكترونية لاستخراج المعلومات ذات الصلة، وفي العِقد الثاني من القرن الحادي والعشرين، طُوِّرت أنظمة التطبيب عن بُعد القائمة على الذكاء الاصطناعي. واليوم، يُساعد الذكاء الاصطناعي الحديث في إنشاء خطط علاجية مُخصصة للمريض، واستخدام أجهزة تقنية يرتديها لمتابعة حالتة الصحية. ويلعب الذكاء الاصطناعي دورًا كبيرًا في مجال الرعاية الصحية، فهو يُمكِّن الأطباء ومقدّمي خدمات الرعاية الصحية الآخرين من تحليل كميات كبيرة من البيانات واتخاذ القرارات حول رعاية المرضى. قد تأتي البيانات من مصادر متنوعة مثل: السجلات الطبية، والفحوصات المعملية، وكذلك الصور مثل: الأشعة السينية أو الأشعة المقطعية، كما تُستخدَم خوارزميات رؤية الحاسب الحديثة بصورة متكررة للكشف عن التشوهات والمساعدة في التشخيص الطبي.

شكل 1.7: تحليل البيانات الصحية

الزراعة والنمذجة المُناخية Agriculture and Climate Modeling

يُستخدَم الذكاء الاصطناعي في الزراعة لتحسين إنتاج المحاصيل الزراعية ورفع كفاءة الممارسات الزراعية. ويتحقق ذلك بالتحليل المستمر للبيانات حول حالة التربة، وأنماط الطقس، والعوامل الأخرى للتنبؤ بأفضل وقت لزراعة المحاصيل الزراعية وريِّها وحصادها. كما يُستخدَم الذكاء الاصطناعي في مراقبة المحاصيل طوال الوقت وتحديد المشكلات التي قد تصيبها مثل: الآفات أو الأمراض، مما يسمح للمزارعين باتخاذ اللازم قَبل أن تؤثر تلك المشكلات على جودة المحاصيل الزراعية، وأحد الأمثلة المُبتكرة على تطبيقات الذكاء الاصطناعي في الزراعة هو استخدام خوارزمية صُنع القرارات البسيطة لتحسين مواعيد الري. ومن الإنجازات الرئيسة الأخرى استخدام شبكات المُستشعرات لمراقبة المحاصيل الزراعية، ومعايرة التطبيقات العلاجية الرئيسة مثل الأسمدة والمبيدات. وفي الآونة الأخيرة، استُخدِمت الصور المُلتقطة بالطائرات المُسيَّرة والأقمار الصناعية لتحليل المحاصيل الزراعية على نطاق واسع، كما في الشكل 1.8 الذي يعرض طائرة مُسيَّرة تُستخدَم لتسميد أحد الحقول.

شكل 1.8: التسميد باستخدام الطائرات المُسيَّرة

أمّا النمذجة المُناخية فهي مجال آخر يرتبط ارتباطًا وثيقًا بالزراعة، وقد تأثر كثيرًا بالذّكاء الاصطناعي الذي بدأت تطبيقاته في هذا المجال في وقت مُبكر، مع تطوير أنظمة التنبؤ بالطقس القائمة عليه. ولاحقًا، استُخدِم الذكاء الاصطناعي لتحليل كميات كبيرة من البيانات حول التغيرات المُناخية والتنبؤ بالأنماط المستقبلية، وتأتي هذه البيانات من مصادر متنوعة، بما في ذلك صور الأقمار الصناعية، وملاحظات محطات الطقس، والمحاكاة الحاسوبية. واليوم، يُستخدَم الذكاء الاصطناعي في مجموعة واسعة من تطبيقات النمذجة المُناخية مثل: التنبؤ بآثار التغيرات المُناخية على مناطق محدَّدة، وتحليل وفهم أسباب الظواهر الجوية المتطرفة وفهمها، ووضع الاستراتيجيات الفعّالة للتخفيف من التغيرات المُناخية أو التكيّف معها.

الطاقة Energy

أثَّر الذكاء الاصطناعي كثيرًا على مجال الطاقة، وذلك عن طريق تمكين الشّركات من ترشيد استخدامها وتقليل الهَدْر، وتحسين الكفاءة. أحد الأمثلة على ذلك استخدام خوارزميات تعلُّم الآلة لتحليل البيانات حول استخدامات الطاقة وتحديد طرائق تقليل الهَدْر وترشيد الاستهلاك. في التسعينيات من القرن الماضي، استُخدِم الذكاء الاصطناعي للتنبؤ بموارد الطاقة المُتجددة وتحسين استخدامها. وكان تطورًا رئيسًا مكَّن شركات الطاقة من التخطيط بصورة أفضل لدمج موارد الطاقة المتجددة في عملياتها.

شكل 1.9: الطاقة الكهربائية النظيفة من الألواح الكهروضوئية الشمسية

شهد العِقد الأول من القرن الحادي والعشرين دمج الذكاء الاصطناعي في الشبكات الذكية، التي تستخدِم خوارزميات تعلُّم الآلة في تحليل البيانات حول استخدام الطاقة وضبط العرض والطلب طوال الوقت، حيث ساهم ذلك في تحسين كفاءة توزيع الطاقة والحد من الهدر، وفي العِقد الثاني من القرن الحادي والعشرين، استُخدِم الذكاء الاصطناعي لتطوير أنظمة تخزين الطاقة التي يمكنها تخزين الطاقة الزائدة واستخدامها عند الحاجة. وكان تطورًا رئيسًا مكَّن شركات الطاقة من إدارة الاستخدام المُتقطِّع بشكل أفضل لموارد الطاقة المتجددة مثل: الطاقة الشمسية وطاقة الرياح. يعرض الشكل 1.9 الألواح الكهروضوئية الشمسية، وفي السنوات الأخيرة، استُخدِم الذكاء الاصطناعي لزيادة كفاءة استخدام الطاقة بتحليل البيانات حول استخدام الطاقة وتحديد طرائق الحد من الهدر، وشمل ذلك تطوير الأنظمة المُستنِدة على الذكاء الاصطناعي التي تُستخدَم في تحسين استخدام الطاقة في المباني، والمصانع، ومن قِبل كبار مُستهلكي الطاقة. كما استُخدِم الذكاء الاصطناعي في صناعة النفط والغاز لتحليل البيانات حول الحفر والإنتاج وتحسين العمليات.

تطبيق القانون Law Enforcement

يُستخدَم الذكاء الاصطناعي بكثافة في مجال تطبيق القانون للتنبؤ بالجرائم والحيلولة دون وقوعها. وعلى وجه التحديد، يُستخدَم الذكاء الاصطناعي لتحليل البيانات من مصادر مختلفة، مثل: سجلات الجرائم، ووسائل التواصل الاجتماعي، وكاميرات المراقبة لتحديد أنماط وتوجُهات الأنشطة الإجرامية والتنبؤ بها. على سبيل المثال طُوِّر الذكاء الاصطناعي في التعرُّف على الوجوه (شكل 1.10). ولاحقًا، دُمِج في أنظمة إرسال قوات الشرطة واستُخدم لمراقبة منصّات وسائل التواصل الاجتماعي بحثًا عن التهديدات المحتملة. وفي الآونة الأخيرة، استُخدِم الذكاء الاصطناعي لتطوير طائرات مُسيَّرة لمراقبة وتحليل تسجيلات الفيديو من الكاميرات التي يرتديها ضباط تطبيق القانون. كما لعِب الذكاء الاصطناعي دورًا كبيرًا في تمكين الجهات المسؤولة من تحليل كميات كبيرة من البيانات، وتحديد الأنماط والتوجهات، واتخاذ القرارات المُستنيرة حول كيفية منع الجريمة والتصدي لها.

شكل 1.10: تقنيات التعرُّف على الوجه وتحديد الهوية الشخصية

تمرينات

1

حدِّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. وضع علماء الرياضيات الأُسس لفهم الحوسبة والمنطق حول الخوارزميات.	○	○
2. يُحدِّد اختبار تورنغ ما إذا كان الحاسب يتمتع بسلوك شبيه بالإنسان أم لا.	○	○
3. كان إلمر (Elmer) وإلسي (Elsie) أول روبوتين يتنقلان حول العقبات باستخدام الضوء واللمس.	○	○
4. استُخدِم الذكاء الاصطناعي فقط في الروبوتات المُستخدَمة في الصناعات التحويلية.	○	○
5. لم يكن للذكاء الاصطناعي أي تأثير يُذكر في مجال الطاقة.	○	○

2 ما الذكاء الاصطناعي (AI)؟

3 اشرح بإيجاز بعض تطبيقات الذكاء الاصطناعي المُستخدَمة في الحياة اليومية.

4 وضّح بعض الأحداث التاريخية الرئيسة التي أثّرت في تطور الذكاء الاصطناعي في الأربعينيات والخمسينيات من القرن الماضي.

5 اشرح كيف استُخدمت التطبيقات التجارية تقنيات الذكاء الاصطناعي للمرة الأولى في العِقد الثاني من القرن الحادي والعشرين.

6 لخّص كيفية استخدام تطبيقات الذكاء الاصطناعي في التصدي لتغيرات المُناخ عبر النمذجة المُناخية والتحسينات في مجال الطاقة.

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الثاني

هياكل البيانات في الذكاء الاصطناعي

أهمية هياكل البيانات في الذكاء الاصطناعي

The Importance of Data Structures in AI

هياكل البيانات (Data Structure):
هياكل البيانات هي تقنية لتخزين وتنظيم البيانات في الذاكرة لاستخدامها بكفاءة.

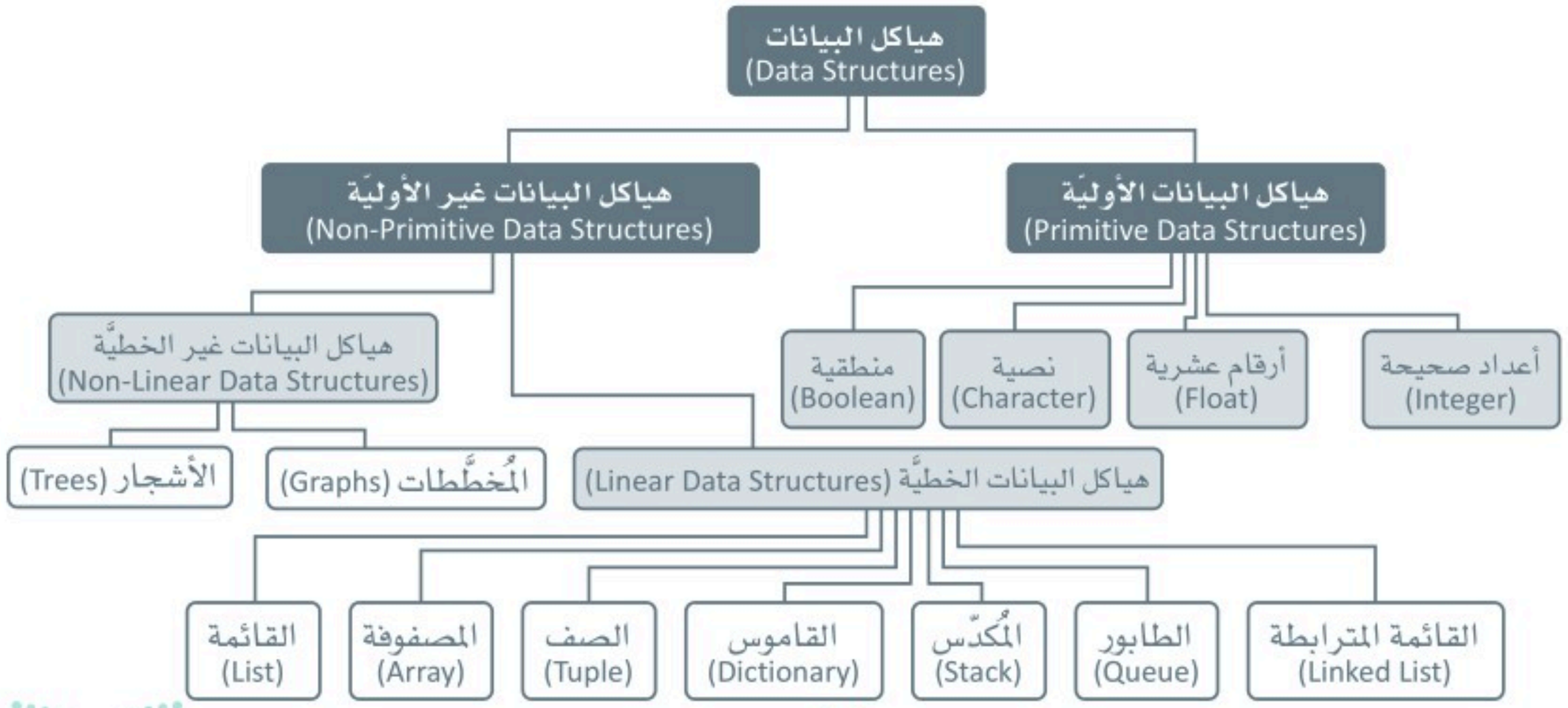
للبيانات أهمية كبرى في مجالات الذكاء الاصطناعي؛ لأنها الأساس المُستخدَم في تدريب نماذج تعلُّم الآلة، حيث تُحدِّد جودة البيانات وكميّتها المتوافرة دقة وفعالية نماذج الذكاء الاصطناعي. ودون بيانات كافية وذات صلة، لن تتعلّم خوارزميات الذكاء الاصطناعي الأنماط، ولن تقوم بالتنبؤات، ولن تتمكن من أداء المهام بفاعلية. وبالتالي، تلعب البيانات دورًا رئيسًا في تشكيل قدرات وإمكانات صُنع القرار لدى أنظمة الذكاء الاصطناعي. هياكل البيانات لها أهمية كبيرة في الذكاء الاصطناعي؛ لأنها توفر طريقة فعَّالة لتنظيم وتخزين البيانات، كما تسمح باسترجاع ومعالجة البيانات بكفاءة. وكذلك، تُحدِّد مدى تعقيد وكفاءة الخوارزميات المُستخدَمة في معالجة البيانات، وبالتالي تؤثر مباشرةً على أداء أنظمة الذكاء الاصطناعي. على سبيل المثال، يمكن تحسين سرعة وقابلية خوارزميات الذكاء الاصطناعي للتوسُّع باستخدام هياكل البيانات المناسبة، مما يجعلها أكثر ملاءمة للتطبيقات في العالم الحقيقي. وكذلك، تساعد هياكل البيانات المُصمَّمة جيدًا في تقليل استخدام الذاكرة وزيادة كفاءة الخوارزميات، لتمكينها من معالجة مجموعات أكبر من البيانات.

تُخزِّن أجهزة الحاسب البيانات وتُعالجها بسرعة ودقة فائقتين. لذلك، من الضروري تخزين البيانات بكفاءة، وتوفير الوصول إليها بطريقة سريعة. يمكن تصنيف هياكل البيانات على النحو التالي:

- هياكل البيانات الأوليّة.
- هياكل البيانات غير الأوليّة.

يُطلَق على البيانات البسيطة كذلك البيانات الأوليّة، أو الخام، أو الأساسية.

يوضِّح المُخطَّط في الشكل 1.11 تصنيف هياكل البيانات.



شكل 1.11: مُخطَّط هياكل البيانات

هياكل البيانات الأوليّة Primitive Data Structures

يُشار إلى هياكل البيانات الأوليّة باسم هياكل البيانات الأساسية في لغة البايثون، ويحتوي هذا النوع من الهياكل على قيم بسيطة للبيانات. تُخبر أنواع البيانات البسيطة المُترجِّم بنوع البيانات التي يُخزِّنها. هياكل البيانات الأوليّة في لغة البايثون هي:

- **الأرقام** (Numbers): تُستخدَم الأرقام لتمثيل البيانات الرقمية وهي:
  - الأعداد الصحيحة
  - الأرقام العشرية
- **السلاسل النصيّة** (Strings): السلاسل النصيّة هي مجموعات من الأحرف والكلمات.
- **البيانات المنطقية** (Boolean): تكون قيم البيانات المنطقية إما صحيحة أو خاطئة.

تُستخدَم أنواع مختلفة من هياكل البيانات لتطبيقات الحاسب ومهامّه المختلفة بناءً على ما يتطلبه المشروع والقيود المفروضة على الذاكرة.

هياكل البيانات غير الأوليّة Non-Primitive Data Structures

هياكل البيانات غير الأوليّة هي هياكل متخصصة تُخزِّن مجموعة من القيم. يكتبها المُبرمِّج ولا تُعرَّف بلغة البايثون مثل هياكل البيانات الأوليّة.

يمكن تقسيم هياكل البيانات غير الأوليّة كذلك إلى نوعين:

- **هياكل البيانات الخطيَّة أو المُتسلسلة** (Linear or Sequential Data Structures): تُخزِّن هياكل البيانات الخطيّة عناصر البيانات في تسلسل معيّن.
- **هياكل البيانات غير الخطيَّة** (Non-linear Data Structures): لا تحتوي هياكل البيانات غير الخطيَّة على ارتباط تسلسلي بين عناصر البيانات، ويُمكن ربط أي زوج أو مجموعة من عناصر البيانات معًا، والوصول إليها دون تسلسل مُحدَّد.

هياكل البيانات الخطيَّة Linear Data Structures

تُخزِّن هياكل البيانات الخطيَّة عناصر البيانات في تسلسل معيّن. في هذا الدرس ستتعلّم بعض هياكل البيانات الخطيّة مثل: **المُكدّس** (Stack) **والطابور** (Queue)، وهما نوعان من هياكل البيانات الأكثر استخدامًا في الحياة اليومية.

شكل 1.12: كومة من الكتب كمثال واقعي على المُكدّس

المُكدّس Stack

يمكن تمثيل المُكدّس في الواقع بمجموعة من الكتب رُصَّت فوق بعضها البعض، كما هو موضّح في الشكل 1.12. فلإنشاء تلك المجموعة، عليك أن تضع الكتب بعضها فوق بعض، وعندما تريد استخدام أحد الكتب، عليك أخذ الكتاب من أعلى المجموعة. وللوصول إلى الكتب الأخرى عليك إنزال الكتب من أعلى المجموعة.

قد يكون حجم المُكدّس ثابتًا أو متغيّرًا ديناميكيًا. تُطبّق لغة البايثون المُكدّسات باستخدام القوائم.

قاعدة المُضاف آخرًا يَخرُج أولًا (Last In First Out (LIFO) Rule):

آخر عنصر مُضاف يمكن الوصول إليه أولًا.

العمليات في المُكدّس Operations on the stack

هناك عمليتان رئيستان في المُكدّس:

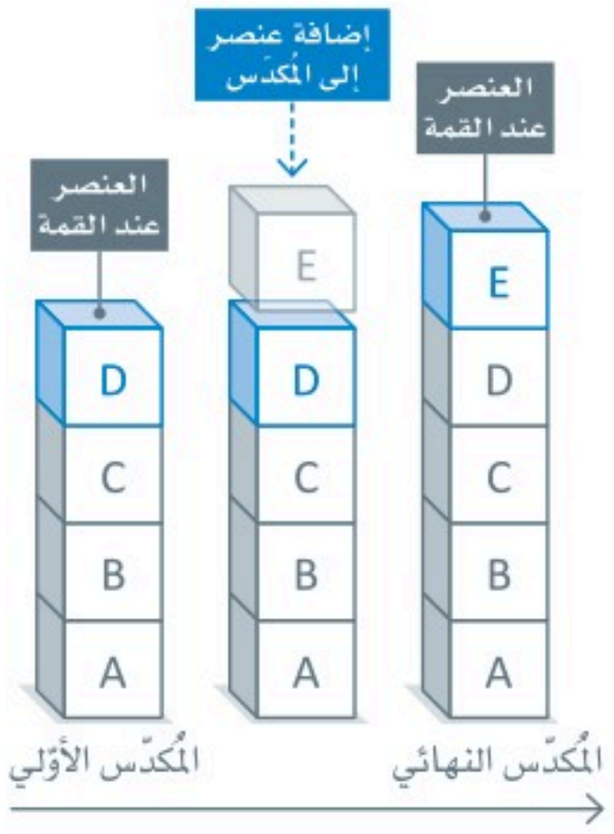
- **إضافة عنصر** (Push): تُستخدَم العملية لإضافة عنصر في قمة المُكدّس.
- **حذف عنصر** (Pop): تُستخدَم العملية لحذف عنصر من قمة المُكدّس.

عملية إضافة عنصر Push operation

يُطلَق على عملية إضافة عنصر جديد إلى المُكدّس اسم **إضافة عنصر** (Push).

يَستخدِم المُكدّس مؤشّرًا يُطلق عليه مؤشر **الأعلى** (Top)، ويُشير إلى العنصر الموجود في قمة المُكدّس، وعند إضافة عنصر جديد إلى المُكدّس:

- تزداد قيمة مؤشر الأعلى بقيمة واحدة لإظهار الموقع الجديد الذي سيُضاف العنصر فيه.
- يُضاف العنصر الجديد إلى قمة المُكدّس.



شكل 1.13: عملية إضافة عنصر إلى المُكدّس

فَيْض المُكدّس Stack Overflow

يتميز المُكدّس بسعة تخزينية مُحدَّدة تعتمد على ذاكرة الحاسب. إذا كانت الذاكرة ممتلئة، فإن إضافة عنصر جديد سينتج عنها مشكلة **فَيْض المُكدّس** (Stack Overflow). ويقصد بها تجاوز السعة؛ لذا يجب التحقق من امتلاء ذاكرة المُكدّس قبل إضافة أي عنصر جديد.

عملية حذف عنصر Pop operation

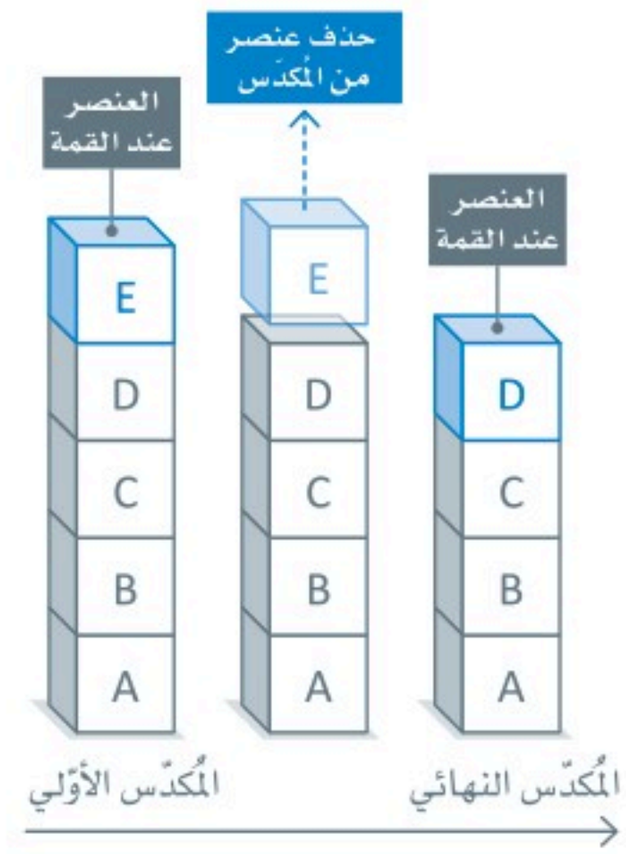
يُطلق على عملية حذف عنصر من المُكدّس اسم **حذف عنصر** (Pop).

عند حذف عنصر من المُكدّس:

- يُحذَف العنصر من قمة المُكدّس.
- تنخفض قيمة مؤشر الأعلى بقيمة واحد لإظهار العنصر التالي عند قمة المُكدّس.

غَيْض المُكدّس Stack Underflow

إذا كنت ترغب في حذف عنصر من المُكدّس، عليك التّحقّق أولًا من أن المُكدّس يحتوي على عنصر واحد على الأقل؛ فإذا كان المُكدّس فارغًا، سينتج عن ذلك مشكلة **غَيْض المُكدّس** (Stack Underflow) ويقصد بها الانخفاض عن الحد الأدنى للسعة.



شكل 1.14: عملية حذف عنصر من المُكدّس

المُكدّس في لغة البايثون Stack in Python

تُمثَّل المُكدّسات في لغة البايثون باستخدام القوائم التي بدورها تُقدِّم بعض العمليات التي يُمكن تطبيقها مباشرةً على المُكدّسات.

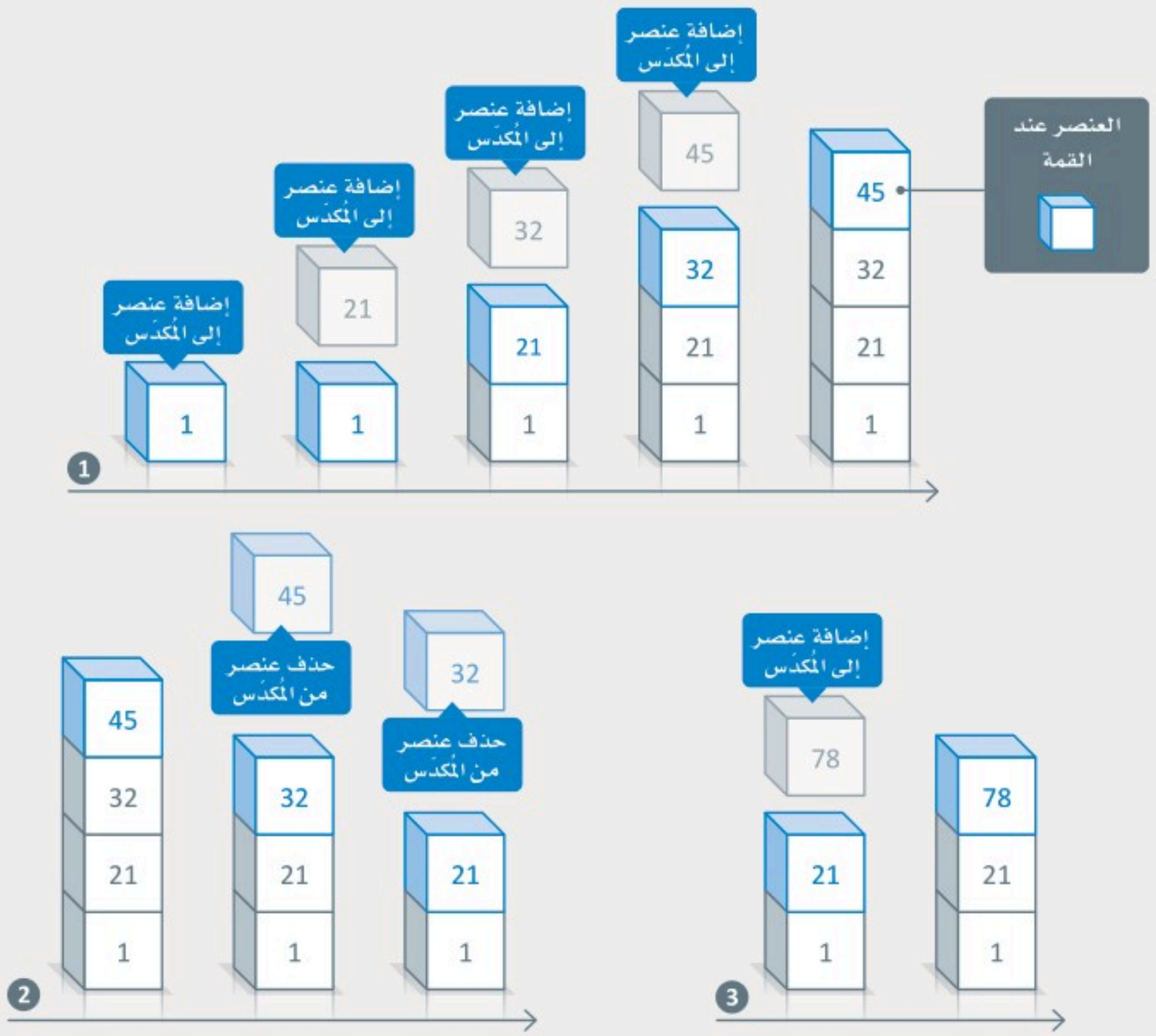
جدول 1.2: طرائق المُكدّس

الطريقة	الوصف
listName.append(x)	إضافة العنصر x إلى نهاية القائمة.
listName.pop( )	حذف العنصر الأخير من القائمة.

تُطبَّق عملية إضافة عنصر للمُكدّس في لغة البايثون باستخدام دالة append.

ستشاهد مثالًا على تطبيق المُكدّس في لغة البايثون:

1. أنشئ المُكدّس لتخزين مجموعة من الأرقام (1، 21، 32، 45).
2. استخدِم عملية **حذف عنصر** (Pop) من المُكدّس مرتين لحذف العنصرين الأخيرين (32، 45) من المُكدّس.
3. استخدِم عملية **إضافة عنصر** (Push) إلى المُكدّس لإضافة عنصر جديد (78) إلى المُكدّس.



شكل 1.15: مثال على المُكدّس

مفكرة جوبيتر Jupyter Notebook

في هذه الوحدة ستكتب برنامجًا بلغة البايثون باستخدام **مفكرة جوبيتر** (Jupyter Notebook). وهي تطبيق الويب المُستخدَم لإنشاء المُستندات الحاسوبية ومشاركتها. كل مُستند يُسمى مفكرة، ويحتوي على المقطع البرمجي الذي كتبتَه، والتعليقات، والبيانات الأوليّة والمُعالجة، وتصوّرات البيانات. ستَستخدِم الإصدار **غير المُتصل بالإنترنت** (Offline) من مفكرة جوبيتر، وأسهل طريقة لتثبيته محليًا هي من خلال **أناكوندا** (Anaconda) وهي منصة توزيع مفتوحة المصدر للطلبة والهواة، ويمكنكَ تنزيلها وتثبيتها من الرابط التالي:

https://www.anaconda.com/products/distribution

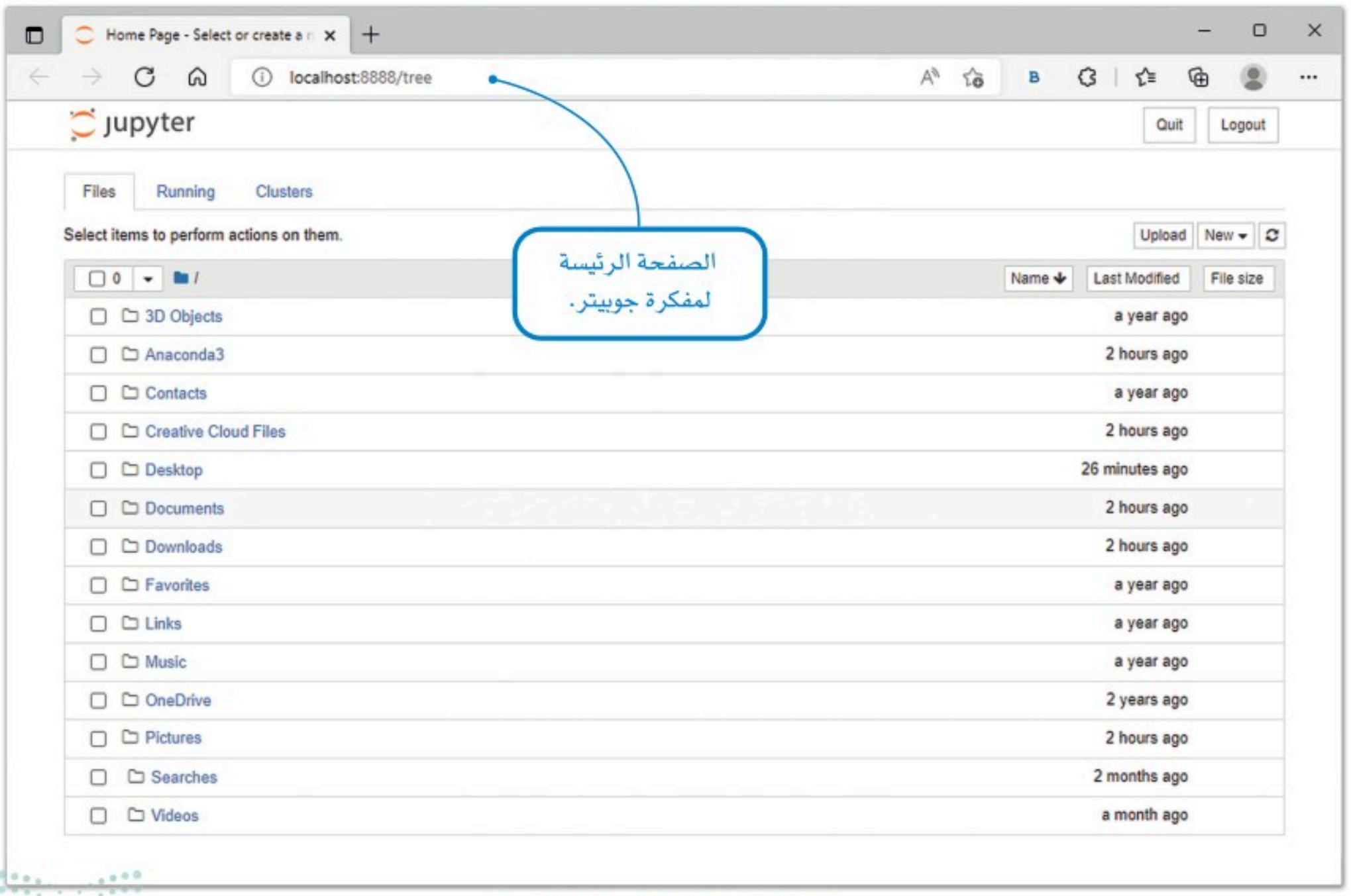
وسيتم تثبيت لغة البايثون ومفكرة جوبيتر تلقائيًا.

لفتح مفكرة جوبيتر (Jupyter Notebook):

> اضغط على Start (بدء)، 1 ثم اضغط على Anaconda3 (أناكوندا 3). 2

> اختر Jupyter Notebook (مفكرة جوبيتر). 3

> ستظهر الصفحة الرئيسة لمفكرة جوبيتر في المتصفح.



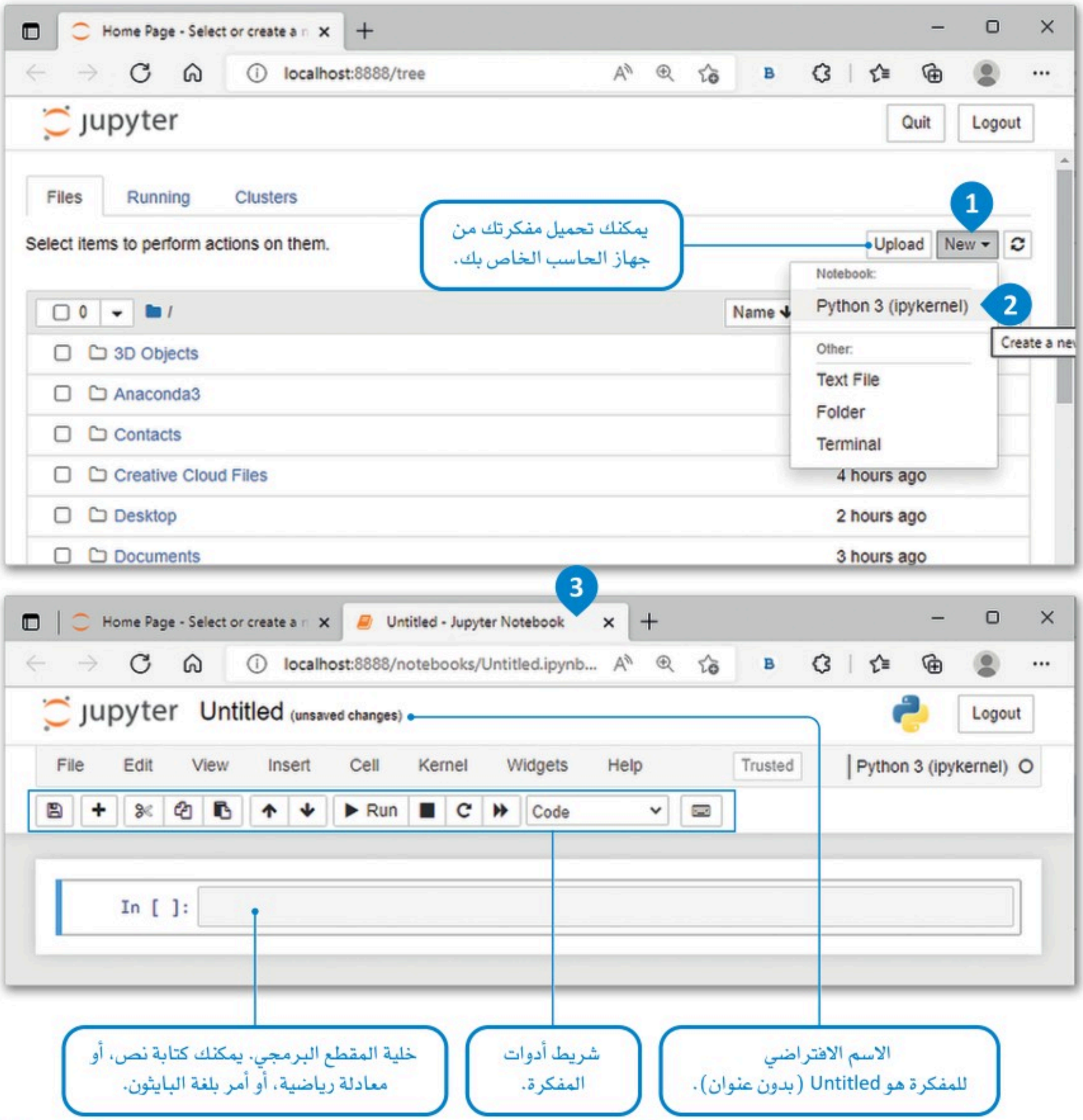
شكل 1.16: الصفحة الرئيسة لمفكرة جوبيتر

لإنشاء مفكرة جوبيتر جديدة:

> في الزاوية اليمنى العلوية من شاشتك، اضغط على New (جديد). 1

> حدِّد Python 3 (ipykernel) (بايثون 3). 2

> سيتم فتح المفكرة الخاصة بك في علامة تبويب جديدة في المتصفح الخاص بك. 3



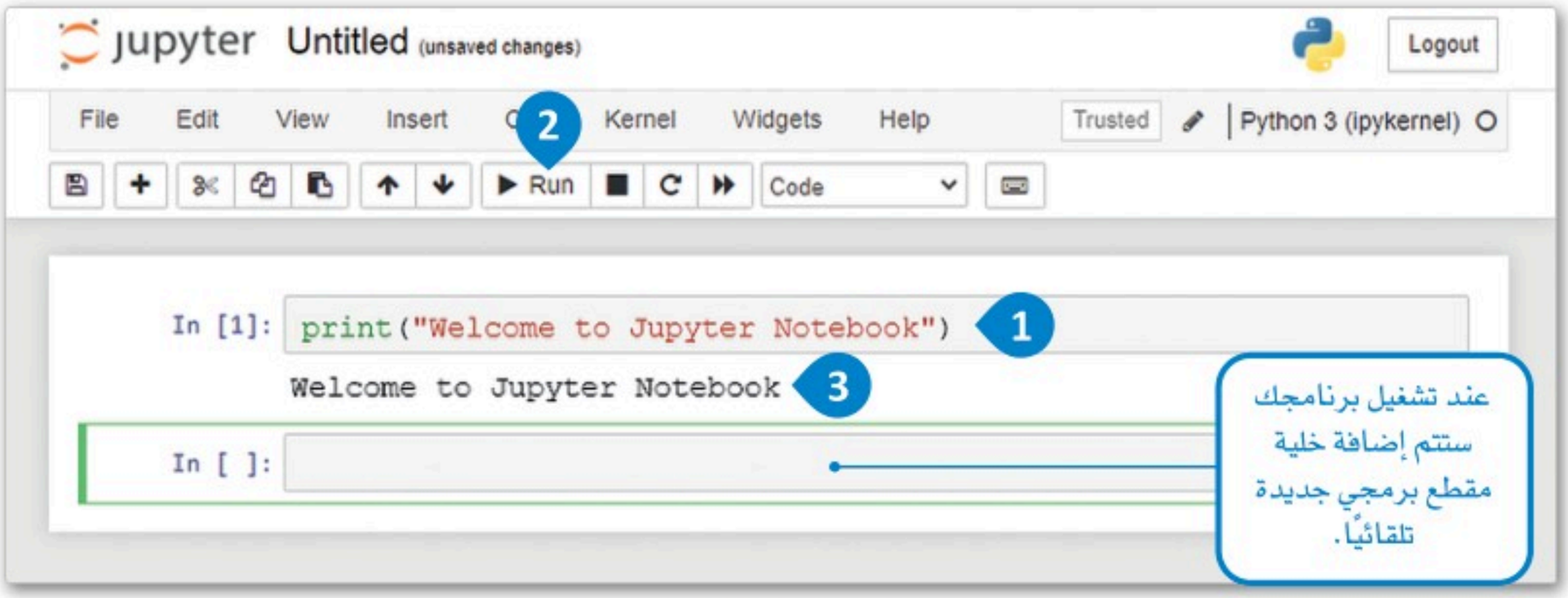
شكل 1.17: إنشاء مفكرة جوبيتر جديدة

الآن وبعد أن أصبحت مُفكرتك جاهزة، حان الوقت لكتابة برنامجك الأول وتشغيله فيها.

يمكنك الحصول على العديد من الخلايا المختلفة التي تحتاجها في نفس المفكرة حيث تحتوي كل خلية على مقطعها البرمجي الخاص.

لانشاء برنامج في مفكرة جوبيتر:

> أكتب الأوامر داخل خلية المقطع البرمجي. 1
> اضغط على Run (تشغيل). 2
> سيتم عرض النتيجة تحت الأوامر. 3



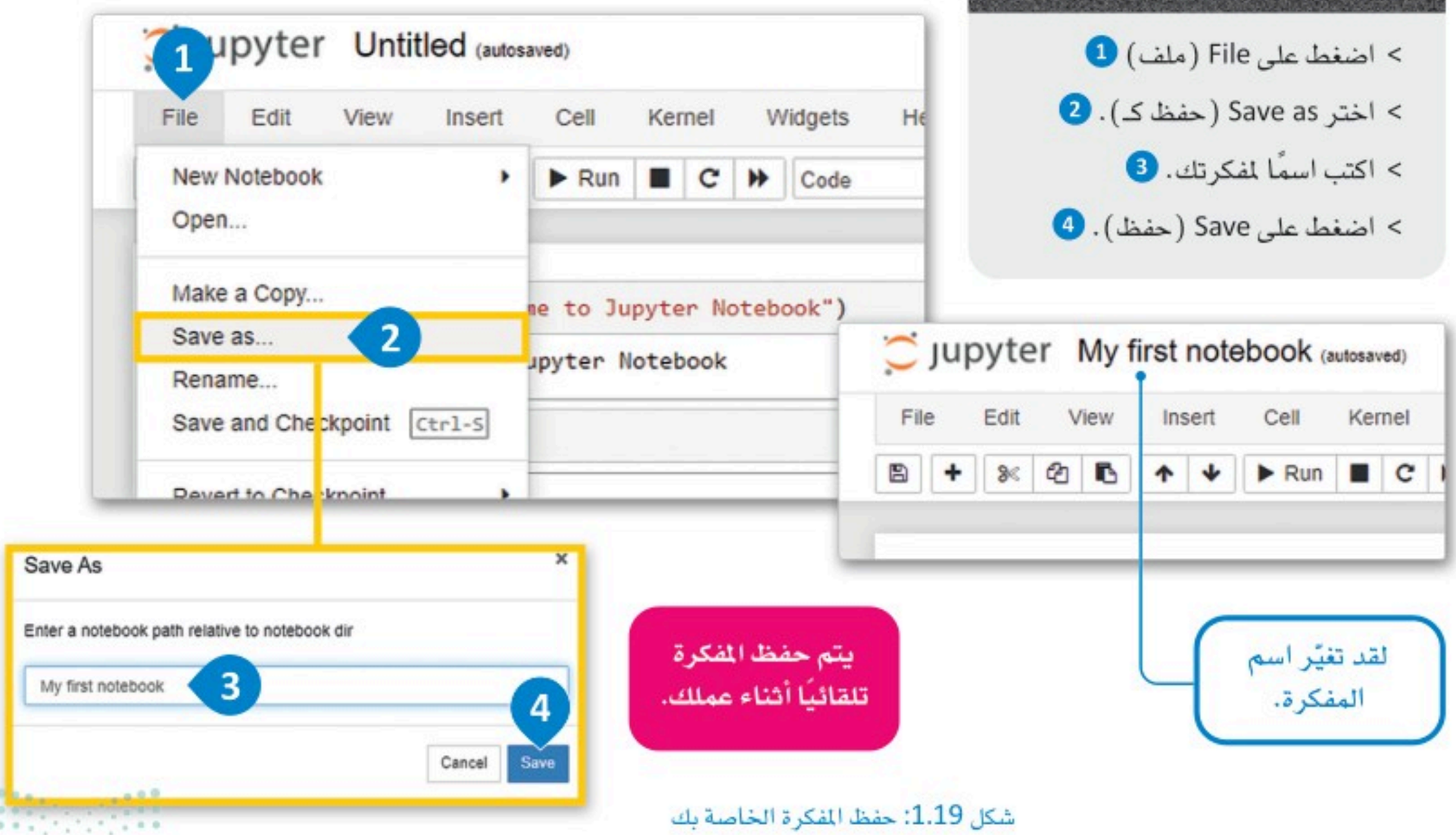
شكل 1.18: إنشاء برنامج في مفكرة جوبيتر

يُمكنك تشغيل برنامجك بالضغط على Shift + Enter ↵.

حان الوقت لحفظ مفكّرتك.

لحفظ المفكرة الخاصة بك:

> اضغط على File (ملف) 1
> اختر Save as (حفظ كـ). 2
> اكتب اسمًا لمفكرتك. 3
> اضغط على Save (حفظ). 4



شكل 1.19: حفظ المفكرة الخاصة بك

لتشاهد المثال في الشكل 1.15 في مفكرة جوبيتر:

1. أنشئ المُكدّس لتخزين مجموعة من الأرقام (1، 21، 32، 45).
2. استخدِم عملية **حذف عنصر** (Pop) من المُكدّس مرتين لحذف العنصرين الأخيرين منه.
3. استخدِم عملية **إضافة عنصر** (Push) إلى المُكدّس لإضافة عنصر جديد إليه.

```
myStack=[1,21,32,45]
print("Initial stack: ", myStack)
print(myStack.pop())
print(myStack.pop())
print("The new stack after pop: ", myStack)
myStack.append(78)
print("The new stack after push: ", myStack)
```

تُستخدَم الدالة print(myStack.pop()) لعرض القيم المُسترجَعة من دالة myStack.Pop().

```
Initial stack: [1, 21, 32, 45]
45
32
The new stack after pop: [1, 21]
The new stack after push: [1, 21, 78]
```

```
myStack=[1,21,32,45]
print("Initial stack:", myStack)
a=len(myStack)
print("size of stack",a)
# empty the stack
for i in range(a):
   myStack.pop()
print(myStack)
myStack.pop()
```

تُستخدَم الدالة len لعرض طول المُكدّس.

يُستخدَم هذا الأمر لحذف كل العناصر من المُكدّس.

```
Initial stack: [1, 21, 32, 45]
size of stack 4
[]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [3], in <cell line: 9>()
      7     myStack.pop()
      8 print(myStack)
----> 9 myStack.pop()

IndexError: pop from empty list
```

يظهر الخطأ؛ لأن المُكدّس فارغ وأنت كتبت أمر حذف عنصر منه.

خطأ الفهرس IndexError

ستلاحظ ظهور خطأ عندما كتبتَ أمر حذف عنصر من المُكدّس الفارغ وتسبب هذا في **غَيْض المُكدّس** (Stack Underflow). عليك دومًا التحقق من وجود عناصر في المُكدّس قبل محاولة حذف عنصر منه.

في البرنامج التالي ستنشئ مُكدّسًا جديدًا وتضيف العناصر إليه، أو تحذفها منه، سيظهر بالبرنامج قائمة تطلب منك تحديد الإجراء الذي تود القيام به في كل مرة.

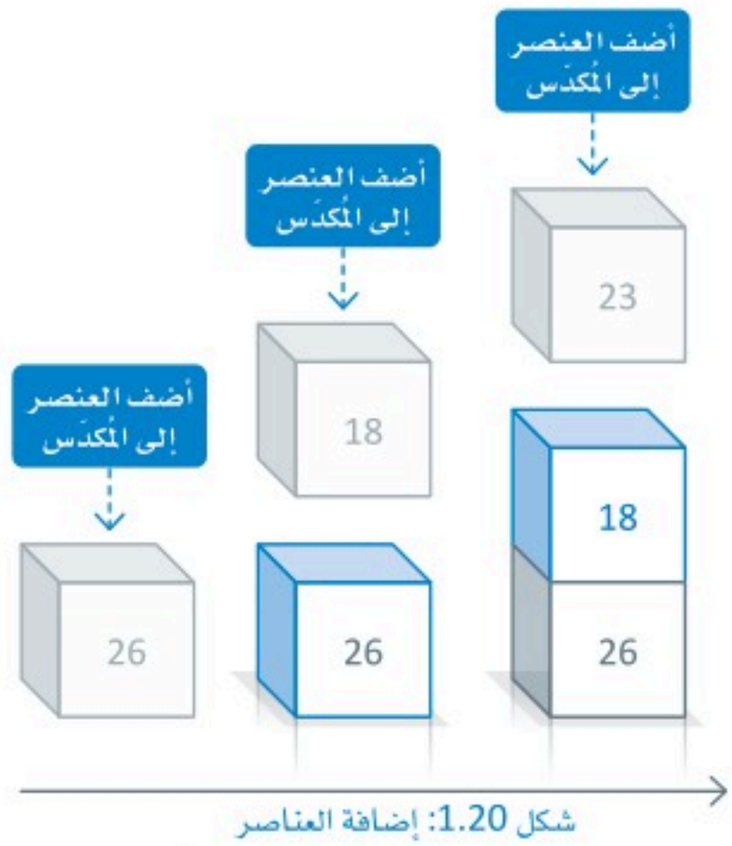
- لإضافة عنصر إلى المُكدّس، اضغط على الرقم 1 من قائمة البرنامج.
- لحذف عنصر من المُكدّس، اضغط على الرقم 2 من قائمة البرنامج.
- للخروج من البرنامج، اضغط على الرقم 3 من قائمة البرنامج.

```
def push(stack,element):
    stack.append(element)
def pop(stack):
    return stack.pop()
def isEmpty(stack):
    return len(stack)==0
def createStack():
    return []

newStack=createStack()
while True:
    print("The stack so far is:",newStack)
    print("----------------------------")
    print("Choose 1 for push")
    print("Choose 2 for pop")
    print("Choose 3 for end")
    print("----------------------------")
    choice=int(input("Enter your choice: "))
    while choice!=1 and choice!=2 and choice!=3:
        print ("Error")
        choice=int(input("Enter your choice: "))
    if choice==1:
        x=int(input("Enter element for push: "))
        push(newStack,x)
    elif choice==2:
        if not isEmpty(newStack):
            print("The pop element is:",pop(newStack))
        else:
            print("The stack is empty")
    else:
        print("End of program")
        break;
```

نَفِّذ البرنامج السابق كما يلي:

- أنشئ مُكدّسًا من ثلاثة أرقام.
- أضف العناصر إلى المُكدّس.

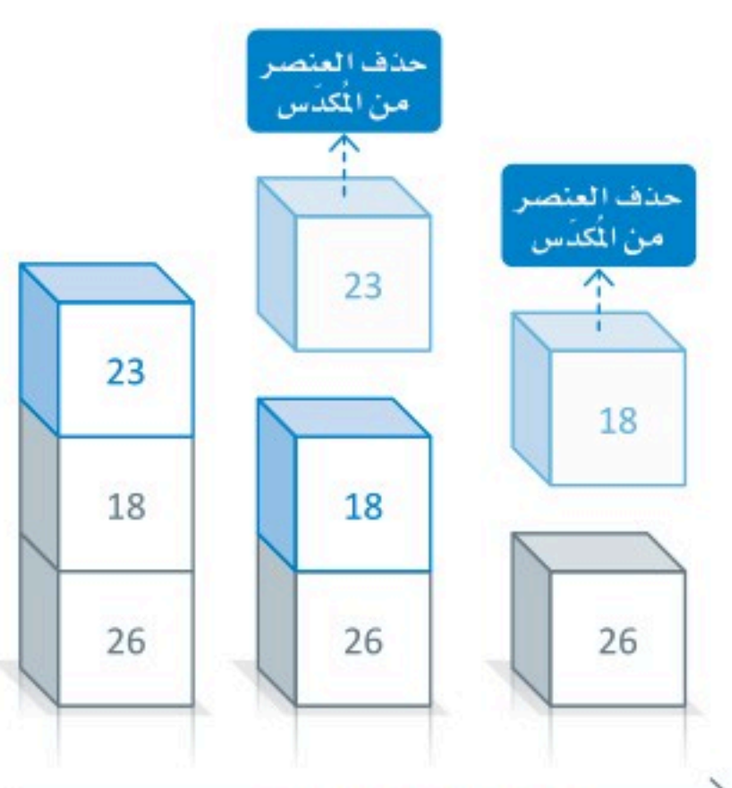
```
The stack so far is: []
----------------------------
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 1
Enter element for push: 26
The stack so far is: [26]
----------------------------
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 1
Enter element for push: 18
The stack so far is: [26, 18]
----------------------------
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 1
Enter element for push: 23
The stack so far is: [26, 18, 23]
----------------------------
```



شكل 1.20: إضافة العناصر

يمكنك الآن حذف عنصرين من المُكدّس، ثم الخروج من البرنامج.

```
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 2
The pop element is: 23
The stack so far is: [26, 18]
----------------------------
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 2
The pop element is: 18
The stack so far is: [26]
----------------------------
Choose 1 for push
Choose 2 for pop
Choose 3 for end
----------------------------
Enter your choice: 3
End of program
```



شكل 1.21: حذف العناصر

الطابور Queue

هيكل البيانات التالي الذي سنستعرضه هو الطابور. تُصادِف عادةً طوابير في حياتك اليومية. الطابور الأكثر شيوعًا هو طابور انتظار السيارات عند إشارة المرور. عندما تتحول إشارة المرور إلى اللون الأخضر، ستكون السيارة التي دخلت إلى الطابور أولًا هي نفسها التي تخرج منه أولًا. الطابور هو هيكل البيانات الذي يَتبع قاعدة **المُضاف أولًا يَخرُج أولًا** (First In First Out - FIFO)، مما يعني أن كل عنصر في الطابور يُقدَّم بالترتيب نفسه الذي وصل به إلى الطابور.

قاعدة المُضاف أولًا يَخرُج أولَ (First In First Out (FIFO) Rule):
العنصر الأول المُضاف إلى القائمة يُعالَج أولًا، والعنصر الأحدث يُعالَج آخرًا.

الفرق بين المُكدَس والطابور هو أنه في المُكدَس تتم إضافة وحذف العنصر من نفس الجانب، وفي الطابور تتم الإضافة من جانب، بينما يتم الحذف من الجانب الآخر. وهكذا، عند الحذف في المُكدَس، يُحذف العنصر المُضاف آخرًا، بينما في الطابور، يُحذف العنصر المُضاف أولًا.

العمليات في الطابور Operations on the Queue

هناك عمليتان رئيستان في الطابور:

- **إضافة عنصر للطابور** (Enqueue): تُستخدَم العملية لإضافة عنصر في آخر الطابور.
- **حذف عنصر من الطابور** (Dequeue): تُستخدَم العملية لحذف عنصر من مقدمة الطابور.

مؤشرات الطابور Queue Pointers

يحتوي الطابور على مؤشرين:

- **المؤشر الأمامي** (Front Pointer): يُشير إلى العنصر الأول في الطابور.
- **المؤشر الأخير** (Rear Pointer): يُشير إلى العنصر الأخير في الطابور.

المؤشر (Pointer):
المؤشر هو مُتغير يُخزِّن أو يُشير إلى عنوان مُتغير آخر. المؤشر يشبه رقم الصفحة في فهرس الكتاب الذي يُسهِّل على القارئ الوصول إلى المحتوى المطلوب.

الفهرس (Index):
الفهرس هو رقم يُحدِّد موضع العنصر في هيكل البيانات.

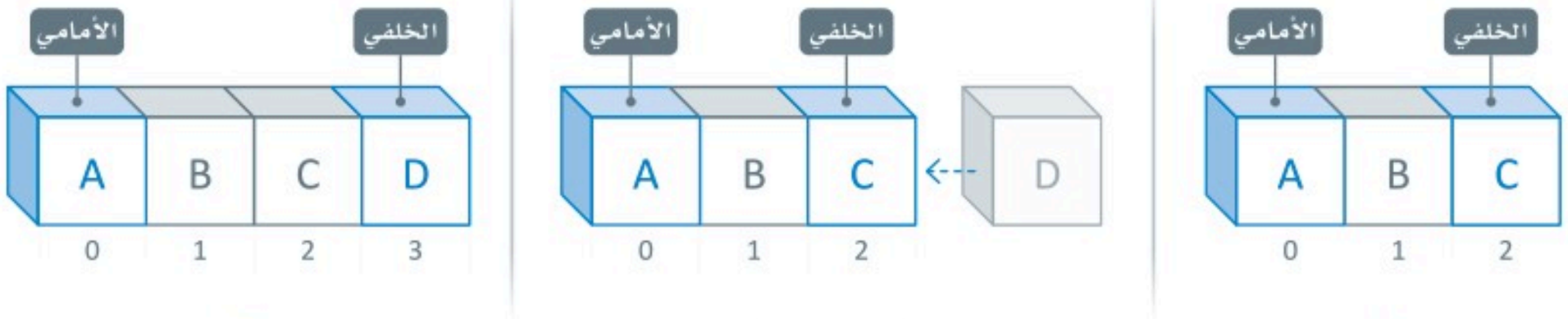
شكل 1.22: العمليات في الطابور

عملية إضافة عنصر للطابور Enqueue Operation

يُطلق على عملية إضافة عنصر جديد إلى الطابور اسم **إضافة عنصر للطابور** (Enqueue). لإضافة عنصر جديد إلى الطابور:

- تتم زيادة قيمة المؤشر الخلفي بقيمة واحد بحيث يشير إلى موضع العنصر الجديد الذي سيُضاف.
- تتمّ إضافة العنصر.

لا يمكنك إضافة عنصر أو حذفه من وسط الطابور.



شكل 1.23: عملية إضافة عنصر للطابور

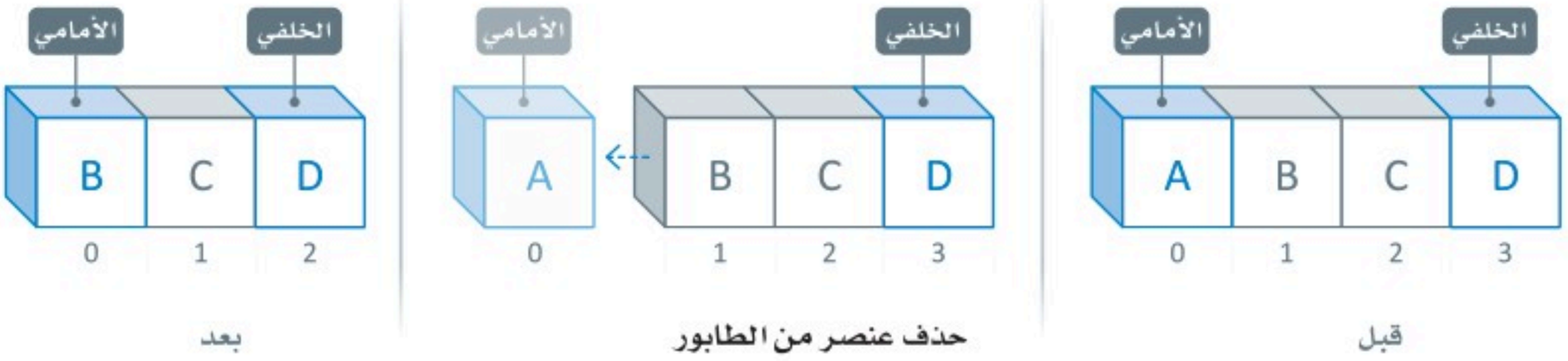
عملية حذف عنصر من الطابور Dequeue Operation

يُطلق على عملية حذف عنصر من الطابور اسم **حذف عنصر من الطابور** (Dequeue).

لحذف عنصر من الطابور:

- يُحذف العنصر المُشار إليه بالمؤشر الأمامي.
- تتم زيادة قيمة المؤشر الأمامي بقيمة واحد بحيث يشير إلى العنصر الجديد التالي في الطابور.

قبل أي إجراء عليك التحقق مما إذا كانت هناك مساحة فارغة في الطابور لإضافة عنصر جديد، أو توافر عنصر واحد على الأقل لتصديره.



شكل 1.24: عملية حذف عنصر من الطابور

الطابور في لغة البايثون Queue in Python

يمكن تمثيل الطابور بعدة طرائق متنوعة في لغة البايثون منها **القوائم** (Lists). ويرجع ذلك إلى حقيقة أن القائمة تمثل مجموعة من العناصر الخطيّة، كما يمكن إضافة عنصر في نهاية القائمة وحذف عنصر من بداية القائمة.

ستتعلّم فيما يلي الصيغ العامة لبعض العمليات التي يمكن تنفيذها على الطابور:

جدول 1.3: طرائق الطابور

الطريقة	الوصف
listName.append(x)	تضيف العنصر x إلى القائمة التي تمثل الطابور.
listName.pop(0)	تحذف العنصر الأول من القائمة.

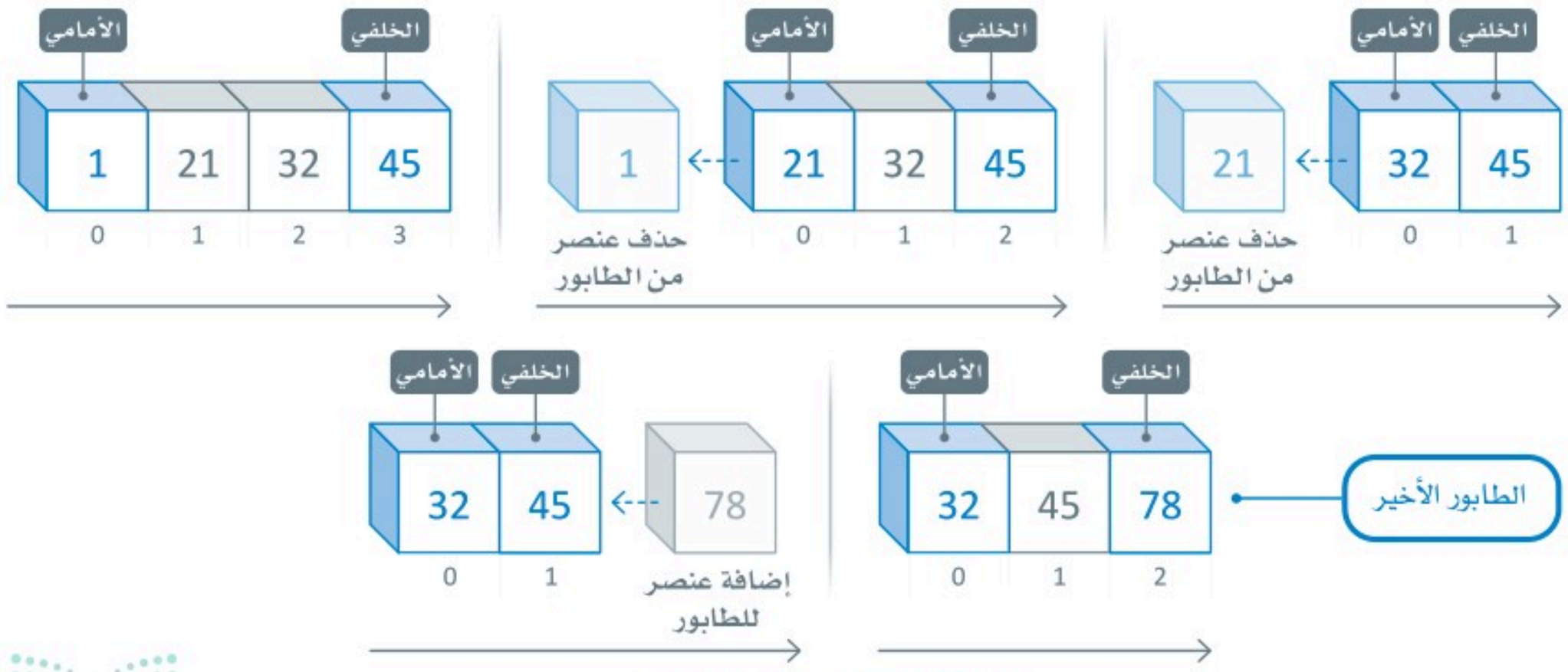
تُستخدَم طريقة listName.pop( ) لكلٍ من هياكل بيانات المُكدّس والطابور. عندما تُستخدَم مع المُكدّس، لا تتطلب الطريقة أي مُعامِل. بينما تتطلّب الطريقة إضافة صفر إلى المُعامل عندما تُستخدم مع الطابور: listName.pop(0). الفرق بين الدالتين مُوضّح في الجدول 1.4 أدناه.

جدول 1.4: طريقة listName.pop( ) مقابل طريقة listName.pop(0)

الطريقة	الوصف
listName.pop( )	إذا كان مُعامِل الدالة فارغًا، يُحذف العنصر الأخير من نهاية القائمة التي تمثل المُكدّس.
listName.pop(0)	إذا كان مُعامِل الدالة صفرًا، يُحذف العنصر الأول من القائمة التي تمثل الطابور.

سنستعرض لك مثالًا على تطبيق الطابور في لغة البايثون:

- أنشئ طابورًا لتخزين مجموعة من الأرقام (1، 21، 32، 45).
- استخدم عملية حذف عنصر من الطابور مرتين لحذف العنصرين الأوَّلَين منه.
- استخدم عملية إضافة عنصر إلى الطابور لإضافة عنصر جديد إليه.



شكل 1.25: مثال توضيحي لمفهوم الطابور

لبرمجة الخطوات الموضحة بالأعلى بلغة البايثون، ستَستخدِم قائمة البايثون لتنفيذ هيكل الطابور، كما فعلت في المُكدّس.

```
myQueue=[1,21,32,45]
print("Initial queue: ", myQueue)
myQueue.pop(0)
myQueue.pop(0)
print("The new queue after pop: ", myQueue)
myQueue.append(78)
print("The new queue after push: ", myQueue)
```

```
Initial queue: [1, 21, 32, 45]
The new queue after pop: [32, 45]
The new queue after push: [32, 45, 78]
```

لكي تشاهد ما قد يحدث عندما تحاول حذف عنصر من طابور فارغ، عليكَ أولًا أن تُفرِغ الطابور من العناصر.

```
myQueue=[1,21,32,45]
print("Initial queue: ", myQueue)
a=len(myQueue)
print("size of queue ",a)
# empty the queue
for i in range(a):
   myQueue.pop(0)
print(myQueue)
myQueue.pop(0)
```

```
Initial queue: [1, 21, 32, 45]
size of queue 4
[]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [6], in <cell line: 9>()
      7     myQueue.pop()
      8 print(myQueue)
----> 9 myQueue.pop()

IndexError: pop from empty list
```

ظهر الخطأ؛ لأنك حاولت حذف عنصر من طابور فارغ.

عليك أن تتحقق دومًا من وجود عناصر في الطابور قبل محاولة حذف عنصر منه.

تطبيقات على الطابور Queue Applications

أحد الأمثلة على تطبيقات الطابور في علوم الحاسب هو طابور الطباعة. على سبيل المثال، لديك معمل حاسب به 30 جهاز حاسب متصلين بطابعة واحدة. عندما يرغب الطلبة في طباعة المُستندات، ستشكّل مهامّ الطّباعةِ طابورًا لمعالجتها وفق قاعدة **المُضاف أولًا يَخرُج أولًا** (FIFO)، أي أنّ تلك المهام ستُنجزُ بالترتيب الزمني الذي أُرسلَت به إلى الطابعة. المهمة المُرسلة أولًا ستُطبع قبل المهمة المُرسلة بعدها ولن تُطبع المهمة في نهاية الطابور قبل طباعة كل المهام التي قبلها. عندما تنتهي الطابعة من أحد الأوامر، ستبحث في الطابور لمعرِفة ما إن كانت هناك أوامر أخرى لمعالجتها.

المُكدّس والطابور باستخدام وحدة الطابور النمطية
Stack and Queue Using Queue Module

يمكن اعتبار القائمة في لغة البايثون بمثابة طابور وكذلك مُكدّس. تُقدِّم لغة البايثون **الوحدة النمطية للطابور** (Queue Module) وهي طريقة أخرى لتنفيذ هيكلَيّ البيانات الموضحين. تتضمن الوحدة النمطية للطابور بعض الدوال الجاهزة للاستخدام التي يمكن تطبيقها على كل من المُكدّس والطابور.

جدول 1.5: طرائق الوحدة النمطية للطابور

الطريقة	الوصف
queueName=queue.Queue()	تنشئ طابورًا جديدًا اسمه queueName.
queueName.put(x)	تضيف العنصر x إلى الطابور.
queueName.qsize()	تعود بقيمة حجم الطابور.
queueName.get()	تعرض وتحذف العنصر الأول من الطابور والعنصر الأخير من المُكدّس.
queueName.full()	تعود بقيمة True (صحيح) إن كان الطابور ممتلئًا، وقيمة False (خطأ) إن كان الطابور فارغًا، ويمكن تطبيقها على المُكدّس كذلك.
queueName.empty()	تعود بقيمة True (صحيح) إن كان الطابور فارغًا والقيمة False (خطأ) إن كان الطابور ممتلئًا، ويمكن تطبيقها على المُكدّس كذلك.

تُستخدَم طرائق الوحدة النمطية للطابور مع كلٍ من المُكدّس والطابور.

ستَستخدِم وحدة الطابور النمطية لإنشاء طابور.

في هذا المثال عليك:

- استيراد الوحدة النمطية **للطابور** (Queue) لاستخدام طرائق الطابور.
- إنشاء طابور فارغ باسم myQueue (طابوري).
- إضافة العناصر a، b، c، d، e إلى الطابور myQueue (طابوري).
- طباعة عناصر الطابور.

```
from queue import *

myQueue = Queue()
# add the elements in the queue
myQueue.put("a")
myQueue.put("b")
myQueue.put("c")
myQueue.put("d")
myQueue.put("e")

# print the elements of the queue
for element in list(myQueue.queue):
   print(element)
```

عليك استيراد الوحدة النمطية للطابور في بداية المقطع البرمجي.

```
a
b
c
d
e
```

أنشئ طابورًا مُكوَّنًا من خمس قيم يقوم المُستخدِم بإدخالها أثناء تنفيذ البرنامج، ثم اطبع هذه القيم، وفي النهاية اطبع حجم الطابور.

```
from queue import *

myQueue = Queue()

# the user enters the elements of the queue for i in range(5):
for i in range(5):
   element=input("enter queue element: ")
   myQueue.put(element)

# print the elements of the queue
for element in list(myQueue.queue):
   print(element)
print ("Queue size is: ",myQueue.qsize())
```

```
enter queue element: 5
enter queue element: f
enter queue element: 12
enter queue element: b
enter queue element: 23
5
f
12
b
23
Queue size is: 5
```

أنشئ برنامجًا للتحقق مما إذا كان الطابور فارغًا أم ممتلئًا.

```
from queue import *

myQueue = Queue()

myQueue.put("a")
myQueue.put("b")
myQueue.put("c")
myQueue.put("d")
myQueue.put("e")

checkFull=myQueue.full()
print("Is the queue full? ", checkFull)
checkEmpty= myQueue.empty()
print("Is the queue empty? ", checkEmpty)
```

```
Is the queue full? False
Is the queue empty? False
```

كما ذُكِر من قبل فإن الوحدة النمطية للطابور تحتوي على بعض الوظائف الجاهزة للاستخدام مع المُكدّس أو الطابور. الجدول 1.6 يوضّح وظائف الوحدة التي يُمكن استخدامها مع هيكل بيانات المُكدّس.

جدول 1.6: طرائق الوحدة المُستخدمة للمُكدّس

الطريقة	الوصف
stackName=queue.LifoQueue( )	تنشئ مُكدّسًا جديدًا اسمه stackName.
stackName.get( )	تحذف العنصر الأخير من المُكدّس.

ستَستخدِم وحدة الطابور لإنشاء مُكدّس فارغ.

```
from queue import *

myStack = LifoQueue()

myStack.put("a")
myStack.put("b")
myStack.put("c")
myStack.put("d")
myStack.put("e")

for i in range(5):
   k=myStack.get()
   print(k)

# empty the stack
checkEmpty= myStack.empty()
print("Is the stack empty?", checkEmpty)
```

تذكّر أن العمليات في المُكدّس تعمل وفقًا لقاعدة المُضاف آخرًا يَخرُج أولًا (LIFO).

عند استخدام دالة get مع الطابور، ستَستنِد عمليات الاستدعاء والطباعة إلى قاعدة المُضاف أولًا يَخرُج أولًا (FIFO).

```
e
d
c
b
a
Is the stack empty? True
```

مثال: الطباعة Print

يظهر أمامك في المثال التالي محاكاة لطابور الطباعة في الطابعة. عندما يُرسِل المُستخدِمون أوامر طباعة، تُضاف إلى طابور الطباعة. تَستخدِم الطابعة هذا الطابور لتحديد الملف الذي سيُطبع أولًا.

- افترض أن سعة الطابعة هي فقط 7 ملفات، ولكن في الوقت نفسه، تحتاج إلى طباعة 10 ملفات من الملف A إلى الملف J.
- اكتب برنامجًا يُمثّل طابور الطباعة منذ بدء أمر الطباعة الأول A حتى الانتهاء من كل أوامر الطباعة.
- أضف اللبنة التي تؤكد أن طابور أوامر الطباعة فارغ.

يُمكنك استخدام الخوارزمية الآتية:

1 أنشئ طابور أوامر الطباعة.

2 أدرِج الملفات من A إلى G في طابور أوامر الطباعة.

3 أخرِج الملف A وأدرج الملف H.

4 أخرِج الملف B وأدرج الملف I.

5 أخرِج الملف C وأدرج الملف J.

6 أخرِج الملفات التي تمت طباعتها (D-E-F-G-H-I-J) واحدًا تلو الآخر.


```
# import the queue library
from queue import *
# import the time library to use the sleep function
import time
# initialize the variables and the queue
printDocument = " "
printQueueSize = 0
printQueueMaxSize = 7
printQueue = Queue(printQueueMaxSize)
# add a document to print the queue
def addDocument(document):
    printQueueSize = printQueue.qsize()
    if printQueueSize == printQueueMaxSize:
        print("!! ", document, " was not sent to print queue.")
        print("The print queue is full.")
        print()
        return
    printQueue.put(document)
    time.sleep(0.5) #Wait 5.0 seconds
    print(document, " sent to print queue.")
    printQueueSizeMessage()
# print a document from the print queue
def printDocument():
    printQueueSize = printQueue.qsize()
    if printQueueSize == 0:
        print("!! The print queue is empty.")
```

```
        print()
        return
    printDocument = printQueue.get()
    time.sleep(1) # wait one second
    print ("OK - ", printDocument, " is printed.")
    printQueueSizeMessage()
# print a message with the size of the queue
def printQueueSizeMessage():
    printQueueSize = printQueue.qsize()
    if printQueueSize == 0:
        print ("There are no documents waiting for printing.")
    elif printQueueSize == 1:
        print ("There is 1 document waiting for printing.")
    else:
        print ("There are ", printQueueSize, " documents waiting for printing.")
    print()
# the main program
# send documents to the print queue for printing
addDocument("Document A")
addDocument("Document B")
addDocument("Document C")
addDocument("Document D")
addDocument("Document E")
addDocument("Document F")
addDocument("Document G")
printDocument()
addDocument("Document H")
printDocument()
addDocument("Document I")
printDocument()
addDocument("Document J")
addDocument("Document K")
printDocument()
printDocument()
printDocument()
printDocument()
printDocument()
printDocument()
```

```
Document A sent to print queue.
There is 1 document waiting for printing.

Document B sent to print queue.
There are 2 documents waiting for printing.

Document C sent to print queue.
There are 3 documents waiting for printing.
```

```
Document D sent to print queue.
There are 4 documents waiting for printing.

Document E sent to print queue.
There are 5 documents waiting for printing.

Document F sent to print queue.
There are 6 documents waiting for printing.

Document G sent to print queue.
There are 7 documents waiting for printing.

OK - Document A is printed.
There are 6 documents waiting for printing.

Document H sent to print queue.
There are 7 documents waiting for printing.

OK - Document B is printed.
There are 6 documents waiting for printing.

Document I sent to print queue.
There are 7 documents waiting for printing.

OK - Document C is printed.
There are 6 documents waiting for printing.

Document J sent to print queue.
There are 7 documents waiting for printing.

!! Document K was not sent to print queue.
The print queue is full.

OK - Document D is printed.
There are 6 documents waiting for printing.

OK - Document E is printed.
There are 5 documents waiting for printing.

OK - Document F is printed.
There are 4 documents waiting for printing.

OK - Document G is printed.
There are 3 documents waiting for printing.

OK - Document H is printed.
There are 2 documents waiting for printing.

OK - Document I is printed.
There is 1 document waiting for printing.

OK - Document J is printed.
There are no documents waiting for printing.

!! The print queue is empty.
```

هياكل البيانات الثابتة والمتغيرة Static and Dynamic Data Structures

سبق توضيح أن هياكل البيانات هي طريقة فعالة لتخزين البيانات وتنظيمها، وبالإضافة إلى ما تعلمته حول تصنيف هياكل البيانات إلى أوليّة وغير أوليّة، فإنه يمكن تصنيفها أيضًا إلى **ثابتة** (Static) و**متغيرة** (Dynamic).

هياكل البيانات الثابتة Static Data Structure

في البيانات الثابتة، يكون حجم الهيكل ثابتًا، وتُخزَّن عناصر البيانات في مواقع الذاكرة المتجاورة. تُعدُّ **المصفوفة** (Array) المثال الأبرز لهياكل البيانات الثابتة.

هياكل البيانات المتغيرة Dynamic Data Structure

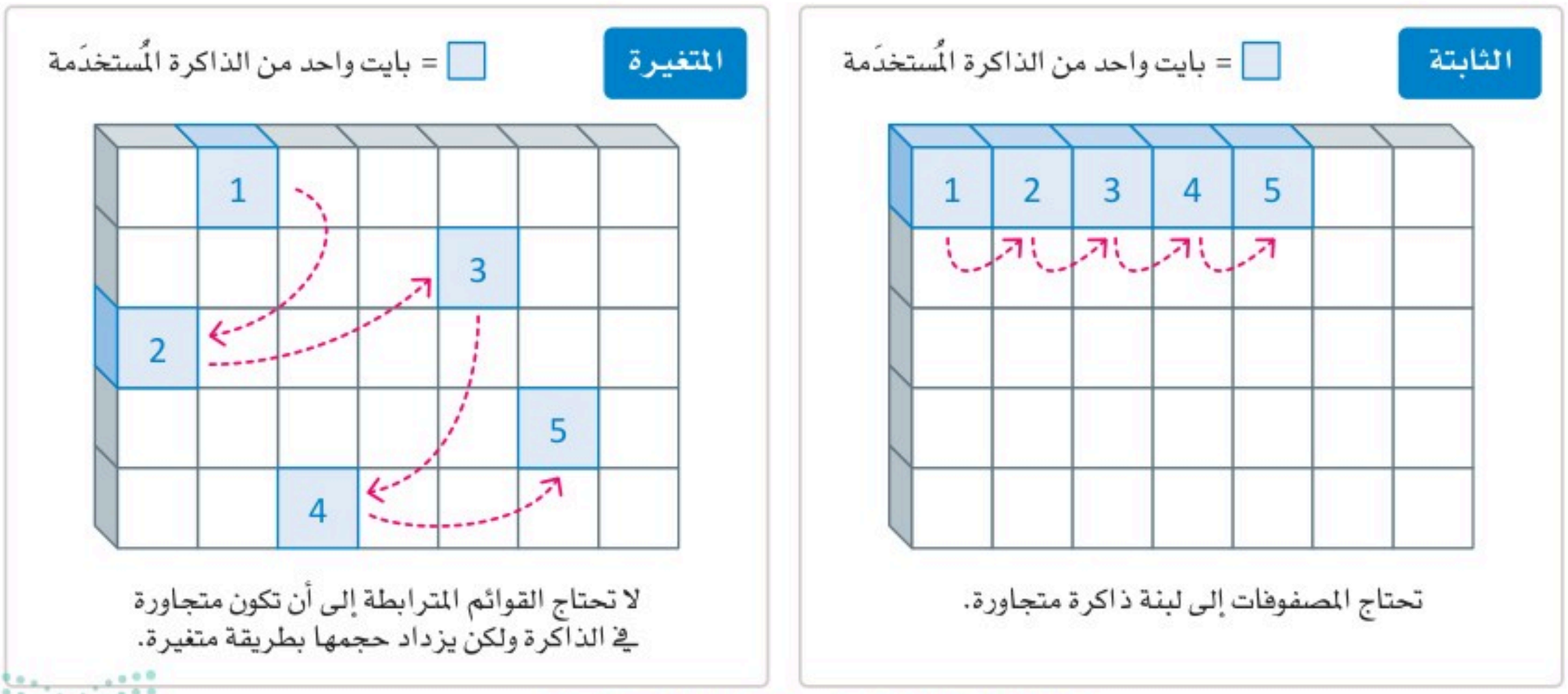
في هياكل البيانات المتغيرة، لا يكون حجم الهيكل ثابتًا ولكن يمكن تعديله أثناء تنفيذ البرنامج، حسب العمليات المُنفَّذة عليه. تُصمَّم هياكل البيانات المتغيرة لتسهيل تغيير حجم هياكل البيانات أثناء التشغيل. وتُعدُّ **القائمة المترابطة** (Linked List) المثال الأبرز لهياكل البيانات المتغيرة.

جدول 1.7: مقارنة بين هياكل البيانات الثابتة والمتغيرة

	الثابتة	المتغيرة
حجم الذاكرة	حجم الذاكرة ثابت.	يمكن تغيير حجم الذاكرة أثناء التشغيل.
أنواع ذاكرة التخزين	تُخزَّن العناصر في مواقع متجاورة في الذاكرة الرئيسة.	تُخزَّن العناصر في مواقع عشوائية في الذاكرة الرئيسة.
سرعة الوصول إلى البيانات	أسرع.	أبطأ.

تخصيص الذاكرة Memory Allocation

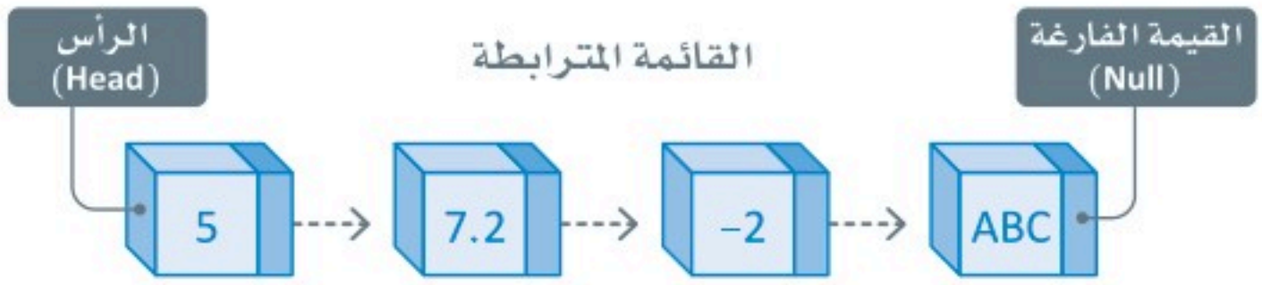
تنتمي **القوائم المترابطة** (Linked Lists) إلى هياكل البيانات المتغيرة، وهذا يعني أن عُقد القائمة المترابطة لا تُخزَّن في مواقع الذاكرة المتجاورة مثل البيانات في المصفوفات. ولهذا السبب، تحتاج إلى تخزين المؤشر من عُقدة إلى أخرى.



شكل 1.26: مثال على تخصيصيّ الذاكرة الثابتة والمتغيرة

القائمة المترابطة Linked List

القائمة المترابطة هي نوع من هياكل البيانات الخطيَّة، وهي واحدة من هياكل البيانات الأكثر شهرة في البرمجة. القائمة المترابطة تشبه سلسلة من العُقد. تحتوي كل عُقدة على حقلين: حقل البيانات حيث تُخزن البيانات، وحقل يحتوي على المؤشر الذي يُشير إلى العُقدة التالية. يُستثنى من هذا العُقدة الأخيرة التي لا يحمل فيها حقل العنوان أي بيانات. إحدى مزايا القائمة المترابطة هي أن حجمها يزداد أو يقل بإضافة أو حذف العُقد.



شكل 1.27: رسم توضيحي للقائمة المترابطة

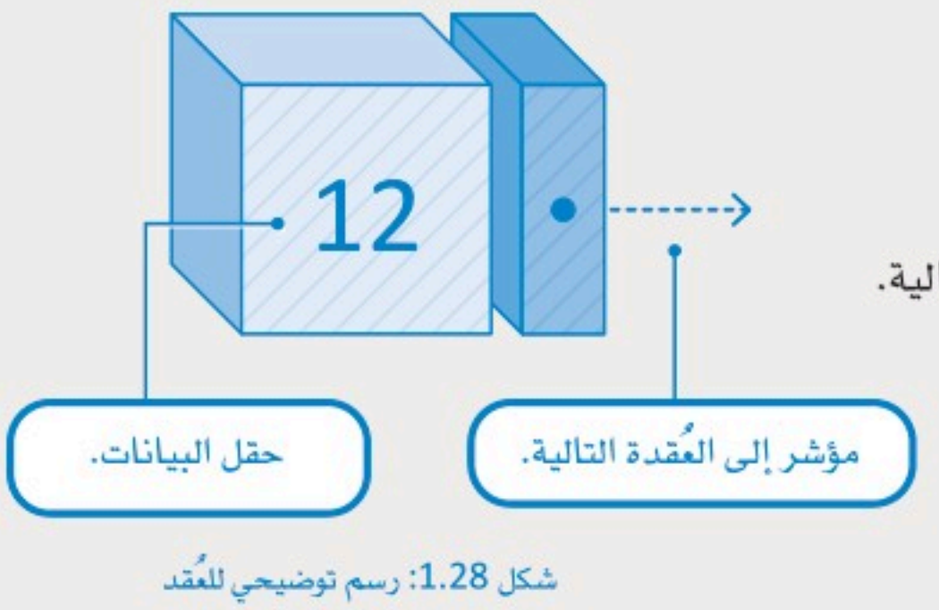
القائمة المترابطة (Linked List):
القائمة المترابطة هي نوع من هياكل البيانات الخطيَّة التي تشبه سلسلة من العُقد.

العُقدة (Node):
العُقدة هي اللبنة الفردية المُكوِّنة لهيكل البيانات وتحتوي على البيانات ورابط واحد أو أكثر من الروابط التي تربطها بالعُقد الأخرى.

العُقدة Node

تتكون كل عُقدة في القائمة المترابطة من جزئين:

- الجزء الأول يحتوي على البيانات.
- الجزء الثاني يحتوي على مؤشر يُشير إلى العُقدة التالية.

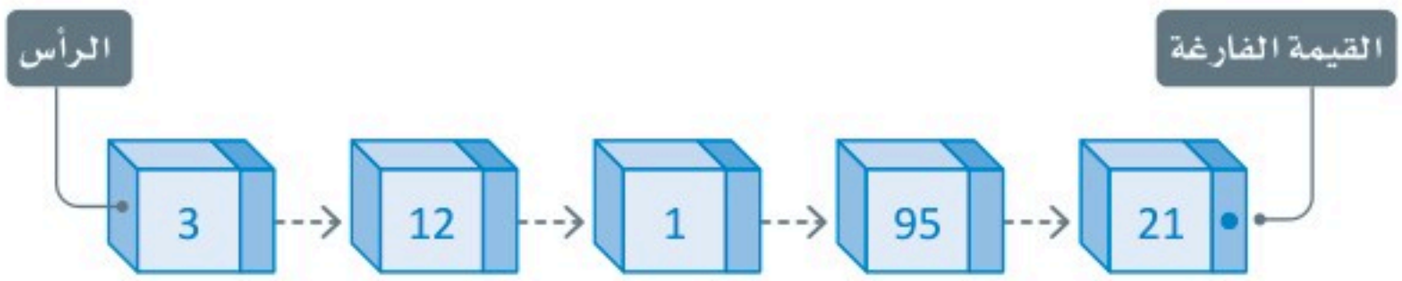


شكل 1.28: رسم توضيحي للعُقد

لقراءة محتوى عُقدة محددة، عليك المرور على كل العُقد السابقة.

لتشاهد مثالًا على القائمة المترابطة للأعداد الصحيحة.

تتكون القائمة المترابطة من خمس عُقد.



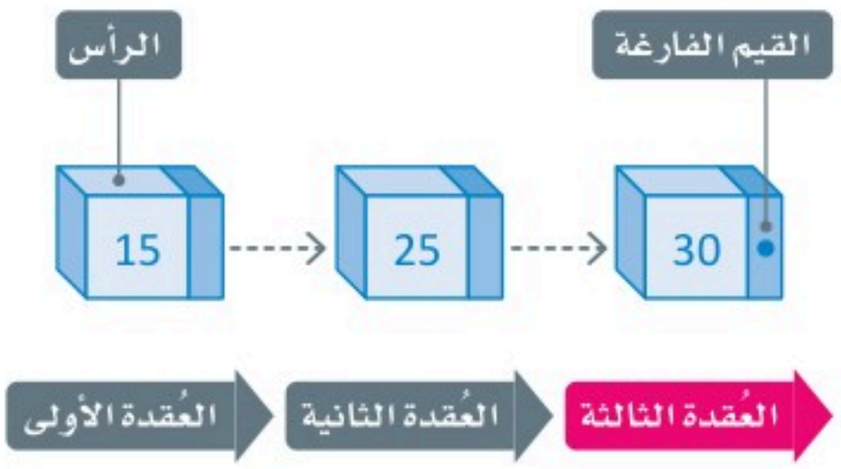
شكل 1.29: رسم توضيحي يُمثّل قائمة مترابطة للأعداد الصحيحة

القيمة الفارغة تعني أنها بلا قيمة، أو غير مُحدَّدة، أو فارغة. على الرغم من أنه في بعض الأحيان يُستخدم الرقم 0 للإشارة إلى القيمة الفارغة، إلا أنه رقم محدَّد وقد يشير إلى قيمة حقيقية.

العُقد في القائمة لا يكون لها اسم، وما تعرفه عنها هو عنوانها (الموقع الذي تخزن فيه العُقدة في الذاكرة). للوصول إلى أي عُقدة بالقائمة، تحتاج فقط إلى معرفة عنوان العُقدة الأولى. ثم تتبع سلسلة العُقد للوصول إلى العُقدة المطلوبة.

على سبيل المثال، إن كنت ترغب في الوصول إلى العُقدة الثالثة في القائمة لمعالجة البيانات التي تحتوي عليها، عليك البدء من العُقدة الأولى في القائمة، ومن العُقدة الأولى للوصول إلى الثانية، ومن الثانية للوصول إلى الثالثة.

- عنوان العُقدة الأولى مُخزَّن في مُتغير خاص (مُستقِل) يُطلق عليه عادةً **الرأس** (Head).
- قيمة مؤشر العُقدة الأخيرة في القائمة **قيمة فارغة** (Null)، ويُمثَّل بالرمز ●.
- عندما تكون القائمة فارغة، يشير مؤشر الرأس إلى **القيمة الفارغة** (Null).



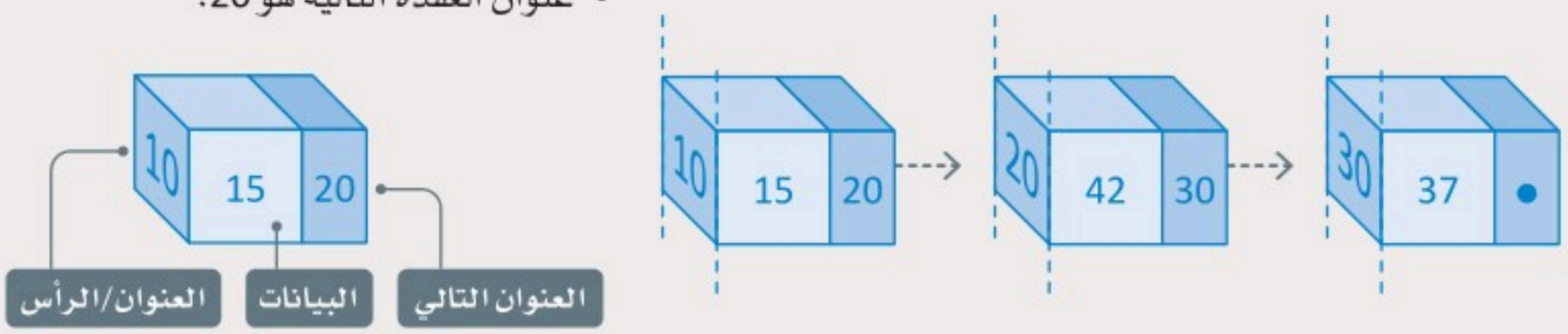
شكل 1.30: الوصول إلى العُقدة الثالثة في القائمة المترابطة

إليك مثالًا توضيحيًا على القائمة المترابطة في الشكل 1.31، كما ذُكر من قبل فإن كل عُقدة تتكون من بيانات ومؤشر يشير إلى العُقدة التالية، بحيث تُخزّن كل عُقدة في الذاكرة في عنوان مُحدَّد.

لتربط العُقدة السابقة بالعُقدة التالية بقيمة بيانات 42، التي بدورها تُشير إلى العُقدة الثالثة والأخيرة عند عنوان 30 بقيمة بيانات 37.

مثال على العُقدة:

- بيانات العُقدة هي الرقم 15.
- عنوان العُقدة في الذاكرة هو 10.
- عنوان العُقدة التالية هو 20.



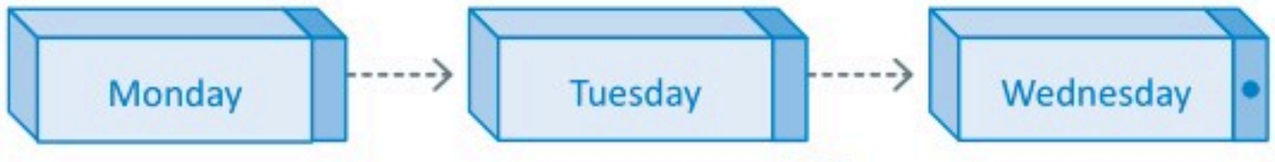
شكل 1.31: المؤشرات في القائمة المترابطة

جدول 1.8: الاختلافات بين القائمة والقائمة المترابطة

الاختلافات	القائمة	القائمة المترابطة
طريقة تخزين الذاكرة	المواقع متجاورة في الذاكرة.	المواقع عشوائية في الذاكرة.
الهيكل	يمكن الوصول إلى كل عنصر برقم **الفهرس** (Index).	يمكن الوصول إلى العناصر من خلال **المؤشر** (Pointer).
الحجم	يُخزَّن كل عنصر تلو الآخر.	تُخزَّن العناصر في صورة عُقد تحتوي على البيانات وعنوان العنصر التالي.
استخدام الذاكرة	تُخزَّن البيانات وحدَها في الذاكرة.	تُخزَّن البيانات والمؤشرات في الذاكرة.
نوع الوصول إلى البيانات	الوصول العشوائي إلى أي عنصر بالقائمة.	الوصول المتسلسل إلى العناصر.
سرعة الإضافة والحذف	بطء إضافة العناصر وحذفها.	سرعة إضافة العناصر وحذفها.

القائمة المترابطة في لغة البايثون Linked List in Python

لا تُوفر لغة البايثون نوع بيانات مُحدَّد مُسبقًا للقوائم المترابطة. عليك إنشاء نوع البيانات الخاص بك، أو استخدام مكتبات البايثون التي توفر تمثيلًا لهذا النوع من البيانات. لإنشاء قائمة مترابطة، استخدم فئات البايثون. في المثال الموضح بالشكل 1.32، ستُنشئ قائمة مترابطة مكونة من ثلاث عُقد، كل واحدة تضم يومًا من أيام الأسبوع.



شكل 1.32: مثال على القائمة المترابطة

الفئة (Class):

الفئة هي هيكل بيانات معرّف بواسطة المُستخدِم، ويحتوي على أعضاء البيانات (**السمات** Properties)، والطرائق (**السلوك** Behavior) الخاصة بها. وتُستخدَم الفئات كقوالب لإنشاء الكائنات.

ستُنشئ أولًا عُقدة باستخدام الفئة.

```
# single node
class Node:
    def __init__(self, data, next=None):
        self.data = data # node data
        self.next = next # Pointer to the next node

# Create a single node
first = Node("Monday")
print(first.data)
```

```
Monday
```

الخطوة التالية هي إنشاء قائمة مترابطة تحتوي على عُقدة واحدة، وهذه المرة ستَستخدِم مؤشر الرأس للإشارة إلى العُقدة الأولى.

```
# single node
class Node:
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next

# linked list with one head node
class LinkedList:
  def __init__(self):
    self.head = None

# list linked with a single node
Linkedlist1 = LinkedList()
Linkedlist1.head = Node("Monday")
print(Linkedlist1.head.data)
```

```
Monday
```

أضِفْ الآن المزيد من العُقد إلى القائمة المترابطة.

```
# single node
class Node:
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next

# an empty linked list with a head node.
class LinkedList:
  def __init__(self):
    self.head = None

# the main program
linked_list = LinkedList()
# the first node
linked_list.head = Node("Monday")
# the second node
linked_list.head.next = Node("Tuesday")
# the third node
linked_list.head.next.next = Node("Wednesday")

# print the linked list
node = linked_list.head
while node:
    print (node.data)
    node = node.next
```

تُستخدَم عبارة while للتنقل من عُقدة إلى أخرى.

```
Monday
Tuesday
Wednesday
```

إضافة العُقدة إلى القائمة المترابطة
Add a Node to a Linked List

لتتمكن من إضافة عُقدة جديدة، اتبع الخطوات التالية:

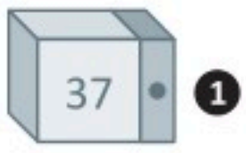
- يجب أن يُشير مؤشر العُقدة الأولى إلى عنوان العُقدة الجديدة، حتى تصبح العُقدة الجديدة هي العُقدة الثانية.
- يجب أن يُشير مؤشر العُقدة الجديدة (الثانية) إلى عنوان العُقدة الثالثة.

بهذه الطريقة، لن تحتاج إلى تغيير العناصر عند إضافة عنصر جديد في المنتصف. تقتصر العملية على تغيير قِيَم العناوين في العُقدة التي تُسرِّع من عملية الإضافة في حالة القوائم المترابطة، مقارنةً بحالة القوائم المتسلسلة.

مثال:

لديك قائمة مترابطة من عنصرين: 12 و99، وتريد إدراج العنصر 37 كعنصر ثانٍ بالقائمة. في النهاية، سيكون لديك قائمة من ثلاثة عناصر: 12 و37 و99.

1. أنشئ العُقدة الجديدة.



2. اربُط العُقدة 37 بالعُقدة 99.
3. اربُط العُقدة 12 بالعُقدة 37 (تمت إضافة العُقدة الجديدة).

```python
# single node
class Node:
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next

# linked list with one head node
class LinkedList:
  def __init__(self):
    self.head = None

def insertAfter(new, prev):
    # create the new node
    new_node = Node(new)
    # make the next of the new node the same as the next of the previous node
    new_node.next = prev.next
    # make the next of the previous node the new node
    prev.next = new_node

# create the linked list
L_list = LinkedList()

# add the first two nodes
L_list.head = Node(12)
second = Node(99)
L_list.head.next = second

# insert the new node after node 12 (the head of the list)
insertAfter(37, L_list.head)

# print the linked list
node = L_list.head
while node:
    print (node.data)
    node = node.next
```

```
12
37
99
```

حذف العُقدة من القائمة المترابطة Delete a Node from a Linked List

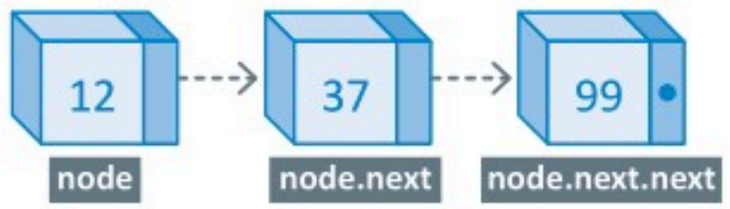
لحذف عُقدة، عليك تغيير مُؤشر العُقدة التي تسبق العُقدة المراد حذفها إلى مؤشر العُقدة التي تلي العُقدة المحذوفة. أصبحت العُقدة المحذوفة (الثانية) عبارة عن **بيانات غير مُفيدة** (Useless Data) وستُخصَّص مساحة الذاكرة التي تشغلها لاستخدامات أخرى.

مثال:

لديك قائمة مترابطة من ثلاثة عناصر: 12 و37 و99، وترغب في حذف العنصر 37. في النهاية، سيكون لديك قائمة من عنصرين: 12 و99.

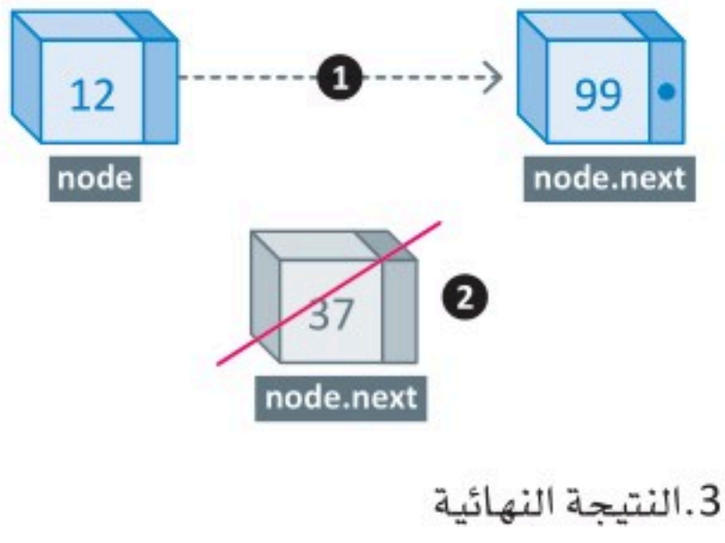
```python
# single node
class Node:
  def __init__(self, data = None, next=None):
    self.data = data
    self.next = next

# linked list with one head node
class LinkedList:
  def __init__(self):
    self.head = None

def deleteNode(key, follow):

    # store the head node
    temp = follow.head

    # find the key to be deleted,
    # the trace of the previous node to be changed
    while(temp is not None):
        if temp.data == key:
            break
        prev = temp
        temp = temp.next

    # unlink the node from the linked list
    prev.next = temp.next
    temp = None

# create the linked list
L_list = LinkedList()

# add the first three nodes
L_list.head = Node(12)
second = Node(37)
third = Node(99)
L_list.head.next = second
second.next = third

# delete node 37
deleteNode(37,L_list)

# print the linked list
node = L_list.head
while node:
    print (node.data)
    node = node.next
```

```
12
99
```

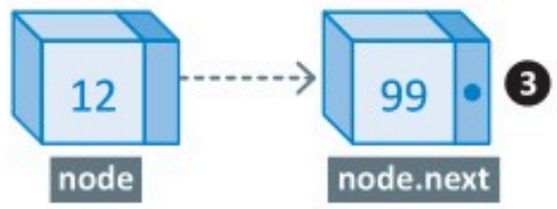


1. اربُط مؤشر العُقدة **12** بالعُقدة **99**.

2. احذف العُقدة **37**.



3. النتيجة النهائية



إذا كنت تريد حذف العُقدة الأولى من القائمة المترابطة، عليك نقل مؤشر الرأس إلى العُقدة الثانية من القائمة.

تمرينات

1

حدّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. لغة البايثون تُعرِّف هياكل البيانات غير الأوليّة.	○	○
2. هياكل البيانات الخطيَّة تُخزِّن عناصر البيانات في ترتيب عشوائي فقط.	○	○
3. إضافة العناصر وحذفها من القائمة المُترابطة (Linked List) أبطأ من القائمة (List).	○	○
4. يمكن الوصول إلى العناصر في القائمة باستخدام رقم الفهرس فقط.	○	○
5. يُمكن تغيير حجم هيكل البيانات الثابتة أثناء تنفيذ البرنامج.	○	○

2 حدّد الاختلافات بين هياكل البيانات الثابتة والمتغيرة.

هياكل البيانات الثابتة	هياكل البيانات المتغيرة

3 اكتب مثالين لاستخدامات القوائم المترابطة.

4 لديك مُكدّس به ست مساحات فارغة.

- ستُضيف الحروف الآتية C وE وB وA وD في المواقع من 0 إلى 4.
- املأ المُكدّس الذي يُشير إلى موقع المؤشر الأعلى.
- نَفّذ العمليات التالية:

pop → pop → push X → push K → pop →

اظهِر المُخرَج النهائي بعد تنفيذ العمليات السابقة للإشارة إلى موقع المؤشر العلوي.

اكتب البرنامج الذي يُنشئ المُكدّس الموضّح بالأعلى، ثم نَفّذ العمليات المذكورة أعلاه باستخدام مكتبة الطابور القياسية.

المُكدّس	
5	
4	
3	
2	
1	
0	

المُخرَج النهائي	
5	
4	
3	
2	
1	
0	

5 لديك التسلسل الرقمي الآتي: 4 و8 و2 و5 و9 و13.

- ما العملية المُستخدَمة لإضافة العناصر الموضحَّة بالأعلى إلى الطابور؟

- أكمل الطابور بعد إضافة العناصر.

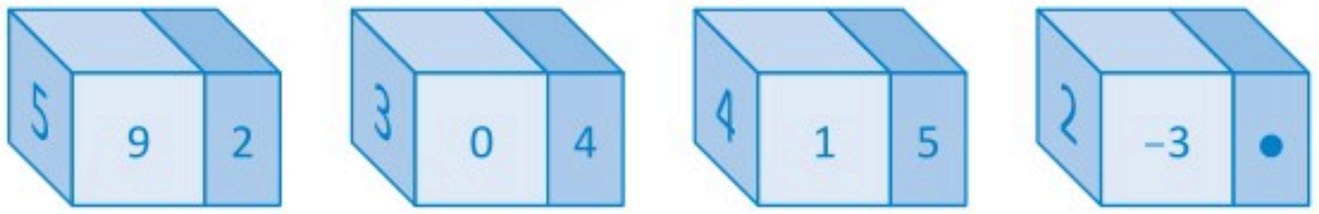
0	1	2	3	4	5

- ما العملية المُستخدَمة لحذف العناصر من الطابور؟

- كم مرة يجب تنفيذ العملية الموضحَّة بالأعلى لحذف العنصر الذي قيمته 5؟

- أكتب المقطع البرمجي بلغة البايثون لإنشاء الطابور السابق.

6 باستخدام العُقد التالية ارسّم القائمة المترابطة، ثم اكتب القيم في القائمة بالترتيب السليم.



الرأس = 3

7 أنشئ قائمة تضُم الأرقام التالية: 5 و20 و45 و8 و1.

- ارسم العُقد في القائمة المترابطة.

- صِفْ عملية إضافة الرقم 7 بعد الرقم 45.

- ارسم القائمة الجديدة.

- صِفْ العملية المطلوبة لحذف العُقدة الثانية من القائمة.

- ارسم القائمة المترابطة النهائية.

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الثالث

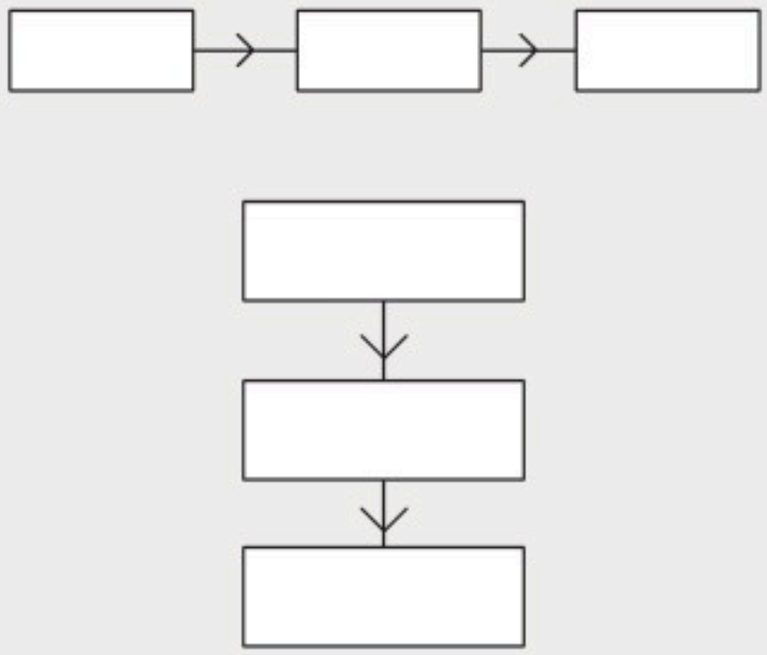
هياكل البيانات غير الخطيَّة

في الدرس السابق تعلّمت بعض هياكل البيانات الخطيَّة، وفيها كل عنصر يَتبع العنصر السابق له بطريقة خطيَّة. هل يمكنك التفكير في حالة لا تسير فيها الأشياء بتسلسل خطيّ؟ على سبيل المثال، هل يمكن لعنصر واحد أن يتبعه أكثر من عنصر؟

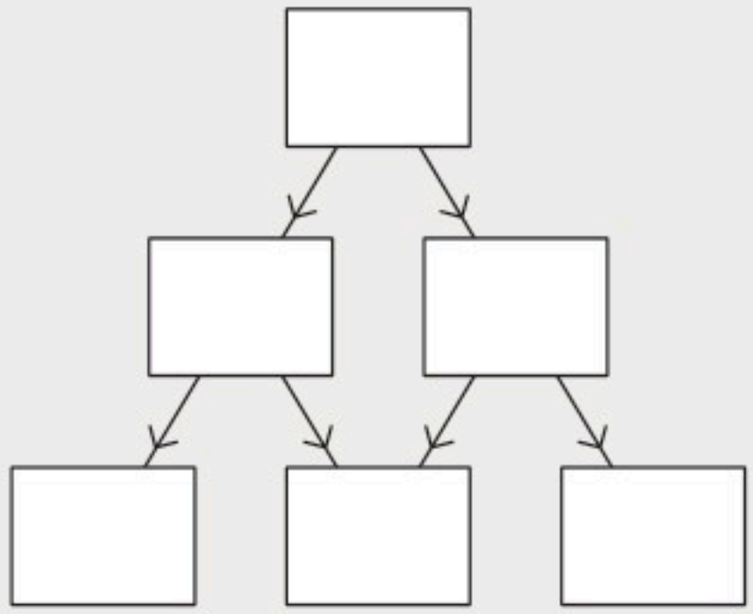
هياكل البيانات غير الخطيَّة Non-Linear Data Structures

هي نوع من هياكل البيانات يتميز بإمكانية ربط عنصر باكثر من عنصر واحد في الوقت نفسه. ومن الأمثلة التوضيحية على هياكل البيانات غير الخطيَّة: الأشجار ومُخطَّطات البيانات. الشكل 1.33 يوضِّح هياكل البيانات الخطيَّة وهياكل البيانات غير الخطيَّة.

هياكل البيانات الخطيَّة



هياكل البيانات غير الخطيَّة

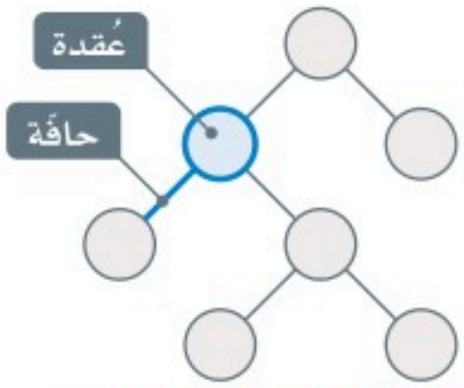


شكل 1.33: الرسم التوضيحي لهياكل البيانات الخطيَّة وغير الخطيَّة

جدول 1.9: الفرق بين هياكل البيانات الخطيَّة وغير الخطيَّة

هياكل البيانات الخطيَّة	هياكل البيانات غير الخطيَّة
تُرتَّب عناصر البيانات في ترتيب خطي يرتبط فيه كل عنصر بالعنصرين السابق والتالي له.	يمكن ربط عناصر البيانات بالعديد من العناصر الأخرى.
تُستَعرض عناصر البيانات في مسار واحد.	لا تُستَعرض عناصر البيانات في مسار واحد.
سهل التنفيذ.	معقّد التنفيذ.

الأشجار Trees



شكل 1.34: العلاقات في الشجرة

الأشجار هي نوع من هياكل البيانات غير الخطيَّة، وتتكون الشجرة من مجموعة من العُقد المُرتَّبة في ترتيب هرمي. ترتبط كل عُقدة بواحدة أو أكثر من العُقد، وترتبط العُقد مع الحواف في نموذج علاقة يربط بين **الأصل** (Parent) و**الفَرع** (Child). تُستخدَم الأشجار في العديد من مجالات علوم الحاسب، بما في ذلك أنظمة التشغيل، والرسوميات، وأنظمة قواعد البيانات، والألعاب، والذكاء الاصطناعي، وشبكات الحاسب.

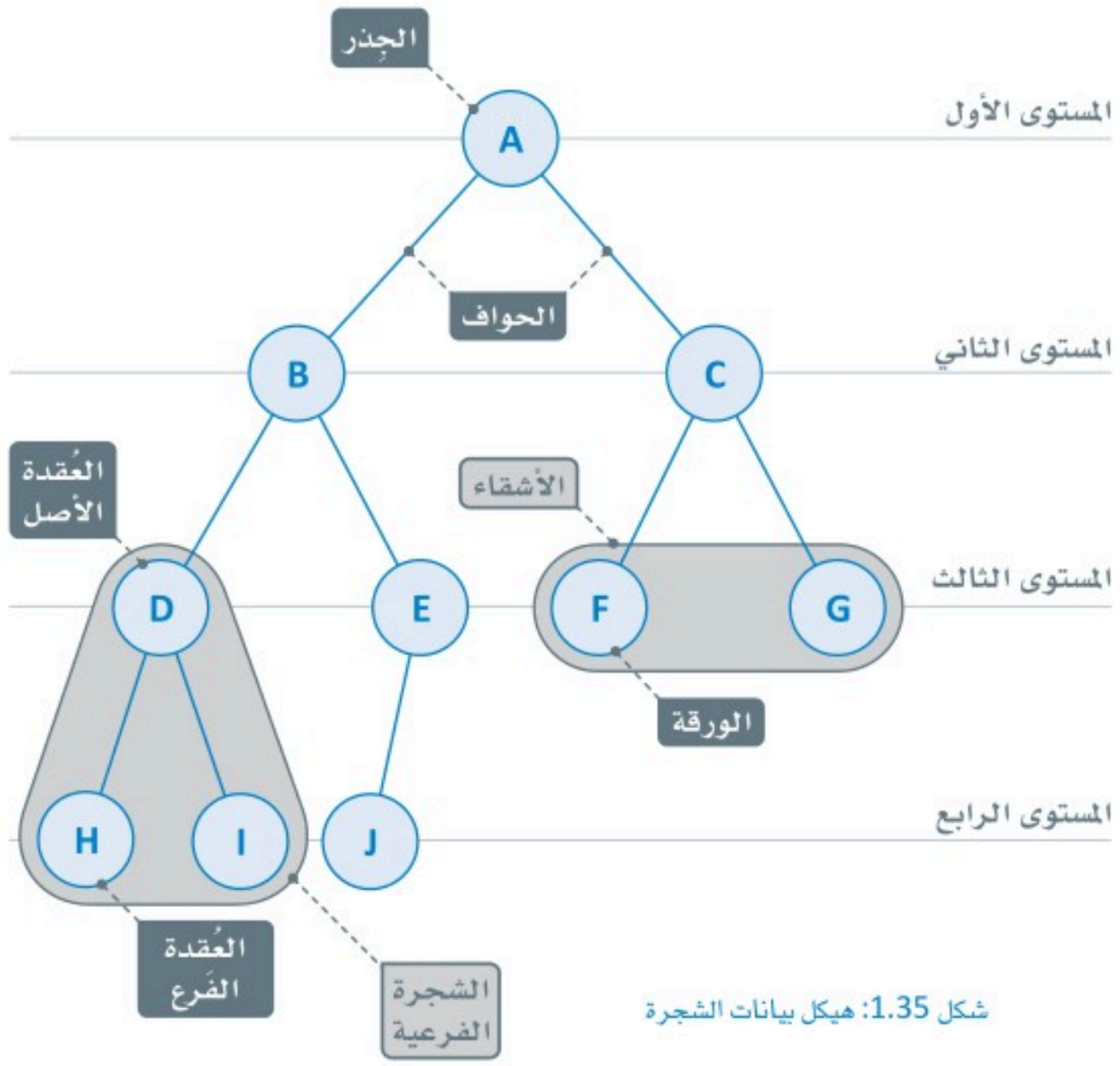
مُصطلحات تقنية الشجرة المُستخدمة في هيكل بيانات الشجرة
Tree Terminology Used in the Tree Data Structure

- **الجِذر** (Root): العُقدة الأولى والوحيدة في الشجرة التي ليس لها أصل وتأتي في المستوى الأول من الشجرة، مثل: العُقدة A في الشكل 1.35.
- **الفَرع** (Child): العُقدة المرتبطة مباشرةً بِعُقدة في المستوى الأعلى، مثل: العُقدة H هي فَرع العُقدة D، والعُقدتان B وC هما فَرعا العُقدة A.
- **الأصل** (Parent): العُقدة التي لها فَرع أو أكثر في المستوى الأقل، مثل: العُقدة B هي أصل العُقدتين D وE.
- **الورقة** (Leaf): العُقدة التي ليس لها أي عُقدة فرعية، مثل: الورقة F.
- **الأشقاء** (Siblings): كل العُقد الفرعية التي تنبثق من الأصل نفسه، مثل:العُقدتان D وE شقيقتان.
- **الحواف** (Edges): الروابط التي تصل بين العُقد والشجرة.
- **الشجرة الفرعية** (Sub-Tree): الشجيرات التي توجد داخل الشجرة الأكبر حجمًا، مثل:الشجرة التي بها العُقدة D هي الأصل والعُقدتان H وا هما الفَرعان.

الشجرة (Tree):
الشجرة هي نوع من هياكل البيانات غير الخطيَّة، وتتكون من مجموعة من العُقد المُرتَّبة في ترتيب هرمي.

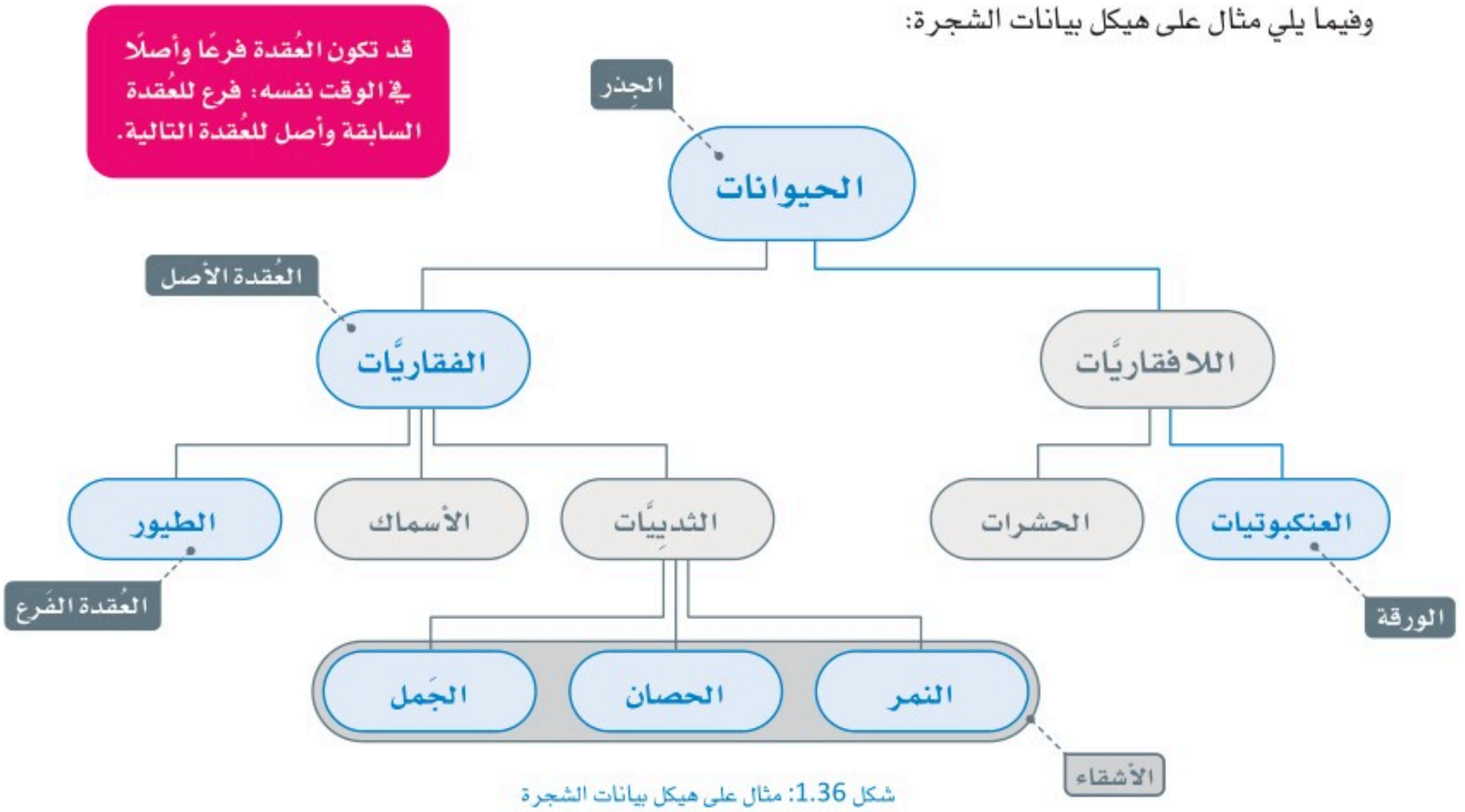
الحافة (Edge):
الحافة تصل بين عُقد هيكل بيانات الشجرة.

قد يكون لديك شجرة بسيطة تتكون من عُقدة واحدة. تكون هذه العُقدة في الوقت نفسه جذر هذه الشجرة البسيطة؛ لأنَها ليس لها أصل.



شكل 1.35: هيكل بيانات الشجرة

وفيما يلي مثال على هيكل بيانات الشجرة:



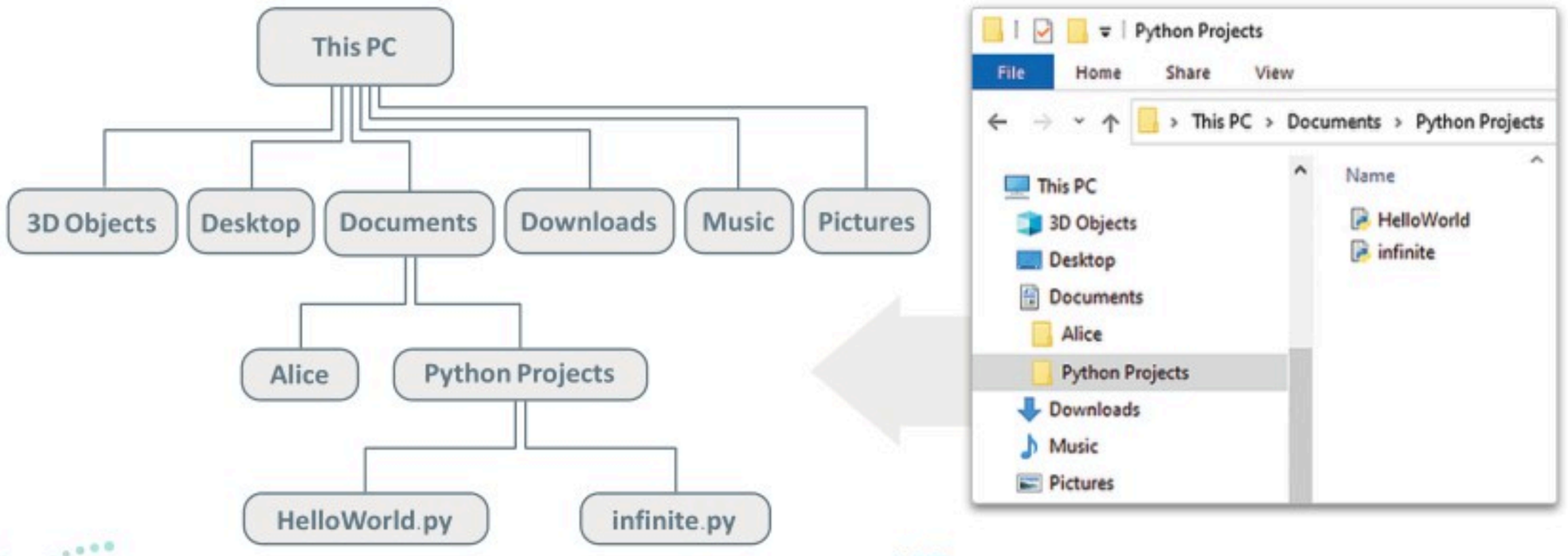
شكل 1.36: مثال على هيكل بيانات الشجرة

خصائص هيكل بيانات الشجرة Tree Data Structure Features

- يُستخدم لتمثيل المُخطَّط الهرمي.
- يتميّز بالمرونة، فمن السهل إضافة عنصر من الشجرة أو حذفه.
- سهولة البحث عن العناصر فيه.
- يعكس العلاقات الهيكلية بين البيانات.

مثال

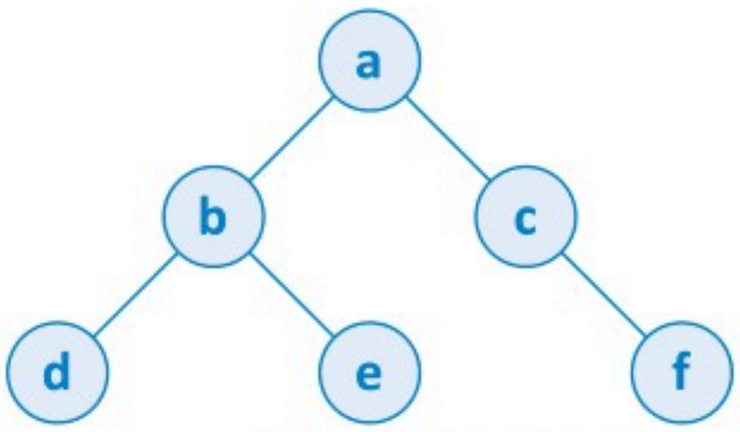
تنظيم الملفات في نظام التشغيل هو مثال عملي على الشجرة. كما يتضح في الشكل 1.37، يوجد داخل مجلد Documents (المستندات) مُجلد آخر اسمه Python Projects (مشروعات البايثون) يحتوي على ملفين آخرين.



شكل 1.37: تنظيم الملفات في نظام التشغيل

هيكل بيانات الشجرة في لغة البايثون

Tree Data Structure in Python



شكل 1.38: شجرة قاموس البايثون

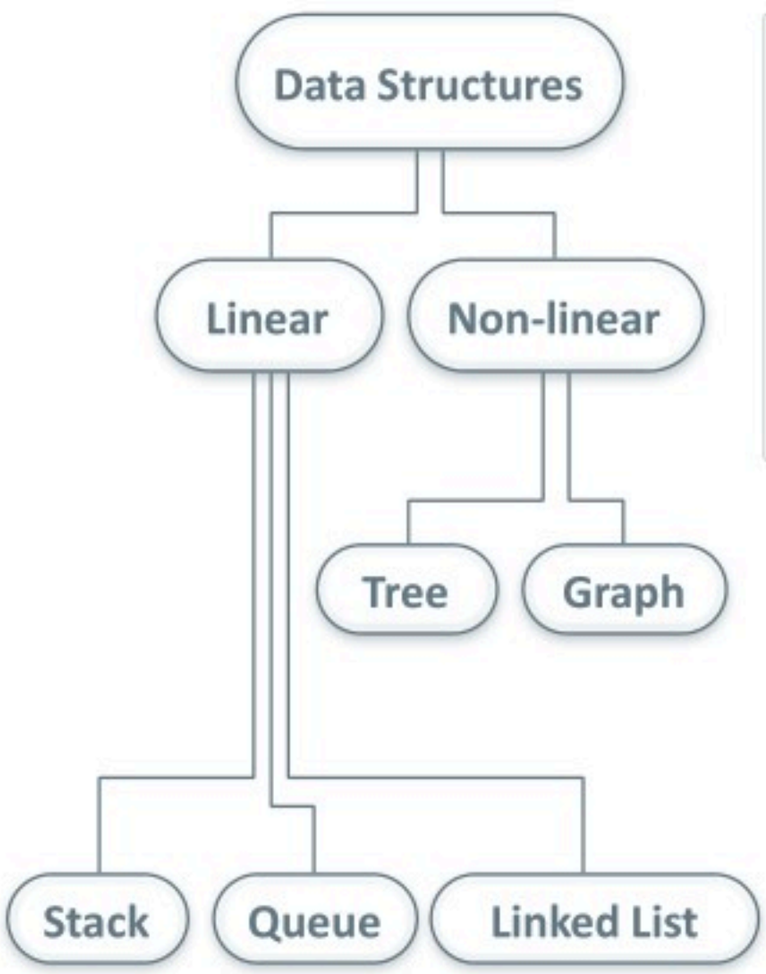
لا تُوفِر لغة البايثون نوعًا محددًا مسبقًا من البيانات لهيكل بيانات الشجرة. ومع ذلك، تُصمَّم الأشجار من القوائم والقواميس بسهولة. يوضِّح الشكل 1.38 تطبيقًا بسيطًا للشجرة باستخدام القاموس.

في هذا المثال، ستُنشئ شجرة باستخدام قاموس البايثون. ستمثّل عُقد الشجرةِ مفاتيحَ القاموس، وستكون القيمة المقابلة لكل مفتاح هي قائمة تحتوي على العُقد المُتصلة بحافة مباشرة من هذه العُقدة.

```
myTree = {
   "a": ["b", "c"], # node
   "b": ["d", "e"],
   "c": [None, "f"],
   "d": [None, None],
   "e": [None, None],
   "f": [None, None],
}
print(myTree)
```

```
{'a': ['b', 'c'], 'b': ['d', 'e'], 'c': [None, 'f'],
'd': [None, None], 'e': [None, None], 'f': [None, None]}
```

في المثال التالي ستُنشئ شجرة مثل تلك الموضحة في الشكل 1.39:



شكل 1.39: شجرة هياكل البيانات

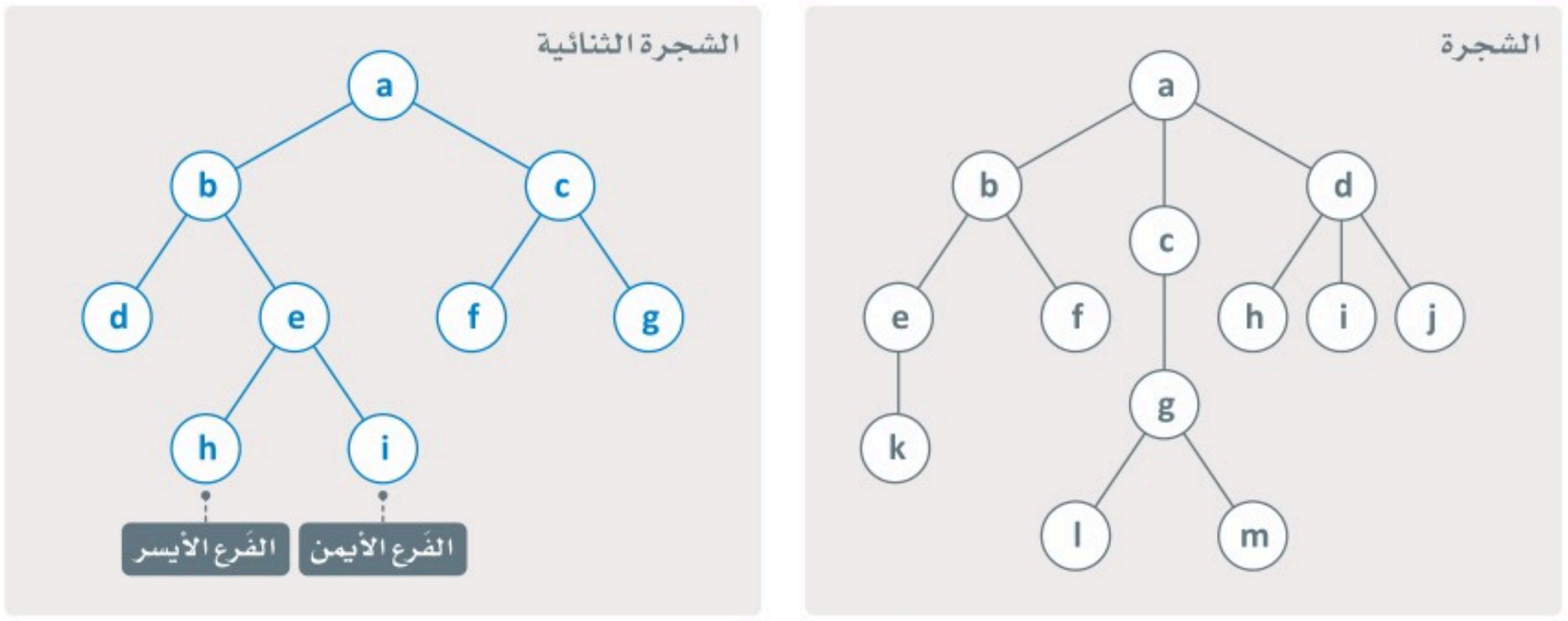
```
myTree = {"Data Structures":["Linear","Non-linear"],
          "Linear":["Stack","Queue","Linked List"],
          "Non-linear":["Tree", "Graph"]}

for parent in myTree:
    print(parent, "has",len(myTree[parent]),"nodes" )
    for children in myTree[parent]:
        print(" ",children)
```

الأصل — الفَرع

```
Data structures has 2 nodes
  Linear
  Non-linear
Linear has 3 nodes
  Stack
  Queue
  Linked List
Non-linear has 2 nodes
  Tree
  Graph
```

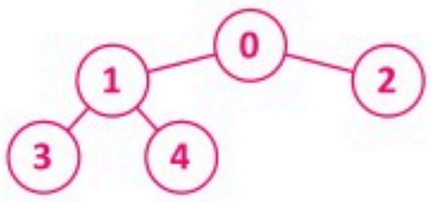
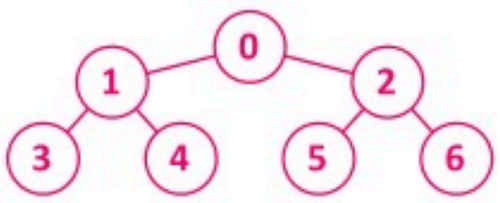
الشجرة الثنائية Binary Tree

الشجرة الثنائية هي نوع خاص من الأشجار، يكون لكلّ عُقدةٍ فيها فَرعان على الأكثر؛ الفَرع الأيمن والفَرع الأيسر. الشكل 1.40 يَعرض مثالًا يوضِّح الشجرة والشجرة الثنائية.



شكل 1.40: الشجرة والشجرة الثنائية

جدول 1.10: أنواع هياكل بيانات الشجرة الثنائية

النوع	الوصف	رسم توضيحي للهيكل
الشجرة الثنائية التَامة (Full Binary Tree)	يكون لكلّ عُقدة إمّا 0 أو 2 من الفروع (Children) بخلاف الأوراق (Leaves).	
الشجرة الثنائية الكاملة (Complete Binary Tree)	يكون كلّ مستوًى من مستويات الشجرة ممتلئًا بالكامل، ربما باستثناء المستوى الأخير، حيث تكون كل العُقد فيه مملوءة من اليسار إلى اليمين.	
الشجرة الثنائية المثالية (Perfect Binary Tree)	يكون لكل العُقد الداخلية فَرعان وتكون كل الأوراق عند المستوى نفسه.	

أمثلة على تطبيقات هياكل بيانات الشجرة

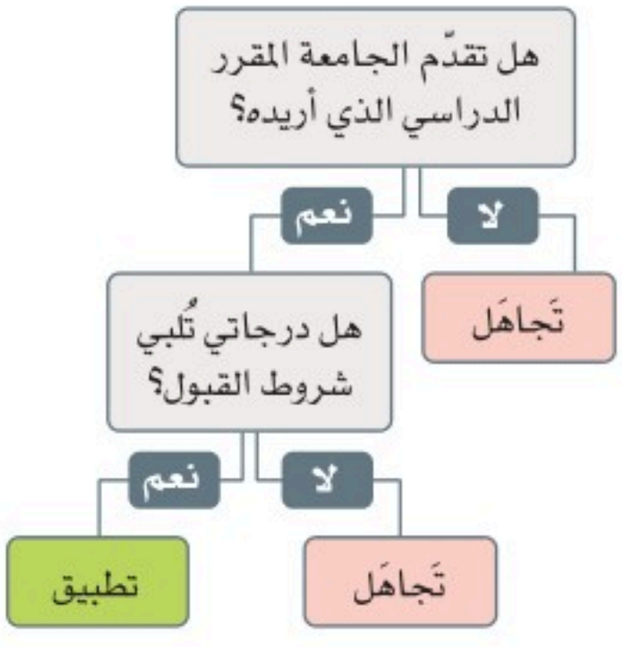
Examples of Applications of Tree Data Structures:

- تخزين البيانات الهرمية مثل: هياكل المجلدات.
- تعريف البيانات في لغة ترميز النص التشعبي (HTML).
- تنفيذ الفهرسة في قواعد البيانات.

شجرة القرار Decision Tree

عبارة القرار if a: else b هي واحدة من العبارات الأكثر استخدامًا في لغة البايثون. ومن خلال تداخل وتجميع هذه العبارات، يمكنك تصميم شجرة القرار.

تُستخدَم أشجار القرار في الذكاء الاصطناعي من خلال إحدى تقنيات تعلُّم الآلة وتُعرف باسم: **تعلُّم شجرة القرار** (Decision Tree Learning). العُقد الأخيرة في هذه التقنية تُسمّى أيضًا الأوراق، وتحتوي على الحلول المُحتملة للمشكلة. كل عُقدة باستثناء الأوراق ترتبط بحالة منطقية يتفرع منها احتمالا الإجابة بنعم أو لا. أشجار القرار تُعدُّ سهلة الفهم، والاستخدام، والتصوير، ويسهل التحقق منها. على سبيل المثال، الشكل 1.41 يوضِّح شجرة القرار التي تُحدِّد ما إذا كنت ستَتقدَّم بطلب الالتحاق بجامعة مُحدَّدة أم لا بناءً على معيارين: المقررات الدراسية التي تُدرَّس في الجامعة، واستيفاء متطلبات القبول.



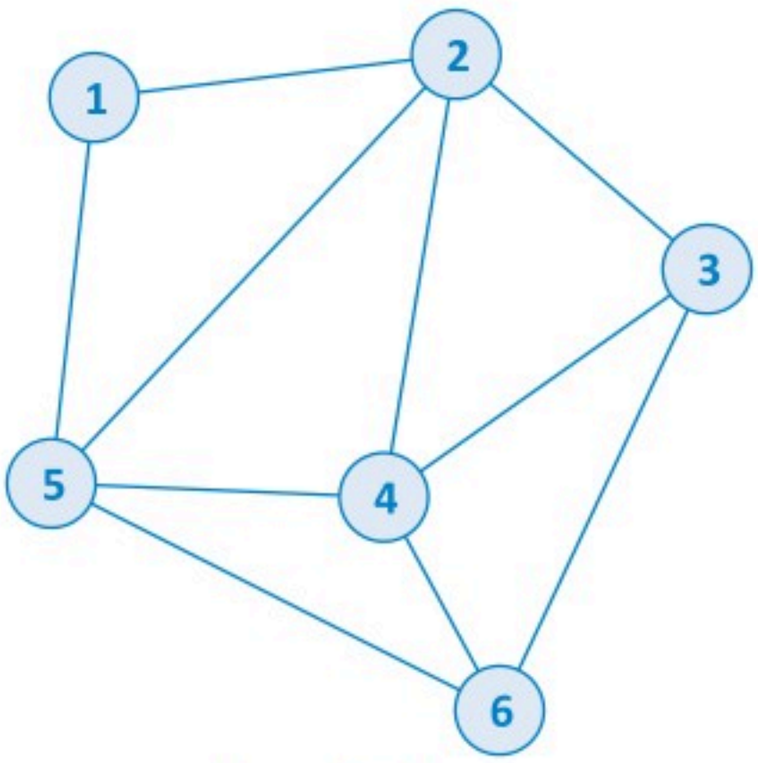
شكل 1.41: مثال على شجرة القرار

المُخطَّطات Graphs

السمَة الأكثر أهمية لهياكل البيانات غير الخطيَّة هي أنّ البيانات الخاصة بها لا تتبّع أي نوع من أنواع التّسلسل، وذلك على خلاف المصفوفات والقوائم المترابطة، كما يمكن ربط عناصرها بأكثر من عنصر وحيد. **الشجرة الجذرية** (Rooted Tree) تبدأ بعُقدة جذرية يمكن ربطها بالعُقد الأخرى. تَتّبع الأشجار قواعد محددة: وهي أن تكون عُقد الشجرة متصلة، وأن تكون الشجرة خالية من **الحلقات**(Loops) و**الحلقات الذاتية** (Self Loops)، كما أن لبعض أنواع الأشجار قواعدها الخاصة (جدول 1.10)، مثلما في حالة الأشجار الثنائية. ولكن ماذا سيحدث إذا لم تَتَّبع قواعد الأشجار؟ في هذه الحالة أنت لا تتحدث عن الأشجار، بل عن نوع جديد من هياكل البيانات المُتغيِّرة التي تُسمى المُخطَّطات. في الحقيقة، الأشجار هي نوع من المُخطَّطات حيث أن المُخطَّط هو الشكل العام لهيكل البيانات، بمعنى أن كل هياكل البيانات السابقة يمكن اعتبارها حالات خاصة من المُخطَّطات. الشكل 1.42 يعرض مُخطَّطًا به ست عُقد وعشر حواف.

المُخطَّط (Graph):
المُخطَّط هو هيكل البيانات المُكوَّن من مجموعة من العُقد ومجموعة من الخطوط التي تصل بين جميع العُقد، أو بعضها.

كل الأشجار مُخطَّطات، ولكن ليست كل المُخطَّطات أشجارًا.



شكل 1.42: مثال على مُخطَّط به ست عُقد وعشر حواف

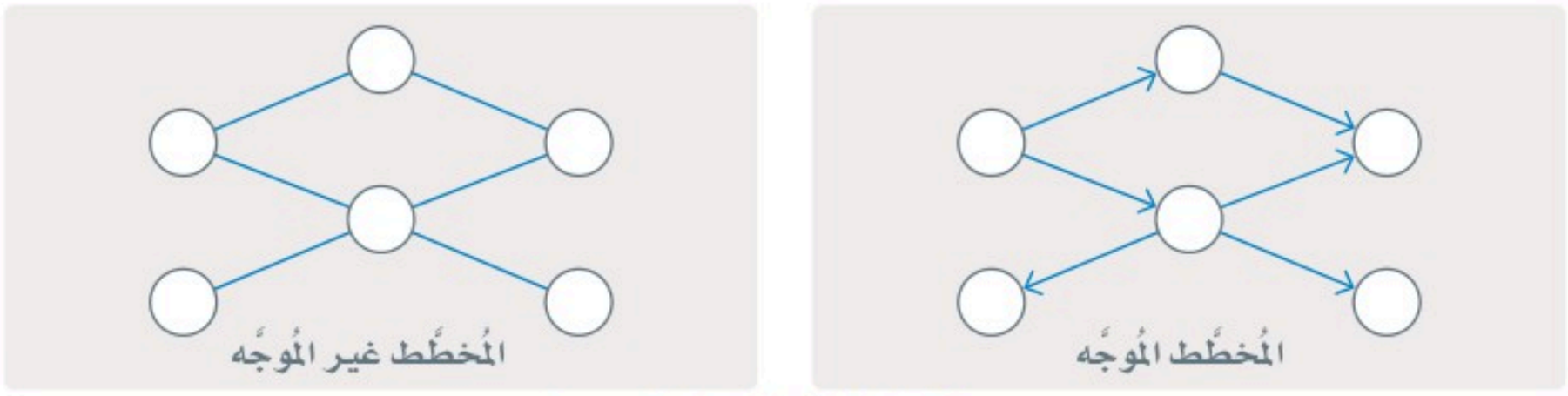
جدول 1.11: الفرق بين الأشجار والمُخطَّطات

الأشجار	المُخطَّطات
تشكّل العُقد المتّصلة فيها نموذجًا هرميًّا.	تشكّل العُقد المتّصلة فيها نموذجًا شبكيًّا.
في الأشجار الجذرية توجد عُقدة فريدة تُسمى الجِذر.	لا توجد فيها عُقدة فريدة أو جذرية.
ترتبط العُقد في صورة علاقة بين الأصل والفَرع.	لا تنطبق علاقة الأصل والفَرع بين العُقد.
تتميز ببساطة التركيب.	تركيب المُخطَّطات أكثر تعقيدًا.
لا يُسمح فيها بالحلقات.	قد تحتوي على الحلقات.

أنواع المُخطَّطات Types of Graphs

- **المُخطَّط المُوجَّه** (Directed Graph): ترتبط العُقد بالحواف الموجهة في المُخطَّط المُوجَّه، بحيث يكون للحافة اتجاه واحد.
- **المُخطَّط غير المُوجَّه** (Undirected graphs): لا تحتوي الوصلات على اتجاه في المُخطَّط غير المُوجَّه، وهذا يعني أن الحواف تشير إلى علاقة ثنائية الاتجاه يمكن من خلالها عرض البيانات في كلا الاتجاهين.

الشكل 1.43 يعرض مُخطَّطًا موجَّهًا، ومُخطَّطًا غير مُوجَّه يتكونان من ست عُقد وست حواف.



شكل 1.43: المُخطَّط المُوجَّه والمُخطَّط غير المُوجَّه

المُخطَّطات في الحياة اليومية Graphs in Everyday Life

شبكة الويب العالمية World Wide Web

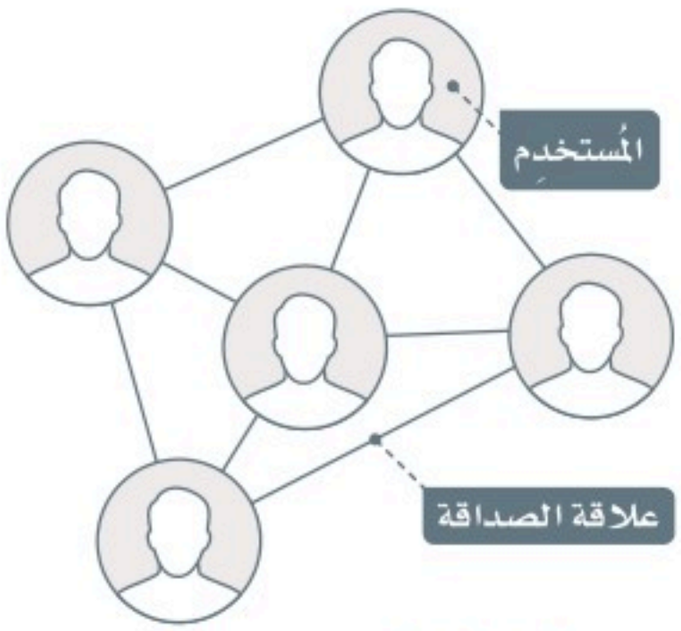
تُعدُّ شبكة الويب العالمية من أبرز الأمثلة للمُخطَّطات، ويمكن اعتبارها بمثابة أحد أنواع المُخطَّطات المُوجَّهة حيث تُمثِّل **الرؤوس** (Vertices) صفحات الويب، وتُمثِّل الارتباطات التشعبية الحواف المُوجَّهة. **تنقيب بُنيَة الويب** (Web Structure Mining) هو اكتشاف المعرِفة المُفيدة من هيكل شبكة الويب المُمَثلَة من خلال الارتباطات التشعبية، ويمكن أن يمثّل هيكل المُخطَّط الارتباطات التشعبية والعلاقات التي تُنشئها بين صفحات الويب المختلفة. يعرض الشكل 1.44 رسمًا توضيحيًا لشبكة الويب العالمية. باستخدام هذه المُخطَّطات يُمكنك حساب الأهمية النسبية لصفحات الويب.

شكل 1.44: شبكة الويب العالمية

يَستخدِم **مُحرِّك البحث قوقل** (Google Search Engine) منهجية مماثلة لتحديد الأهمية النسبية لصفحات الويب ومن ثمَّ ترتيب نتائج البحث حسب أهميتها. الخوارزمية المُستخدَمة بواسطة قوقل هي خوارزمية **تصنيف الصفحة** أو **بيج رانك** (PageRank) التي ابتكرها مؤسِّسو قوقل.

فيسبوك Facebook

فيسبوك هو مثال آخر على المُخطَّطات غير المُوجَّهة. يظهر بالشكل 1.45 العُقد التي تُمثِّل مُستخدِمي فيسبوك، بينما تُمثِّل الحواف علاقات الصداقة. عندما تريد إضافة صديق، يجب عليه قبول طلب الصداقة؛ ولن يكون ذلك الشخص صديقك على الشبكة دون قبول طلب الصداقة. العلاقة هنا بين اثنين من المُستخدِمين (عُقدتين) هي علاقة ثنائية الاتجاه. تُستخدَم خوارزمية مقترحات الأصدقاء في فيسبوك نظرية المُخطَّطات. تَدرس تحليلات الشبكات الاجتماعية العلاقات الاجتماعية باستخدام نظرية المُخطَّطات أو الشبكات من علوم الحاسب.



شكل 1.45: مُخطَّط فيسبوك غير المُوجَّه

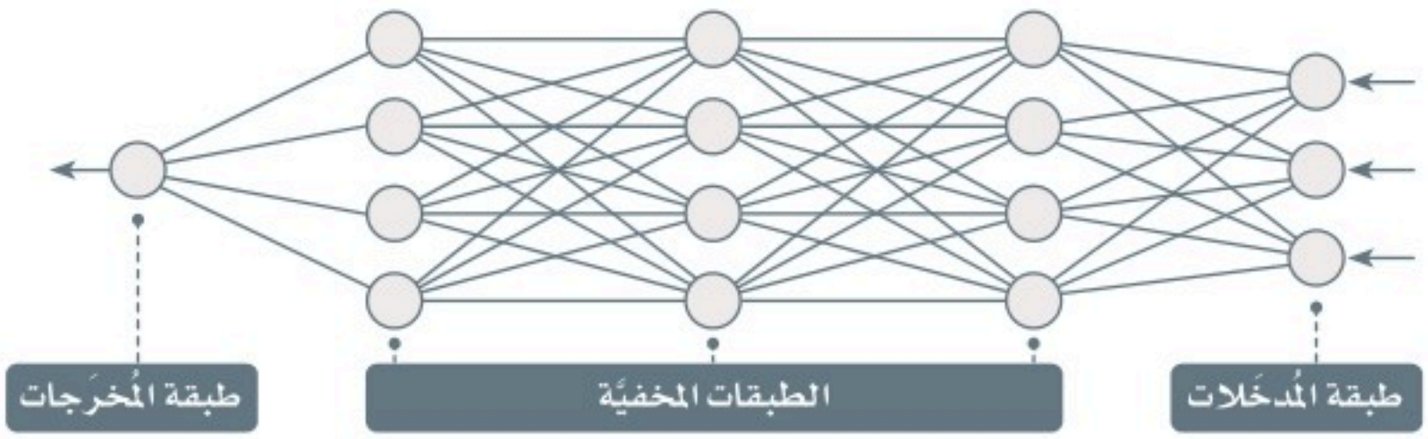
خرائط قوقل Google Maps

يستخدِم تطبيق خرائط قوقل وكل التطبيقات المُشابهة له المُخطَّطات لعرض أنظمة النقل والمواصلات لحساب المسار الأقصر بين موقعين. تَستخدِم هذه التطبيقات المُخطَّطات التي تحتوي على عدد كبير جدًا من العُقد والحواف التي لا يُمكن تمييزها بالعين المُجردة.

شكل 1.46: خرائط قوقل

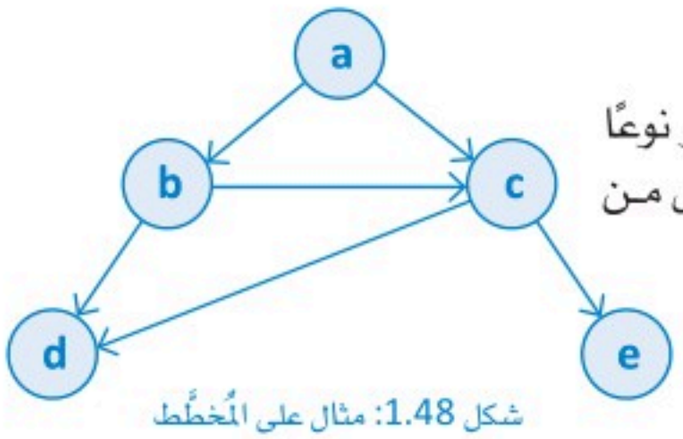
الشبكة العصبية Neural Network

الشبكة العصبية هي نوع مُخطَّط تعلُّم الآلة الذي يُحاكي الدماغ البشري. الشبكات العصبية يُمكن أن تكون شبكات مُوجَّهة أو غير مُوجَّهة وفقًا للغرض من التعلُّم، وتتكوّن هذه الشّبكات من الخلايا العصبية والأوزان المُوزعة في الطبقات المختلفة. تُمثَّل الخلايا العصبية بالعُقد، بينما تُمثَّل الأوزان بالحواف. يتم حساب تدفقات الإشارة وتحسينها في جميع أنحاء بُنية الشبكات العصبية لتقليل الخطأ. تُستخدَم الشبكات العصبية في العديد من التطبيقات الذكية مثل: الترجمة الآلية، وتصنيف الصور، وتحديد الكائنات، والتعرّف عليها. الشكل 1.47 يوضِّح مثالًا على هيكل الشبكات العصبية.



شكل 1.47: هيكل الشبكات العصبية

المُخطَّطات في لغة البايثون Graphs in Python



شكل 1.48: مثال على المُخطَّط

لا تُوفِر لغة البايثون نوعًا محددًا مسبقًا من البيانات للأشجار، كما أنّها لا تُوفِر نوعًا محددًا مسبقًا من البيانات للمُخطَّطات، (تذكر أن الأشجار هي نوع خاص من المُخطَّطات). ومع ذلك، يُمكن بِناء المُخطَّطات باستخدام القوائم والقواميس.

في المثال التالي، ستقوم بتنفيذ التالي:

1. إنشاء مُخطَّط مُوجَّه مثل المُوضح بالشكل 1.48.
2. إنشاء دالة لإضافة عُقدة إلى المُخطَّط.
3. إنشاء كائن يحتوي على كل مسارات المُخطَّط.

```
myGraph = { "a" : ["b","c"],
            "b" : ["c", "d"],
            "c" : ["d", "e"],
            "d" : [],
            "e" : [],
            }
print(myGraph)
```

```
{'a': ['b', 'c'], 'b': ['c', 'd'], 'c': ['d', 'e'],
 'd': [], 'e': []}
```

وسيتولّى البرنامج الرئيس:

1. إنشاء المُخطَّط.
2. طباعة المُخطَّط.
3. استدعاء دالة الإضافة.
4. طباعة كل مسارات المُخطَّط.

ستَستخدِم القاموس الذي تُمثِّل مفاتيحه العُقد بالمُخطَّط. تكون القيمة المقابِلة لكل مفتاح هي قائمة تحتوي على العُقد المتصلة بحافة مباشرة من هذه العُقدة.

```
# function for adding an edge to a graph
def addEdge(graph,u,v):
    graph[u].append(v)

# function for generating the edges of a graph
def generate_edges(graph):
    edges = []

    # for each node in graph
    for node in graph:
```

```
            # for each neighbouring node of a single node
            for neighbour in graph[node]:

                # if edge exists then append to the list
                edges.append((node, neighbour))
        return edges

# main program
# initialisation of graph as dictionary
myGraph = {"a" : ["b","c"],
           "b" : ["c", "d"],
           "c" : ["d", "e"],
           "d" : [],
           "e" : [],
          }

# print the graph contents
print("The graph contents")
print(generate_edges(myGraph))

# add more edges to the graph
addEdge(myGraph,'a','e')
addEdge(myGraph,'c','f')

# print the graph after adding new edges
print("The new graph after adding new edges")
print(generate_edges(myGraph))
```

```
The graph contents
[('a', 'b'), ('a', 'c'), ('b', 'c'), ('b', 'd'), ('c', 'd'), ('c', 'e')]
The new graph after adding new edges
[('a', 'b'), ('a', 'c'), ('a', 'e'), ('b', 'c'), ('b', 'd'), ('c', 'd'),
('c', 'e'), ('c', 'f')]
```

تمرينات

1

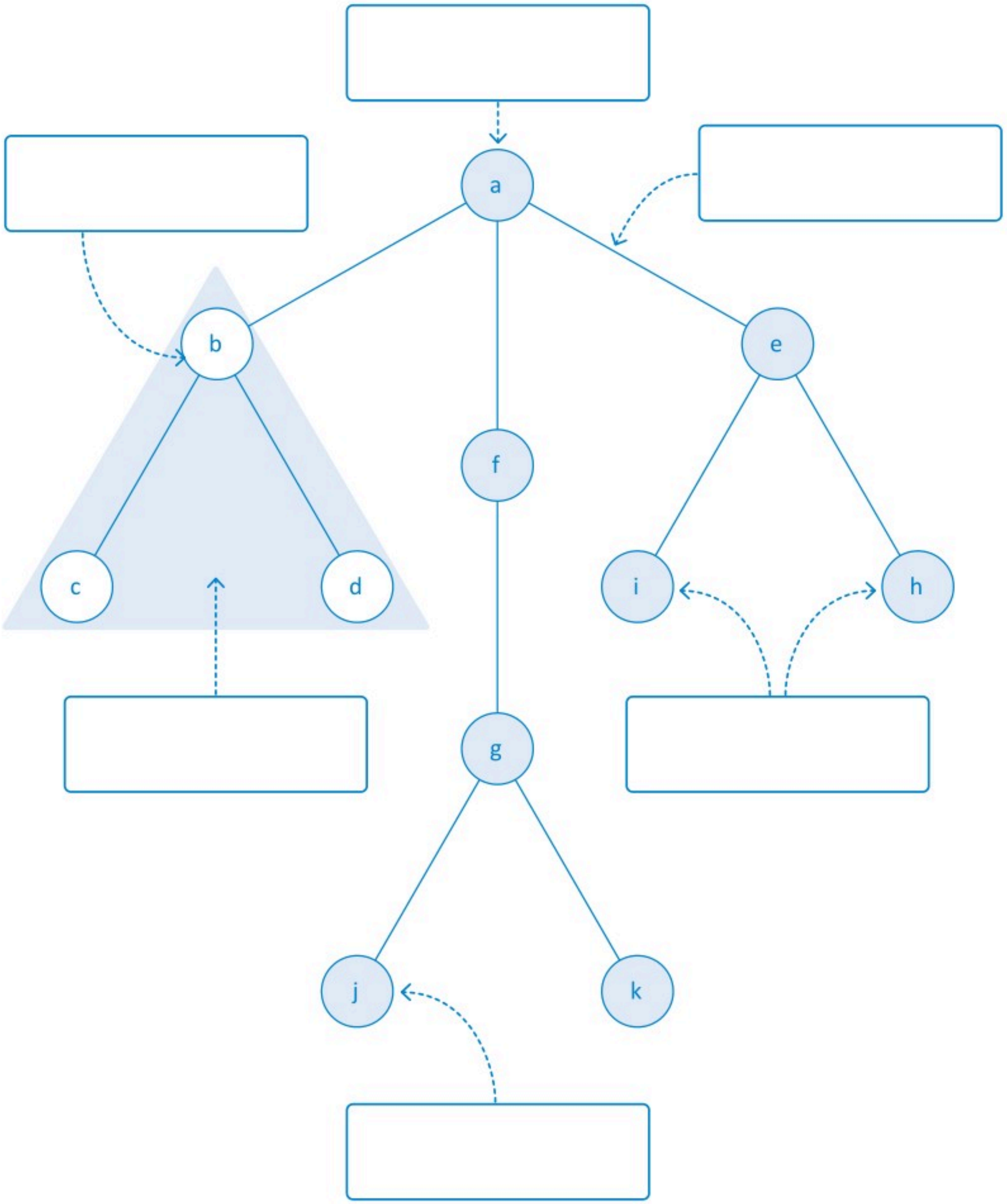
حدِّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. يمكن ربط العنصر في هياكل البيانات غير الخطيَّة بأكثر من عنصر واحد.	○	○
2. تنفيذ هياكل البيانات الخطيَّة يكون أكثر تعقيدًا من تنفيذ هياكل البيانات غير الخطيَّة.	○	○
3. الأوراق في تعلُّم شجرة القرار تحتوي على حلول المشكلة.	○	○
4. تَحسِب خوارزمية قوقل تصنيف الصفحة (PageRank) الأهمية النسبية لصفحة ويبَ على شبكة الويب العالمية.	○	○
5. الشبكات العصبية هي نوع المُخطَّطات المُستخدَم لتصوير المشكلات الأخرى.	○	○

2 وضِّح الاختلافات بين الأشجار والمُخطَّطات.

الأشجار	المُخطَّطات

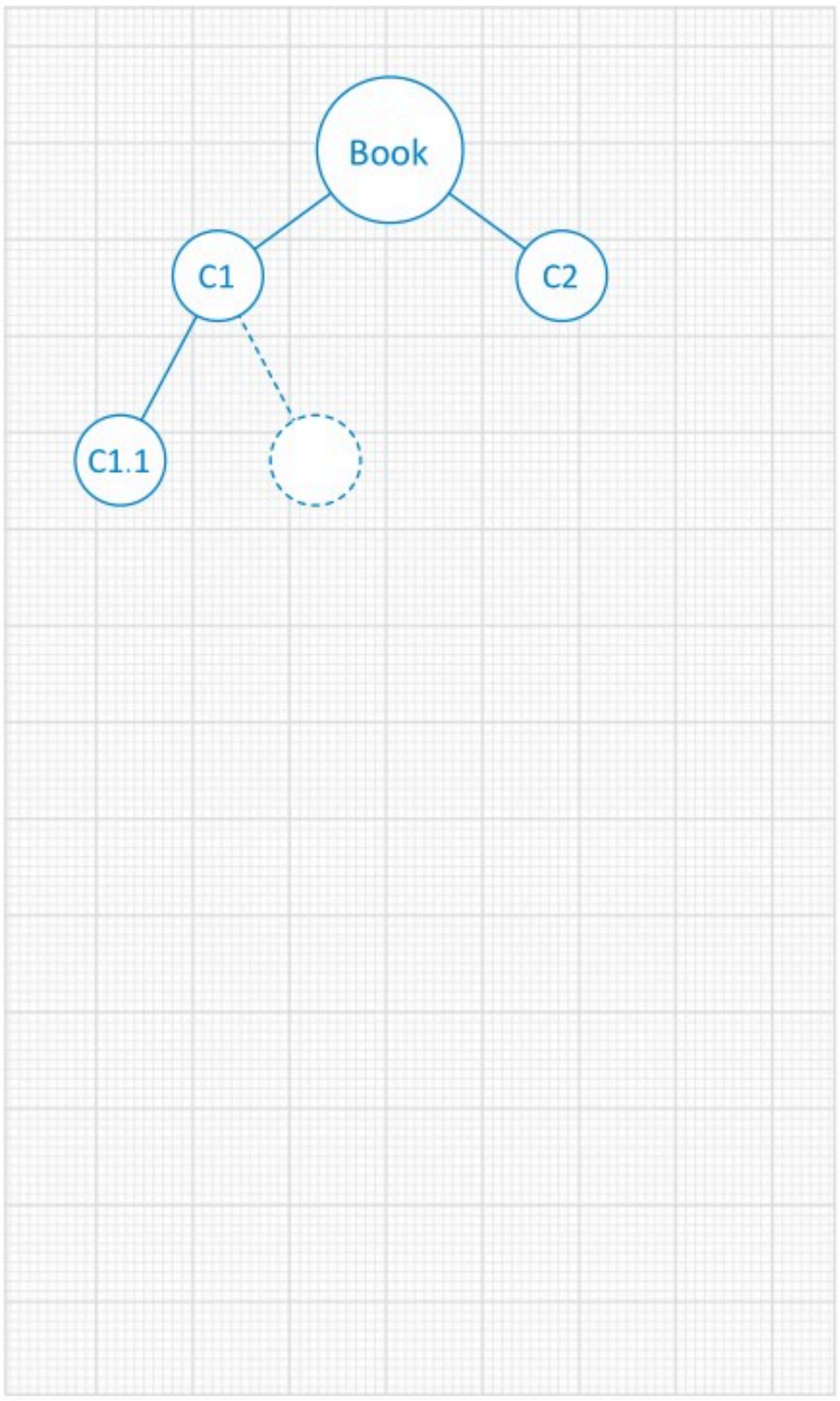
3 صِفْ كيف تُستخدَم خوارزميات المُخطَّطات في التطبيقات التجارية.

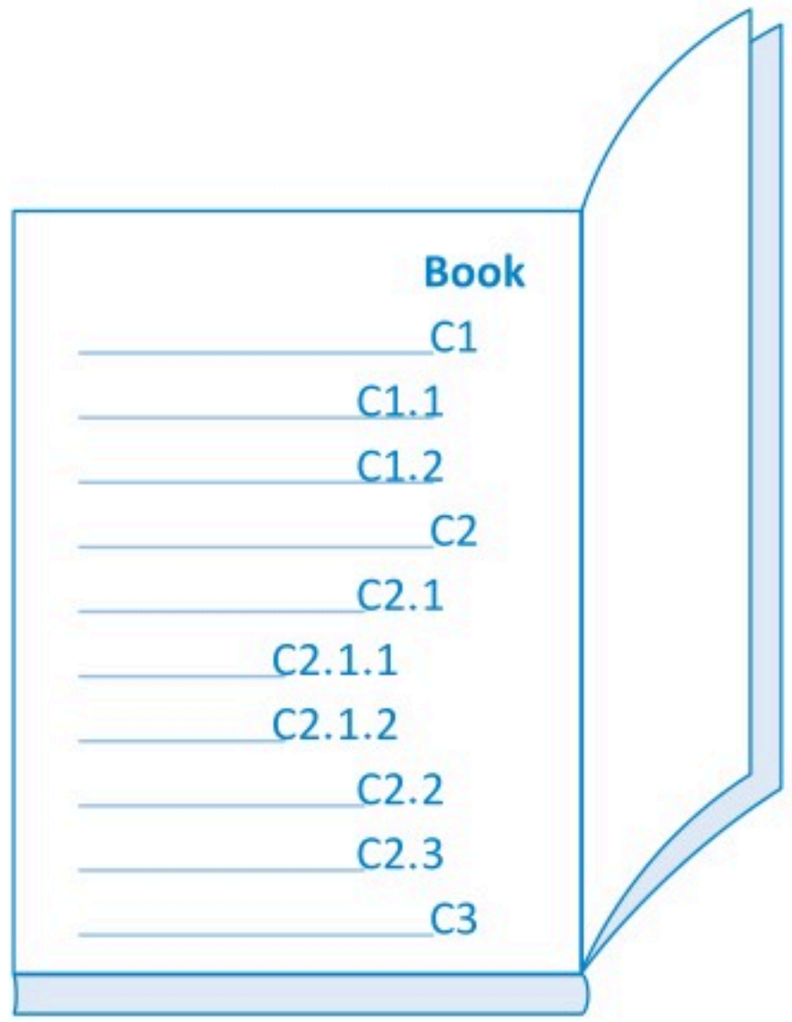
4 املأ الفراغات بالأسماء الصحيحة لأجزاء الشجرة.



5 يظهر أمامك في الصورة التالية صفحة محتويات الكتاب.

- أكمل تمثيل الشجرة.





- هل هي شجرة ثنائية؟ عَلّل إجابتك.

6 ارسم الشجرة الناتجة عن المعطيات التالية:

- العُقدة A لها فَرعان B وC.
- العُقدتان D وE لهما الأصل نفسه وهو العُقدة B.
- العُقدتان F وG شقيقتان، ولهما الأصل نفسه وهو العُقدة C.
- العُقدة H لها عُقدتان فَرعيتان I وJ ولها عُقدة أصل F.

ما نوع الشجرة المرسومة في الأعلى؟

باستخدام القاموس في لغة البايثون اكتب البرنامج المناسب لتمثيل هذه الشجرة، ثم أضف العُقدة الأصل والعُقد الفرعية.

المشروع

تُقدَّم الخدمة للعملاء في أحد البنوك بناءً على وقت وصولهم إلى فرع البنك. يعمل بالبنك موظف وحيد، ومُتوسط وقت الخدمة لكل عميل هو دقيقتان.

لا يُسمح بأن يتجاوز الطابور في البنك 40 عميلًا.

1

أنشئ برنامجًا بلغة البايثون يستدعي إحدى قيم الاستيراد: ENTRY (دخول) أو NEXT (التالي).

- إن أدخلت القيمة ENTRY (دخول)، سيقرأ البرنامج اسم العميل وبعدها مباشرةً يُظهِر عدد الأشخاص في قائمة الانتظار أمامه. إن كان الطابور مُمتلئًا، تظهر رسالة The branch is full. Come another day (الفَرع مُمتلئ. الرجاء العودة في يوم آخر).
- إن أدخلت القيمة NEXT (التالي)، لابد أن يظهر اسم العميل التالي الذي ستُقدَّم له الخدمة.

2

كَرِّر العملية الموضحة أعلاه حتى لا يكون هناك عملاء في قائمة الانتظار.

3

في النهاية، سيَعرض البرنامج على الشاشة:

- عدد العملاء الذين قُدِّمت لهم الخدمة.
- متوسط وقت انتظار العميل.

ماذا تعلّمت

- مفهوم الذكاء الاصطناعي.
- تصنيف تطبيقات الذكاء الاصطناعي.
- تصنيف هياكل البيانات.
- تحديد الاختلافات بين هيكل بيانات المُكدّس وهيكل بيانات الطابور.
- تحديد الاختلافات بين هيكل بيانات القائمة وهيكل بيانات القائمة المترابطة.
- تحديد الاختلافات بين هيكل بيانات الشجرة وهيكل بيانات المُخطَّط.
- تطبيق هياكل البيانات المُعقَّدة باستخدام لغة برمجة البايثون.

المصطلحات الرئيسة

غير أوليّ	Non-Primitive
قيمة فارغة	Null
مؤشر	Pointer
حذف عنصر	Pop
أوليّ	Primitive
إضافة عنصر	Push
أخير	Rear
الجذر	Root
أشقاء	Siblings
المُكدّس	Stack
شجرة فرعية	Sub-Tree
أعلى	Top
غَيْض المُكدّس	Underflow
المُخطَّط غير الموجَّه	Undirected Graph

الشجرة الثنائية	Binary Tree
فَرع (ابن)	Child
هيكل البيانات	Data Structure
شجرة القرار	Decision Tree
حذف عنصر من الطابور	Dequeue
المُخطَّط المُوجَه	Directed Graph
متغير	Dynamic
الأمامي	Front
مُخطَّط	Graph
فهرس	Index
رأس	Head
ورقة	Leaf
خطي	Linear
قائمة مترابطة	Linked List
غير خطي	Non-Linear

2. خوارزميات الذكاء الاصطناعي

سيتعرف الطالب في هذه الوحدة على بعض الخوارزميات الأساسية المُستخدَمة في الذكاء الاصطناعي (AI). كما سيتعلّم كيف يُنْشِئ نظام تشخيص طبي بسيط مُستنِد إلى القواعد بِطرائق برمجية مُتعددة ثم يقارن النتائج. وفي الختام سيتعلّم خوارزميات البحث وطرائق حل ألغاز المتاهة مع أخذ معايير معيّنة في الاعتبار.

أهداف التعلُّم

بنهاية هذه الوحدة سيكون الطالب قادرًا على أن:

- يُنشئ مقطعًا برمجيًا تكراريًا.
- يُقارن بين خوارزمية البحث بأولوية الاتساع وخوارزمية البحث بأولوية العُمق.
- يَصف خوارزميات البحث وتطبيقاتها.
- يُقارن بين خوارزميات البحث.
- يَصف النظام القائم على القواعد.
- يُدرِّب نماذج الذكاء الاصطناعي حتى تتعلّم حل المشكلات المُعقدة.
- يُقيِّم نتائج المقطع البرمجي وكفاءة البرنامج الذي أنشأه.
- يُطوِّر البرامج لمحاكاة حلّ مشكلات الحياة الواقعية.
- يُقارِن بين خوارزميات البحث.

الأدوات

- مفكّرة جوبيتر (Jupyter Notebook)

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الأول

الاستدعاء الذاتي

تقسيم المُشكلة Dividing the Problem

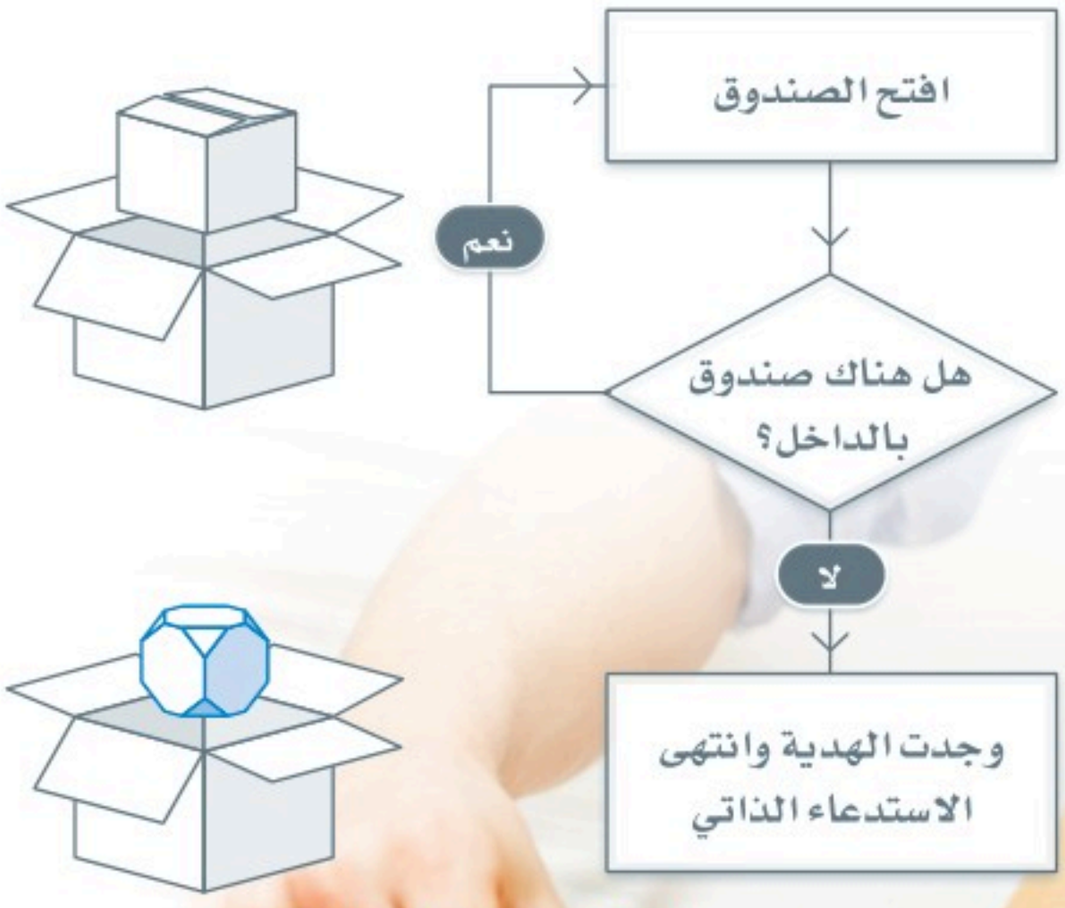
في هذا الدرس، ستتعلّم استخدام الدوال التكرارية لتبسيط البرنامج وزيادة كفاءته.

تخيّل أن والداكَ قد أحضرا لكَ هديّة، وكنت مُتلهفًا لمعرفتها، ولكن عندما فتحت الصندوق، وجدت صندوقًا جديدًا بداخله، وعندما فتحته، وجدتَ آخرَ بداخله، وهكذا حتى عجزت أن تعرف في أي صندوق توجد الهدية.

الاستدعاء الذاتي Recursion

الاستدعاء الذاتي هو أحد طرائق حل المشكلات في علوم الحاسب، ويتم عن طريق تقسيم المشكلة إلى مجموعة من المشكلات الصغيرة المُشابهة للمشكلة الأصلية حتى يُمكنك استخدام الخوارزمية نفسها لحل تلك المشكلات. يُستخدَم الاستدعاء الذاتي بواسطة أنظمة التشغيل والتطبيقات الأخرى، كما تدعمه معظم لغات البرمجة.

يحدث الاستدعاء الذاتي عندما تتكرر التعليمات نفسها، ولكن مع بيانات مختلفة وأقل تعقيدًا.



شكل 2.1: مثال على الاستدعاء الذاتي

لتُلقِ نظرة على مثال لدالة تستدعي دالة أخرى.

```
def mySumGrade (gradesList):
    sumGrade=0
    l=len(gradesList)
    for i in range(l):
        sumGrade=sumGrade+gradesList[i]
    return sumGrade

def avgFunc (gradesList):
    s=mySumGrade(gradesList)
    l=len(gradesList)
    avg=s/l
    return avg

# program section
grades=[89,88,98,95]
averageGrade=avgFunc(grades)
print ("The average grade is: ",averageGrade)
```

استدعاء الدالة .mySumGrade

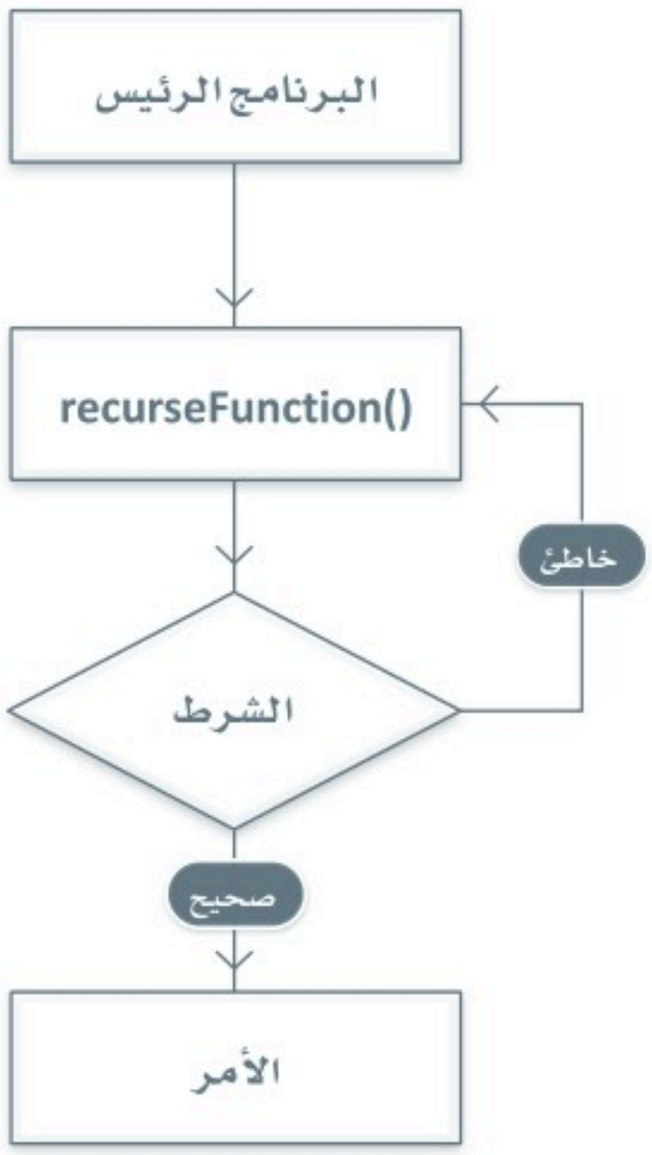
تستخدِم دالة len() قائمة كمُعامِل مُدخَل، لحساب وتحديد عدد العناصر في القائمة.

```
The average grade is: 92.5
```

دالة الاستدعاء التكرارية Recursive Function

في بعض الحالات تستدعي الدّالةُ نفسَها وهذه الخاصية تُسمى **الاستدعاء التكراري** (Recursive Call).

يكون بِناء الجملة العام لدالة الاستدعاء التكرارية على النحو التالي:



شكل 2.2: تمثيل الاستدعاء التكراري

```
# recursive function
def recurseFunction():
    if (condition): # base case
        statement
    else:
        #recursive call
        recurseFunction()

# main program
.......

# normal function call
recurseFunction()
.........
```

الاستدعاء التكراري هو عملية استدعاء الدالة لنفسها.

تتكون دالة الاستدعاء التكرارية من حالتين:

الحالة الأساسية Base Case

وفي هذه الحالة تتوقف الدالة عن استدعاء نفسها، ويتأكّد الوصول إلى هذه الحالة من خلال الأمر المشروط. بدون الحالة الأساسية، ستَتَكرَّر عملية الاستدعاء الذاتي إلى ما لا نهاية.

حالة الاستدعاء التكرارية Recursive Case

وفي هذه الحالة تستدعي الدالةُ نفسَها عندما لا تُحقق شرط التوقف، وتظل الدالة في حالة الاستدعاء الذاتي حتى تصل إلى الحالة الأساسية.

أمثلة شائعة على الاستدعاء الذاتي Recursion Common Examples

أحد الأمثلة الأكثر شيوعًا على استخدام الاستدعاء الذاتي هو عملية حساب مضروب رقم مُعيّن. مضروب الرقم هو ناتج ضرب جميع الأعداد الطبيعية الأقل من أو تساوي ذلك الرقم. يُعبَّر عن المضروب بالرقم متبوعًا بالعلامة "!"، على سبيل المثال، مضروب الرقم 5 هو !5 ويساوي 5*4*3*2*1.

جدول 2.1: مضروب الأرقام من 0 إلى 5

0!	0! = 1	
1!	1! = 1*1 = 1	أو 1! = 0! *1
2!	2! = 2*1 = 2	أو 2! = 1! *2
3!	3! = 3*2*1 = 6	أو 3! = 2! *3
4!	4! = 4*3*2*1 = 24	أو 4! = 3! * 4
5!	5!= 5*4*3*2*1 = 120	أو 5! = 4! * 5

ستلاحظ أن عملية حساب المضروب تستند إلى القاعدة أدناه:

$$n! = \begin{cases} 1 & ,if\ n=0 \\ (n-1)! * n & if\ n>0 \end{cases}$$

الحالة الأساسية.

حالة الاستدعاء التكرارية.

شكل 2.3: قاعدة حساب المضروب

لإنشاء برنامج يقوم باحتساب مضروب العدد باستخدام حلقة التكرارfor، اتّبع ما يلي:

```
# calculate the factorial of an integer using iteration

def factorialLoop(n):
    result = 1
    for i in range(2,n+1):
            result = result * i

    return result

# main program
num = int(input("Type a number: "))
f=factorialLoop(num)
print("The factorial of ", num, " is:", f)
```

```
Type a number: 3
The factorial of 3 is:6
```

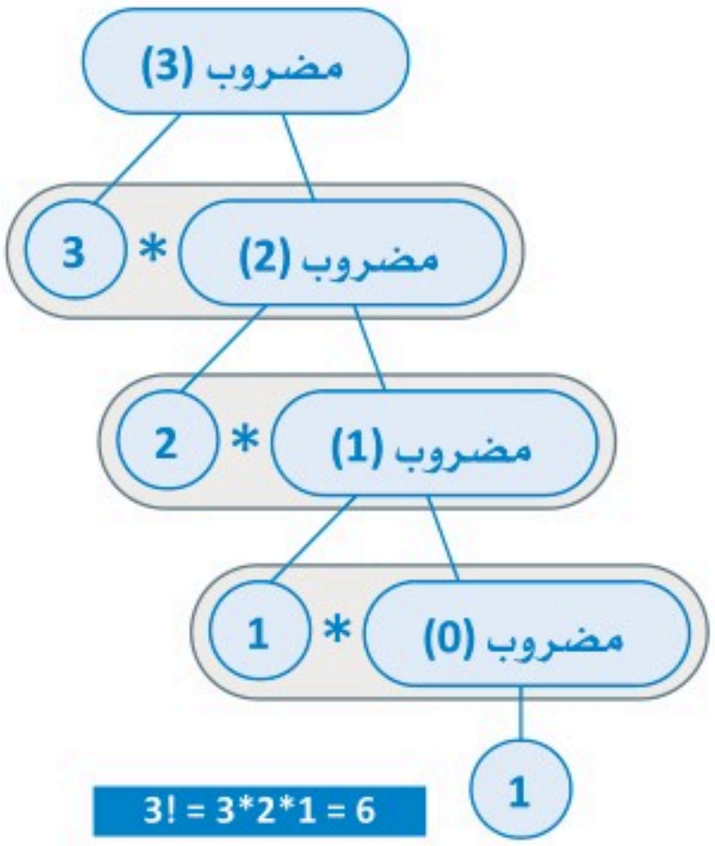
الآن احسِب مضروب العدد باستخدام دالة المضروب.

```
# calculate the factorial of an integer using a
# recursive function
def factorial(x):
    if x == 0:
        return 1
    else:
        return (x * factorial(x-1))

# main program
num = int(input("Type a number: "))
f=factorial(num)
print("The factorial of ", num, " is: ", f)
```

الحالة الأساسية.

حالة الاستدعاء التكرارية.

```
Type a number: 3
The factorial of 3 is: 6
```



شكل 2.4: شجرة الاستدعاء الذاتي

جدول 2.2: مزايا الاستدعاء الذَاتي وعيوبه

المزايا	العيوب
• تقلل دوال الاستدعاء التكرارية من عدد التعليمات في المقطع البرمجي. • يمكن تقسيم المهمة إلى مجموعة من المشكلات الفرعية باستخدام الاستدعاء الذاتي. • في بعض الأحيان، يَسهُل استخدام الاستدعاء الذاتي لاستبدال التكرارات المُتداخلة.	• في بعض الأحيان، يَصعُب تَتبُّع منطق دوال الاستدعاء التكرارية. • يتطلب الاستدعاء الذاتي مزيدًا من الذاكرة والوقت. • لا يسهل تحديد الحالات التي يمكن فيها استخدام دوال الاستدعاء التكرارية.

الاستدعاء الذاتي والتكرار Recursion and Iteration

يُستخدم كلٌّ من الاستدعاء الذاتي والتكرار في تنفيذ مجموعة من التعليمات لعدة مرات، والفارق الرئيس بين الاستدعاء الذاتي والتكرار هو طريقة إنهاء الدالة التكرارية. دالة الاستدعاء التكرارية تستدعي نفسها وتُنهي التنفيذ عندما تصل إلى الحالة الأساسية. أما التكرار فيُنفِّذ لِبِنَةَ المقطع البرمجي باستمرار حتى يتحقق شرط مُحدَّد أو ينقضي عدد مُحدَّد من التكرارات.

الجدول التالي يعرض بعض الاختلافات بين الاستدعاء الذاتي والتكرار.

جدول 2.3: التكرار والاستدعاء الذاتي

التكرار	الاستدعاء الذاتي
سريع التنفيذ.	بطيء التنفيذ مقارنةً بالتكرار.
يتطلب حجم ذاكرة أقل.	يتطلب حجم ذاكرة أكبر.
حجم المقطع البرمجي أكبر.	حجم المقطع البرمجي أصغر.
ينتهي باستكمال العدد المُحدَّد من التكرارات أو تحقيق شرط مُعيَّن.	ينتهي بمجرد الوصول إلى الحالة الأساسية.

متى تَستخدِم الاستدعاء الذاتي؟

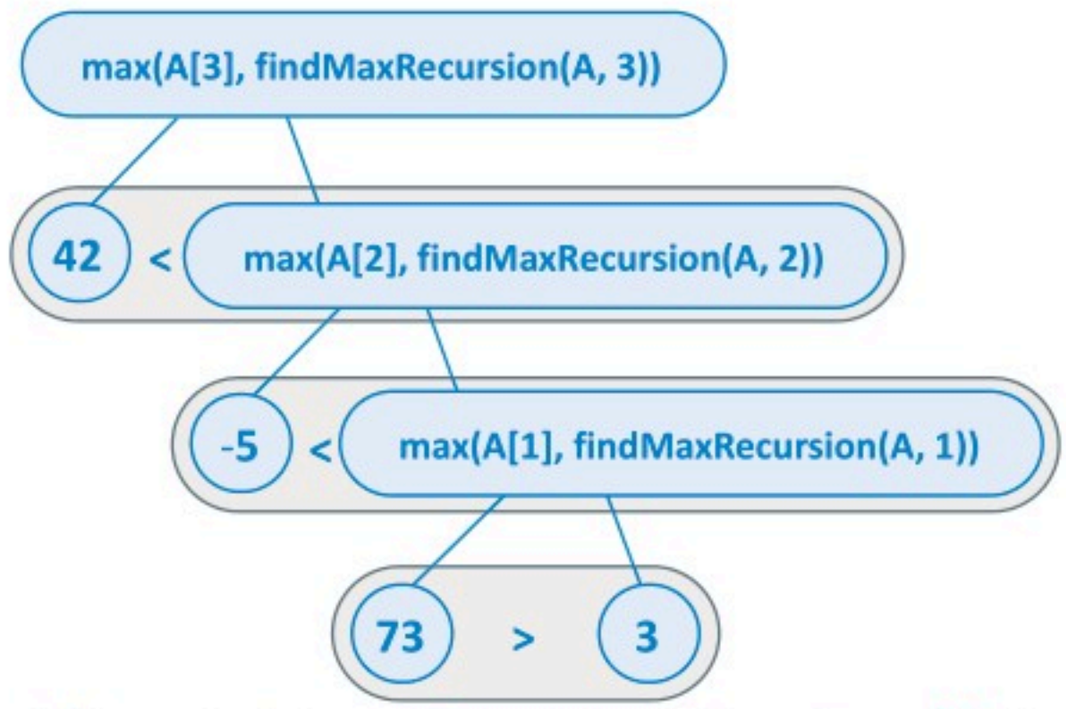
- يُعدُّ الاستدعاء الذاتي الطريقة الأكثر ملائمة للتعامل مع المشكلة في العديد من الحالات.
- يَسهُل استكشاف بعض هياكل البيانات باستخدام الاستدعاء الذاتي.
- بعض **خوارزميات التصنيف** (Sorting Algorithms)، تَستخدِم الاستدعاء الذاتي، مثل: **التصنيف السريع** (Quick Sort).

في المثال التالي، ستستخرج أكبر رقم موجود في قائمة مكونة من الأرقام باستخدام دالة الاستدعاء التكرارية.
كما يظهر في السطر الأخير من المثال دالة أخرى للتكرار لغرض المقارنة.

```
def findMaxRecursion(A,n):

    if n==1:
        m = A[n-1]
    else:
        m = max(A[n-1],findMaxRecursion(A,n-1))
    return m

def findMaxIteration(A,n):

    m = A[0]
    for i in range(1,n):
        m = max(m,A[i])
    return m

# main program
myList = [3,73,-5,42]
l = len(myList)
myMaxRecursion = findMaxRecursion(myList,l)
print("Max with recursion is: ", myMaxRecursion)
myMaxIteration = findMaxIteration(myList,l)
print("Max with iteration is: ", myMaxIteration)
```

تستخرج الدالة ()max العنصر ذا القيمة الأكبر (العنصر ذو القيمة الأكبر في myList).

```
Max with recursion is:  73
Max with iteration is:  73
```



شكل 2.5: شجرة الاستدعاء الذاتي لدالة استخراج أكبر رقم في قائمة مكونة من الأرقام

في البرنامج التالي، ستُنشئ دالة استدعاء تكرارية لحساب مُضاعَف الرقم.
ستقوم بإدخال رقمًا (الأساس) وفهرسًا (الأُس أو القُوَّة) يقبلهما البرنامج، ومن ثَمَّ ستَستخدِم دالة الاستدعاء التكرارية powerFunRecursive( ) التي ستَستخدِم هذين المُدخَلَين لحساب مُضاعَف الرقم. يمكن تحقيق الأمر نفسه باستخدام التكرار، والمثال التالي يوضّح ذلك:

```
def powerFunRecursive(baseNum,expNum):
    if(expNum==1):
        return(baseNum)
    else:
        return(baseNum*powerFunRecursive(baseNum,expNum-1))

def powerFunIteration(baseNum,expNum):

    numPower = 1
    for i in range(exp):
        numPower = numPower*base
    return numPower

# main program
base = int(input("Enter number: "))
exp = int(input("Enter exponent: "))
numPowerRecursion = powerFunRecursive(base,exp)
print( "Recursion: ", base, " raised to ", exp, " = ",numPowerRecursion)
numPowerIteration = powerFunIteration(base,exp)
print( "Iteration: ", base, " raised to ", exp, " = ",numPowerIteration)
```

```
Enter number: 10
Enter exponent: 3
Recursion: 10 raised to 3 = 1000
Iteration: 10 raised to 3 = 1000
```

دالة الاستدعاء التكرارية اللانهائية Infinite Recursive Function

يجب أن تكون حذرًا للغاية عند تنفيذ الاستدعاء التكراري، كما يجب عليك استخدام طريقة معينة لإيقاف التكرار عند تحقيق شرط مُحدَّد لتجنب حدوث الاستدعاء التكراريّ اللانهائيّ، الذي يسبّب توقُّف النظام عن الاستجابة بسبب كثرة استدعاءات الدالة، مما يؤدي إلى **فَيْض الذاكرة** (Memory Overflow) وإنهاء التطبيق.

تمرينات

1

حدِّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. تتكون دالة الاستدعاء التكرارية من حالتين.	○	○
2. تستدعي دالة الاستدعاء التكرارية دالة أخرى.	○	○
3. دوال الاستدعاء التكرارية أسرع في التنفيذ.	○	○
4. استدعاء الدوال يجعل لِبنة المقطع البرمجي أصغر حجمًا.	○	○
5. كتابة مقطع برمجي مُتكرِّر يتطلب استدعاءً ذاتيًّا أقل.	○	○

2 ما الاختلافات بين التكرار والاستدعاء الذاتي؟

3 متى يجب استخدام الاستدعاء الذاتي؟

4 وَضِّح مزايا استخدام الاستدعاء الذاتي وعيوبه.

5 اكتب دالة استدعاء تكرارية بلغة البايثون تقوم بحساب الرقم الأكبر بترتيب محدد (مثلًا ثاني أكبر رقم) في قائمة من الأرقام.

6 اكتُب دالة استدعاء تكرارية بلغة البايثون لحساب مجموع كل الأرقام الزوجية في قائمة معيّنة.

رابط الدرس الرقمي

www.ien.edu.sa

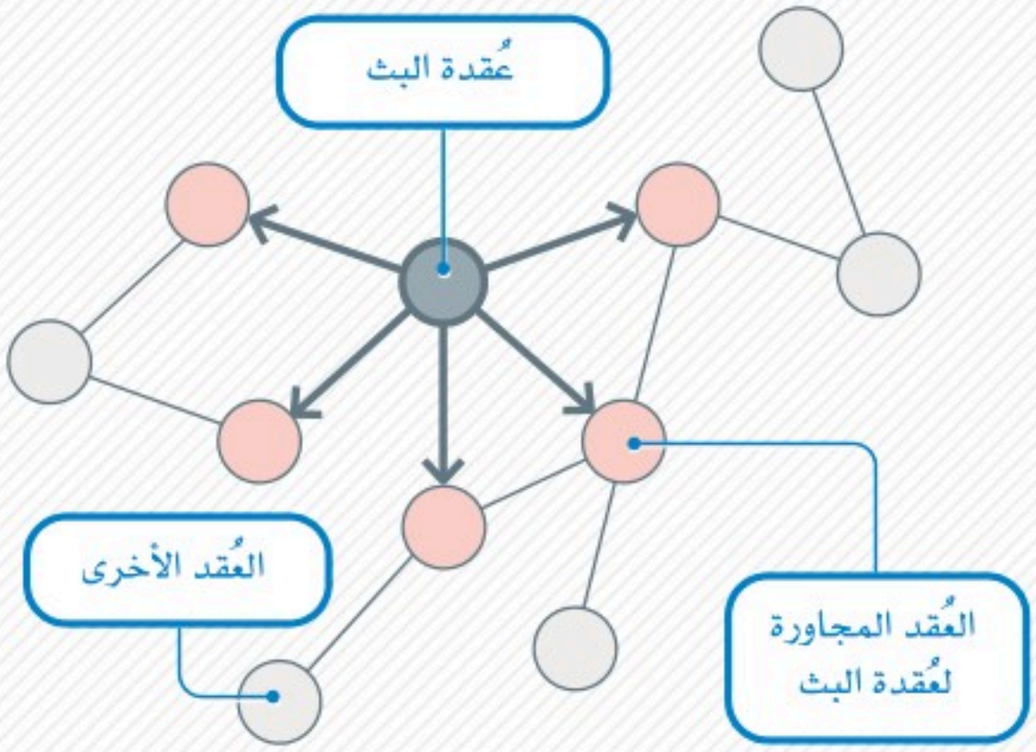
الدرس الثاني

خوارزمية البحث بأولوية العمق والبحث بأولوية الاتساع

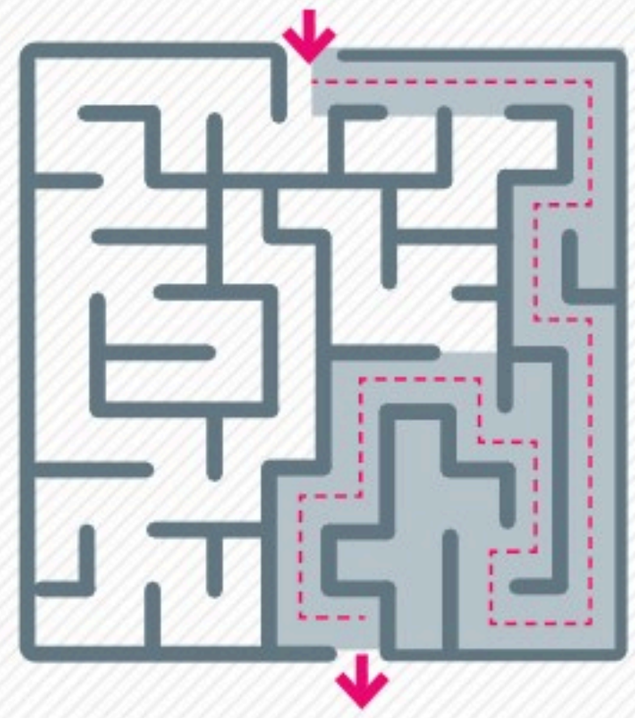
البحث في المُخطَّطات Searching in Graphs

هناك بعض الحالات التي تحتاج فيها إلى البحث عن عُقدة مُحدَّدة في المُخطَّط، أو تفحُّص كل عُقدة في المُخطَّط لإجراء عملية بعينها مثل طباعة عُقد المُخطَّط، فتكون حالتكَ كشخصٍ يبحث عن المدينة التي يريد السّفر إليها؛ وليتحقق هذا، تحتاج إلى فحص كل عُقدة في المُخطَّط حتى تجد تلك التي تحتاج إليها. يُطلق على هذا الإجراء: البحث في المُخطَّط أو مسح المُخطَّط، وهناك العديد من خوارزميات البحث التي تساعد على تنفيذه، مثل:

- **خوارزمية البحث بأولوية الاتساع** (Breadth-First Search - BFS).
- **خوارزمية البحث بأولوية العمق** (Depth-First Search - DFS).



مثال على خوارزمية البحث بأولوية الاتساع (BFS): البث الشبكي



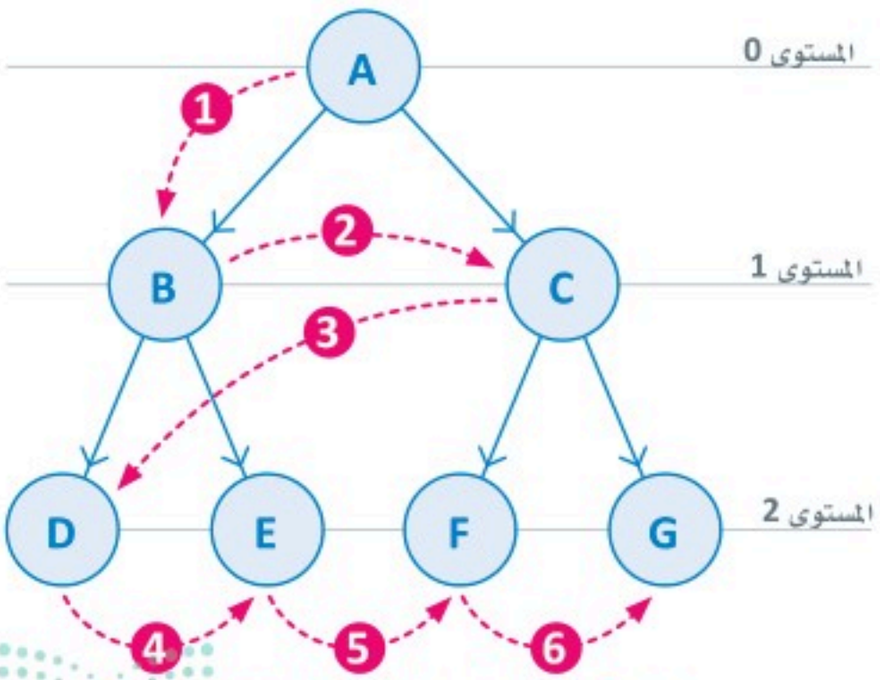
مثال على خوارزمية البحث بأولوية العمق (DFS): حل المتاهة

خوارزمية البحث بأولوية الاتساع
Breadth-First Search (BFS) Algorithm

تستكشف خوارزمية **البحث بأولوية الاتساع** (BFS) المُخطَّط بحسب المستوى واحدًا تلو الآخر، حيث تبدأ بفحص عُقدة الجذر (عُقدة البداية)، ثم تفحص جميع العُقد المرتبطة بها بشكل مباشر واحدة تلو الأخرى.

بعد الانتهاء من فحص كل العُقد في المستوى، تنتقل إلى المستوى التالي، وتتبع الإجراءات نفسها المُوضّحة في الشكل 2.6.

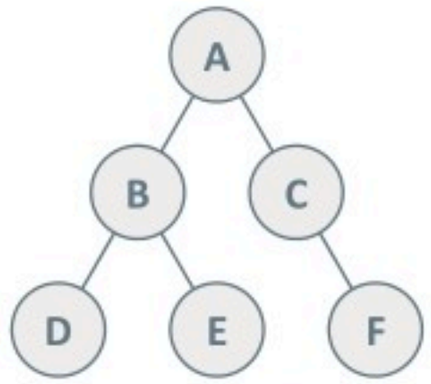
يُستخدَم الطّابور لتتبّع العُقد التي تمّ فحصها، وبمجرّد استكشاف العُقدة، ستتم إضافة العُقَد الفرعية إلى الطابور، ثم تحذف العُقدة التالية الموجودة في أول الطابور التي تم استكشافها سابقًا.



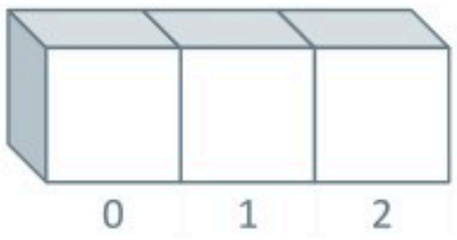
شكل 2.6: خوارزمية البحث بأولوية الاتساع (BFS)

المثال التالي يوضِّح طريقة عمل خوارزمية **البحث بأولوية الاتساع** (BFS). باستخدام المُخطَّط التالي، حدِّد العُقد التي يجب فحصها للانتقال من عُقدة الجذر A إلى العُقدة F:

ملاحظة: استخدِم هيكل البيانات المُناسب.



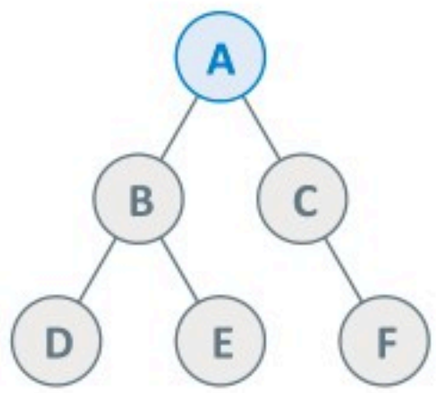
المُخطَّط



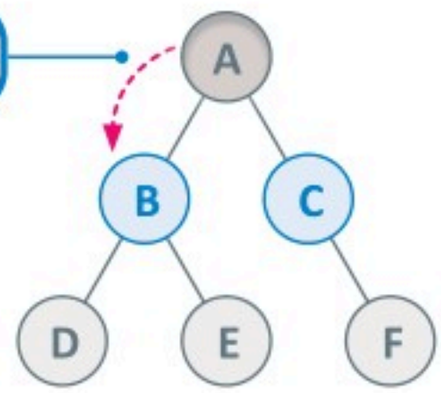
الطابور

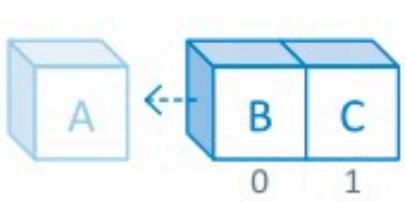
عليك فحص كل العُقد في المستوى 1 قبل الانتقال إلى العُقد في المستوى 2.

1 البداية من العُقدة الجذرية (العُقدة A). أضف العُقدة الجذريّة إلى الطابور.

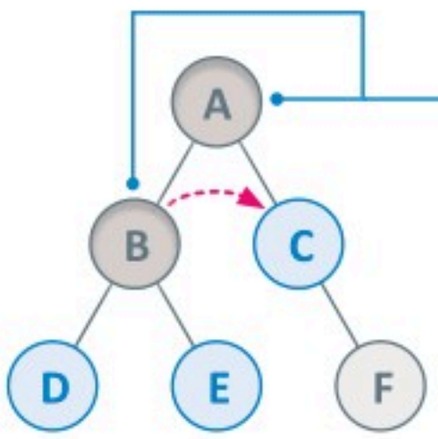


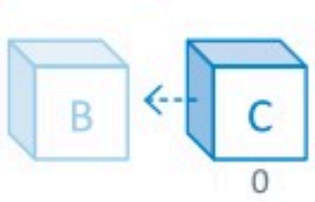
2 احذف العُقدة الجذريّة من الطابور لمعالجتها، ثم أضف فروع هذه العُقدة إلى الطابور (العُقدتين B وC).



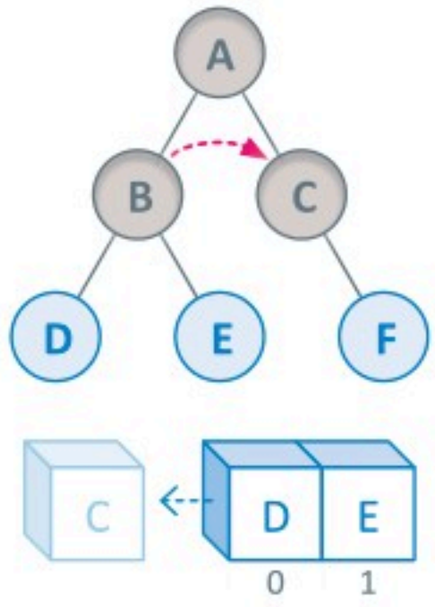


3 احذف العُقدة من مقدمة الطابور (العُقدة B) لمعالجتها، ثم أضف فروع هذه العُقدة إلى الطابور (العُقدتين D وE).

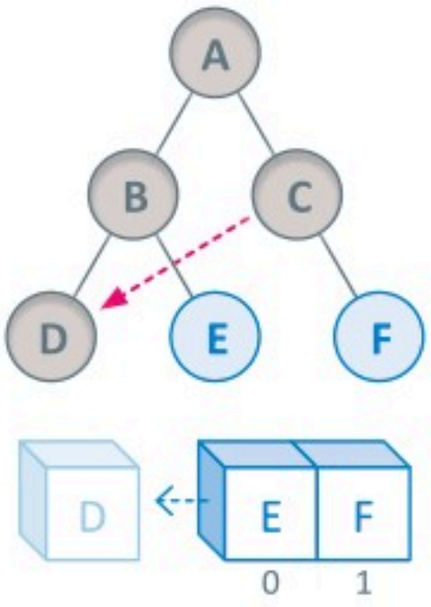




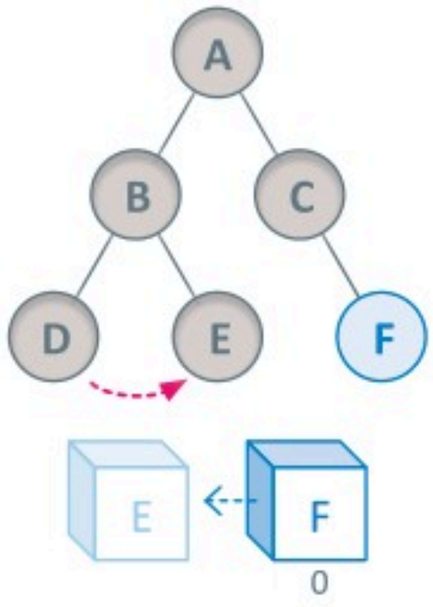
❹ احذف العُقدة C وعالجها، ثم أضف فَرعها إليها.



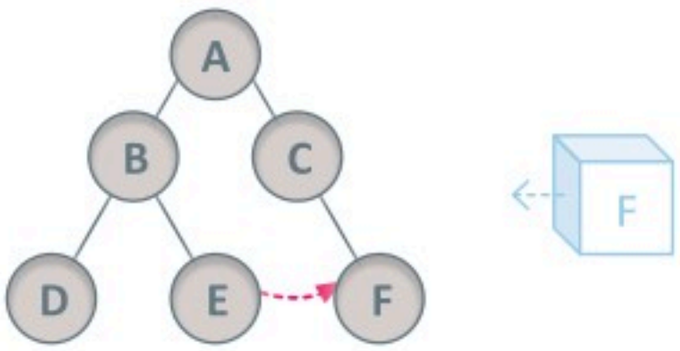
❺ احذف العُقدة D لمعالجتها. (ليس لديها فروع).



❻ احذف العُقدة E لمعالجتها. (ليس لديها فروع).



❼ احذف العُقدة F لمعالجتها، وبذلك أصبح الطابور الآن فارغًا وانتهت عملية البحث.



العُقد التي فُحصَت باستخدام خوارزمية البحث بأولوية الاتساع (BFS) هي: A، B، C، D، E، F.

لاحظ كيف يُمكنك تطبيق خوارزمية **البحث بأولوية الاتساع** (BFS) بلغة **البايثون** (Python) في المثال التالي:

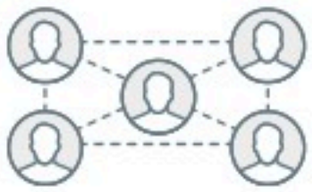
```
graph = {
    "A" : ["B","C"],
    "B" : ["D","E"],
    "C" : ["F"],
    "D" : [],
    "E" : [],
    "F" : []
}

visitedBFS = [] # List to keep track of visited nodes
queue = []      # Initialize a queue

 # bfs function
def bfs(visited, graph, node):
    visited.append(node)
```

```
  queue.append(node)

  while queue:
    n = queue.pop(0)
    print (n, end = " ")

    for neighbor in graph[n]:
      if neighbor not in visited:
        visited.append(neighbor)
        queue.append(neighbor)

# main program
bfs(visitedBFS, graph, "A")
```

```
A B C D E F
```

التطبيقات العملية لخوارزمية البحث بأولوية الاتساع
Practical Applications of the BFS Algorithm

تُستخدَم في **شبكات النَظير للنَظير** (Peer-to-Peer Networks) للعثور على كل العُقد المجاورة من أجل تأسيس الاتصال.



تُستخدَم في **وسائل التواصل الاجتماعي** (Social Media) لربط عُقد المُستخدِمين المُرتبطين، مثل أولئك الذين لهم الاهتمامات نفسها أو الموقع نفسه.

تُستخدَم في **نُظم الملاحة باستخدام مُحدِّد المواقع العالمي** (GPS Navigation Systems) للبحث عن الأماكن المتجاورة حتى تُحدِّد الاتجاهات التي يتبعها المُستخدِم.

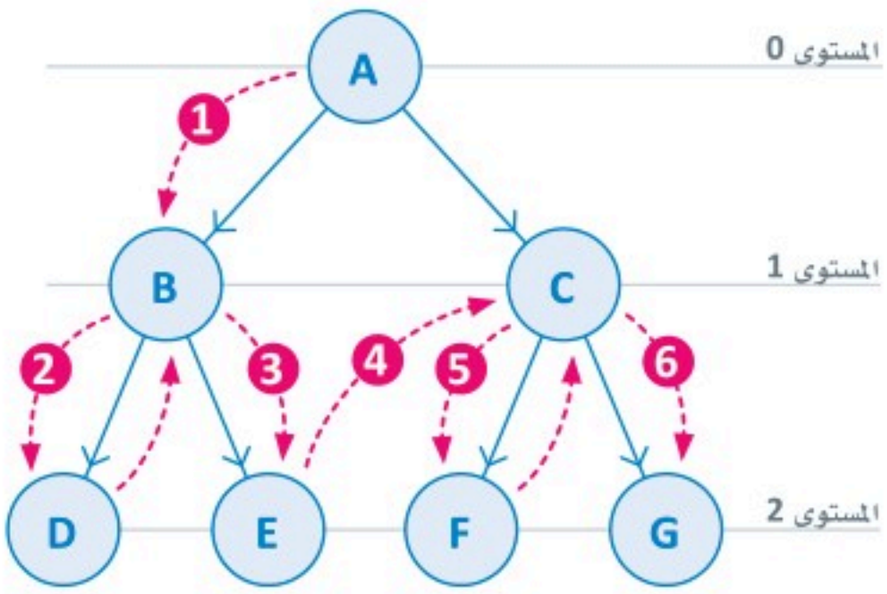
تُستخدَم للحصول على **البث الشبكي** (Network Broadcasting) لبعض الحُزم.

معلومة

يُمكن تطوير خوارزمية البحث بأولوية الاتساع (BFS) بتحديد نقطة البداية (الحالة الأوليّة) ونقطة الهدف (الحالة المُستهدَفة) لإيجاد المسار بينهما.

خوارزمية البحث بأولوية العمق
Depth-First Search (DFS) Algorithm

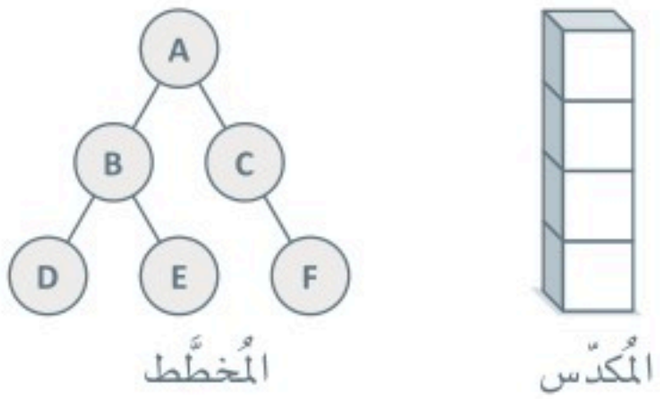
في **البحث بأولوية العمق** (DFS)، ستقوم باتباع الحواف، وتتعمق أكثر وأكثر في المُخطَّط. يَستخدِم البحث بأولوية العمق إجراء استدعاء تكراري للتنقل عبر العُقد. عند الوصول إلى عُقدة لا تحتوي على حواف لأي عُقدة جديدة، ستعود إلى العُقدة السابقة وتستمر العملية. تَستخدِم خوارزمية البحث بأولوية العمق هيكل بيانات المُكدّس لتتبع مسار الاستكشاف. بمجرد استكشاف عُقدة، ستُضاف إلى المُكدّس. عندما ترغب في العودة، ستحذف العُقدة من المُكدّس كما هو موضَّح في الشكل 2.7.



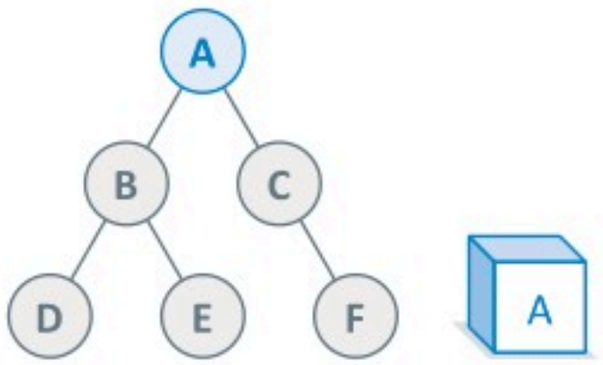
شكل 2.7: خوارزمية البحث بأولوية العمق (DFS)

المثال التالي يوضِّح طريقة عمل خوارزمية **البحث بأولوية العمق** (DFS)، باستخدام المُخطَّط التالي، تَتَبّع ترتيب استكشاف العُقد (Traversal) بحسب خوارزمية البحث بأولوية العمق.

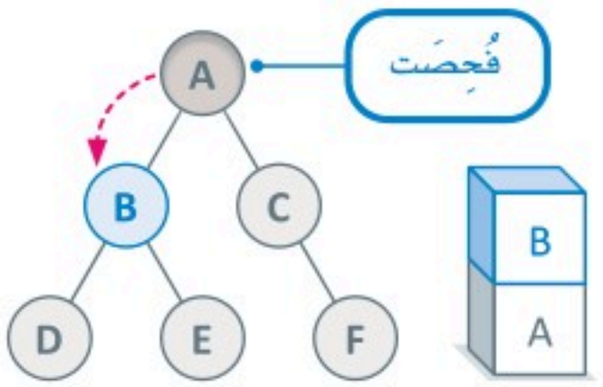
ملاحظة: استخدِم هيكل البيانات المُناسب.



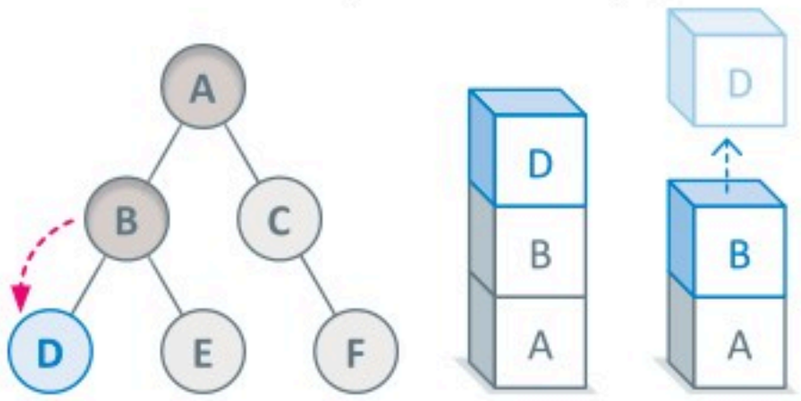
❶ عالِج الجِذر A ثم أضفه إلى المُكدّس.



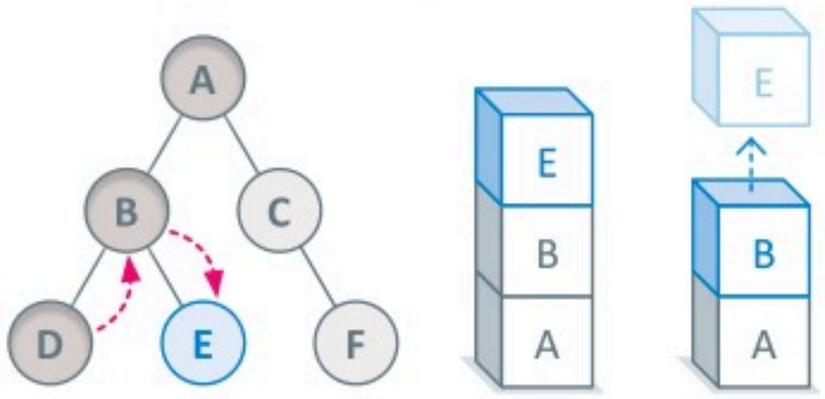
❷ عالِج العُقدة B ثم أضفها إلى المُكدّس.



❸ عالِج العُقدة D ثم أضفها إلى المُكدّس. ستُحذَف العُقدة التي فُحِصَت وليس لها فروع من المُكدّس. (احذف العُقدة D).



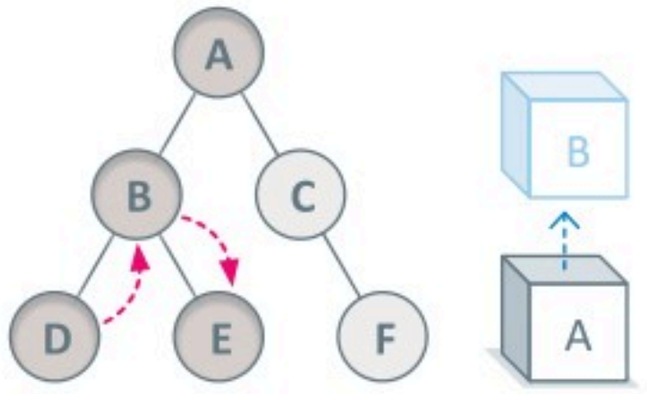
❹ عالِج العُقدة E ثم أضفها إلى المُكدّس. ستُحذَف العُقدة التي فُحِصَت وليس لها فروع من المُكدّس. (احذف العُقدة E).



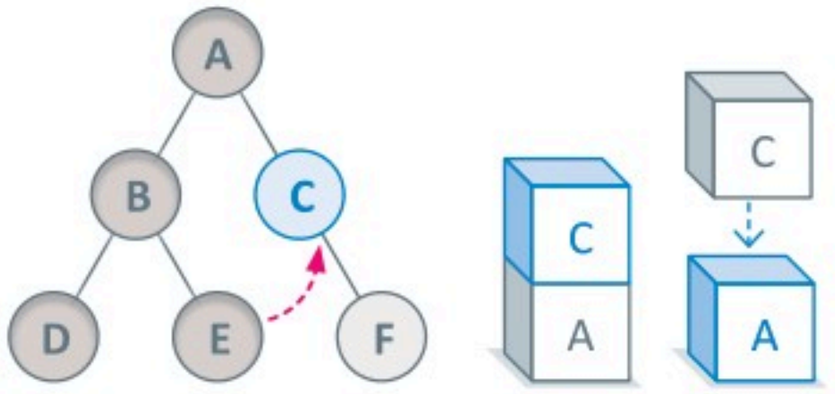
لمحة تاريخية

طُورَت النسخة الأولى من خوارزمية البحث بأولوية العمق (DFS) في القرن التاسع عشر بواسطة عالم رياضيات فرنسي كاستراتيجية لحل المتاهات.

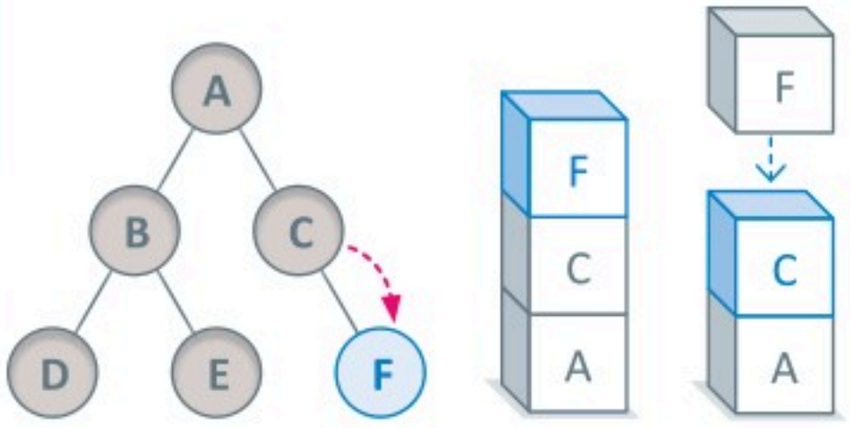
❺ احذف العُقدة B.



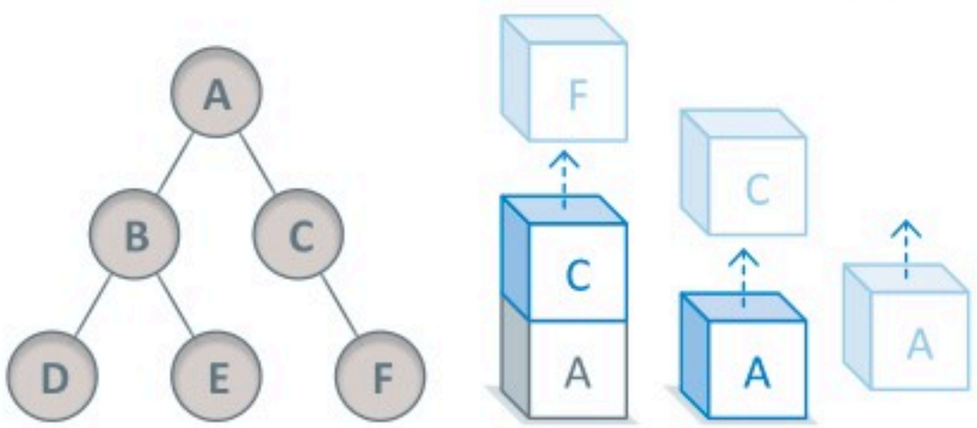
❻ عالِج العُقدة C ثم أضفها إلى المُكدّس.



❼ عالِج العُقدة F ثم أضفها إلى المُكدّس.



❽ المُكدّس خالي وبالتالي ستتوقف خوارزمية **البحث بأولوية العمق** (DFS).



والآن ستتعلّم طريقة تنفيذ خوارزمية **البحث بأولوية العمق** (DFS) في لغة البايثون.

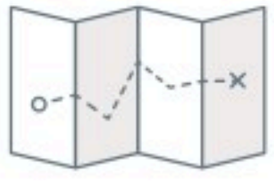
العُقد التي فُحِصَت باستخدام خوارزمية البحث بأولوية العمق (DFS) هي: A، B، D، E، C، F.

```
graph = {
    "A" : ["B","C"],
    "B" : ["D","E"],
    "C" : ["F"],
    "D" : [],
    "E" : [],
    "F" : []
}

visitedDFS = [] # list to keep track of visited nodes

# dfs function
def dfs(visited, graph, node):
    if node not in visited:
        print(node, end = " ")
        visited.append(node)
        for neighbor in graph[node]:
            dfs(visited, graph, neighbor)


# main program
dfs(visitedDFS, graph, "A")
```

يُستخدم المُكدّس بصورة غير مباشرة عبر مُكدّس أثناء التشغيل (Runtime Stack) لتتبُّع الاستدعاءات التكرارية.

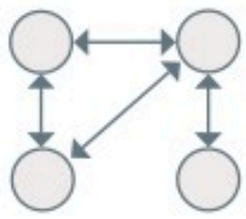
```
A B D E C F
```

التطبيقات العملية لخوارزمية البحث بأولوية العمق
Practical Applications of the DFS Algorithm



تُستخدَم خوارزمية البحث بأولوية العمق في **إيجاد المسارات** (Path Finding) لاستكشاف المسارات المختلفة في العمق للخرائط والطرقات والبحث عن المسار الأفضل.

تُستخدَم خوارزمية البحث بأولوية العمق في **حل المتاهات** (Solve Mazes) من خلال اجتياز كل الطُرُق الممكنة.



يُمكِن تحديد **الدورات** (Cycles) في المُخطَّط باستخدام خوارزمية البحث بأولوية العمق من خلال وجود **حافة خلفية** (Back Edge)، تمُر من خلال العُقدة نفسها مرتين.

جدول 2.4: مقارنة بين خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS)

معايير المقارنة	خوارزمية البحث بأولوية الاتساع (BFS)	خوارزمية البحث بأولوية العمق (DFS)
طريقة التنفيذ	التنقّل حسب مستوى الشجرة.	التنقّل حسب عُمق الشجرة.
هيكل البيانات	تَستخدِم هيكل بيانات الطابور لتتبُّع الموقع التالي لفحصه.	تَستخدِم هيكل بيانات المُكدّس لتتبُّع الموقع التالي لفحصه.
الاستخدام	يُفضَّل استخدامها عندما يكون هيكل المُخطَّط واسعًا وقصيرًا.	يُفضَّل استخدامها عندما يكون هيكل المُخطَّط ضيقًا وطويلًا.
طريقة البحث	تبحث عن مسار الوجهة باستخدام أقل عدد من الحواف.	يتجة البحث إلى قاع الشجرة الفرعية، ثم يتراجع.
العُقد التي تُفحص في البداية	فحص عُقد الأشقاء قبل الفروع.	فحص عُقد الفروع قبل الأشقاء.

تمرينات

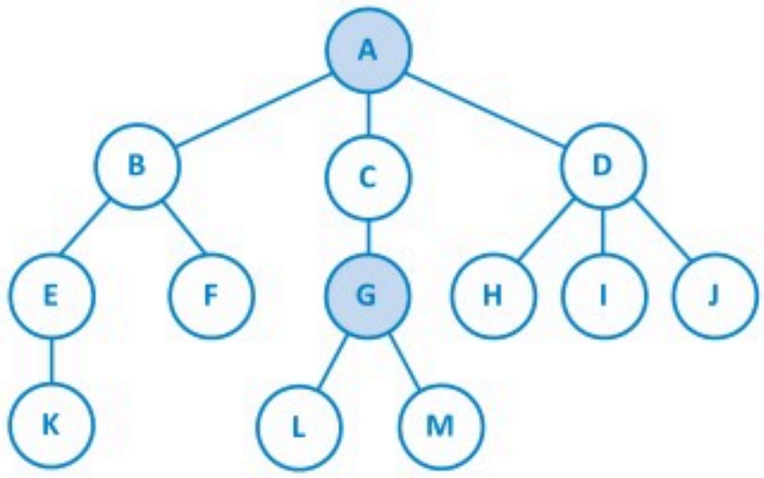
1

حدّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. تُنفَّذ خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS) باستخدام الاستدعاء الذاتي.	○	○
2. لا يمكن استخدام خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS) في هيكل بيانات الشجرة.	○	○
3. تُنفَّذ خوارزمية البحث بأولوية الاتساع (BFS) بمساعدة هيكل بيانات القائمة المترابطة.	○	○
4. يمكن تنفيذ خوارزمية البحث بأولوية العمق (DFS) بمساعدة هيكل بيانات المُكدّس.	○	○
5. لا يمكن استخدام خوارزمية البحث بأولوية الاتساع (BFS) في البث الشبكي.	○	○

2 اشرح كيف تعمل خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS).

3 قارن بين خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS).

4 في المُخطَّط على اليسار، انتقِل من عُقدة البداية A إلى عُقدة الهدف G. طبِّق خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية العمق (DFS) باستخدام هيكل البيانات المناسب (المُكدّس أو الطابور)، مع الإشارة إلى العُقد التي فُحِصَت.



5 اكتب دالة بلغة البايثون تَستخدِم خوارزمية البحث بأولوية الاتساع (BFS) في مُخطَّط للتحقق مما إذا كان هناك مسارٌ بين عُقدتين مُعطاتين.

6 اكتب دالة بلغة البايثون تَستخدِم خوارزمية البحث بأولوية العمق (DFS) لإيجاد المسار الأقصر في مُخطَّط غير موزون.

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الثالث

اتخاذ القرار القائم على القواعد

الأنظمة القائمة على القواعد Rule-Based Systems

تُركِّز أنظمة الذكاء الاصطناعي القائمة على القواعد على استخدام مجموعة من القواعد المُحدَّدة مُسبقًا لاتخاذ القرارات وحل المشكلات. **الأنظمة الخبيرة** (Expert Systems) هي المثال الأكثر شهرة للذكاء الاصطناعي القائم على القواعد، وهي إحدى صور الذكاء الاصطناعي الأولى التي طُوِّرت وانتشرت في فترة الثمانينيات والتسعينيات من القرن الماضي. وغالبًا ما كانت تُستخدَم لأتمتة المهام التي تتطلب عادةً خبرات بشرية مثل: تشخيص الحالات الطبية أو تحديد المشكلات التقنية وإصلاحها. واليوم لم تَعُد الأنظمة القائمة على القواعد التقنية هي الأحدث، حيث تفوّقت عليها منهجيات الذكاء الاصطناعي الحديثة. ومع ذلك، لا تزال الأنظمة الخبيرة شائعة الاستخدام في العديد من المجالات نظرًا لقدرتها على الجمع بين الأداء المعقول وعملية اتخاذ القرار البديهية والقابلة للتفسير.

قاعدة المعرِفة Knowledge Base

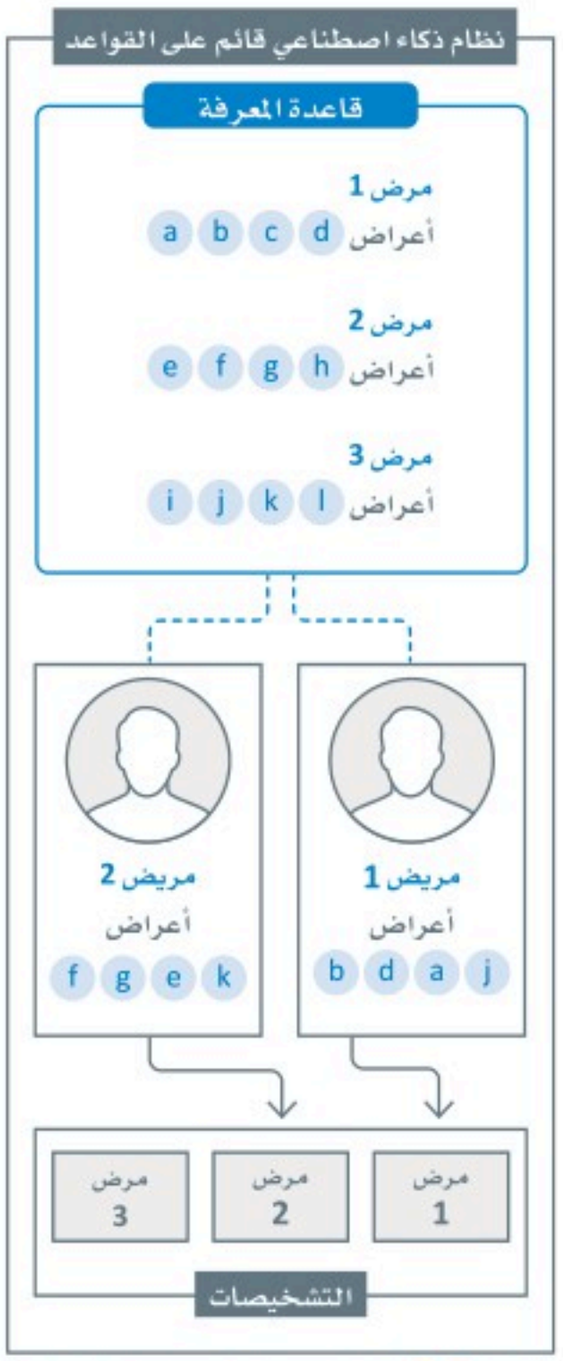
أحد المكونات الرئيسة لأنظمة الذكاء الاصطناعي القائمة على القواعد هي قاعدة المعرِفة، وهي مجموعة من الحقائق والقواعد التي يَستخدِمها النظام لاتخاذ القرارات. تُدخَل هذه الحقائق والقواعد في النظام بواسطة الخبراء البشريين المسؤولين عن تحديد المعلومات الأكثر أهمية وتحديد القواعد التي يتَّبعها النظام. لاتخاذ القرار أو حل المُشكلة، يبدأ النظام الخبير بالتحقق من الحقائق والقواعد في قاعدة البيانات وتطبيقها على الموقف الحالي. إن لم يتمكن النظام من العثور على تطابق بين الحقائق والقواعد في قاعدة المعرِفة، فقد يطلب من المُستخدِم معلومات إضافية أو إحالة المشكلة إلى خبير بشري لمزيد من المساعدة، وإليكَ بعض مزايا وعيوب الأنظمة القائمة على القواعد موضحة في الجدول 2.5:

الأنظمة الخبيرة (Expert Systems):
النظام الخبير هو أحد أنواع الذكاء الاصطناعي الذي يُحاكي قدرة اتخاذ القرار لدى الخبير البشري. يَستخدِم النظام قاعدة المعرِفة المُكوَّنة من قواعد وحقائق ومحركات الاستدلال لتقديم المشورة أو حل المشكلات في مجال معرفي مُحدَّد.

جدول 2.5: المزايا والعيوب الرئيسة للأنظمة القائمة على القواعد

المزايا	العيوب
• يُمكنها اتخاذ القرارات وحل المشكلات بسرعة وبدقة أفضل من البشر، خاصةً عندما يتعلق الأمر بالمهام التي تتطلب قدرًا كبيرًا من المعرِفة أو البيانات. • تَعمل هذه الأنظمة باستمرار، دون تحيُّز أو أخطاء قد تؤثر في بعض الأحيان على اتخاذ القرار البشري.	• تَعمل هذه الأنظمة بكفاءة طالما كانت مُدخَلات المعرِفة والقواعد جيدة، وقد لا تستطيع التعامل مع المواقف التي تقع خارج نطاق خبراتها. • لا يُمكنها التعلُّم أو التكيُّف بالطريقة نفسها مثل البشر، وهذا يجعلها أقل قابلية للتطبيق على الأحداث المُتغيِّرة حيث تتغير مُدخَلات البيانات والمنطق كثيرًا بمرور الوقت.

في هذا الدرس ستتعلّم المزيد حول الأنظمة القائمة على القواعد في سياق أحد تطبيقاتها الرئيسة، وهو: التشخيص الطبي. سيعرض النظام تشخيصًا طبيًا وفقًا للأعراض التي تظهر على المريض، كما هو مُوضّح في الشكل 2.8. بدءًا بنظام تشخيص بسيط مُستنِد إلى القواعد، وستكتشف بعض الأنظمة الأكثر ذكاءً وكيف يُحقّق كل تكرار نتائج أفضل.



شكل 2.8: التشخيص الطبي بواسطة نظام الذكاء الاصطناعي القائم على القواعد

الإصدار 1

في الإصدار الأول ستبني نظامًا بسيطًا قائمًا على القواعد يمكنه تشخيص ثلاثة أمراض مُحتملة: KidneyStones (حصى الكُلى)، وAppendicitis (التهاب الزائدة الدودية)، وFood Poisoning (التسمُّم الغذائي). ستكون المُدخَلات إلى النظام هي قاعدة معرفة بسيطة تربط كل مرض بقائمة من الأعراض المُحتملة. يتوفّر ذلك في ملف بتنسيق JSON (جيسون) يُمكنك تحميله وعرضه كما هو مُوضَّح بالأسفل.

```python
import json # a library used to save and load JSON files

# the file with the symptom mapping
symptom_mapping_file='symptom_mapping_v1.json'

# open the mapping JSON file and load it into a dictionary
with open(symptom_mapping_file) as f:
    mapping=json.load(f)

# print the JSON file
print(json.dumps(mapping, indent=2))
```

```
{
  "diseases": {
    "food poisoning": [
      "vomiting",
      "abdominal pain",
      "diarrhea",
      "fever"
    ],
    "kidney stones": [
      "lower back pain",
      "vomiting",
      "fever"
    ],
    "appendicitis": [
      "abdominal pain",
      "vomiting",
      "fever"
    ]
  }
}
```

سيتَّبع الإصدار الأول القائم على القواعد قاعدة بسيطة ألا وهي: إذا كان لدى المريض على الأقل ثلاثًا من جميع الأعراض المحتملة للمرض، فيجب إضافة المرض كتشخيص مُحتَمل. يمكنك العثور أدناه على دالة Python (البايثون) التي تَستخدِم هذه القاعدة لإجراء التشخيص، بالاستناد إلى قاعدة المعرِفة المذكورة أعلاه وأعراض المرض الظاهرة على المريض.

```
def diagnose_v1(patient_symptoms:list):

    diagnosis=[] # the list of possible diseases

    if "vomiting" in patient_symptoms:

        if "abdominal pain" in patient_symptoms:

            if "diarrhea" in patient_symptoms:

                    # 1:vomiting, 2:abdominal pain, 3:diarrhea
                    diagnosis.append('food poisoning')

            elif 'fever' in patient_symptoms:

                    # 1:vomiting, 2:abdominal pain, 3:fever
                    diagnosis.append('food poisoning')
                    diagnosis.append('appendicitis')

    elif "lower back pain" in patient_symptoms and 'fever' in patient_symptoms:

        # 1:vomiting, 2:lower back pain, 3:fever
        diagnosis.append('kidney stones')

elif "abdominal pain" in  patient_symptoms and\
     "diarrhea" in patient_symptoms and\
      "fever" in patient_symptoms:\
    # 1:abdominal pain, 2:diarrhea, 3:fever
    diagnosis.append('food poisoning')

    return diagnosis
```

في هذه الحالة، تكون قاعدة المعرِفة محددةً **بتعليمات برمجية ثابتة** (Hard-Coded) داخل الدالة في شكل عبارات IF. تَستخدِم هذه العبارات الأعراض الشائعة بينَ الأمراضَ الثلاثة للتوصل تدريجيًا إلى التشخيص في أسرع وقت ممكن. على سبيل المثال، عَرض Vomiting (القيء) مشترك بين جميع الأمراض. لذلك، إذا كانت عبارة IF الأولى صحيحة فقد تم بالفعل حساب أحد الأعراض الثلاثة المطلوبة لجميع الأمراض. بعد ذلك، ستبدأ في البحث عن Abdominal Pain (ألم البطن) المرتبط بمرضين وتستمر بالطريقة نفسها حتى يتم النظر في جميع مجموعات الأعراض الممكنة.

يُمكنك بعد ذلك اختبار هذه الدالة على ثلاثة مرضى مختلفين:

```
# Patient 1
my_symptoms=['abdominal pain', 'fever', 'vomiting']
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)

# Patient 2
my_symptoms=['vomiting', 'lower back pain', 'fever' ]
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)

# Patient 3
my_symptoms=['fever', 'cough', 'vomiting']
diagnosis=diagnose_v1(my_symptoms)
print('Most likely diagnosis:',diagnosis)
```

```
Most likely diagnosis: ['food poisoning', 'appendicitis']
Most likely diagnosis: ['kidney stones']
Most likely diagnosis: []
```

شكل 2.9: تمثيل الإصدار الأول

يتضمن التشخيص الطبي للمريض الأول التسمُم الغذائي والتهاب الزائدة الدودية؛ لأن الأعراض الثلاثة التي تظهر على المريض ترتبط بكلا المرضين. يُشخَّص المريض الثاني بحصى الكُلى، فهو المرض الوحيد الذي تجتمع فيه الأعراض الثلاثة. في النهاية، لا يُمكن تشخيص الحالة الطبية للمريض الثالث؛ لأن الأعراض الثلاثة التي ظهرت على المريض لا تجتمع في أيٍ من الأمراض الثلاثة.

يتميز الإصدار الأول القائم على القواعد بالبديهية والقابلية للتفسير، كما يتضمن استخدام قاعدة المعرِفة والقواعد في التشخيص الطبي دون تَحيُّز أو انحراف عن الخط المعياري. ومع ذلك، يشوب هذا الإصدار العديد من العيوب: أولًا، أن قاعدة ثلاثة أعراض على الأقل هي تمثيل مُبسَّط للغاية لكيفية التشخيص الطبي على يد الخبير البشري. ثانيًا، أن قاعدة المعرِفة داخل الدالة تكون محددةً بتعليماتٍ برمجيةٍ ثابتة، وعلى الرغم من أنه يسهُل إنشاء عبارات شَرطيَّة بسيطة لقواعد المعرِفة الصغيرة، إلا أن المهمة تصبح أكثر تعقيدًا وتستغرق وقتًا طويلًا عند تشخيص الحالات التي تعاني من العديد من الأمراض والأعراض المرضية.

الإصدار 2

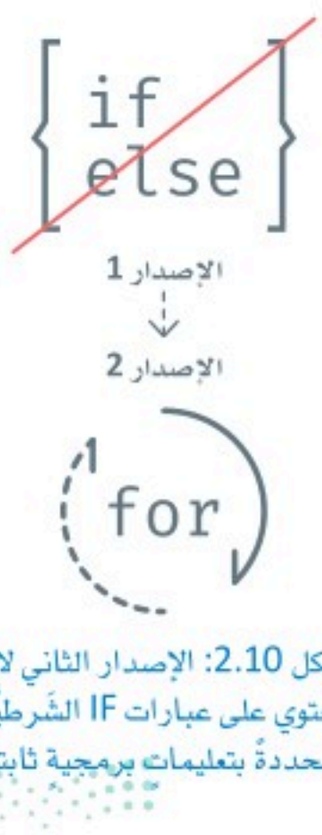
في الإصدار الثاني، ستُعزِّز مرونة وقابلية تطبيق النظام القائم على القواعد بتمكينه من قراءة قاعدة المعرِفة المُتغيِّرة مباشرةً من ملف JSON (جسون). سيؤدي هذا إلى الحد من عملية الهندسة اليدوية لعبارات IF الشَرطيَّة حسب الأعراض ضمن الدالة. وهذا يُعدُّ تحسُّنًا كبيرًا يجعل النظام قابلًا للتطبيق على قواعد المعرِفة الأكبر حجمًا مع تزايد عدد الأمراض والأعراض. وفي الأسفل، مثال يوضِّح قاعدة المعرِفة.

```
symptom_mapping_file='symptom_mapping_v2.json'

with open(symptom_mapping_file) as f:
    mapping=json.load(f)

print(json.dumps(mapping, indent=2))
```

```
{
  "diseases": {
    "covid19": [
      "fever",
      "headache",
      "tiredness",
      "sore throat",
      "cough"
    ],
    "common cold": [
      "stuffy nose",
      "runny nose",
      "sneezing",
      "sore throat",
      "cough"
    ],
   "flu": [
      "fever",
      "headache",
      "tiredness",
      "stuffy nose",
      "sneezing",
      "sore throat",
      "cough",
      "runny nose"
    ],
    "allergies": [
      "headache",
      "tiredness",
      "stuffy nose",
      "sneezing",
      "cough",
      "runny nose"
    ]
  }
}
```

قاعدة المعرِفة الجديدة هذه أكبر قليلًا من سابقتها. ومع ذلك، يتَّضح أن محاولة إنشاء عبارات IF الشَرطيَّة في هذه الحالة ستكون أصعب بكثير. على سبيل المثال، تضمنت قاعدة المعرِفة السابقة ربط أحد الأمراض بأربعة أعراض، ومرضين بثلاثة أعراض. وعند تطبيق قاعدة ثلاثة أعراض على الأقل المُطبَّقة في الإصدار الأول، تحصل على 6 مجموعات ثلاثية من الأعراض المحتملة التي تؤخَذ في الاعتبار. في قاعدة المعرِفة الجديدة بالأعلى، تكون للأمراض الأربعة 5 و5 و8 و6 أعراض، على التوالي. وبهذا، تحصل على 96 مجموعة ثلاثية من الأعراض المحتملة. وفي حال التعامل مع مئات أو حتى آلاف الأمراض، ستجدُ أنّه من المستحيل إنشاء نظام مثل الموجود في الإصدار الأول.

وكذلك، لا يوجد سبب طبي وجيه لقِصَر التشخيص الطبي على مجموعات ثلاثية من الأعراض. ولذلك، ستجعل **منطق التشخيص** (Diagnosis Logic) أكثر تنوعًا بحساب عدد الأعراض المُطابقة لكل مرض، والسماح للمُستخدِم بتحديد عدد الأعراض المُطابقة التي يجب توافرها في المرض لتضمينه في التشخيص.



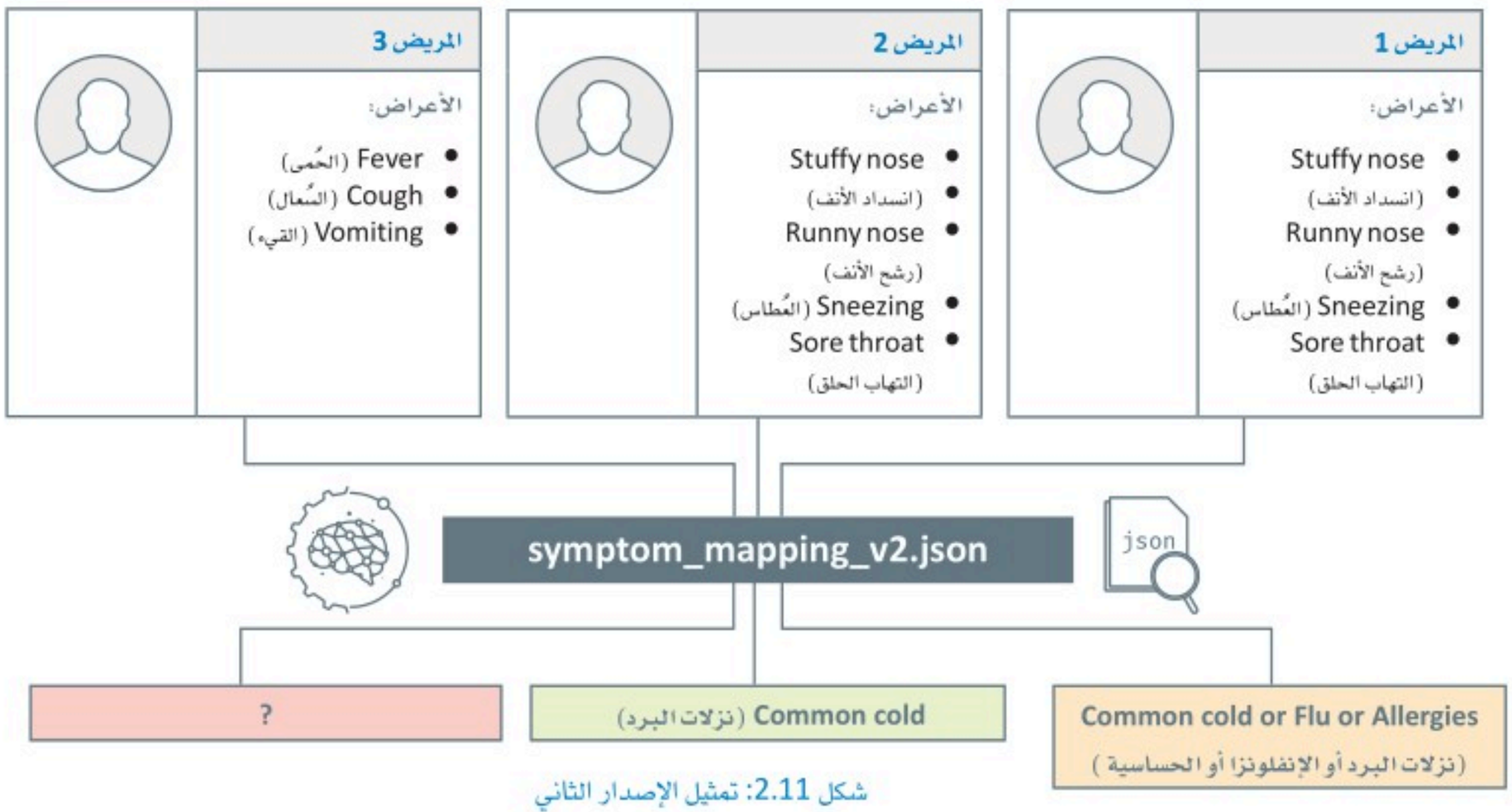
شكل 2.10: الإصدار الثاني لا يحتوي على عبارات IF الشَرطيَّة المحددةً بتعليمات برمجية ثابتة

```python
def diagnose_v2(patient_symptoms:list,
                symptom_mapping_file:str,
                matching_symptoms_lower_bound:int):

    diagnosis=[]

    with open(symptom_mapping_file) as f:
        mapping=json.load(f)

    # access the disease information
    disease_info=mapping['diseases']

    # for every disease
    for disease in disease_info:

        counter=0

        disease_symptoms=disease_info[disease]

        # for each patient symptom
        for symptom in patient_symptoms:

            # if this symptom is included in the known symptoms for the disease
            if symptom in disease_symptoms:

                counter+=1

        if counter>=matching_symptoms_lower_bound:
            diagnosis.append(disease)

    return diagnosis
```

لا يحتوي هذا الإصدار على عبارات IF الشَرطيَّة المحددة بتعليماتٍ برمجيةٍ ثابتة. بعد تحميل مُخطَّط الأعراض من ملف JSON (جسون)، يبدأ الإصدار في أخذ كلّ مرض محتمل في الاعتبار باستخدام حلقة التكرار الأولى FOR. تتحقق الحلقة من كل عَرَض على حدة بمقارنته بالأعراض المعروفة للمرض وزيادة **العدَّاد** (Counter) في كل مرة يجد فيها النظام تطابقًا.

```python
# Patient 1
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v2(my_symptoms,'symptom_mapping_v2.json' , 3)
print('Most likely diagnosis:',diagnosis)

# Patient 2
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json' , 4)
print('Most likely diagnosis:',diagnosis)

# Patient 3
my_symptoms=['fever', 'cough', 'vomiting']
diagnosis=diagnose_v2(my_symptoms, 'symptom_mapping_v2.json' , 3)
print('Most likely diagnosis:',diagnosis)
```

```
Most likely diagnosis: ['common cold', 'flu', 'allergies']
Most likely diagnosis: ['common cold']
Most likely diagnosis: []
```



شكل 2.11: تمثيل الإصدار الثاني

لاحظ أن الإصدار الثاني هو نسخة مُعمَّمة من الإصدار الأول. ومع ذلك، يُعدُّ هذا الإصدار أكثر قابلية للتطبيق على نطاق واسع، ويمكن استخدامه كما هو مع أي قاعدة معرِفة أخرى بالتنسيق نفسه، حتى لو كانت تشمل الآلاف من الأمراض مع عدد ضخم من الأعراض. كما يَسمح للمُستخدِم بزيادة أو تقليل عدد القيود على التشخيص بضبط المُتغيِّر matching_symptoms_lower_bound. يمكن ملاحظة ذلك في حالة المريض 1 والمريض 2: فعلى الرغم من أنهما يعانيان من الأعراض نفسها، إلا أنه عند ضبط هذا المُتغيِّر، ستحصل على تشخيص مختلف تمامًا.

على الرغم من هذه التحسينات،إلا إنّ بعض العيوب لا تزال موجودة في هذا الإصدار، ولا يُعدُّ تمثيلًا دقيقًا للتشخيص الطبي الحقيقي.

الإصدار 3

في الإصدار الثالث، ستزيد من ذكاء النظام القائم على القواعد بمنحه إمكانية الوصول إلى نوع مُفصَّل من قاعدة المعرِفة. هذا النوع الجديد يأخذ بعين الاعتبار الحقيقة الطبية التي تقول: إنّ بعض الأعراض تكون أكثر شيوعًا من أخرى للمرض نفسه.

```
symptom_mapping_file='symptom_mapping_v3.json'

with open(symptom_mapping_file) as f:
    mapping=json.load(f)

print(json.dumps(mapping, indent=2))
```

```
{
  "diseases": {
    "covid19": {
      "very common": [
        "fever",
        "tiredness",
        "cough"
      ],
      "less common": [
        "headache",
        "sore throat"
      ]
    },
    "common cold": {
      "very common": [
        "stuffy nose",
        "runny nose",
        "sneezing",
        "sore throat"
      ],
      "less common": [
        "cough"
      ]
    },
    "flu": {
      "very common": [
        "fever",
        "headache",
        "tiredness",
        "sore throat",
        "cough"
      ],
      "less common": [
        "stuffy nose",
        "sneezing",
        "runny nose"
      ]
    },
    "allergies": {
      "very common": [
        "stuffy nose",
        "sneezing",
        "runny nose"
      ],
      "less common": [
        "headache",
        "tiredness",
        "cough"
      ]
    }
  }
}
```

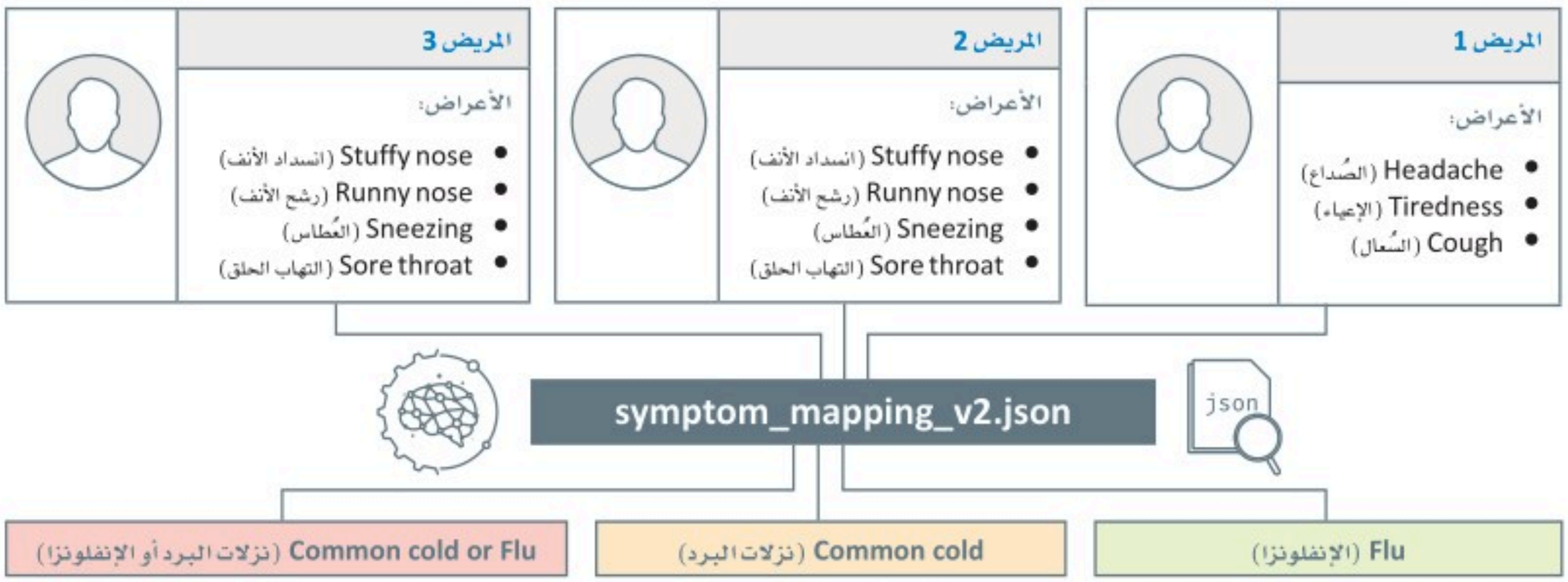
لن يُنظر إلى المنطق الذي يقتصر على عدد الأعراض، وسيُستبدَل بدالة تسجيل النقاط التي تعطي أوزانًا مُخصَّصة للأعراض الأكثر والأقل شيوعًا. ستَتوفر للمُستخدِم كذلك المرونة لتحديد الأوزان التي يراها مناسبة. سيتمّ تضمين المرض أو الأمراض ذات المجموع الموزون الأعلى في التشخيص.

```
from collections import defaultdict

def diagnose_v3(patient_symptoms:list,
                symptom_mapping_file:str,
                very_common_weight:float=1,
                less_common_weight:float=0.5
               ):

    with open(symptom_mapping_file) as f:
        mapping=json.load(f)

    disease_info=mapping['diseases']

    # holds a symptom-based score for  each potential disease
    disease_scores=defaultdict(int)

    for disease in disease_info:

        # get the very common symptoms of the disease
        very_common_symptoms=disease_info[disease]['very common']

        # get the less common symptoms for this disease
        less_common_symptoms=disease_info[disease]['less common']

        for symptom in patient_symptoms:

            if symptom in very_common_symptoms:
                disease_scores[disease]+=very_common_weight

            elif symptom in less_common_symptoms:
                disease_scores[disease]+=less_common_weight

        # find the max score all candidate diseases
        max_score=max(disease_scores.values())

        if max_score==0:
            return []

        else:
            # get all diseases that have the max score
            diagnosis=[disease for disease in disease_scores if disease_scores
[disease]==max_score]

            return diagnosis, max_score
```

لكل مرض محتمل في قاعدة المعرِفة، تُحدِّد هذه الدالة الجديدة الأعراض الأكثر والأقل ظهورًا على المريض، ثم تزيد من درجة المرض وفقًا للأوزان المُقابِلة، وفي الأخير تُدرَج الأمراض ذات الدرجة الأعلى في التشخيص. يُمكنك الآن اختبار تنفيذ الدالة مع بعض الأمثلة:

```
# Patient 1
my_symptoms=["headache", "tiredness", "cough"]
diagnosis=diagnose_v3(my_symptoms, 'symptom_mapping_v3.json')
print('Most likely diagnosis:',diagnosis)

# Patient 2
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v3(my_symptoms, 'symptom_mapping_v3.json')
print('Most likely diagnosis:',diagnosis)

# Patient 3
my_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]
diagnosis=diagnose_v3(my_symptoms, 'symptom_mapping_v3.json', 1, 1)
print('Most likely diagnosis:',diagnosis)
```

```
Most likely diagnosis: (['flu'], 3)
Most likely diagnosis: (['common cold'], 4)
Most likely diagnosis: (['common cold', 'flu'], 4)
```



شكل 2.12: تمثيل الإصدار الثالث

قد تلاحظ أنه على الرغم من أن الأعراض الثلاثة على المريض 1: Headache (الصداع)، وTiredness (الإعياء)، وCough (السعال) تظهر عند الإصابة بكل من Flu (الإنفلونزا)، وCovid19 (كوفيد- 19). والحساسية، إلّا أنّ الظّاهر في نتائج التّشخيص هي الإنفلونزا فقط. هذا لأن جميع الأعراض الثلاثة شائعة جدًا في قاعدة المعرِفة، مما يؤدي إلى درجة قصوى قدرها 3. وبالمثل، في ظل معاناة المريض الثاني والثالث من الأعراض نفسها، تؤدي مُدخَلات الأوزان المختلفة للأعراض الأكثر والأقل شيوعًا إلى تشخيصات مختلفة. وعلى وجه التحديد، يَنتج عن استخدام وزن متساوٍ لنوعين من الأعراض إضافة الإنفلونزا إلى التشخيص.

الإصدار 4

يمكن تحسين النظام القائم على القواعد بزيادة كفاءة قاعدة المعرِفة وتجربة **دوال تسجيل النقاط** (Scoring Functions) المختلفة. وعلى الرغم من أن ذلك سيؤدي إلى تحسين النظام، إلا أنه سيتطلب الكثير من الوقت والجهد اليدوي. ولحسن الحظ، هناك طريقة آلية لبناء نظام مبني على القواعد يكون ذكيًا بما يكفي لتصميم قاعدة معرِفة ودالة تسجيل نقاط خاصة به: باستخدام تعلُّم الآلة. يُطبِّق **تعلُّم الآلة القائم على القواعد** (Rule-Based Machine Learning) خوارزمية تعلُّم لتحديد القواعد المُفيدة تلقائيًا، بدلًا من الحاجة إلى الإنسان لتطبيق المعرِفة والخبرات السابقة في المجال لبناء القواعد وتنظيمها يدويًا.

فبدلًا من قاعدة المعرِفة ودالة تسجيل النقاط المُصمَّمتان يدويًا، تَتوقَّع خوارزمية تعلُّم الآلة مُدخَلًا واحدًا فقط وهو مجموعة البيانات التاريخيّة للحالات المَرَضيَّة. فالتعلُّم من البيانات مباشرةً يحوُل دون حدوث المشكلات المرتبطة باكتساب المعرِفة الأساسية والتحقق منها. تتكون كل حالة من بيانات أعراض المريض والتشخيص الطبي الذي يمكن أن يقدمه أي خبير بشري مثل الطبيب. وباستخدام مجموعة بيانات التدريب، تتعلّم الخوارزمية تلقائيًا كيف تتنبأ بالتشخيص المُحتمَل لحالة مريض جديد.

```
import pandas as pd # import pandas to load and process spreadsheet-type data

medical_dataset=pd.read_csv('medical_data.csv') # load a medical dataset.

medical_dataset
```

	fever	cough	tiredness	headache	stuffy nose	runny nose	sneezing	sore throat	diagnosis
0	1	1	1	0	0	0	0	0	covid19
1	0	1	1	1	0	0	0	0	covid19
2	1	1	1	0	0	0	0	0	covid19
3	1	1	1	0	0	0	0	0	covid19
4	1	1	1	0	0	0	0	0	covid19
...	...	...	...	...	...	...	...	...	...
1995	0	1	0	0	1	0	1	1	common cold
1996	0	0	0	1	1	1	1	0	common cold
1997	0	0	1	0	1	0	0	1	common cold
1998	0	0	0	0	1	0	0	1	common cold
1999	0	1	0	0	0	0	1	1	common cold

في المثال أعلاه، تحتوي مجموعة البيانات على 2,000 حالة مرضية، بحيث تتكون كل حالة من 8 أعراض محتملة: Fever (الحُمى)، وCough (السُعال)، وTiredness (الإعياء)، وHeadache (الصُداع)، وStuffy nose (انسداد الأنف)، وRunny nose (رشح الأنف)، وSneezing (العُطاس)، وSore throat (التهاب الحلق). تُرمَّز كل واحدة من هذه الأعراض في عمود ثنائي مُنفصل. العدد الثنائي 1 يشير إلى أن المريض يُعاني من الأعراض، بينما العدد الثنائي 0 يشير إلى أن المريض لا يُعاني من الأعراض.

يحتوي العمود الأخير على تشخيص الخبير البشري، وهناك أربعة تشخيصات محتملة:
Covid19 (كوفيد- 19)، وFlu (الإنفلونزا)، وAllergies (الحساسية)، وCommon cold (نزلات البرد).
يمكنك التحقق من ذلك بسهولة باستخدام المقطع البرمجي التالي بلغة البايثون:

```
set(medical_dataset['diagnosis'])
```

على الرغم من أن هناك العشرات من خوارزميات تعلُّم الآلة المحتملة التي يمكن استخدامها مع مجموعة البيانات هذه، إلا أنك ستستخدِم تلك التي تتبع المنهجية المُستنِدة على منطق **شجرة القرار** (Decision Tree)، كما ستَستخدِم DecisionTreeClassifier (مصنّف شجرة القرار) من مكتبة البايثون **سكليرن** (Sklearn) على وجه التحديد.

```
from sklearn.tree import DecisionTreeClassifier
def diagnose_v4(train_dataset:pd.DataFrame):

    # create a DecisionTreeClassifier
    model=DecisionTreeClassifier(random_state=1)

    # drop the diagnosis column to get only the symptoms
    train_patient_symptoms=train_dataset.drop(columns=['diagnosis'])

    # get the diagnosis column, to be used as the classification target
    train_diagnoses=train_dataset['diagnosis']

    # build a decision tree
    model.fit(train_patient_symptoms, train_diagnoses)

    # return the trained model
    return model
```

يُعدُّ تطبيق البايثون في الإصدار الرابع أقصر وأبسط بكثير من التطبيقات السابقة، فهو ببساطة يقرأ الملف التدريبي، ويستخدمه لبناء نموذج شجرة القرار استنادًا إلى العلاقات بين الأعراض والتشخيصات، ومن ثَمَّ ينتج نموذجًا مخصَّصًا. لاختبار هذا الإصدار بشكل صحيح، ابدأ بتقسيم مجموعة البيانات إلى مجموعتين منفصلتين، واحدة للتدريب، وأخرى للاختبار.

```
from sklearn.model_selection import train_test_split

# use the function to split the data, get 30% for testing and 70% for training.
train_data, test_data = train_test_split(medical_dataset, test_size=0.3,
random_state=1)

#print the shapes (rows x columns) of the two datasets
print(train_data.shape)
print(test_data.shape)
```

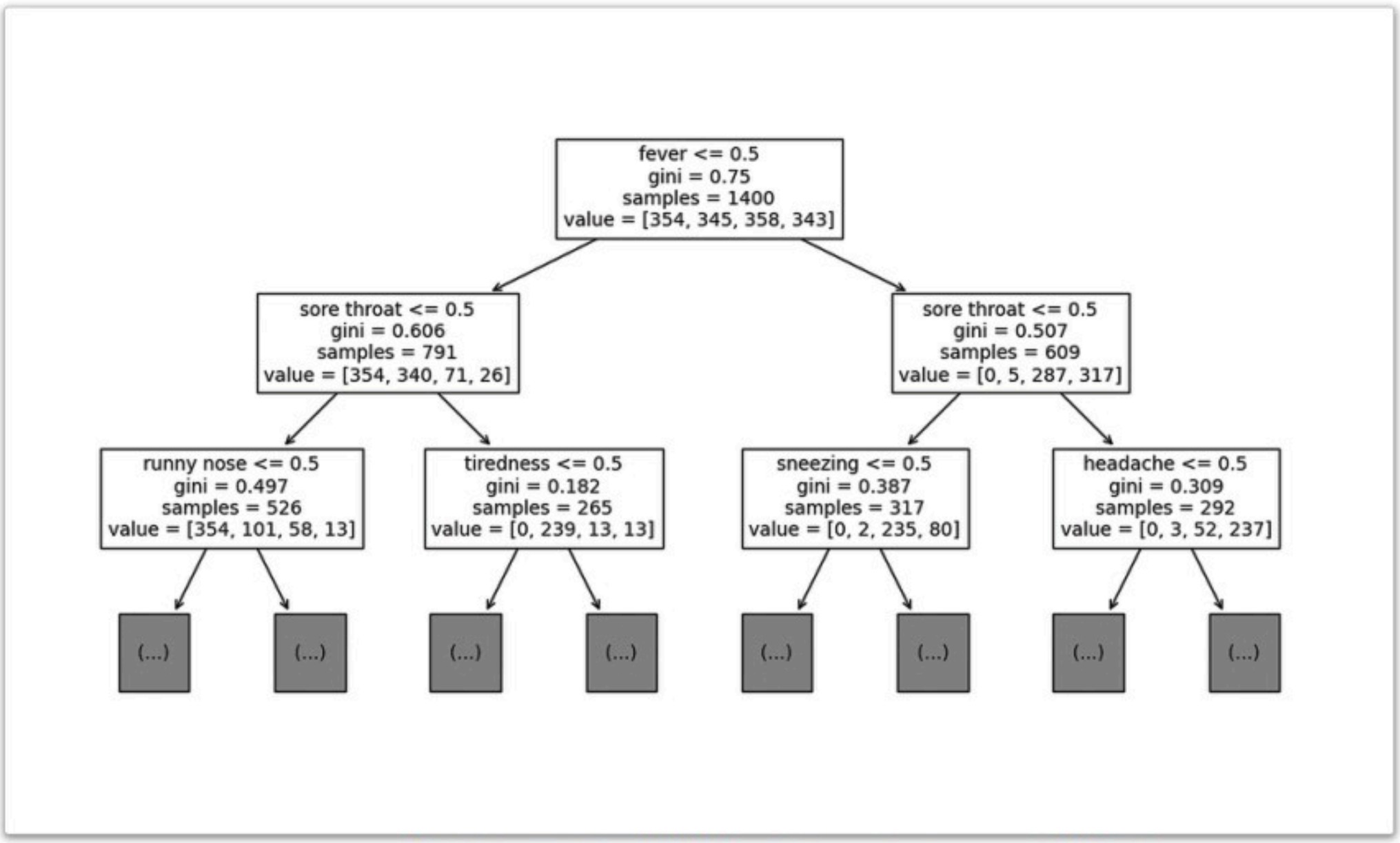
```
(1400, 9)
(600, 9)
```

لديك الآن 1,400 نقطة بيانات ستُستخدَم لتدريب النموذج و600 نقطة ستُستخدَم لاختباره.

ابدأ بتدريب نموذج شجرة القرار وتمثيله:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

my_tree=diagnose_v4(train_data) # train a model

print(my_tree.classes_) # print the possible target labels (diagnoses)

plt.figure(figsize=(12,6)) # size of the visualization, in inches

# plot the tree
plot_tree(my_tree,
          max_depth=2,
          fontsize=10,
          feature_names=medical_dataset.columns[:-1]
         )
```

```
['allergies' 'common cold' 'covid19' 'flu']
```



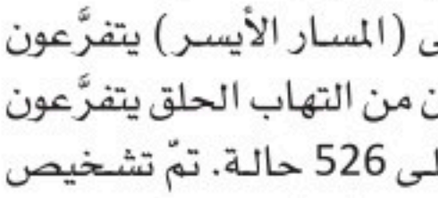
شكل 2.13: نموذج شجرة القرار لمجموعة بيانات medical_data (البيانات- الطبية) بعُمق مستويين

تُستخدَم دالة plot_tree( ) لرسم وعرض شجرة القرار. ولعدم توفر مساحة كافية للعرض سيتم تمثيل المستويين الأوّلين فقط، بالإضافة إلى الجِذر. يمكن ضبط هذا الرقم بسهولة باستخدام المُتغيِّر max_depth.

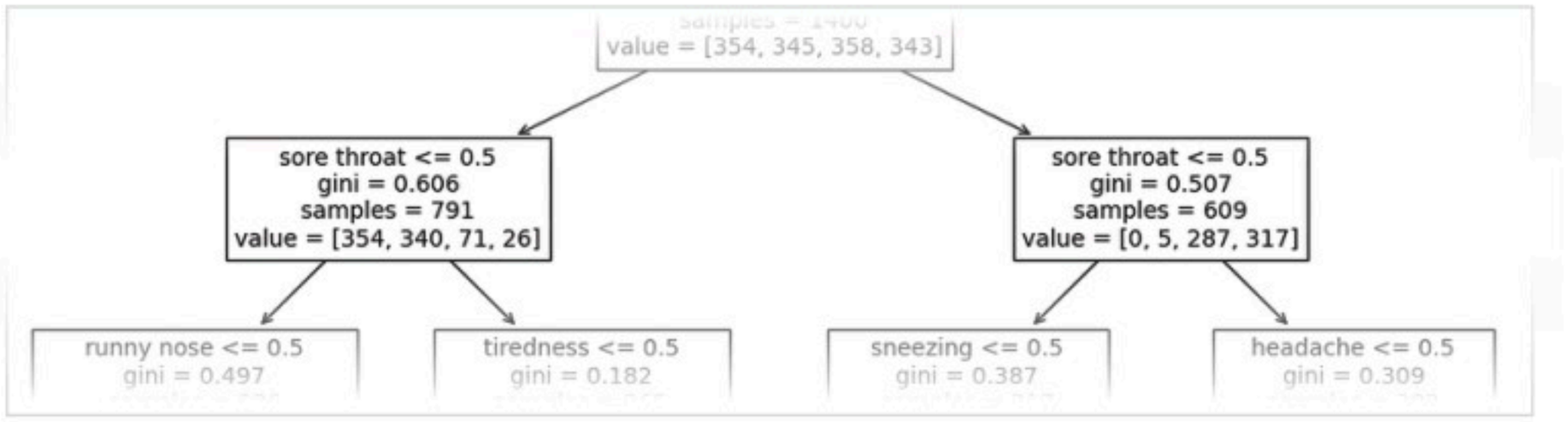
```
# plot the tree
plot_tree(my_tree,
          max_depth=2,
          fontsize=10
```

عُمق شجرة القرار.

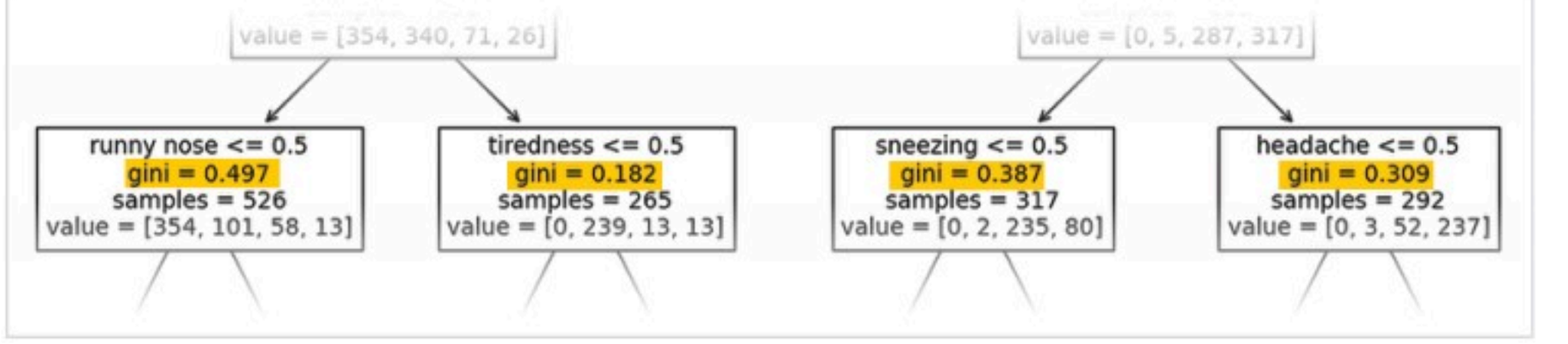
كل عُقدة في الشجرة تُمثِّل مجموعة فرعية من المرضى، فعلى سبيل المثال،



تُمثِّل عُقدة الجِذر إجمالي عدد 1,400 مريض في مجموعة بيانات التدريب. من بينهم، 354، و345، و358، و343 شُخِّصوا بـ Allergies (الحساسية)، وCommon cold (نزلات البرد)، وCovid19 (كوفيد-19)، وFlu (الإنفلونزا)، على التوالي.



بُنيَّت الشجرة باستخدام نمط من الأعلى إلى الأسفل عبر **التفرُّع الثنائي** (Binary Splits). يَستنِد التفرُّع الأول إلى ما إذا كان المريض يُعاني من الحُمى أم لا. ونظرًا لأن كل خصائص الأعراض ثنائية، يكون التحقق a <=0.5 صحيحًا إذا لم يكن المريض يعاني من الأعراض. أما المرضى الذين لا يعانون من الحُمى (المسار الأيسر) يتفرَّعون مرة أخرى بناءً على ما إذا كانوا يعانون من التهاب الحلق أم لا. المرضى الذين لا يعانون من التهاب الحلق يتفرَّعون بناءً على ما إذا كانوا يعانون من رشح الأنف أم لا. في هذه المرحلة، تحتوي العُقدة على 526 حالة. تمّ تشخيص 354، و101، و58، و13 من بينهم بالحساسية، ونزلات البرد، وكوفيد-19، والإنفلونزا، على التوالي.



يستمر التفرُّع حتى تُحدِّد الخوارزمية الحالات التي انقسمت بالفعل إلى عُقد نقيَّة تمامًا. العُقدة النقيَّة بالكامل تحتوي على الحالات التي لها التشخيص نفسه. قِيَم مؤشر gini (جيني) المُحدَّدة على كل عُقدة، تُمثِّل مؤشرات على مقياس جيني، وهي صيغة شهيرة تُستخدَم لتقييم درجة نقاء العُقدة.

يقيس مؤشر جيني (Gini Index) الشوائب بالعُقدة، وبالتحديد احتمالية تصنيف محتويات العُقدة بصورة خاطئة. يشير انخفاض مُعامِل جيني إلى ارتفاع درجة تأكُّد الخوارزمية من التصنيف.

ستَستخدِم الآن شجرة القرار للتنبؤ بالتشخيص الأكثر احتمالًا للمرضى في مجموعة الاختبار.

تُستخدَم مجموعة الاختبار لتقييم أداء النموذج. تَستنِد طريقة التقييم الدقيقة على ما إذا كان المقصود من المهمة **الانحدار** (Regression) أم **التصنيف** (Classification). في مثل مشكلات التصنيف المعروضة هنا، تُستخدَم طرائق التقييم الشهيرة مثل: حساب **دقة النموذج** (Model's Accuracy) و**مصفوفة الدقة** (Confusion Matrix).

- الدقة هي نسبة التنبؤات الصحيحة التي يقوم بها المُصنِّف. تَحقُق دقة عالية قريبة من 100% يعني أن معظم التنبؤات التي يقوم بها المُصنِّف صحيحة.
- مصفوفة الدقة هي جدول يقارن بين القيم الحقيقية (الفعلية) وبين التنبؤات التي يقوم بها المُصنِّف في مجموعة البيانات. يحتوي الجدول على صف واحد لكل قيمة صحيحة وعمود واحد لكل قيمة مُتوقَّعة. كل مُدخَل في المصفوفة يُمثِّل عدد الحالات التي لها قيم فعلية ومُتوقَّعة.

```
# functions used to evaluate a classifier
from sklearn.metrics import accuracy_score,confusion_matrix

# drop the diagnosis column to get only the symptoms
test_patient_symptoms=test_data.drop(columns=['diagnosis'])

# get the diagnosis column, to be used as the classification target
test_diagnoses=test_data['diagnosis']

# guess the most likely diagnoses
pred=my_tree.predict(test_patient_symptoms)

# print the achieved accuracy score
accuracy_score(test_diagnoses,pred)
```

```
0.8166666666666667
```

ستلاحِظ أن شجرة القرار تُحقِّق دقة تصل إلى 81.6%، وهذا يعني أنه من بين 600 حالة تمّ اختبارها، شَخَّصت الشجرة 490 منها بشكل صحيح. يُمكنك كذلك طباعة مصفوفة الدقة للنموذج لتستعرض بشكل أفضل الأمثلة المُصنَّفة بشكل خاطئ.

```
confusion_matrix(test_diagnoses,pred)
```

```
array([[143,   3,   0,   0],
       [ 48,  98,   5,   4],
       [  2,   1, 127,  12],
       [  1,   3,  31, 122]])
```

	الحساسية المُتوقَّعة	نزلات البرد المُتوقَّعة	كوفيد19-المُتوقَّع	الإنفلونزا المُتوقَّعة
الحساسية الفعلية	**143**	3	0	0
نزلات البرد الفعلية	48	**98**	5	4
كوفيد-19 الفعلي	2	1	**127**	12
الإنفلونزا الفعلية	1	3	31	**122**

شكل 2.14: مصفوفة الدقة للحالات المُتوقَّعة والحالات الفعلية

الأرقام الواقعة في الخط القُطري (المُظللة باللون الوردي) تُمثِّل الحالات المُتوقَّعة بشكل صحيح، أما الأرقام التي تقع خارج الخط القُطري فتُمثِّل أخطاء النموذج.

على سبيل المثال، بالنظر إلى ترتيب التشخيصات الأربعة المُحتملة [Allergies (الحساسية)، Common cold (نزلات البرد)، Covid19 (كوفيد-19)، Flu (الإنفلونزا)]، توضِّح المصفوفة أن النموذج أخطأ في تصنيف 48 حالة من المُصابين بنزلات البرد بأنهم مصابون بالحساسية، كما أخطأ في تصنيف 31 حالة من المُصابين بالإنفلونزا بأنهم مصابون بكوفيد-19.

وعلى الرغم من أنّ هذا النموذج ليس مثاليًا، فمن المُثير للدهشة أنّه قادر على تحقيق مثل هذه الدرجة العالية من الدقة بتعلُّم مجموعة القواعد الخاصة به، دون الحاجة إلى قاعدة معرِفة أنشئت يدويًا. بالإضافة إلى تحقيق مثل هذه الدقة دون محاولة ضبط مُتغيرات الأداء المتنوعة لـ DecisionTreeClassifier (مُصنِّف شجرة القرار). وبالتالي، يُمكن تحسين دقة النموذج لأفضل من ذلك. كما يُمكن تحسين النموذج بتجاوز قيود النموذج القائم على القواعد وتجربة أنواع مختلفة من خوارزميات تعلُّم الآلة. وستتعلّم بعض هذه الطرائق في الوحدة التالية.

تمرينات

1 اذكر بعض مزايا وعيوب الأنظمة القائمة على القواعد.

2 ما مزايا وعيوب الإصدار الأول؟

3 أضف إلى المقطع البرمجي الخاص بالإصدار الأول لنظام قائم على القواعد مريضًا يُعاني من الأعراض التالية [Vomiting (القيء)، وAbdominal pain (آلام البطن)، وDiarrhea (الإسهال)، وFever (الحُمى)، وLower back pain (ألم بأسفل الظهر)]. ما التشخيص الطبي لحالة المريض؟ دَوِّن ملاحظاتك بالأسفل.

4 في الإصدار الثاني، كم عدد الأمراض المُوضحَّة في تشخيص كل مريض إذا غَيَّرت قيمة المُتغيِّر matching_symptoms_lower_bound إلى 2 و3 و4؟ عَدِّل المقطع البرمجي ثم دَوِّن ملاحظاتك.

5 في الإصدار الثالث، غَيِّر كلا الوزنين إلى 1 للمريضين الأول والثاني، تمامًا مثل المريض الثالث.

عَدِّل المقطع البرمجي ثم دَوِّن ملاحظاتك.

6 صفْ بإيجاز كيف يُمكن تحسين كل إصدار بالنسبة للإصدار السابق له (الأول إلى الثاني، والثاني إلى الثالث، والثالث إلى الرابع).

رابط الدرس الرقمي

www.ien.edu.sa

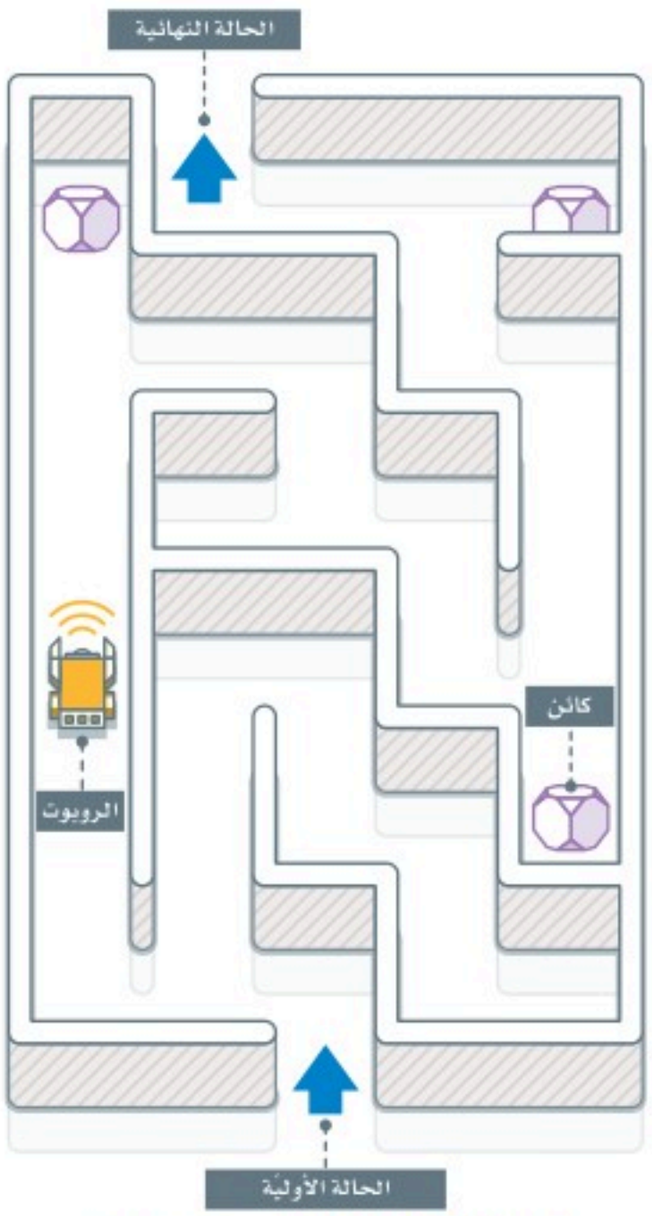
الدرس الرابع

خوارزميات البحث المستنيرة

تطبيقات خوارزميات البحث
Applications of Search Algorithms

خوارزميات البحث هي أحد المكونات الرئيسة لأنظمة الذكاء الاصطناعي، فباستخدامها يُمكن اكتشاف الاحتمالات المختلفة لإيجاد الحلول المناسبة للمشكلات المُعقدة في العديد من التطبيقات السائدة. وفيما يلي أمثلة على بعض تطبيقات خوارزميات البحث:

- **الروبوتية** (Robotics): قد يَستخدِم الروبوت خوارزمية البحث لتحديد طريقه عبر المتاهة أو لتحديد موقع أحد الكائنات في نطاق بيئته.
- **مواقع التجارة الإلكترونية** (E-Commerce Websites): تَستخدِم مواقع التسوق عبر الإنترنت خوارزميات البحث لتُطابق بين استفسارات العملاء وبين المُنتجات المتوفرة، ولتصفية نتائج البحث وفق بعض المعايير مثل: السعر، والعلامة التجارية، والتقييمات، واقتراح المُنتجات ذات الصلة.
- **منصّات مواقع التواصل الاجتماعي** (Social Media Platforms): تَستخدِم مواقع التواصل الاجتماعي خوارزميات البحث لعرض التدوينات، والأشخاص، والمجموعات للمُستخدِمين وفقًا للكلمات المفتاحية واهتمامات المُستخدِم.
- **تمكين الآلة من ممارسة الألعاب بمستوى عالٍ من المهارة** (Enabling a Machine to Play Games at a High Skill Level): يَستخدِم الذكاء الاصطناعي خوارزمية البحث أثناء لعب الشطرنج أو **قو** (Go) لتقييم الحركات المختلفة واختيار الخطوات التي من المرجح أن تؤدي إلى الفوز.
- **نُظم الملاحة باستخدام مُحدّد المواقع العالمي** (GPS Navigation Systems): تَستخدِم نُظم الملاحة القائمة على مُحدِّد المواقع العالمي خوارزميات البحث لتحديد أقصر وأسرع طريق بين موقعين، مع مراعاة بيانات حركة المرور في الوقت الحالي.
- **نُظم إدارة الملفات** (File Management Systems): تُستخدم خوارزميات البحث في نُظم إدارة الملفات لتحديد موقع الملفات باستخدام اسم، ومحتوى الملف، وبعض السمات الأخرى.



شكل 2.15: استخدام الروبوت خوارزمية البحث لتحديد طريقه

أنواع خوارزميات البحث وأمثلتها Types and Examples of Search Algorithms

هناك نوعان رئيسان من خوارزميات البحث وهما: **غير المُستنيرة** (Uninformed) و**المُستنيرة** (Informed).

خوارزميات البحث غير المُستنيرة Uninformed Search Algorithms

خوارزميات البحث غير المُستنيرة، وتسمّى أيضًا: خوارزميات البحث العمياء، وهي تلك التي لا تحتوي على معلومات إضافية حول حالات المشكلة بإستثناء المعلومات المستفادة من تعريف المشكلة. وتقوم هذه الخوارزميات بإجراء فحص شامل لمساحة البحث استنادًا إلى مجموعة من القواعد المُحدَّدة مُسبقًا. وتُعدُّ تقنيات **البحث بأولوية الاتساع** (BFS) و**البحث بأولوية العمق** (DFS) المُشار إليها في الدرس الثاني أمثلة على خوارزميات البحث غير المُستنيرة.

على سبيل المثال، تبدأ خوارزمية **البحث بأولوية العمق** (DFS) عند عُقدة الجِذر بالشجرة أو المُخطَّط وتتوسَّع حتى تصل للعُقدة الأعمق التي لم تُفحَص. ويستمر الأمر بهذه الطريقة حتى تستنفد الخوارزمية مساحة البحث بأكملها بعد فحص كل العُقد المتاحة. ثم تُخرِج الحل الأمثل الذي وجدته أثناء البحث. فالحقيقة أن خوارزمية **البحث بأولوية العمق** (DFS) تَتبع دومًا هذه القواعد ولا يمكن ضبط استراتيجتها بصرف النظر عن نتائج البحث، وهذا ما يجعلها خوارزمية غير مُستنيرة.

ومثال آخر ملحوظ على هذا النوع من الخوارزميات هو خوارزمية **البحث بأولوية العمق التكراري المُتعمِّق** (Iterative Deepening Depth-First Search – IDDFS) التي يمكن اعتبارها مزيجًا بين خوارزميتي **البحث بأولوية العمق** (DFS) و**البحث بأولوية الاتساع** (BFS)، فهي تَستخدِم استراتيجة العُمق أولًا للبحث في جميع الخيارات الموجودة في النطاق الكامل بصورة متكررة حتى تصل إلى عُقدة مُحدَّدة.

خوارزميات البحث المُستنيرة Informed Search Algorithms

على النقيض من خوارزميات البحث غير المُستنيرة، تَستخدِم خوارزميات البحث المستنيرة المعلومات حول المشكلة ومساحة البحث لتوجيه عملية البحث. والأمثلة على هذه الخوارزميات تشمل:

- **خوارزمية البحث بأولوية الأفضل** (A* search) تَستخدِم دالة استدلالية لتقدير المسافة بين كل عُقدة من العُقد المُرشَّحة والعُقدة المُستهدَفة. ثم تُوسِّع العُقدة المُرشَّحة بالتقدير الأقل. إن فعَّالية خوارزمية **البحث بأولوية الأفضل** (A* search) مرتبطة بجودة دالتها الاستدلالية. على سبيل المثال، إذا كنت تضمن أن الاستدلال لن يتجاوز المسافة الفعلية إلى الهدف، فبالتالي ستعثر الخوارزمية على الحل الأمثل. بخلاف ذلك، قد لا يكون الحل الناتج من الخوارزمية هو الأفضل.
- **خوارزمية ديكسترا** (Dijkstra's Algorithm) تُوسِّع العُقدة بناء على أقصر مسافة فعلية إلى الهدف في كل خطوة. ولذلك، على النقيض من خوارزمية البحث بأولوية الأفضل، تَحسب خوارزمية **ديكسترا** (Dijkstra) المسافة الفعلية ولا تَستخدِم التقديرات الاستدلالية. وبينما يجعل هذا خوارزمية ديكسترا أبطأ من خوارزمية البحث بأولوية الأفضل، إلا أن ذلك يعني ضمان العثور على الحل الأمثل دومًا (ممثلًا بالمسار الأقصر من البداية حتى الهدف).
- **خوارزمية تسلُّق التلال** (Hill Climbing) تبدأ بتوليد حل عشوائي، ثم تحاول تحسين هذا الحل بصورة متكررة بإجراء تغييرات بسيطة تُحسِّن من دالة استدلالية مُحدَّدة. وبالرغم من أن هذه المنهجية لا تضمن إيجاد الحل الأمثل، إلا أنها سهلة التنفيذ وتتميز بفعالية كبيرة عند تطبيقها على أنواع مُعينة من المشكلات.

الدالة الاستدلالية (Heuristic Function):
هي الدالة التي تُصنِّف البدائل في خوارزميات البحث عند كل مرحلة فرعية استنادًا إلى تقديرات استدلالية مبنية على البيانات المتوفرة لتحديد الفَرع الذي ستسلكه.

شكل 2.16: حل المتاهة نفسها باستخدام خوارزمية البحث بأولوية الأفضل وخوارزمية ديكسترا

الخلايا ذات اللون البنفسجي هي الخلايا التي تمّ فحصها، والخلية ذات اللون الأخضر هي موضع البدء، والخلية ذات اللون الأحمر هي موقع الهدف، بينما الخلايا ذات اللون الأصفر تمثل المسار الذي تم العثور عليه.

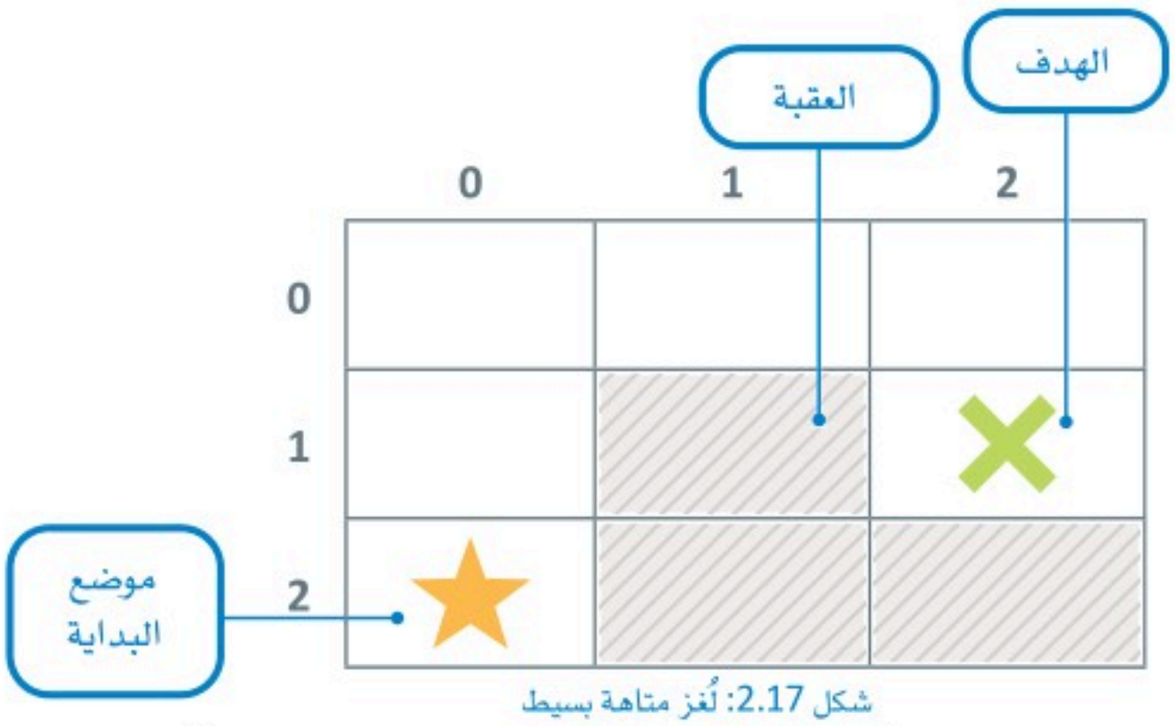
في هذه الوحدة، ستشاهد بعض الأمثلة المرئية وتطبيقات البايثون على خوارزمية **البحث بأولوية الاتساع** (BFS) وخوارزمية **البحث بأولوية الأفضل** (A* search) لمعرِفة الاختلافات بين خوارزميتي البحث المُستنيرة وغير المُستنيرة.

إنشاء ألغاز المتاهة بواسطة البايثون

Creating Maze Puzzles in Python

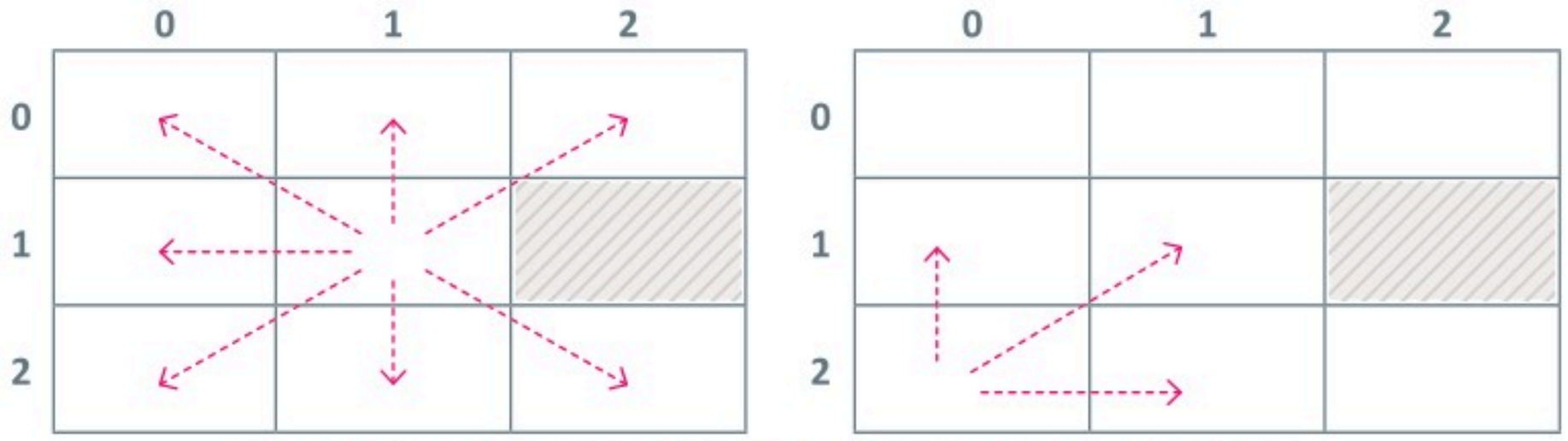
تُعرَّف المتاهة في صورة إطار شبكي 3×3.

يُحدَّد موضع البداية بنجمة في أسفل يسار المتاهة. الهدف هو الوصول إلى الخلية المُستهدَفة المُحدَّدة بالعلامة X، ويمكن للّاعب الانتقال إلى أي خلية فارغة مجاورة لموقعه الحالي.



شكل 2.17: لُغز متاهة بسيط

تكون الخلية فارغة إذا لم تحتوي على عائق. على سبيل المثال، المتاهة الموضَّحة في شكل 2.17 تحتوي على 3 خلايا تشغلُها **الحواجز** (Blocks). هذه الحواجز الملوّنة باللون الرمادي تُشكِّل عائقًا يجب على اللاعب تجاوزه للوصول إلى الهدف X، ويمكن للّاعب الانتقال بشكل أفقي أو رأسي أو قطري إلى أي خلية فارغة مجاورة لموقعه الحالي كما يظهر في الشكل 2.18، على سبيل المثال:



شكل 2.18: يمكن للاعب الانتقال بشكل أفقي أو رأسي أو قطري إلى أي خلية فارغة مجاورة لموقعه الحالي

الهدف هو إيجاد المسار الأقصر والأقلّ عددًا لمرات فحص الخلايا. وبالرغم من أن المتاهة الصغيرة 3×3 قد تبدو بسيطة للّاعب البشري، إلا أنه يتوجب على الخوارزمية الذكية إيجاد حلول للتعامل مع المتاهات الكبيرة والمُعقدة للغاية، مثل: متاهة 10,000×10,000 التي تحتوي على ملايين الحواجز المُوزَّعة في أشكال مُعقَّدة ومتنوعة.

يمكن استخدام المقطع البرمجي التالي بلغة البايثون لإنشاء مجموعة بيانات تُصوّر المثال المُوضَّح في الشكل 2.18.

```
import numpy as np

# create a numeric 3 x 3 matrix full of zeros.
small_maze=np.zeros((3,3))

# coordinates of the cells occupied by blocks
blocks=[(1, 1), (2, 1), (2, 2)]

for block in blocks:
    # set the value of block-occupied cells to be equal to 1
    small_maze[block]=1

small_maze
```

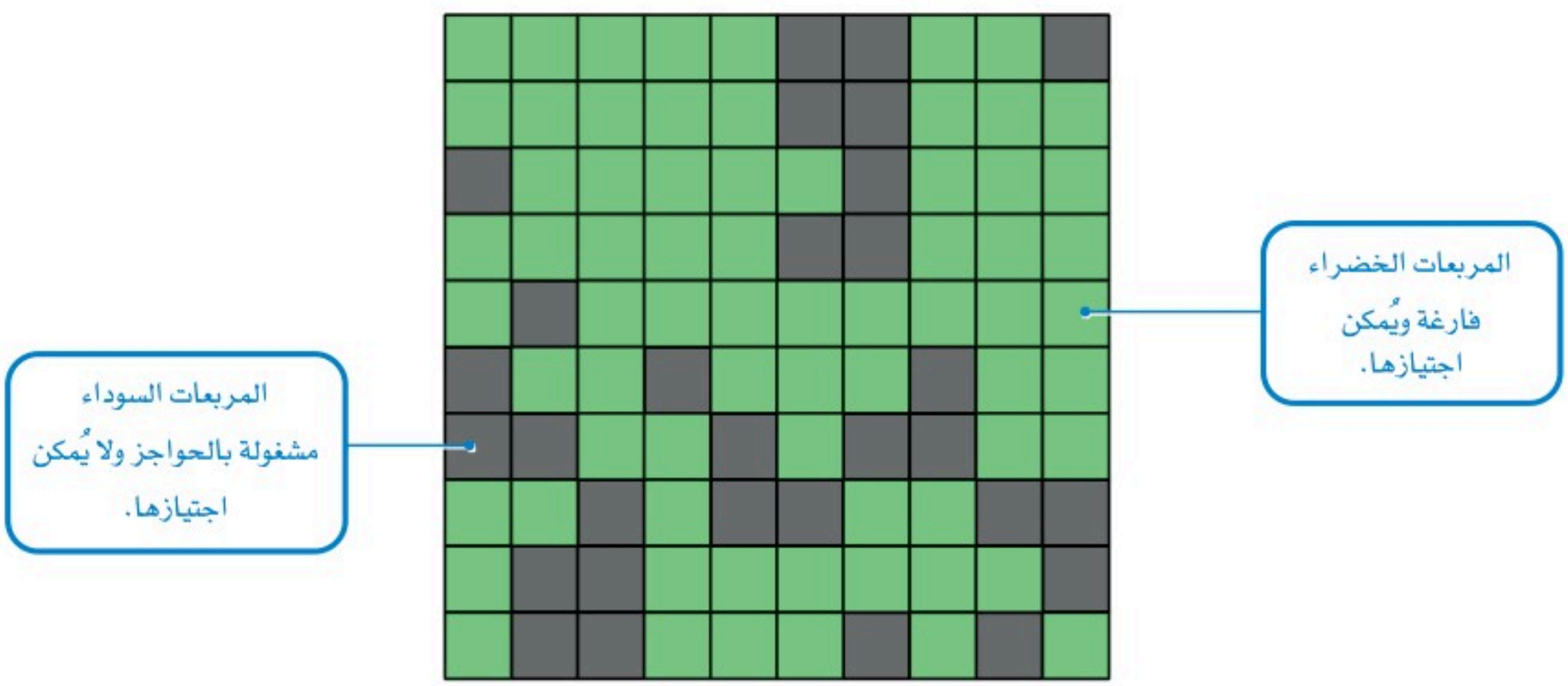
```
array([[0., 0., 0.],
       [0., 1., 0.],
       [0., 1., 1.]])
```

في هذا التمثيل الرقمي للمتاهة، تُمثَّل الخلايا الفارغة **بالأصفار** (Zeros) والمشغولة **بالآحاد** (Ones). يمكن تحديث المقطع البرمجي نفسه بسهولة لإنشاء متاهات كبيرة ومُعقدة للغاية، مثل:

```
import random

random_maze=np.zeros((10,10))

# coordinates of 30 random cells occupied by blocks
blocks=[(random.randint(0,9),random.randint(0,9)) for i in range(30)]

for block in blocks:
    random_maze[block]=1
```

تُستخدَم الدالة التالية لتمثيل المتاهة:

```
import matplotlib.pyplot as plt # library used for visualization

def plot_maze(maze):
    ax = plt.gca()          # create a new figure
    ax.invert_yaxis()       # invert the y-axis to match the matrix
    ax.axis('off')          # hide the axis labels
    ax.set_aspect('equal') # make sure the cells are rectangular

    plt.pcolormesh(maze, edgecolors='black', linewidth=2,cmap='Accent')
    plt.show()

plot_maze(random_maze)
```



شكل 2.19: تمثيل متاهة 10×10 باستخدام حواجز عشوائية

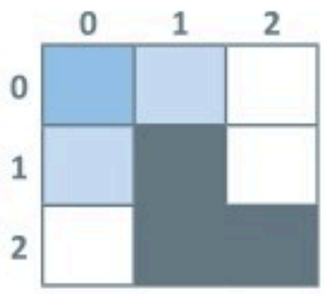
يُمكن استخدام الدالة التالية لاستدعاء قائمة تحتوي على كل الخلايا الفارغةِ والمُجاورة لخلية مُحدَّدة في أي متاهة:

```
def get_accessible_neighbors(maze:np.ndarray, cell:tuple):

    # list of accessible neighbors, initialized to empty
    neighbors=[]

    x,y=cell

    # for each adjacent cell position
    for i,j in [(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y),
(x+1,y+1)]:

        # if the adjacent cell is within the bounds of the grid and is not occupied by a block
        if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and
maze[(i,j)]==0:

            neighbors.append(((i,j),1))

    return neighbors
```

x-1, y-1	x-1, y	x-1, y+1
x, y-1	**x, y**	x, y+1
x+1, y-1	x+1, y	x+1, y+1

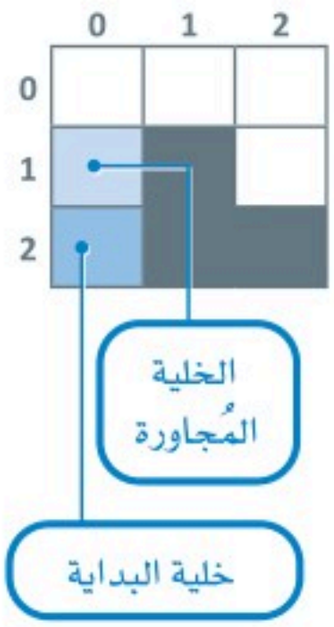
يَفترض هذا التطبيق أن كل عملية انتقال من خلية إلى أخرى مجاورة سواءً أفقيًا أو رأسيًا أو قطريًا يتم بتكلفة مقدارها وِحدَةٌ وَاحِدَة فقط. سيتم إعادة النظر في هذه الفرضية في وقت لاحق من هذا الدرس بعرض حالات أكثر تعقيدًا مع شروط انتقال مُتغيرة.

تَستخدِم كل خوارزميات البحث دالة ( )get_accessible_neighbors في محاولة حل المتاهة. في الأمثلة التالية تُستخدم المتاهة 3×3 المُصمَّمة بالأعلى للتحقق من أن الدالة تستدعي الخلية الصحيحة الفارغة والمجاورة للخلية المُحدَّدة.




```
# this cell is the northwest corner of the grid and has only 2 accessible neighbors
get_accessible_neighbors(small_maze, (0,0))
```

```
[((0, 1), 1), ((1, 0), 1)]
```



شكل 2.20: الخلايا المجاورة

```
# the starting cell (in the southwest corner) has only 1 accessible neighbor
get_accessible_neighbors(small_maze, (2,0))
```

```
[((1, 0), 1)]
```

بعد أن تعلّمت كيفية إنشاء المتاهات، وكذلك استدعاء الخلايا المجاورة لأي خلية في المتاهة، فإنّ الخطوة التالية هي تطبيق خوارزميات البحث التي يمكنها حل المتاهة من خلال إيجاد المسار الأقصر من خلية البداية إلى خلية الهدف المُحدَّدة.

استخدام خوارزمية البحث بأولوية الاتساع في حل ألغاز المتاهة
Using BFS to Solve Maze Puzzles

تَستخدِم دالة bfs_maze_solver( ) المُشار إليها في هذا الجزء خوارزمية **البحث بأولوية الاتساع** (BFS) لحل ألغاز المتاهة باستخدام خلية البداية وخلية الهدف. يَستخدِم هذا النموذج دالة get_accessible_neighbors( ) المُحدَّدة بالأعلى لاستدعاء الخلايا المُجاورة التي يمكن فحصها عند أي نقطة أثناء البحث، وبمجرد عثور خوارزمية **البحث بأولوية الاتساع** (BFS) على الخليّة الهدف، ستُستخدم دالة reconstruct_shortest_path( ) المُوضحة بالأسفل لإعادة بِناء المسار الأقصر واستدعائه، وذلك بتتبع المسار بصورة عكسية من خلية الهدف إلى خلية البداية:

```
def reconstruct_shortest_path(parent:dict, start_cell:tuple, target_cell:tuple):

    shortest_path = []
    my_parent=target_cell # start with the target_cell

    # keep going from parent to parent until the search cell has been reached
    while my_parent!=start_cell:

        shortest_path.append(my_parent) # append the parent

        my_parent=parent[my_parent] # get the parent of the current parent

    shortest_path.append(start_cell) # append the start cell to complete the path

    shortest_path.reverse() # reverse the shortest path

    return shortest_path
```

ستُستخدَم دالة reconstruct_shortest_path( ) أيضًا لإعادة بِناء الحل لخوارزمية **البحث بأولوية الأفضل** (A* search) المُشار إليها سلفًا في هذا الدرس. وبالنظر إلى تعريف الدالتين get_accessible_ neighbors( ) و reconstruct_shortest_path( ) المُساعدتين، يُمكن تنفيذ دالة bfs_maze_solver( ) على النحو التالي:

```
from typing import Callable # used to call a function as an argument of another function

def bfs_maze_solver(start_cell:tuple,
                    target_cell:tuple,
                    maze:np.ndarray,
                    get_neighbors: Callable,
                    verbose:bool=False): # by default, suppresses descriptive output text

    cell_visits=0 # keeps track of the number of cells that were visited during the search
    visited = set() # keeps track of the cells that have already been visited
    to_expand = [] # keeps track of the cells that have to be expanded

    # add the start cell to the two lists
    visited.add(start_cell)
    to_expand.append(start_cell)
    # remembers the shortest distance from the start cell to each other cell
    shortest_distance = {}
    # the shortest distance from the start cell to itself, zero
```

```
    shortest_distance[start_cell] = 0

    # remembers the direct parent of each cell on the shortest path from the start_cell to the cell
    parent = {}
    #the parent of the start cell is itself
    parent[start_cell] = start_cell

    while len(to_expand)>0:

        next_cell = to_expand.pop(0) # get the next cell and remove it from the expansion list

        if verbose:
            print('\nExpanding cell', next_cell)

        # for each neighbor of this cell
        for neighbor,cost in get_neighbors(maze, next_cell):

            if verbose:
                print('\tVisiting neighbor cell',neighbor)

            cell_visits+=1

            if neighbor not in visited: # if this is the first time this neighbor is visited

                visited.add(neighbor)
                to_expand.append(neighbor)
                parent[neighbor]= next_cell
                shortest_distance[neighbor]=shortest_distance[next_cell]+cost

                # target reached
                if neighbor==target_cell:

                    # get the shortest path to the target cell, reconstructed in reverse.
                    shortest_path = reconstruct_shortest_path(parent,
                                                              start_cell, target_cell)

                    return shortest_path, shortest_distance[target_cell],cell_visits

            else: # this neighbor has been visited before

                # if the current shortest distance to the neighbor is longer than the shortest
                # distance to next_cell plus the cost of transitioning from next_cell to this neighbor
                if shortest_distance[neighbor]>shortest_distance[next_cell]
                                                                +cost:


                    parent[neighbor]=next_cell
                    shortest_distance[neighbor]=shortest_distance[next_cell]+cost

    # search complete but the target was never reached, no path exists
    return None,None,None
```

تتّبع الدالة منهجية **البحث بأولوية الاتساع** (BFS) للبحث في كل الخيارات في العُمق الحالي قبل الانتقال إلى مستوى العُمق التالي، وتَستخدم هذه المنهجية مجموعة واحدة تُسمّى visited وقائمة تُسمى to_expand.

تتضمن المجموعة الأولى كل الخلايا التي فُحِصَت مرة واحدة على الأقل من قبل الخوارزمية، بينما تتضمن القائمة الثانية كل الخلايا التي لم تُوَسَّع بعد، مما يعني أن الخلايا المُجاورة لم تُفحص بعد. تَستخدم الخوارزمية كذلك قاموسين shortest_distance وparent، يحفظ الأوّل منهما طول المسار الأقصر من خلية البداية إلى كل خلية أخرى، بينما يحفظ الثاني عُقدة الخلية الأصل في المسار الأقصر.

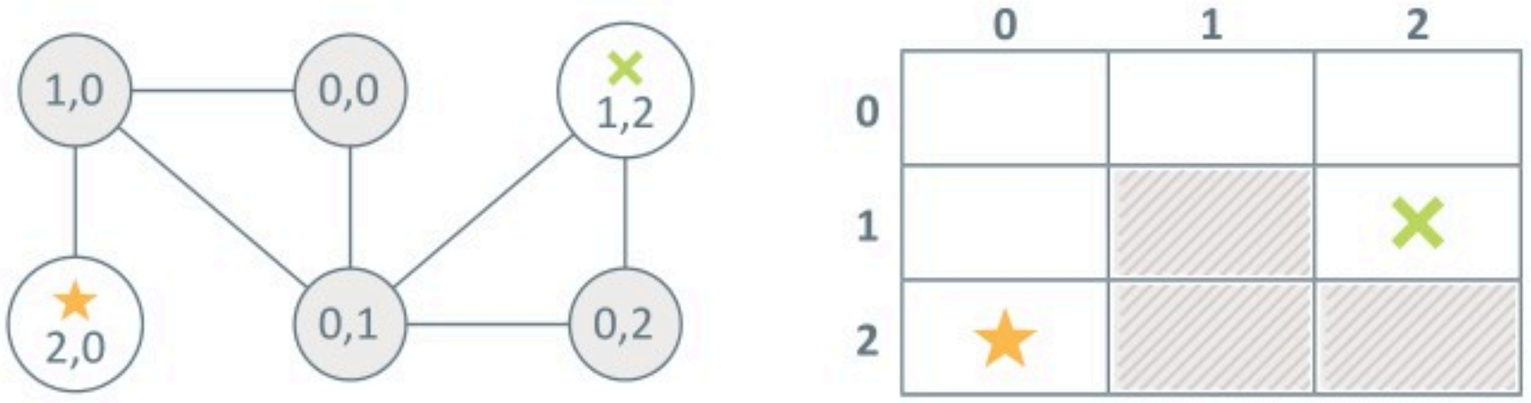
بمجرد الوصول إلى الخلية الهدف وانتهاء البحث، سيُخزِّن المتغيِّر shortest_distance[target_cell] طول الحل والذي يمثل طول المسار الأقصر من البداية إلى الهدف.

يَستخدِم المقطع البرمجي التالي دالة bfs_maze_solver( ) لحل المتاهة الصغيرة 3×3 المُوضَّحة بالأعلى:

```
start_cell=(2,0) # start cell, marked by a star in the 3x3 maze
target_cell=(1,2) # target cell, marked by an "X" in the 3x3 maze

solution, distance, cell_visits=bfs_maze_solver(start_cell,
                                        target_cell,
                                        small_maze,
                                        get_accessible_neighbors,
                                        verbose=True)

print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

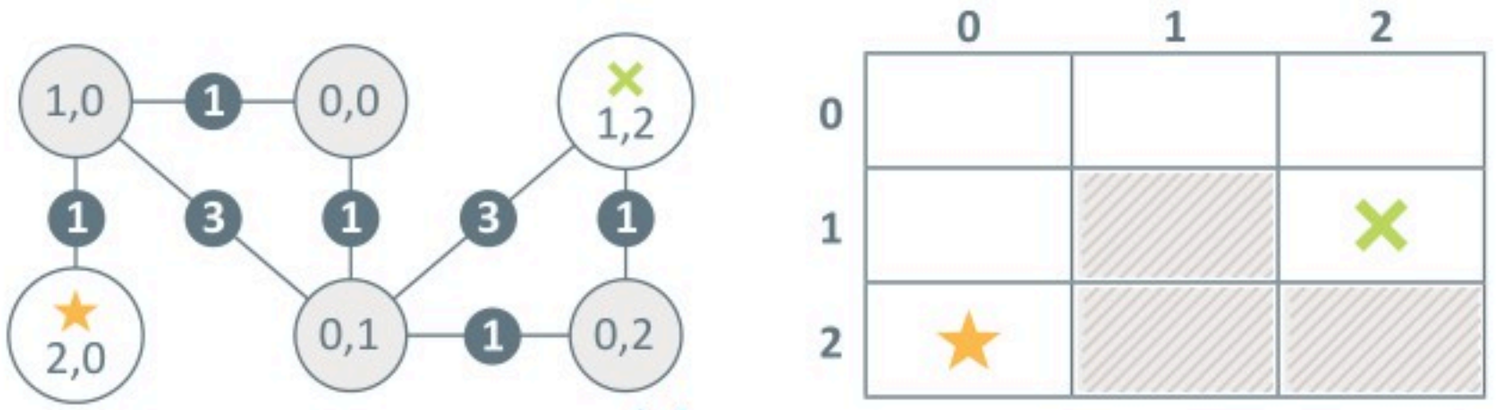
```
Expanding cell (2, 0)
      Visiting neighbor cell (1, 0)

Expanding cell (1, 0)
      Visiting neighbor cell (0, 0)
      Visiting neighbor cell (0, 1)
      Visiting neighbor cell (2, 0)

Expanding cell (0, 0)
      Visiting neighbor cell (0, 1)
      Visiting neighbor cell (1, 0)

Expanding cell (0, 1)
      Visiting neighbor cell (0, 0)
      Visiting neighbor cell (0, 2)
      Visiting neighbor cell (1, 0)
      Visiting neighbor cell (1, 2)

Shortest Path: [(2, 0), (1, 0), (0, 1), (1, 2)]
Cells on the Shortest Path: 4
Shortest Path Distance: 3
Number of cell visits: 10
```

تنجح خوارزمية **البحث بأولوية الاتساع** (BFS) في إيجاد المسار الأقصر بعد فحص 10 خلايا. يُمكن تصوير عملية البحث المطبّقة بخوارزمية **البحث بأولوية الاتساع** (BFS) بسهولة عند تصوير المتاهة بالتمثيل المُستنِد إلى مُخطَّط. المثال التالي يعرض متاهة 3×3 وتمثيلها بالمُخطَّط:



يتضمن تمثيل المُخطَّط عُقدة واحدة لكل خلية غير مشغولة. تُوضِّح القيمة على العُقد إحداثيات خلية المصفوفة المُقابلة. ستظهر حافة غير مُوجَّهة من عُقدة إلى أخرى في حال كانت الخلايا المُقابلة يُمكن الوصول إليها من خلال الانتقال من واحدة إلى الأخرى. إحدى الملاحظات المُهمّة حول خوارزمية **البحث بأولوية الاتساع** (BFS) هي أنه في حالة **المُخطَّطات غير الموزونة** (Unweighted Graphs) يكون المسار الأول الذي تُحدِّده الخوارزمية بين خلية البداية وأي خلية أخرى هو المسار الذي يتضمن أقل عدد من الخلايا التي تمّ فحصها. وهذا يعني أنه إذا كانت كلّ الحواف في المُخطَّط لها الوزن نفسه، أي كان لكلّ الانتقالات من خلية إلى أخرى التكلفة نفسها، فإنّ المسار الأول الذي تُحدِّده الخوارزمية إلى عُقدة مُحدَّدة يكون هو المسار الأقصر إلى تلك العُقدة. ولهذا السبب، تتوقف دالة bfs_maze_solver() عن البحث، وتَعرض نتيجة المرة الأولى التي فَحصَت فيها العُقدة المُستهدَفة.

ومع ذلك، لا يمكن تطبيق هذه المنهجية على **المُخطَّطات الموزونة** (Weighted Graphs). المثال التالي يوضِّح **إصدارًا موزونًا** (Weighted Version) لتمثيل مُخطَّط متاهة 3×3:



شكل 2.21: المتاهة ومُخطَّطها الموزون

في هذا المثال، يكون وزن كل الحواف المُقابلة للحركات الرأسية أو الأفقية (جنوبًا، شمالًا، غربًا، شرقًا) يساوي 1.

ومع ذلك، يكون وزن كل الحواف المُقابلة للحركات القُطرية (جنوبية غربية، جنوبية شرقية، شمالية غربية، شمالية شرقية) يساوي 3. في هذه الحالة الموزونة، سيكون المسار الأقصر هو [(2,0)، (1,0)، (0,0)، (0,1)، (0,2)، (1,2)]، بمسافة إجمالية: 1+1+1+1=5.

يمكن ترميز هذه الحالة الأكثر تعقيدًا باستخدام الإصدار الموزون من الدالة get_accessible_neighbors() المُوضَّحة بالأسفل.

```
def get_accessible_neighbors_weighted(maze:np.ndarray,
                                      cell:tuple,
                                      horizontal_vertical_weight:float,
                                      diagonal_weight:float):
```

```
    neighbors=[]
    x,y=cell

    for i,j in [(x-1,y-1), (x-1,y+1), (x+1,y-1), (x+1,y+1)]: # for diagonal neighbors

        # if the cell is within the bounds of the grid and it is not occupied by a block
        if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:

            neighbors.append(((i,j), diagonal_weight))

    for i,j in [(x-1,y), (x,y-1), (x,y+1), (x+1,y)]: # for horizontal and vertical neighbors

        if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:

            neighbors.append(((i,j), horizontal_vertical_weight))

    return neighbors
```

تسمح الدالة للمُستخدِم بتعيين وزن مُخصّص للحركات الأفقية والحركات الرأسية، وكذلك وزن مُخصّص مختلف للحركات القُطرية. إذا استُخدِم **الإصدار الموزون** (Weighted Version) المُشار إليه بواسطة **أداة الحل في البحث بأولوية الاتساع** (BFS solver)، فإنّ النتائج ستكون كما يلي:

```
from functools import partial

start_cell=(2,0)
target_cell=(1,2)
horz_vert_w=1  # weight for horizontal and vertical moves
diag_w=3  # weight for diagonal moves

solution, distance, cell_visits=bfs_maze_solver(start_cell,
                                                target_cell,
                                                small_maze,
                                                partial(get_accessible_neighbors_weighted,
                                                        horizontal_vertical_weight=horz_vert_w,
                                                        diagonal_weight=diag_w),
                                                verbose=False)

print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

```
Shortest Path: [(2, 0), (1, 0), (0, 1), (1, 2)]
Cells on the Shortest Path: 4
Shortest Path Distance: 7
Number of cell visits: 6
```

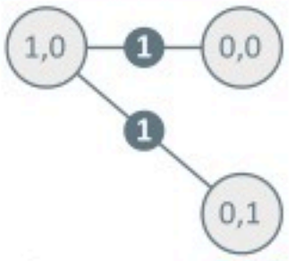
وكما هو مُتوقَّع، أخطأت أداة **الحل في البحث بأولوية الاتساع** (BFS solver) في عرض المسار السابق نفسه بالضبط، على الرغم من أن التكلفة تساوي 7، ومن الواضح أنه ليس المسار الأقصر. ويرجع ذلك إلى الطبيعة غير المستنيرة لخوارزمية **البحث بأولوية الاتساع** (BFS)، حيث لا تأخذ الخوارزمية الأوزان بعين الاعتبار عند تحديد الخلية المُقرَّر توسيعها في الخطوة التالية؛ لأنها تُطبِّق ببساطة منهجية البحث بالعرض نفسِها والتي تؤدي إلى المسار نفسه الذي وجدته الخوارزمية في **الإصدار غير الموزون** (Unweighted Version) من المتاهة. القسم التالي يصف طريقة معالجة نقطة الضعف هذه باستخدام خوارزمية **البحث بأولوية الأفضل** (A* search)، وهي خوارزمية مُستنيرة وأكثر ذكاءً تضبط سلوكها وفقًا للأوزان المُحدَّدة، وبالتالي يُمكنها حل المتاهات باستخدام **الانتقالات الموزونة** (Weighted Transitions) و**الانتقالات غير الموزونة** (Unweighted Transitions).

استخدام خوارزمية البحث بأولوية الأفضل في حل ألغاز المتاهة

Using A* Search to Solve Maze Puzzles

كما في خوارزمية **البحث بأولوية الاتساع** (BFS)، تَفحَص خوارزمية **البحث بأولوية الأفضل** (A* search) خلية واحدة في كل مرة بفحص كل خلية مجاورة يمكن الوصول إليها. فبينما تستخدِم خوارزمية **البحث بأولوية الاتساع** (BFS) منهجية بحث عمياء بأولوية العرض لتحديد الخلية التالية التي ستَفحَصها، تَفحَص خوارزمية **البحث بأولوية الأفضل** (A* search) الخلية التي يكون بينها وبين الخلية المُستهدَفة أقصر مسافة محسوبة بواسطة **الدالة الاستدلالية** (Heuristic Function). يعتمد التعريف الدقيق للدالة الاستدلالية على التطبيق. في حالة ألغاز المتاهة، توفِّر الدالة الاستدلالية تقديرًا دقيقًا لمدى قُرب الخلية المُرشَّحة إلى الخلية المُستهدَفة. يضمن الاستدلال المُطبَّق عدم **المبالغة في تقدير** (Overestimate) المسافة الفعلية إلى الخلية المُستهدَفة مثل: عرض مسافة تقديرية أكبر من المسافة الحقيقية إلى الهدف، وبالتالي ستُحدِّد الخوارزمية أقصر مسار مُحتمل لكلٍ من المُخطَّطين **الموزون** (Weighted) و**غير الموزون** (Unweighted). إذا كان الاستدلال يُبالِغ في بعض الأحيان في تقدير المسافة، ستُقدِّم خوارزمية **البحث بأولوية الأفضل** (A* search) حلًا، ولكن قد لا يكون الأفضل. الاستدلال المُحتمل الأبسط الذي لن يؤدي إلى المُبالغة في تقدير المسافة هو دالة بسيطة تعطي دومًا مسافة تقديرية قدرها وِحدَة واحدة.

```
def constant_heuristic(candidate_cell:tuple, target_cell:tuple):
    return 1
```



شكل 2.22: الاستدلال الثابت

على الرغم من أن هذا الاستدلال شديد التفاؤل، إلا أنه لن يُقدم أبدًا تقديرًا أعلى من المسافة الحقيقية، وبالتالي سيؤدي إلى أفضل حل ممكن. سيتم تقديم استدلال متطور يُمكنه العثور على أفضل حل بشكلٍ سريع في هذا القسم لاحقًا.

تَستخدِم الدالة التالية دالة استدلالية معطاة للعثور على الخلية التي يجب توسيعها بعد ذلك:

```
def get_best_candidate(expansion_candidates:set,
                       shortest_distance:dict,
                       heuristic:Callable):

    winner = None
    # best (lowest) distance estimate found so far. Initialized to a very large number
    best_estimate= sys.maxsize

    for candidate in expansion_candidates:

        # distance estimate from start to target, if this candidate is expanded next
```

```
        candidate_estimate=shortest_distance[candidate]+heuristic(candidate,target_cell)
         if candidate_estimate < best_estimate:

             winner = candidate
             best_estimate=candidate_estimate

    return winner
```

يَستخدِم التطبيق المُشار إليه بالأعلى حلقة التكرار For لفحص الخلايا المُرشَّحة في المجموعة وتحديد الأفضل. ولِتطبيق أكثر كفاءة، قد يُستخدم **طابور الأولوية** (Priority Queue) في تحديد المُرشَّح الأفضل دون الحاجة إلى فحص كل المُرشَّحِين بصورة متكررة. تُستخدَم دالة ( )get_best_candidate كدالةٍ مُساعدة بواسطة دالة ( )astar_maze_solver المُوضَّحة فيما يلي. وبالإضافة إلى الدالة الاستدلالية، يَستخدِم هذا التطبيق كذلك الدالتين المُساعدتين ( )get_accessible_neighbors_weighted و ( )reconstruct_shortest_path المُشار إليهما في القسم السابق.

```
import sys

def astar_maze_solver(start_cell:tuple,
                      target_cell:tuple,
                      maze:np.ndarray,
                      get_neighbors: Callable,
                      heuristic:Callable,
                      verbose:bool=False):

    cell_visits=0

    shortest_distance = {}
    shortest_distance[start_cell] = 0

    parent = {}
    parent[start_cell] = start_cell

    expansion_candidates = set([start_cell])

    fully_expanded = set()

    # while there are still cells to be expanded
    while len(expansion_candidates) > 0:

        best_cell = get_best_candidate(expansion_candidates,shortest_distance,heuristic)

         if best_cell == None:  break

         if verbose: print('\nExpanding cell', best_cell)

         # if the target cell has been reached, reconstruct the shortest path and exit
         if best_cell == target_cell:
```

```
            shortest_path=reconstruct_shortest_path(parent,start_cell,target_cell)
            return shortest_path, shortest_distance[target_cell],cell_visits
        for neighbor,cost in get_neighbors(maze, best_cell):
            if verbose: print('\nVisiting neighbor cell', neighbor)
            cell_visits+=1
            # first time this neighbor is reached
            if neighbor not in expansion_candidates and neighbor not in fully_expanded:
                expansion_candidates.add(neighbor)
                parent[neighbor] = best_cell # mark the best_cell as this neighbor's parent
                # update the shortest distance for this neighbor
                shortest_distance[neighbor] = shortest_distance[best_cell] + cost
            # this neighbor has been visited before, but a better (shorter) path to it has just been found
            elif shortest_distance[neighbor] > shortest_distance[best_cell] + cost:
                shortest_distance[neighbor] = shortest_distance[best_cell] + cost
                parent[neighbor] = best_cell
                if neighbor in fully_expanded:
                    fully_expanded.remove(neighbor)
                    expansion_candidates.add(neighbor)
        # all neighbors of best_cell have been inspected at this point
        expansion_candidates.remove(best_cell)
        fully_expanded.add(best_cell)
    return None, None, None   # no solution was found
```

وكما الحال في الدالة ()bfs_maze_solver، تَستخدِم الدالة المُوضَّحة بالأعلى كذلك القاموسين shortest_distance وparent لحفظ طول المسار الأقصر من خلية البداية إلى أي خلية أخرى، وحفظ عُقدة أصل الخلية في هذا المسار الأقصر.

ورغم ذلك، تَتبع الدالة ()astar_maze_solve منهجية مختلفة لفحص الخلايا وتوسيعها، فهي تَستخدِم expansion_candidates لتتبُّع كل الخلايا التي يمكن توسيعها بعد ذلك. في كل تكرار، تُستخدَم دالة ()get_best_candidate لتحديد أيٍّ من الخلايا المُرشَّحة ستُوسَّعُها بعد ذلك.

بعد اختيار الخليةِ المرشحةِ best_cell، تُستخدَم حلقة التكرار For لفحص كل الخلايا المجاورة لها. إذا كانت الخلية المجاورة تُفحَص للمرة الأولى، فستصبح best_cell عُقدة الأصل للخلية المجاورة في المسار الأقصر.

يحدث الأمر نفسه إذا تم فحص الدالة المجاورة من قبل، ولكن فقط إذا كان المسار إلى هذه الخلية المجاورة من خلال best_cell أقصر من المسار السابق. إذا عثرت الدالة بالفعل على مسار أفضل، فسيتعين على الخلية المجاورة العودة إلى مجموعة expansion_candidates لإعادة تقييم المسار الأقصر إلى الخلايا المجاورة لها. يُستخدم المقطع البرمجي التالي ( )astar_maze_solver لحل **الحالة غير الموزونة** (Unweighted Case) للُغز المتاهة 3x3:

```
start_cell=(2,0)
target_cell=(1,2)

solution, distance, cell_visits=astar_maze_solver(start_cell,
                                        target_cell,
                                        small_maze,
                                        get_accessible_neighbors,
                                        constant_heuristic,
                                        verbose=False)

print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

```
Shortest Path: [(2, 0), (1, 0), (0, 1), (1, 2)]
Cells on the Shortest Path: 4
Shortest Path Distance: 3
Number of cell visits: 12
```

ستبحث **أداة الحل في البحث بأولوية الأفضل** (A* search solver) عن المسار المُحتمل الأقصر والأفضل بعد فحص 12 خلية. وهذا أكثر قليلًا من **أداة الحل في البحث بأولوية الاتساع** (BFS solver) التي وجدت الحل بعد فحص 10 خلايا فقط. هذا يعود إلى بساطة الاستدلال الثابت المُستخدَم لإرشاد ( )astar_maze_solver. وكما سيتضح لاحقًا في هذا القسم، يُمكن استخدام دالة استدلال أخرى لتمكين الخوارزمية من إيجاد الحل بشكلٍ أسرع.

الخطوة التالية هي تقييم ما إذا كانت خوارزمية **البحث بأولوية الأفضل** (A* search) قادرة على حل المتاهة الموزونة التي فشلت خوارزمية **البحث بأولوية الاتساع** (BFS) في العثور على أقصر مسار لها أم لا:

```
start_cell=(2,0)
target_cell=(1,2)

horz_vert_w=1  # weight for horizontal and vertical moves
diag_w=3  # weight for diagonal moves

solution, distance, cell_visits=astar_maze_solver(start_cell,
                                        target_cell,
                                        small_maze,
                                        partial(get_accessible_neighbors_weighted,
                                                horizontal_vertical_weight=horz_vert_w,
                                                diagonal_weight=diag_w),
                                        constant_heuristic,
                                        verbose=False)
```

```
print('\nShortest Path:', solution)
print('Cells on the Shortest Path:', len(solution))
print('Shortest Path Distance:', distance)
print('Number of cell visits:', cell_visits)
```

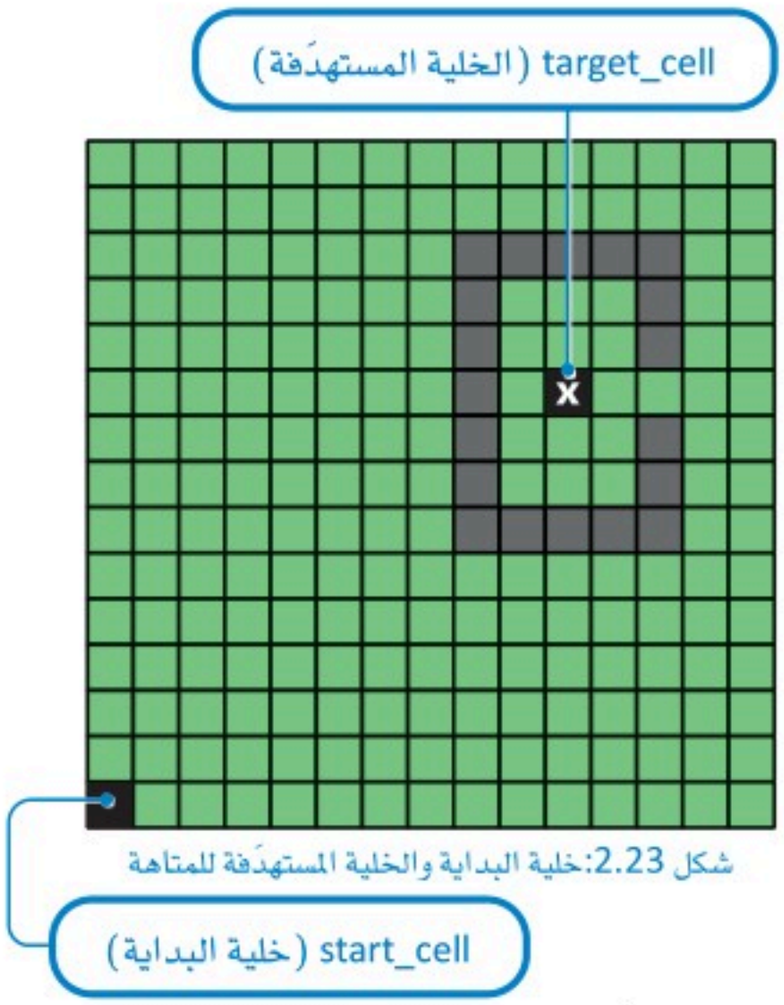
```
Shortest Path: [(2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (1, 2)]
Cells on the Shortest Path: 6
Shortest Path Distance: 5
Number of cell visits: 12
```

تُوضِّح النتائج قدرة astar_maze_solver( ) على حل الحالة الموزونة بالعثور على المسار الأقصر المُحتمل [(2، 0)، (1، 0)، (0، 0)، (0، 1)، (0، 2)، (1، 2)] بتكلفة إجمالية قدرها 5. وهذا يوضِّح مزايا استخدام خوارزمية بحث مستنيرة، فهي تمكنك من إيجاد الحل الأمثل باستخدام أبسط طريقة ممكنة.

المقارنة بين الخوارزميات Algorithm Comparison

الخطوة التالية هي المقارنة بين خوارزمية **البحث بأولوية الاتساع** (BFS) وخوارزمية **البحث بأولوية الأفضل** (A* search) في متاهة أكبر حجمًا وأكثر تعقيدًا. يُستخدَم المقطع البرمجي التالي بلغة البايثون لإنشاء تمثيل رقمي لهذه المتاهة:

```
big_maze=np.zeros((15,15))

# coordinates of the cells occupied by blocks
blocks=[(2,8), (2,9), (2,10), (2,11), (2,12),
        (8,8), (8,9), (8,10), (8,11), (8,12),
        (3,8), (4,8), (5,8), (6,8), (7,8),
        (3,12), (4,12), (6,12), (7,12)]

for block in blocks:
    # set the value of block-occupied cells to be equal to 1
    big_maze[block]=1
```



شكل 2.23:خلية البداية والخلية المستهدَفة للمتاهة

هذه المتاهة 15×15 تحتوي على قسم من الحواجز على شكل حرف C ينبغي على اللّاعب تجاوزها للوصول إلى الهدف المُحدَّد بالعلامة X. ثم تُستخدَم **أداة الحل في البحث بأولوية الاتساع** (BFS solver) و**أداة الحل في البحث بأولوية الأفضل** (A* search solver) لحل الإصدارات الموزونة وغير الموزونة من هذه المتاهة كبيرة الحجم:

الإصدار غير الموزون

```
start_cell=(14,0)
target_cell=(5,10)

solution_bfs_unw, distance_bfs_unw, cell_visits_bfs_unw=bfs_maze_solver(start_cell,
                                                target_cell,
                                                big_maze,
                                                get_accessible_neighbors,
```

```
                                        verbose=False)

print('\nBFS unweighted.')
print('\nShortest Path:', solution_bfs_unw)
print('Cells on the Shortest Path:', len(solution_bfs_unw))
print('Shortest Path Distance:', distance_bfs_unw)
print('Number of cell visits:', cell_visits_bfs_unw)

solution_astar_unw, distance_astar_unw, cell_visits_astar_unw=astar_maze_solver(
                                       start_cell,
                                       target_cell,
                                       big_maze,
                                       get_accessible_neighbors,
                                       constant_heuristic,
                                       verbose=False)

print('\nA* Search unweighted with a constant heuristic.')
print('\nShortest Path:', solution_astar_unw)
print('Cells on the Shortest Path:', len(solution_astar_unw))
print('Shortest Path Distance:', distance_astar_unw)
print('Number of cell visits:', cell_visits_astar_unw)
```

```
BFS unweighted.

Shortest Path: [(14, 0), (13, 1), (12, 2), (11, 3), (10, 4), (9, 5), (8,
6), (8, 7), (9, 8), (9, 9), (9, 10), (9, 11), (9, 12), (8, 13), (7, 13),
(6, 13), (5, 12), (4, 11), (5, 10)]
Cells on the Shortest Path: 19
Shortest Path Distance: 18
Number of cell visits: 1237

A* Search unweighted with a constant heuristic.

Shortest Path: [(14, 0), (13, 1), (12, 2), (11, 3), (10, 4), (10, 5), (10,
6), (9, 7), (9, 8), (10, 9), (9, 10), (9, 11), (9, 12), (8, 13), (7, 13),
(6, 13), (5, 12), (6, 11), (5, 10)]
Cells on the Shortest Path: 19
Shortest Path Distance: 18
Number of cell visits: 1272
```

الإصدار الموزون

```
start_cell=(14,0)
target_cell=(5,10)

horz_vert_w=1
diag_w=3

solution_bfs_w, distance_bfs_w, cell_visits_bfs_w=bfs_maze_solver(start_cell,
                                       target_cell,
```

```
                                    big_maze,
                                    partial(get_accessible_neighbors_weighted,
                                            horizontal_vertical_weight=horz_vert_w,
                                            diagonal_weight=diag_w),
                                    verbose=False)

print('\nBFS weighted.')
print('\nShortest Path:', solution_bfs_w)
print('Cells on the Shortest Path:', len(solution_bfs_w))
print('Shortest Path Distance:', distance_bfs_w)
print('Number of cell visits:', cell_visits_bfs_w)

solution_astar_w, distance_astar_w, cell_visits_astar_w=astar_maze_solver(start_cell,
                                    target_cell,
                                    big_maze,
                                    partial(get_accessible_neighbors_weighted,
                                            horizontal_vertical_weight=horz_vert_w,
                                            diagonal_weight=diag_w),
                                    constant_heuristic,
                                    verbose=False)

print('\nA* Search weighted with constant heuristic.')
print('\nShortest Path:', solution_astar_w)
print('Cells on the Shortest Path:', len(solution_astar_w))
print('Shortest Path Distance:', distance_astar_w)
print('Number of cell visits:', cell_visits_astar_w)
```

```
BFS weighted.

Shortest Path: [(14, 0), (14, 1), (14, 2), (13, 2), (13, 3), (12, 3), (12,
4), (11, 4), (11, 5), (10, 5), (10, 6), (9, 6), (9, 7), (9, 8), (9, 9), (9,
10), (9, 11), (9, 12), (9, 13), (8, 13), (7, 13), (6, 13), (5, 13), (5,
12), (4, 11), (5, 10)]
Cells on the Shortest Path: 26
Shortest Path Distance: 30
Number of cell visits: 1235

A* Search weighted with constant heuristic.

Shortest Path: [(14, 0), (13, 0), (12, 0), (11, 0), (10, 0), (9, 0), (9,
1), (9, 2), (9, 3), (9, 4), (9, 5), (9, 6), (9, 7), (9, 8), (9, 9), (9,
10), (9, 11), (9, 12), (9, 13), (8, 13), (7, 13), (6, 13), (5, 13), (5,
12), (5, 11), (5, 10)]
Cells on the Shortest Path: 26
Shortest Path Distance: 25
Number of cell visits: 1245
```

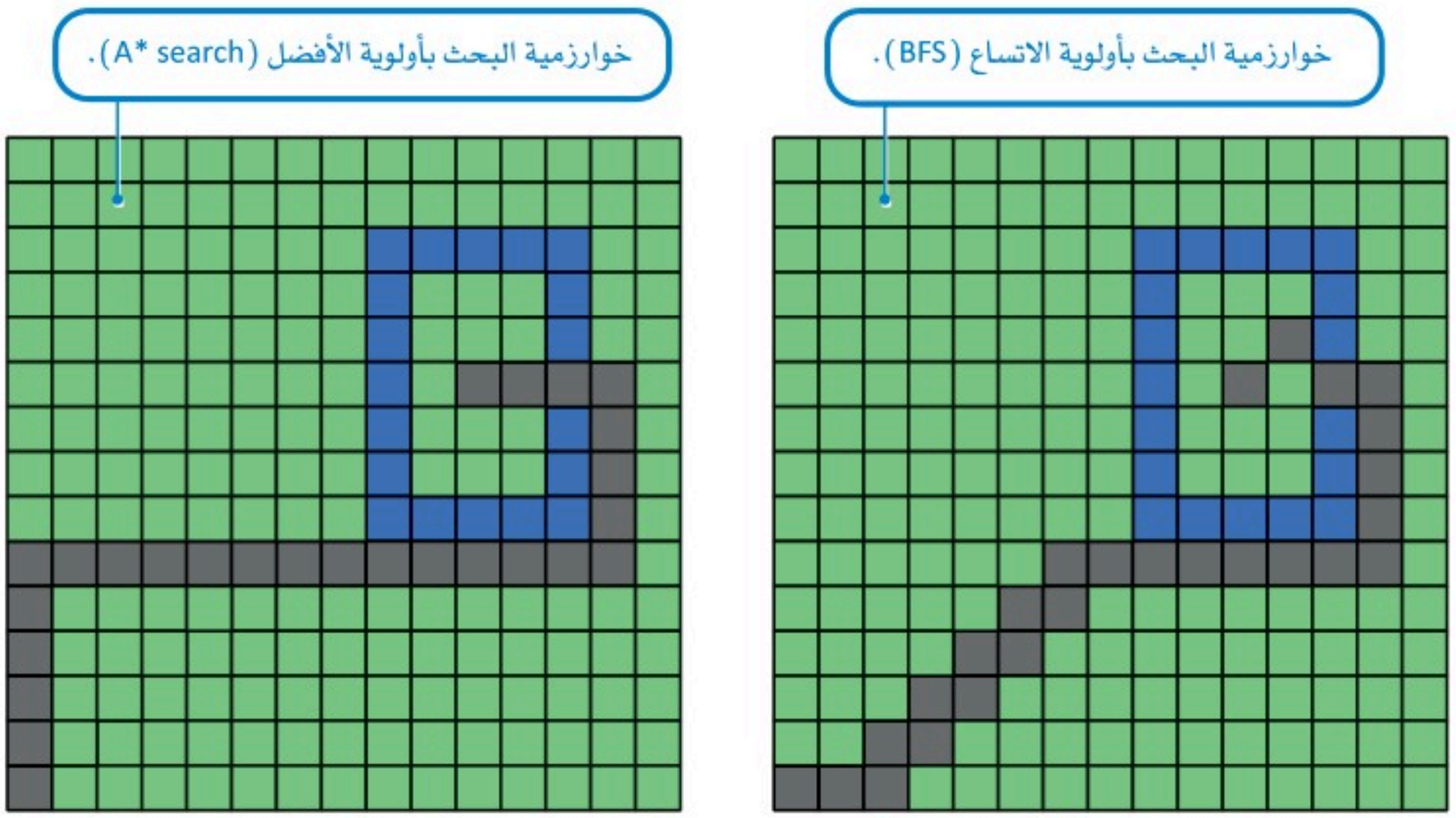
تتوافق النتائج مع تلك التي حصلت عليها في المتاهة الصغيرة وهي كالتالي:

- نجحت خوارزميّتا البحث بأولوية الاتساع (BFS) والبحث بأولوية الأفضل (A* search) في العثور على المسار الأقصر للإصدار غير الموزون.
- وجدت خوارزمية **البحث بأولوية الاتساع** (BFS) الحل بعد فحص عدد أقلّ من الخلايا وهو 1237 مقابل 1272 في خوارزمية **البحث بأولوية الأفضل** (A* search).
- فشلت خوارزمية **البحث بأولوية الاتساع** (BFS) في العثور على المسار الأقصر للإصدار الموزون، حيث عثرت على مسار بطول 30 وحدة.
- عثرت خوارزمية **البحث بأولوية الأفضل** (A* search) على المسار الأقصر للإصدار الموزون، حيث عثرت على مسار بطول 25 وحدة.

يُستخدَم المقطع التالي لتمثيل المسار الأقصر الذي وجدته الخوارزميتان؛ خوارزمية **البحث بأولوية الاتساع** (BFS) وخوارزمية **البحث بأولوية الأفضل** (A* search) للإصدار الموزون كالتالي:

```
maze_bfs_w=big_maze.copy()

for cell in solution_bfs_w:
    maze_bfs_w[cell]=2

plot_maze(maze_bfs_w)
```

```
maze_astar_w=big_maze.copy()

for cell in solution_astar_w:
    maze_astar_w[cell]=2

plot_maze(maze_astar_w)
```



شكل 2.24: مقارنة بين حلَّي خوارزميتي البحث بأولوية الاتساع والبحث بأولوية الأفضل

يؤكد التّمثيلان أن الطبيعة المُستنيرة لخوارزمية **البحث بأولوية الأفضل** (A* search) تسمح لها بتجنب الحركة القُطرية؛ لأن تكلفتها أعلى من الحركتين الأفقية والرأسية. ومن ناحية أخرى، تتجاهل خوارزمية **البحث بأولوية الأفضل** (BFS) غير المُستنيرة تكلفة كل حركة وتُعطي حلًا أعلى تكلفة. وفيما يلي مقارنة عامة بين الخوارزميات المُستنيرة وغير المُستنيرة كما هو موضَّح في الجدول 2.6:

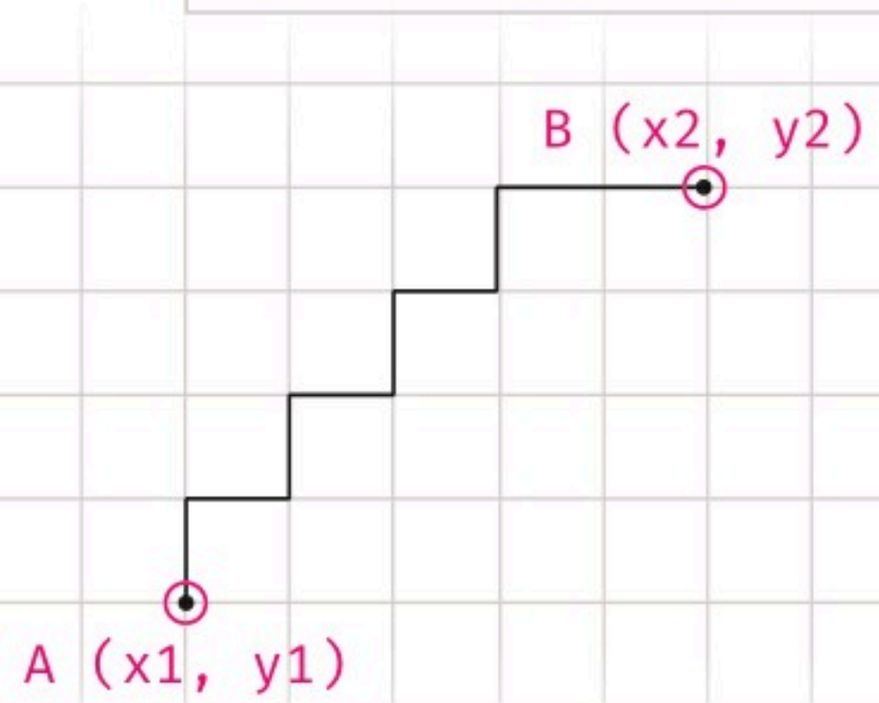
جدول 2.6: مقارنة بين الخوارزميات المُستنيرة وغير المُستنيرة

معايير المقارنة	المُستنيرة	غير المُستنيرة
التعقيد الحسابي (Computational Complexity)	أقل تعقيدًا.	أكثر تعقيدًا حسابيًا.
الكفاءة (Efficiency)	أسرع في عمليات البحث.	أبطأ من الخوارزميات المُستنيرة.
الأداء (Performance)	أفضل في حل مشكلات البحث واسع النطاق.	غير عملية لحل مشكلات البحث واسع النطاق.
الفعالية (Effectiveness)	تحقّقُ حلولا مُناسبةً بشكلٍ عام.	تُحقِق الحل الأمثل.

ومع ذلك، تُظهِر النتائج أن خوارزمية **البحث بأولوية الاتساع** (BFS) يمكنها العثور على الحل الأمثل بشكلٍ سريع بفحص عدد أقَل من الخلايا في الحالة غير الموزونة. يمكن مُعالجة ذلك بتوفير استدلال أكثر ذكاءً لخوارزمية **البحث بأولوية الأفضل** (A* search). والاستدلال الشهير في التطبيقات المُستنِدة إلى المسافة هو **مسافة مانهاتن** (Manhattan Distance)، وهي مجموع الفروقات المُطلقة بين إحداثِيَّي نقطتين مُعطاتين. يوضِّح الشكل أدناه مثالًا على كيفية حساب مسافة مانهاتن:

مسافة مانهاتن Manhattan Distance

```
Manhattan (A, B) = |x1-x2| + |y1-y2|
```



شكل 2.25: مسافة مانهاتن

يمكن تطبيق هذا بسهولة في صورة دالة البايثون كما يلي:

```
def manhattan_heuristic(candidate_cell:tuple,target_cell:tuple):

    x1,y1=candidate_cell
    x2,y2=target_cell
    return  abs(x1 - x2) + abs(y1 - y2)
```

يُستخدَم المقطع البرمجي التالي لاختبار إمكانية استخدام هذا الاستدلال الذكي لدعم astar_maze_solver( ) في البحث بشكل أسرع في كلٍ من الحالات الموزونة وغير الموزونة:

```
start_cell=(14,0)
target_cell=(5,10)

solution_astar_unw_mn, distance_astar_unw_mn, cell_visits_astar_unw_mn=astar_
maze_solver(start_cell,
            target_cell,
            big_maze,
            get_accessible_neighbors,
            manhattan_heuristic,
            verbose=False)

print('\nA* Search unweighted with the Manhattan heuristic.')
print('\nShortest Path:', solution_astar_unw_mn)
print('Cells on the Shortest Path:', len(solution_astar_unw_mn))
print('Shortest Path Distance:', distance_astar_unw_mn)
print('Number of cell visits:', cell_visits_astar_unw_mn)

horz_vert_w=1  # weight for horizontal and vertical moves
diag_w=3  # weight for diagonal moves

solution_astar_w_mn, distance_astar_w_mn, cell_visits_astar_w_mn=astar_maze_
solver(start_cell,
       target_cell,
       big_maze,
       partial(get_accessible_neighbors_weighted,
               horizontal_vertical_weight=horz_vert_w,
               diagonal_weight=diag_w),
       manhattan_heuristic,
       verbose=False)

print('\nA* Search weighted with the Manhattan heuristic.')
print('\nShortest Path:', solution_astar_w_mn)
print('Cells on the Shortest Path:', len(solution_astar_w_mn))
print('Shortest Path Distance:', distance_astar_w_mn)
print('Number of cell visits:', cell_visits_astar_w_mn)
```

```
A* Search unweighted with the Manhattan heuristic.

Shortest Path: [(14, 0), (13, 1), (12, 2), (11, 3), (10, 4), (9, 5), (8,
6), (8, 7), (9, 8), (9, 9), (9, 10), (9, 11), (9, 12), (8, 13), (7, 13),
(6, 13), (5, 12), (5, 11), (5, 10)]
Cells on the Shortest Path: 19
Shortest Path Distance: 18
Number of cell visits: 865

A* Search weighted with the Manhattan heuristic.

Shortest Path: [(14, 0), (14, 1), (13, 1), (12, 1), (12, 2), (12, 3), (12,
4), (12, 5), (12, 6), (12, 7), (11, 7), (11, 8), (10, 8), (9, 8), (9, 9),
(9, 10), (9, 11), (9, 12), (9, 13), (8, 13), (7, 13), (6, 13), (5, 13), (5,
12), (5, 11), (5, 10)]
Cells on the Shortest Path: 26
Shortest Path Distance: 25
Number of cell visits: 1033
```

تؤكد النتائج أن استدلال **مسافة مانهاتن** (Manhattan Distance) يمكن استخدامه لدعم خوارزمية **البحث بأولوية الأفضل** (A* search) في العثور على المسارات الأقصر المُحتمَلة بفحص أقل عدد من الخلايا في كلٍ من الحالات الموزونة وغير الموزونة. علمًا بأن استخدام هذا الاستدلال الأكثر ذكاءً يفحص عددًا أقل من الخلايا من ذلك المُستخدَم في خوارزمية **البحث بأولوية الاتساع** (BFS).

يُلخِّص الجدول 2.7 النتائج حول مُتغيرات الخوارزميّات المختلفة في المتاهة الكبيرة:

جدول 2.7: مقارنة بين أداء الخوارزميات

	خوارزمية البحث بأولوية الاتساع (BFS)	خوارزمية البحث بأولوية الأفضل (A* search) بالاستدلال الثابت	خوارزمية البحث بأولوية الأفضل (A* search) باستدلال مانهاتن
الموزونة	المسافة=30، وفَحصَت 1235	المسافة=25، وفَحصَت 1245	المسافة=25، وفَحصَت 1033
غير الموزونة	المسافة=18، وفَحصَت1237	المسافة=18، وفَحصَت 1272	المسافة=18، وفَحصَت865

يُوضِّح الجدول مزايا استخدام الطَّرائق الأكثر ذكاءً لحل المشكلات المُستنِدة إلى البحث مثل تلك المُوضَّحة بهذا الدرس:

- التَحوُّل من خوارزمية **البحث بأولوية الاتساع** (BFS) غير المُستنيرة إلى خوارزمية **البحث بأولوية الأفضل** (A* search) المُستنيرة حَقَّق نتائجَ أفضل، كما أتاح إمكانية حل المشكلات الأكثر تعقيدًا.
- يُمكن تحسين ذكاء خوارزميات البحث المُستنيرة باستخدام دوال الاستدلال الأفضل التي تسمح لها بالعثور على الحل الأمثل بشكلٍ أسرع.

تمرينات

1 اذكر تطبيقين لخوارزميات البحث.

2 حدّد الاختلافات بين خوارزميات البحث المُستنيرة وغير المُستنيرة، ثم اذكر مثالًا على كل خوارزمية.

3 اشرح بإيجاز كيف تعمل خوارزمية البحث بأولوية الأفضل (A* search).

4 عَدِّل المقطع البرمجي بتغيير الوزن القُطري (Diagonal Weight) من 3 إلى 1.5. ماذا تُلاحظ؟
هل يتغير المسار الأقصر في حالتي خوارزمية البحث بأولوية الاتساع (BFS) وخوارزمية البحث بأولوية الأفضل (A* search)؟

5 عَدِّل المقطع البرمجي بتبديل إحداثيات خلية البداية مع احداثيات الخلية المُستهدَفة. ماذا تُلاحظ؟
هل المسار هو نفسه كما كان سابقًا للحالات الموزونة من خوارزميتي البحث بأولوية الاتساع (BFS) والبحث بأولوية الأفضل (A* search)؟

المشروع

1

عدِّل المقطع البرمجي لخوارزمية **البحث بأولوية الاتساع** (BFS) وخوارزمية **البحث بأولوية الأفضل** (A* search) الموزونتين بتغيير الأوزان الأفقية والرأسية إلى 3 والأوزان القُطرية إلى 5، وكذلك عدِّل نقطة البداية إلى (2، 7).

2

ما المسار الجديد ذو المسافة الأقصر، وما عدد الخلايا التي فُحِصَت في الإصدارات غير الموزونة لخوارزميتي **البحث بأولوية الاتساع** (BFS) و**البحث بأولوية الأفضل** (A* search) باستخدام دالة الاستدلال الثابت؟ حَدِّد هذه القيم ودوِّن ملاحظاتك.

3

اتبع الخطوات نفسها للإصدارات الموزونة من خوارزميتي **البحث بأولوية الاتساع** (BFS) و**البحث بأولوية الأفضل** (A* search) باستخدام دالة الاستدلال الثابت.

4

كرِّر العملية للإصدارات غير الموزونة والموزونة من خوارزميتي **البحث بأولوية الاتساع** (BFS) و**البحث بأولوية الأفضل** (A* search) باستخدام دالة **استدلال مانهاتن** (Manhattan Heuristic).

ماذا تعلّمت

- استخدام الاستدعاء الذاتي لحل المشكلات.
- تطبيق خوارزميات اجتياز المُخطَّط المُتقدمة.
- تطبيق الأنظمة القائمة على القواعد البسيطة والمتقدمة.
- تصميم نموذج الذكاء الاصطناعي.
- قياس فعالية نموذج الذكاء الاصطناعي الذي صمَّمتَه.
- استخدام خوارزميات البحث لمحاكاة حلّ مشكلات الحياة الواقعية.

المصطلحات الرئيسة

Model Training	تدريب النموذج
Path Finding	إيجاد المسار
Recursion	الاستدعاء الذاتي
Rule-Based Systems	الأنظمة القائمة على القواعد
Scoring Function	دالة تسجيل النقاط
Search Algorithms	خوارزميات البحث
Uninformed Search	البحث غير المُستنير
Unweighted Graph	مُخطَّط غير موزون
Weighted Graph	مُخطَّط موزون

A* Search	البحث بأولوية الأفضل
Algorithm Performance	أداء الخوارزمية
Breadth-First Search (BFS)	البحث بأولوية الاتساع
Confusion Matrix	مصفوفة الدقة
Depth-First Search (DFS)	البحث بأولوية العمق
Heuristic Function	دالة استدلالية
Informed Search	البحث المُستنير
Knowledge Base	قاعدة المعرفة
Maze Solving	حل المتاهات

3. معالجة اللغات الطبيعية

سيتعلّم الطالب في هذه الوحدة عملية تدريب شاملة لنموذج التعلُّم الموجَّه والتعلُّم غير الموجَّه لفهم المعنى الكامن في أجزاء النصوص. وكذلك سيتعلّم كيفية استخدام تعلُّم الآلة (Machine Learning - ML) في دعم التطبيقات ذات الصلة بمعالجة اللغات الطبيعية (Natural Language Processing - NLP).

أهداف التعلُّم

بنهاية هذه الوحدة سيكون الطالب قادرًا على أن:

- يُعرِّف التعلُّم الموجَّه.
- يُدرِّب نموذج التعلُّم الموجَّه على فهم النص.
- يُعرِّف التعلُّم غير الموجَّه.
- يُدرِّب نموذج التعلُّم غير الموجَّه على فهم النص.
- يُنشئ روبوت دردشة بسيط.
- يُنتج النصوص باستخدام تقنيات توليد اللغات الطبيعية (Natural Language Generation - NLG).

الأدوات

- مفكّرة جوبيتر (Jupyter Notebook)

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الأول

التعلُّم الموجَّه

استخدام التعلُّم الموجَّه لفهم النصوص

Using Supervised Learning to Understand Text

معالجة اللغات الطبيعية (Natural Language Processing – NLP) هي إحدى مجالات **الذكاء الاصطناعي** (Artificial Intelligence – AI) التي تركّز على تمكين أجهزة الحاسب لتصبح قادرة على فهم اللغات البشريّة، وتفسيرها، وإنتاجها. حيث تُعنى معالجة اللغات الطبيعية بعدد من المهام، مثل: تصنيف النصوص، وتحليل المشاعر، والترجمة الآلية، والإجابة على الأسئلة. سيركز هذا الدرس بشكل خاص على كيفية استخدام التعلُّم الموجَّه الذي يُعدُّ أحد الأنواع الرئيسة **لتعلُّم الآلة** (Machine Learning – ML) في تحقيق الفهم والتنبؤ التلقائي لخصائص النصوص.

لقد تعلّمت في الوحدة الأولى أن الذكاء الاصطناعي هو مصطلح يشملُ كلًّا من تعلُّم الآلة والتعلُّم العميق، كما يتضح في الشكل 3.1، فالذكاء الاصطناعي هو ذلك المجال الواسع من علوم الحاسب الذي يُعنى بابتكار آلات ذكية، بينما تعلُّم الآلة هو أحد فروع الذكاء الاصطناعي الذي يركّز على تصميم الخوارزميات وبِناء النماذج التي تُمكِّن الآلة من التعلُّم من البيانات دون الحاجة إلى برمجتها بشكل صريح.

التعلُّم العميق (Deep Learning):
التعلُّم العميق هو أحد أنواع تعلُّم الآلة الذي يستخدِم الشبكات العصبية العميقة للتعلُّم تلقائيًا من مجموعات كبيرة من البيانات، فهو يسمح لأجهزة الحاسب بالتعرّف على الأنماط واتخاذ القرارات بطريقة تحاكي الإنسان، عبر تصميم نماذج مُعقدة من البيانات.

شكل 3.1: فروع الذكاء الاصطناعي

تعلُّم الآلة Machine Learning

تعلُّم الآلة هو أحد فروع الذكاء الاصطناعي المعني بتطوير الخوارزميات التي تُمكِّن أجهزة الحاسب من التعلُّم من البيانات المُدخَلة، بدلًا من اتباع التعليمات البرمجية الصريحة، فهو يعمل على تدريب نماذج الحاسب للتعرُّف على الأنماط والقيام بالتنبؤات وفقًا للبيانات المُدخَلة مما يسمح للنموذج بتحسين الدقة مع مرور الوقت، وكذلك يتيح للآلة أداء مهام متعددة، مثل: التصنيف، والانحدار، والتجميع، وتقديم التوصيات دون الحاجة إلى برمجة الآلة بشكل صريح للقيام بكل مُهِمَّة على حدة. يمكن تصنيف تعلُّم الآلة إلى ثلاثة أنواع رئيسة:

التعلُّم الموجَّه (Supervised Learning) هو نوع من تعلُّم الآلة تتعلّم فيه الخوارزمية من بيانات تدريب **مُعنوَنة** (Labelled) بهدف القيام بالتنبؤات حول بيانات جديدة غير موجودة في مجموعة التدريب أو الاختبار كما هو موضَّح في الشكل 3.2، ومن الأمثلة عليه:

- **تصنيف الصور** (Image Classification)، مثل: التعرف على الكائنات في الصور.
- **كشف الاحتيال** (Fraud Detection)، مثل: تحديد المُعامَلات المالية المشبوهة.
- **تصفية البريد الإلكتروني العشوائي** (Spam Filtering)، مثل: تحديد رسائل البريد الإلكتروني غير المرغوب فيها.

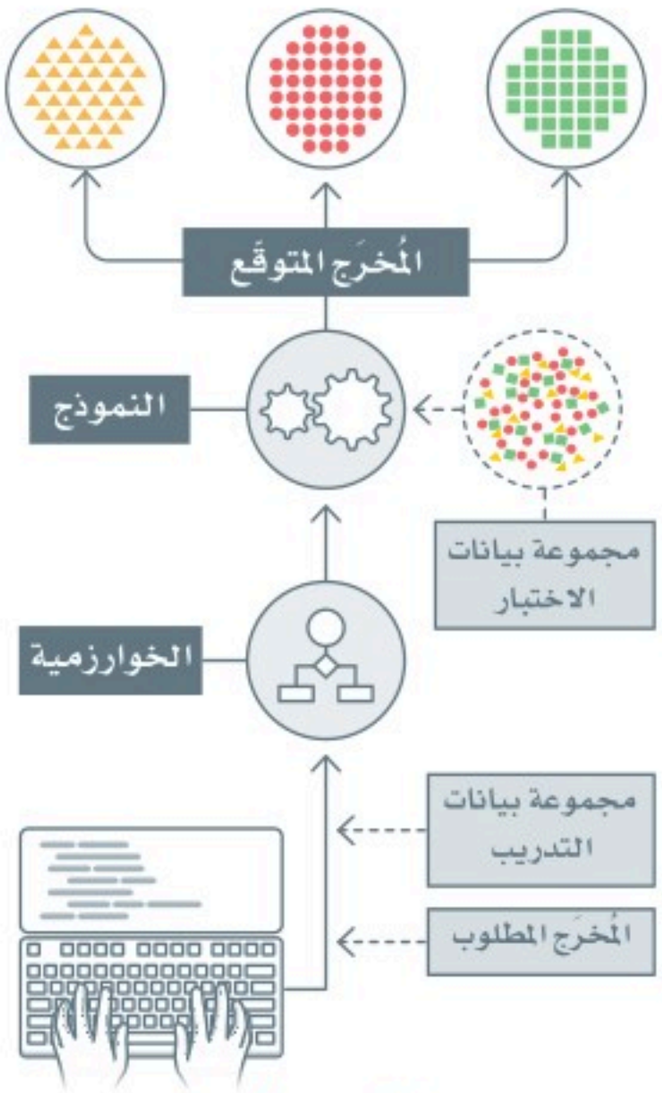
التعلُّم غير الموجَّه (Unsupervised Learning) هو نوع من تعلُّم الآلة تعمل فيه الخوارزمية بموجب بيانات **غير مُعنوَنة** (Unlabeled) في محاولة لإيجاد الأنماط والعلاقات بين البيانات، ومن الأمثلة عليه:

- **الكشف عن الاختلاف** (Anomaly Detection)، مثل: تحديد الأنماط غير العادية في البيانات.
- **التجميع** (Clustering)، مثل: تجميع البيانات ذات الخصائص المتشابهة.
- **تقليص الأبعاد** (Dimensionality Reduction)، مثل: اختيار الأبعاد المُستخدَمة للحدِّ من تعقيد البيانات.

التعلُّم المعزَّز (Reinforcement Learning) هو نوع من تعلُّم الآلة تتفاعل فيه الآلة مع البيئة المحيطة وتتعلّم عبر المحاولة والخطأ أو تلقّي المكافأة والعقاب، ومن الأمثلة عليه:

- لعب الألعاب، مثل: لعبة الشطرنج أو لعبة **قو** (GO).
- الروبوتية، مثل: تعليم الروبوت كيف يتنقل في البيئة المحيطة به.
- تخصيص الموارد، مثل: تحسين استخدام الموارد في شبكة ما.

جدول 3.1 يلخص مزايا أنواع تعلُّم الآلة وعيوبها.



شكل 3.2: تمثيل التعلُّم الموجَّه

جدول 3.1: مزايا أنواع تعلُّم الآلة، وعيوبها

المزايا	العيوب
التعلُّم الموجَّه	
• أثبت كفاءة وفعالية كبيرة ويُستخدم على نطاق واسع. • سهل الفهم والتطبيق. • يُمكنه التعامل مع البيانات الخطية وغير الخطية على حد سواء.	• يتطلب بيانات مُعنوَنة، والتي قد تكون مرتفعة التكلفة. • يقتصر استخدامه على المُهِمَّة التي تم تدريبه عليها، وقد لا يمكنه إعطاء التنبؤ الصحيح للبيانات الجديدة. • يصعب تكيفه مع المشكلات الأخرى في حالات النماذج المُعقدة جدًا.
التعلُّم غير الموجَّه	
• لا يتطلب بيانات مُعنوَنة، مما يجعله أكثر مرونة. • يُمكنه اكتشاف الأنماط الخفية في البيانات. • يُمكنه التعامل مع البيانات الضخمة والمُعقدة.	• أصعب من التعلُّم الموجَّه من حيث الفهم والتفسير. • يقتصر على التحليل الاستكشافي، وقد لا يناسب عمليات صُنع القرار. • يصعب تكيفه مع المشكلات الأخرى في حالات النماذج المُعقدة جدًا.
التعلُّم المعزَّز	
• يتّسم بالمرونة، ويُمكنه التعامل مع البيئات المُعقدة والمتغيرة باستمرار. • يمكنه التعلُّم من التجارب السابقة وتحسين الكفاءة مع مرور الوقت. • يتناسب مع عمليات صُنع القرار مثل لعب الألعاب والروبوتية.	• أكثر تعقيدًا من التعلُّم الموجَّه وغير الموجَّه. • صعوبة تصميم نُظم مكافآت تُحدد السلوك المطلوب بشكل دقيق. • قد يتطلب مجموعات كبيرة من بيانات التدريب والموارد الحسابية.

التعلُّم الموجَّه Supervised Learning

التعلُّم الموجَّه هو أحد أنواع تعلُّم الآلة الذي يعتمد على استخدام البيانات المُعنوَنة لتدريب الخوارزميات للقيام بالتنبؤات. يتم تدريب الخوارزمية على مجموعة من البيانات المُعنوَنة ثم اختبارها على مجموعة بيانات جديدة لم تكن جزءًا من بيانات التدريب. يُستخدَم التعلُّم الموجَّه عادةً في معالجة اللغات الطبيعية للقيام بمهام مثل: تصنيف النصوص، وتحليل المشاعر، **والتعرّف على الكيانات المسماة** (Named Entity Recognition – NER). في هذه المهام يتم تدريب الخوارزمية على مجموعة من البيانات المُعنوَنة، حيث يتم إدراج كل مثال تحت عنوان التصنيف المناسب أو المشاعر المناسبة. يُطلَق على عملية التعلُّم الموجَّه اسم **الانحدار** (Regression) عندما تكون القيم التي تتنبأ بها الآلة رقميّة، بينما يطلق عليها اسم **التصنيف** (Classification) عندما تكون القيم متقطّعه.

التعلُّم الموجَّه (Supervised Learning):
ستستخدِم في التعلُّم الموجَّه مجموعات البيانات المُعنوَنة والمُنظمة بشكل يدوي لتدريب خوارزميات الحاسب على التنبؤ بالقيم الجديدة.

الانحدار

على سبيل المثال، قد يُستخدم الانحدار في التنبؤ بسعر بيع المنزل وفقًا لمساحته، وموقعه، وعدد غرف النوم فيه. كما يمكن استخدامه في التنبؤ بحجم الطلب على أحد المنتجات استنادًا إلى بيانات المبيعات التاريخية وحجم الإنفاق الإعلاني. وفي مجال معالجة اللغات الطبيعية، يَستخدِم الانحدار النصوص المُدخَلة المتوفرة للتنبؤ بتقييم الجمهور للفيلم أو مدى التفاعل مع المنشورات الخاصة به على وسائل التواصل الاجتماعي.

التصنيف

من ناحية أخرى، يُستخدم التصنيف في التطبيقات مثل: تشخيص الحالات الطبية وفقًا للأعراض ونتائج الفحوصات. وعندما يتعلق الأمر بفهم النصوص، يمكن استخدام التعلُّم الموجَّه في تصنيف النصوص المُدخَلة إلى فئات أو عناوين أو التنبؤ بها بناءً على الكلمات أو العبارات الموجودة في المُستنَد. على سبيل المثال، يمكن تدريب نموذج التعلُّم الموجَّه لتصنيف رسائل البريد الإلكتروني إلى رسائل مزعجة أو غير مزعجة وفقًا للكلمات أو العبارات المُستخدَمة في رسالة البريد الإلكتروني. ويُعدّ تصنيف المشاعر أحد التطبيقات الشهيرة كذلك، حيث يمكن التنبؤ بالانطباع العام حول مستند ما سواء كان سلبيًّا أم إيجابيًّا. وسَيُستخدم هذا التطبيق كمثال عملي في هذه الوحدة، لشرح كل خطوات عملية بِناء واستخدام نموذج التعلُّم الموجَّه بشكل شامل من بداية رحلة التعلُّم حتى نهايتها.

في هذه الوحدة ستَستخدم مجموعة بيانات من مراجعات الأفلام على موقع IMDb.com الشهير. ستجد البيانات مُقسّمه إلى مجموعتين؛ الأولى ستُستخدم لتدريب النموذج، والثانية لاختبار أداء النموذج. في البداية لابد أن تُحَمّل البيانات إلى DataFrame، لِذا عليك استخدام مكتبة **باندا بايثون** (Pandas Python) والتي استخدمتها سابقًا. مكتبة باندا هي إحدى الأدوات الشهيرة التي تُستخدم للتعامل مع جداول البيانات. التعليمات البرمجية التالية ستقوم باستيراد المكتبة إلى البرنامج، ثم تحميل مجموعتي البيانات:

```
%%capture  # capture is used to suppress the installation output.

# install the pandas library, if it is missing.
!pip install pandas
import pandas as pd
```

مكتبة باندا هي مكتبة شهيرة تُستخدم لقراءة ومعالجة البيانات الشبيهة بجداول البيانات.

```
# load the train and testing data.
imdb_train_reviews=pd.read_csv('imdb_data/imdb_train.csv')
imdb_test_reviews=pd.read_csv('imdb_data/imdb_test.csv')

imdb_train_reviews
```

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1
...	...	...
39995	"Western Union" is something of a forgotten cl...	1
39996	This movie is an incredible piece of work. It ...	1
39997	My wife and I watched this movie because we pl...	0
39998	When I first watched Flatliners, I was amazed....	1
39999	Why would this film be so good, but only gross...	1

40000 rows × 2 columns

وكما يتضح في الشكل 3.3، فإن مجموعة بيانات DataFrame تحتوي على عمودين:

- نصّ التقييم.
- القيم (الصَّنف).

تقييم إيجابي

تقييم سلبي

القيمة 0 تمثل تقييمًا سلبيًّا

بينما القيمة 1 تمثل تقييمًا إيجابيًّا.

شكل 3.3: مجموعة بيانات التدريب المُعنوَنة

الخطوة التالية هي إسناد أعمدة النص والقيم إلى متغيرات مستقلة في أمثلة التدريب والاختبار المُمثّلة كمجموعة بيانات DataFrame كما يلي:

```
# extract the text from the 'text' column for both training and testing.
X_train_text=imdb_train_reviews['text']
X_test_text=imdb_test_reviews['text']

# extract the labels from the 'label' column for both training and testing.
Y_train=imdb_train_reviews['label']
Y_test=imdb_test_reviews['label']

X_train_text # training data in text format
```

يُستخدم الرمزان X وY عادةً في التعلُّم الموجَّه فيعبّر X عن البيانات المدخلة للتنبؤ، وY عن القيم المستهدفة.

```
0        I grew up (b. 1965) watching and loving the Th...
1        When I put this movie in my DVD player, and sa...
2        Why do people who do not know what a particula...
3        Even though I have great interest in Biblical ...
4        Im a die hard Dads Army fan and nothing will e...
                              ...
39995    "Western Union" is something of a forgotten cl...
39996    This movie is an incredible piece of work. It ...
39997    My wife and I watched this movie because we pl...
39998    When I first watched Flatliners, I was amazed....
39999    Why would this film be so good, but only gross...
Name: text, Length: 40000, dtype: object
```

شكل 3.4: صورة من أمثلة التدريب (X_train_text) من مجموعة بيانات DataFrame

تجهيز البيانات والمعالجة المُسبقة Data Preparation and Pre-Processing

على الرغم من أن تنسيق النص الأولي كما في الشكل 3.4 بديهي للقارئ البشري، إلا أنَّ خوارزميات التعلُّم الموجَّه لا تستطيع التعامل معه بصورته الحالية. فبدلًا من ذلك، تحتاج الخوارزميات إلى تحويل هذه المُستنَدات إلى تنسيق **متَّجَه رقمي** (Numeric Vector). فيما يُعرف بعملية **البرمجة الاتجاهية** (Vectorization). ويمكن تطبيق عملية البرمجة الاتجاهية بعدة طرائق مختلفة، وتتميز بأن لها تأثيرًا إيجابيًّا كبيرًا على أداء النموذج المُدرّب.

مكتبة سكليرن Sklearn Library

سيتم بناء النموذج الموجَّه باستخدام مكتبة سكليرن وتُعرف كذلك باسم مكتبة **سايكيت ليرن** (Scikit-Learn)، وهي مكتبة شهيرة في البايثون تختص بتعلُّم الآلة. توفر المكتبة مجموعة من الأدوات والخوارزميات لأداء مهام متعددة، مثل: التصنيف، والانحدار، والتجميع، وتقليص الأبعاد. إحدى الأدوات المفيدة في مكتبة سكليرن هي أداة تُسمى CountVectorizer، ويمكن استخدامها في تهيئة عملية المعالجة وتمثيل البيانات النصية بالمتَّجَهات.

أداة CountVectorizer

تُستخدم أداة CountVectorizer في تحويل مجموعة من المُستنَدات النصية إلى مصفوفة من رموز متعددة، حيث يمثّل كلّ صفّ مستندًا وكل عمود يمثل رمزًا خاصًا. قد تكون الرموز كلمات فردية أو عبارات أو بُنيات أكثر تعقيدًا تقوم بالتقاط الأنماط المتعددة من البيانات النصية الأساسية. تُشير المُدخَلات في المصفوفة إلى عدد مرات ظهور الرمز في كل مستند. ويُعرف ذلك أيضًا باسم تمثيل **حقيبة الكلمات** (BoW) "bag-of-words"، حيث يتجاهل ترتيب الكلمات في النص مع المحافظة على تكرارها فيه. على الرغم من أن تمثيل حقيبة الكلمات هو تبسيط شديد للغة البشرية، إلا أنه يحقق نتائج تنافسية للغاية عند التطبيق العملي.

البرمجة الاتجاهية (Vectorization):

البرمجة الاتجاهية هي عملية تحويل السلاسل النصية المكوَّنة من الكلمات أو العبارات (النص) إلى متَّجَه متجانس من الأرقام الحقيقية يُستخدم لترميز خصائص النص باستخدام تنسيق تفهمه خوارزميات تعلُّم الآلة.

شكل 3.5: تمثيل حقيبة الكلمات (bag-of-words)

يستخدِم المقطع البرمجي التالي أداة CountVectorizer لتمثيل مجموعة بيانات التدريب IMDb بالمتَّجَهات:

```
from sklearn.feature_extraction.text import CountVectorizer

# the min_df parameter is used to ignore terms that appear in less than 10 reviews.
vectorizer_v1 = CountVectorizer(min_df=10)

vectorizer_v1.fit(X_train_text) # fit the vectorizer on the training data.
# use the fitted vectorizer to vectorize the data.
X_train_v1 = vectorizer_v1.transform(X_train_text)

X_train_v1
```

```
<40000x23392 sparse matrix of type '<class 'numpy.int64'>'
        with 5301561 stored elements in Compressed Sparse Row format>
```

```
# expand the sparse data into a sparse matrix format, where each column represents a different word.
X_train_v1_dense=pd.DataFrame(X_train_v1.toarray(),
                              columns=vectorizer_v1.get_feature_names_out())
X_train_v1_dense
```

	00	000	007	01	02	04	05	06	07	08	...	zoo	zoom	zooming	zooms	zorro	zu	zucco	zucker	zulu	über
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
39995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
39996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
39997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
39998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
39999	0	2	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

40000 rows × 23392 columns

شكل 3.6: تمثيل مجموعة بيانات التدريب بالمتَّجهات

يُعبِّر هذا التنسيق **الكثيف** (Dense) للمصفوفة عن 40,000 تقييم ومراجعة في بيانات التدريب. تحتوي المصفوفة على عمود لكل كلمة تظهر في 10 مراجعات على الأقل (مُنفذة بواسطة المتغير min_df). كما يتضح بالأعلى، ينتج عن ذلك 23,392 عمودًا، مرتبة في ترتيب أبجدي رقمي. يُعبِّر مُدخَل المصفوفة في الموضع [i،j] عن عدد المرات التي تظهر فيها كلمة j في المراجعة i. وعلى الرغم من إمكانيّة استخدام هذه المصفوفة مباشرةً من قِبَل خوارزمية التعلُّم الموجَّه، إلا أنها غير فعّالة من حيث استخدام الذاكرة. والسبب في ذلك أن الغالبية العظمى من المُدخَلات في هذه المصفوفة تساوي 0. وهذا يحدث لأن نسبة ضئيلة جدًا فقط من بين 23,392 كلمة محتملة ستظهر فعليًا في كل مراجعة. ولمعالجة هذا القصور، تُخزِّن أداة CountVectorizer البيانات الممثلة بالمتَّجهات في مصفوفة متباعدة، حيث تحتفظ فقط بالمُدخَلات غير الصفرية في كل عمود. يستخدِم المقطع البرمجي بالأسفل الدالة ()getsizeof التي تحدِّد حجم الكائنات في لغة **البايثون** (Python) **بالبايت** (Bytes) لتوضيح مدى التوفير في الذاكرة عند استخدام المصفوفة المتباعدة لبيانات IMDb:

```
from sys import getsizeof
print('\nMegaBytes of RAM memory used by the raw text format:',
       getsizeof(X_train_text)/1000000)
print('\nMegaBytes of RAM memory used by the dense matrix format:',
       getsizeof(X_train_v1_dense)/1000000)
print('\nMegaBytes of RAM memory used by the sparse format:',
       getsizeof(X_train_v1)/1000000)
```

```
MegaBytes of RAM memory used by the raw text format: 54.864133

MegaBytes of RAM memory used by the dense matrix format: 7485.440144

MegaBytes of RAM memory used by the sparse format: 4.8e-05
```

وبحسب المتوقّع تحتاج المصفوفة المتباعدة إلى ذاكرة أقل بكثير وتحديدًا 0.000048 ميجابايت، بينما تشغل المصفوفة الكثيفة 7 جيجابايت، كما أنّ هذه المصفوفة لن تُستخدَم مرة أخرى وبالتالي يمكن حذفها لتوفير هذا الحجم الكبير من الذاكرة:

```
# delete the dense matrix.
del X_train_v1_dense
```

بِناء خط أنابيب التنبؤ

Building a Prediction Pipeline

الآن بعد أن تمكَّنت من تمثيل بيانات التدريب بالمتَّجَهات فإن الخطوة التالية هي بِناء خط أنابيب التنبؤ الأول. وللقيام بذلك، ستستخدِم نوعًا من المُصنِّفات يسمى **مُصنِّف بايز الساذج** (Naive Bayes Classifier)، حيث يَستخدِم هذا المصنِّف احتمالات الكلمات أو العبارات المحددة الواردة في النَّص للتنبؤ باحتمال انتمائه إلى تصنيف محدد. جاءت كلمة **الساذج** (Naive) في اسم المُصنِّف من افتراض أن وجود كلمة بعينها في النَّص مستقل عن وجود أي كلمة أخرى. وهذا افتراض قوي، ولكنه يسمح بتدريب الخوارزمية بسرعة وبفعالية كبيرة.

المُصنِّف (Classifier):
المُصنِّف في تعلُّم الآلة هو نموذج يُستخدم لتمييز نقاط البيانات في فئات أو تصنيفات مختلفة. الهدف من المُصنِّف هو التعلُّم من بيانات التدريب المُعنوَنة، ومن ثَمَّ القيام بالتنبؤات حول قيم التصنيف لبيانات جديدة.

يستخدِم المقطع البرمجي التالي تطبيق **مُصنِّف بايز الساذج** (Multinomial NB) من **مكتبة سكليرن** (Sklearn Library) لتدريب نموذج التعلُّم الموجَّه على بيانات التدريب IMDb بالمتَّجَهات:

```
from sklearn.naive_bayes import MultinomialNB

model_v1=MultinomialNB() # a Naive Bayes Classifier

model_v1.fit(X_train_v1, Y_train) # fit the classifier on the vectorized training data.

from sklearn.pipeline import make_pipeline

# create a prediction pipeline: first vectorize using vectorizer_v1, then use model_v1 to predict.
prediction_pipeline_v1 = make_pipeline(vectorizer_v1, model_v1)
```

على سبيل المثال، سيُنتج هذا المقطع البرمجي مصفوفة نتائج يرمز فيها الرقم 1 للتقييم الإيجابي و0 للتقييم السلبي:

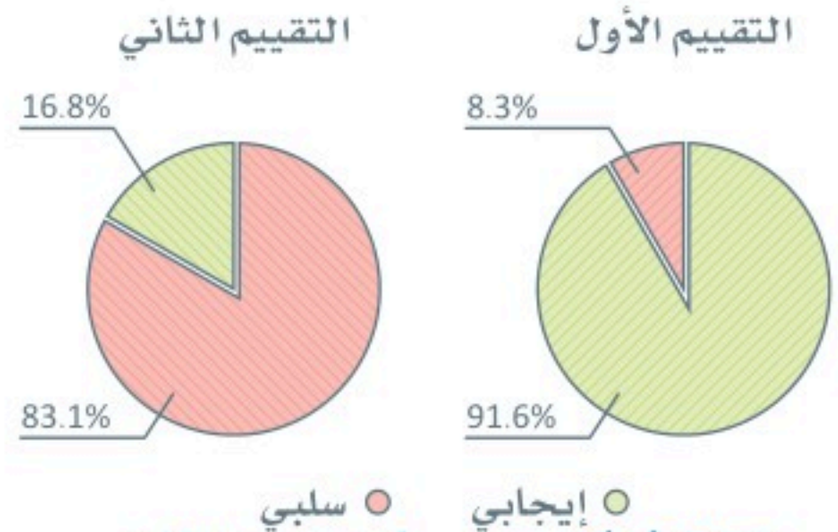
```
prediction_pipeline_v1.predict(['One of the best movies of the year. Excellent
                                 cast and very interesting plot.',
                                'I was very disappointed with his film. I
                                 lost all interest after 30 minutes' ])
```

```
array([1, 0], dtype=int64)
```

يتنبأ خط الأنابيب بشكل صحيح بالقيمة الإيجابية للتقييم الأول والقيمة السلبية للتقييم الثاني. يُمكن استخدام الدالة المُضمّنة predict_proba( ) لتحديد جميع الاحتمالات التي يقوم خط الأنابيب بتخصيصها لكل واحدة من القيمتين المحتملتين. العنصر الأول هو احتمال تعيين 0 والعنصر الثاني هو احتمال تعيين 1:

```
prediction_pipeline_v1.predict_proba(['One of the best movies of the year. Ex
                                      cellent cast and very interesting plot.',
                                      'I was very disappointed with his film.
                                       I lost all interest after 30 minutes' ])
```

```
array([[0.08310769, 0.91689231],
       [0.83173475, 0.16826525]])
```



النموذج يؤكد بنسبة 8.3% أن التقييم الأول سلبي، بينما يؤكد بنسبة 91.7% أنه إيجابي. وبالمثل، يؤكد النموذج بنسبة 83.1% أن التقييم الثاني سلبي، بينما يؤكد بنسبة 16.8% أنه إيجابي.

شكل 3.7: مُخطَّطان دائريان يوضِّحان النسب المئوية للتقييمين

الخطوة التالية هي اختبار دقة خط الأنابيب الجديد في تصنيف التقييمات في مجموعة بيانات اختبار IMDb. المُخرَج هو مصفوفة تشمل جميع قيم نتائج تصنيف التقييمات الواردة في بيانات الاختبار:

```
# use the pipeline to predict the labels of the testing data.
predictions_v1 = prediction_pipeline_v1.predict(X_test_text) # vectorize the text
data, then predict.

predictions_v1
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

توفر لغة البايثون العديد من الأدوات لتحليل وتصوير نتائج خطوط أنابيب التصنيف. تشمل الأمثلة دالة accuracy_score( ) من مكتبة سكليرن وتمثيل **مصفوفة الدقة** (Confusion Matrix) من مكتبة **سايكيت بلوت** (Scikit-Plot)، وهناك مقاييس تقييم أخرى مثل: الدقة، والاستدعاء، والنوعية، والحساسية، ومقياس درجة F1، وفقًا لحالة الاستخدام التي يمكن حسابها من مصفوفة الدقة. المُخرَج التالي هو تقريب دقيق لدرجة التنبؤ:

```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test, predictions_v1) # get the achieved accuracy.
```

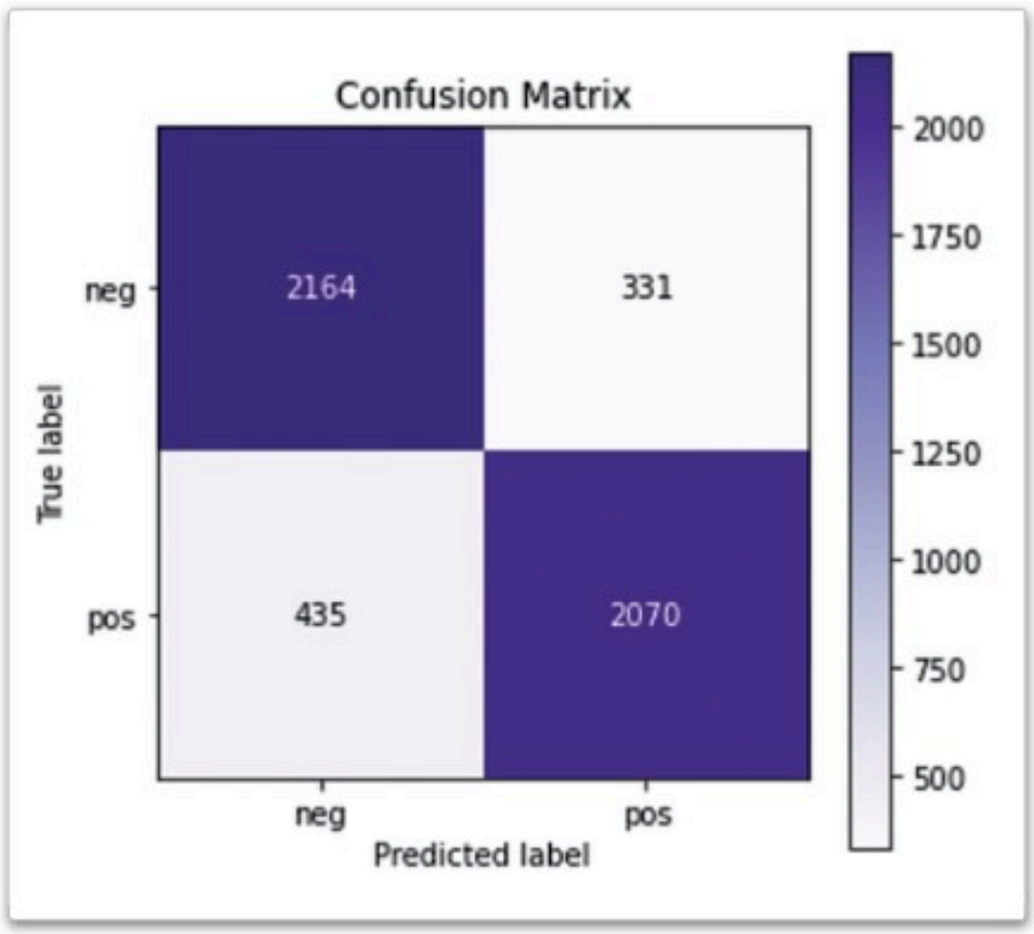
```
0.8468
```

```
%%capture
!pip install scikit-plot; # install the scikit-plot library, if it is missing.
import scikitplot; # import the library

class_names=['neg','pos'] # pick intuitive names for the 0 and 1 labels.

# plot the confusion matrix.
scikitplot.metrics.plot_confusion_matrix(
                                [class_names[i] for i in Y_test],
                                [class_names[i] for i in predictions_v1],
                                 title="Confusion Matrix", # title to use
                                 cmap="Purples", # color palette to use
                                 figsize=(5,5) # figure size
                                 );
```

القيم الحقيقية.

القيم المُتوقَّعة.

تحتوي مصفوفة الدقة على عدد التصنيفات الحقيقية مقابل المُتوقَّعة. في مُهِمَّة التصنيف الثنائية (مثل: مسألة احتواء قيمتين، الموجودة في مُهِمَّة IMDb)، ستحتوي مصفوفة الدقة على أربع خلايا:



- **التنبؤات السالبة الصحيحة (أعلى اليسار):** عدد المرات التي تنبأ فيها المُصنِّف بالحالات السالبة بشكل صحيح.
- **التنبؤات السالبة الخاطئة (أعلى اليمين):** عدد المرات التي تنبأ فيها المُصنِّف بالحالات السالبة بشكل خاطئ.
- **التنبؤات الموجبة الخاطئة (أسفل اليسار):** عدد المرات التي تنبأ فيها المُصنِّف بالحالات الموجبة بشكل خاطئ.
- **التنبؤات الموجبة الصحيحة (أسفل اليمين):** عدد المرات التي تنبأ فيها المُصنِّف بالحالات الموجبة بشكل صحيح.

شكل 3.8: نتائج مصفوفة الدقة بتطبيق مصنَّف بايز الساذج على بيانات الاختبار باستخدام مجموعة بيانات IMDb

تُظهر النتائج أنه على الرغم من أن خط الأنابيب الأول يحقق دقة تنافسية تصل إلى 84.68%، إلا أنه لا يزال يُخطئ في تصنيف مئات التقييمات. فهناك 331 تنبّؤًا غير صحيح في الربع الأيمن العلوي و435 تنبّؤًا غير صحيح في الربع الأيسر السفلي، بإجمالي 766 تنبّؤًا غير صحيح. الخطوة الأولى نحو تحسين الأداء هي دراسة سلوك خط أنابيب التنبؤ، لمعرفة كيف يقوم بمعالجة النصّ وفهمه.

الدقة (Accuracy):

الدقة هي نسبة التنبؤات الصحيحة إلى إجمالي عدد التنبؤات.

$$\text{الدقة} = \frac{(\text{التنبؤات الموجبة الصحيحة} + \text{التنبؤات السالبة الصحيحة})}{(\text{التنبؤات الموجبة الصحيحة} + \text{التنبؤات السالبة الصحيحة} + \text{التنبؤات الموجبة الخاطئة} + \text{التنبؤات السالبة الخاطئة})}$$

شرح مُتنبِّئات الصندوق الأسود Explaining Black-Box Predictors

يستخدِم مصنَّف بايز الساذج الصيغ الرياضية البسيطة لتجميع احتمالات آلاف الكلمات وتقديم تنبؤاتها. وبالرغم من بساطة النموذج، إلا أنه لا يزال غير قادر على تقديم شرح بسيط ومباشر لكيفية قيام النموذج بتوقُّع القيمة الموجبة أو السالبة لجزء محدد من النص. قارِن ذلك مع مُصنِّفات شجرة القرار الأكثر وضوحًا، حيث يتم تمثيل القواعد التي تعلمها النموذج في الهيكل الشجري، مما يُسهِّل على الأشخاص فهم كيف يقوم المُصنِّف بالتنبؤات. يتيح هيكل الشجرة كذلك الحصول على تصور مرئي للقرارات المُتخذَّة في كل فرع، ممّا يكون مفيدًا في فهم العلاقات بين الخصائص المُدخَلة والمتغير المستهدف.

الافتقار إلى قدرة التفسير تمثل تحديًا كبيرًا في الخوارزميات الأكثر تعقيدًا، كتلك المُستنِدة إلى التجميعات مثل: توليفات من الخوارزميات المتعددة أو الشبكات العصبية. فبدون القدرة على التفسير، تتقلص خوارزميات التعلُّم الموجَّه إلى متنبئات الصندوق الأسود: على الرغم من أنها تفهم النص بشكل كافٍ للتنبؤ بالقيم، إلا أنها لا تزال غير قادرة على تفسير كيف تقوم باتخاذ القرار. أجريت العديد من الأبحاث للتغلب على هذه التحديات بتصميم وسائل قادرة على التفسير تستطيع فهم نماذج الصندوق الأسود. واحدة من الوسائل الأكثر شهرة هي **النموذج المحايد المحلي القابل للتفسير والشرح** (Local Interpretable Model-Agnostic Explanations – LIME).

النموذج المحايد المحلي القابل للتفسير والشرح
Local Interpretable Model-Agnostic Explanations - LIME

النموذج المحايد المحلي القابل للتفسير والشرح (LIME) هو طريقة لتفسير التنبؤات التي قامت بها نماذج الصندوق الأسود. وذلك من خلال النظر في نقطة بيانات واحدة في وقت محدد، وإجراء تغييرات بسيطة عليها لمعرفة كيف يؤثر ذلك على قدرة تنبؤ النموذج، ثم تُستخدم هذه المعلومات لتدريب نموذج مفهوم وبسيط مثل الانحدار الخطي على تفسير هذه التنبؤات. بالنسبة للبيانات النصية، يقوم النموذج المحايد المحلي القابل للتفسير والشرح بالتعرّف على الكلمات أو العبارات التي لها الأثر الأكبر على القيام بالتنبؤات.

وفيما يلي، تطبيق بلغة البايثون يوضِّح ذلك:

```
%%capture

!pip install lime # install the lime library, if it is missing
from lime.lime_text import LimeTextExplainer

# create a local explainer for explaining individual predictions
explainer_v1 = LimeTextExplainer(class_names=class_names)

# an example of an obviously negative review
easy_example='This movie was horrible. The actors were terrible and the plot
was very boring.'

# use the prediction pipeline to get the prediction probabilities for this example
print(prediction_pipeline_v1.predict_proba([easy_example]))
```

```
[[0.99874831 0.00125169]]
```

كما هو مُتوقَّع، يقدم نموذج التنبؤ تنبؤًا سلبيًا مؤكدًا بدرجة كبيرة في هذا المثال البسيط.

```
# explain the prediction for this example.
exp = explainer_v1.explain_instance(easy_example.lower(),
                                    prediction_pipeline_v1.predict_proba,
                                    num_features=10)
# print the words with the strongest influence on the prediction.
exp.as_list()
```

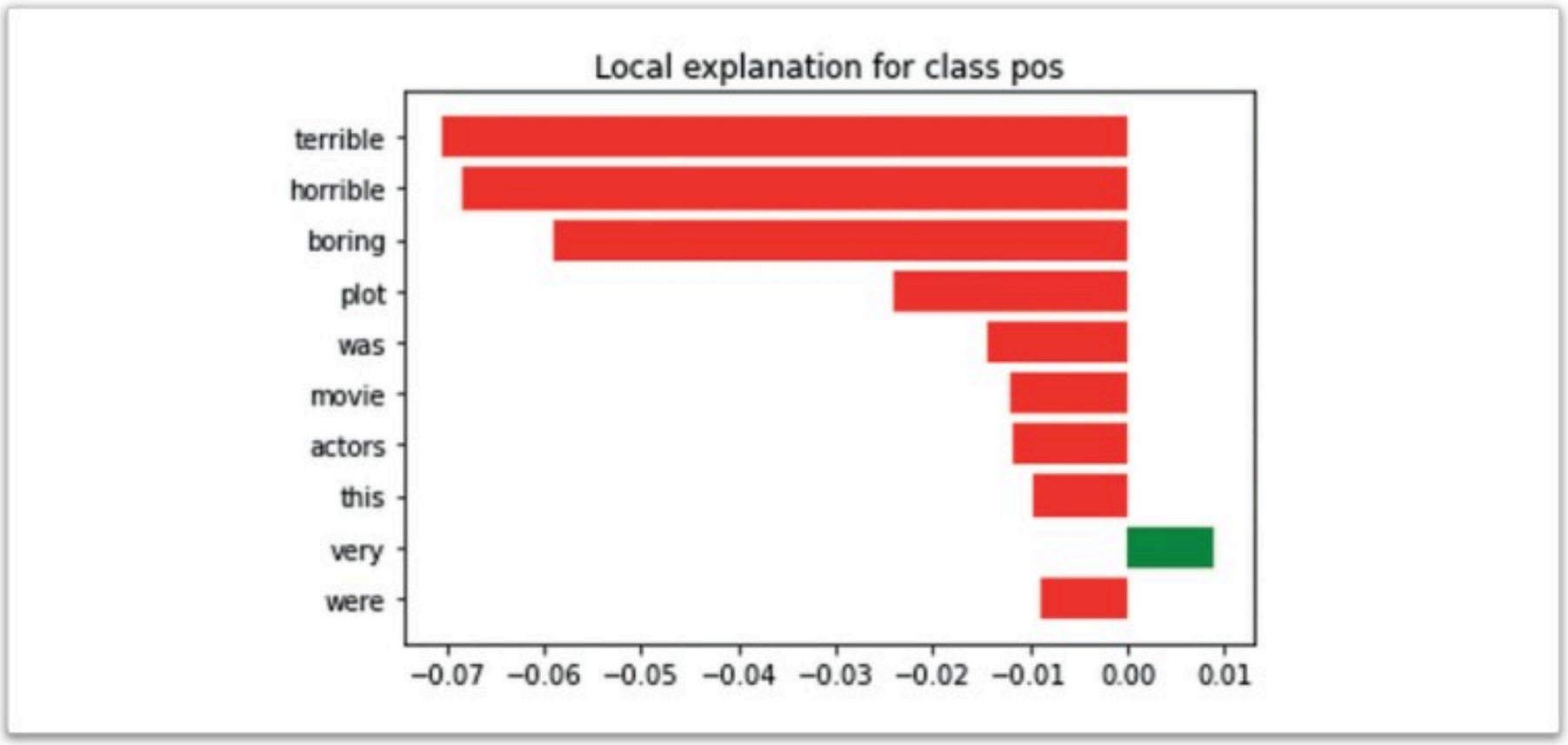
```
[('terrible', -0.07046118794796816),
 ('horrible', -0.06841672591649835),
 ('boring', -0.05909016205135171),
 ('plot', -0.02406309557799637 6),
 ('was', -0.014436071624747861),
 ('movie', -0.011956911011210977),
 ('actors', -0.011682594571408675),
 ('this', -0.009712387273986628),
 ('very', 0.008956707731803237),
 ('were', -0.008897098392433257)]
```

الخصائص العشرة الأكثر تأثيرًا.

الدرجة المقابلة لكل كلمة تمثل مُعاملًا في نموذج الانحدار الخطي البسيط المُستخدَم لتقديم التفسير.

يمكن الحصول على تصور مرئي أكثر دقةً على النحو التالي:

```
# visualize the impact of the most influential words.
fig = exp.as_pyplot_figure()
```

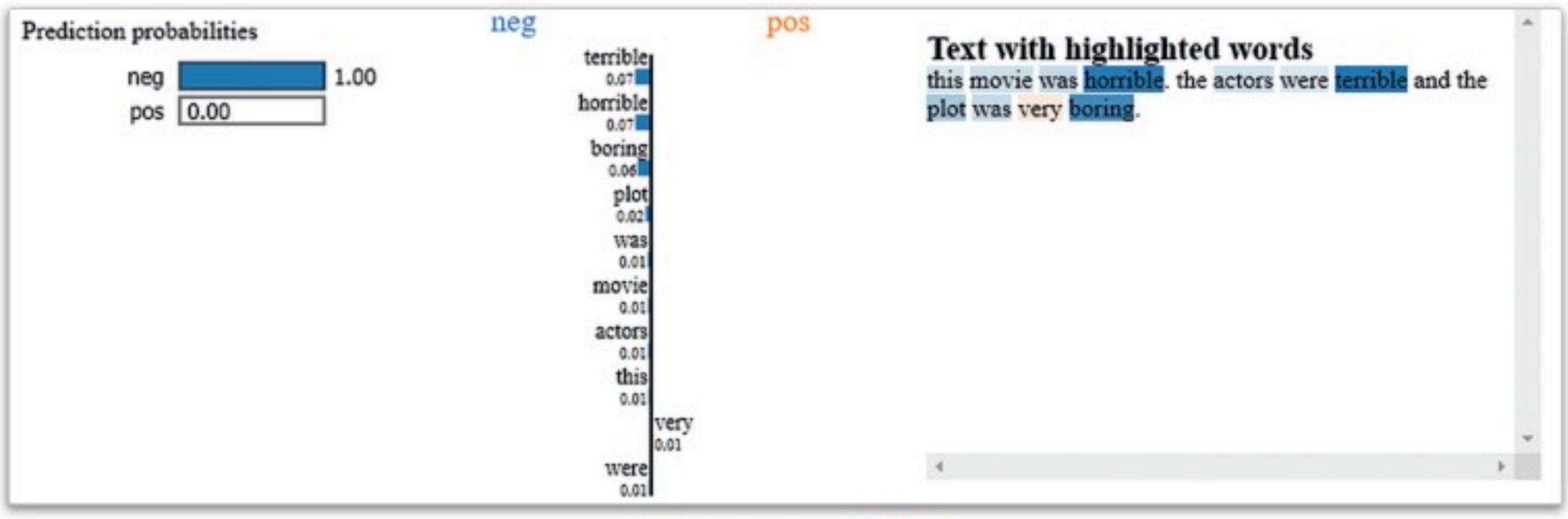


شكل 3.9: الكلمات الأعلى تأثيرًا في القيام بالتنبؤات

يَزيد المُعامِل السالب من احتمالية التصنيف السالب، بينما يُقلل المُعامِل الموجب منه. على سبيل المثال، الكلمات: horrible (فظيع)، وterrible (مريع)، وboring (ممل) لها التأثير الأقوى على قرار النموذج بالتنبؤ بالقيمة السالبة. الكلمة very (جدًا) دفعت النموذج قليلًا في اتجاه آخر إيجابي، ولكنها لم تكن كافية لتغيير القرار. بالنسبة للمراقب البشري، قد يبدو غريبًا أن الكلمات الخالية من المشاعر مثل: plot (الحبكة الدرامية) أو was (كان) لها مُعامِلات مرتفعة نسبيًا. ومع ذلك، من الضروري أن تتذكر أن تعلُّم الآلة لا يتبع دومًا الوعي البشري السليم.

وقد تكشف هذه المُعامِلات المرتفعة بالفعل عن قصور في منطق الخوارزمية وقد تكون مسؤولة عن بعض أخطاء نموذج التنبُّؤ. وعلى نحوٍ بديل، يُعدُّ نموذج التنبُّؤ بمثابة مؤشرٍ على الأنماط التنبؤية الكامنة والغنيّة في الوقت نفسه بالمعلومات. على سبيل المثال، قد يبدو الواقع وكأن المُقيّمِين البشريين أكثر استخدامًا لكلمة plot (الحبكة الدرامية) أو صيغة الماضي was (كان) عند الحديث في سياق سلبي. ويمكن لمكتبة **النموذج المحايد المحلي القابل للتفسير والشرح** (LIME) في لغة البايثون تصوير الشروحات بطرائق أخرى. على سبيل المثال:

```
exp.show_in_notebook()
```



شكل 3.10: التمثيلات المرئية الأخرى

التقييم المُستخدَم في المثال السابق كان سلبيًا بشكل واضح ويَسهُل التنبؤ به. خُذْ بعين الاعتبار التقييم التالي الأكثر صعوبةً والذي يمكن أن يتسبب في تذبذب دقة الخوارزمية، وهو مأخوذ من مجموعة بيانات اختبار IMDb:

```
# an example of a positive review that is mis-classified as negative by prediction_pipeline_v1
mistake_example= X_test_text[4600]
mistake_example
```

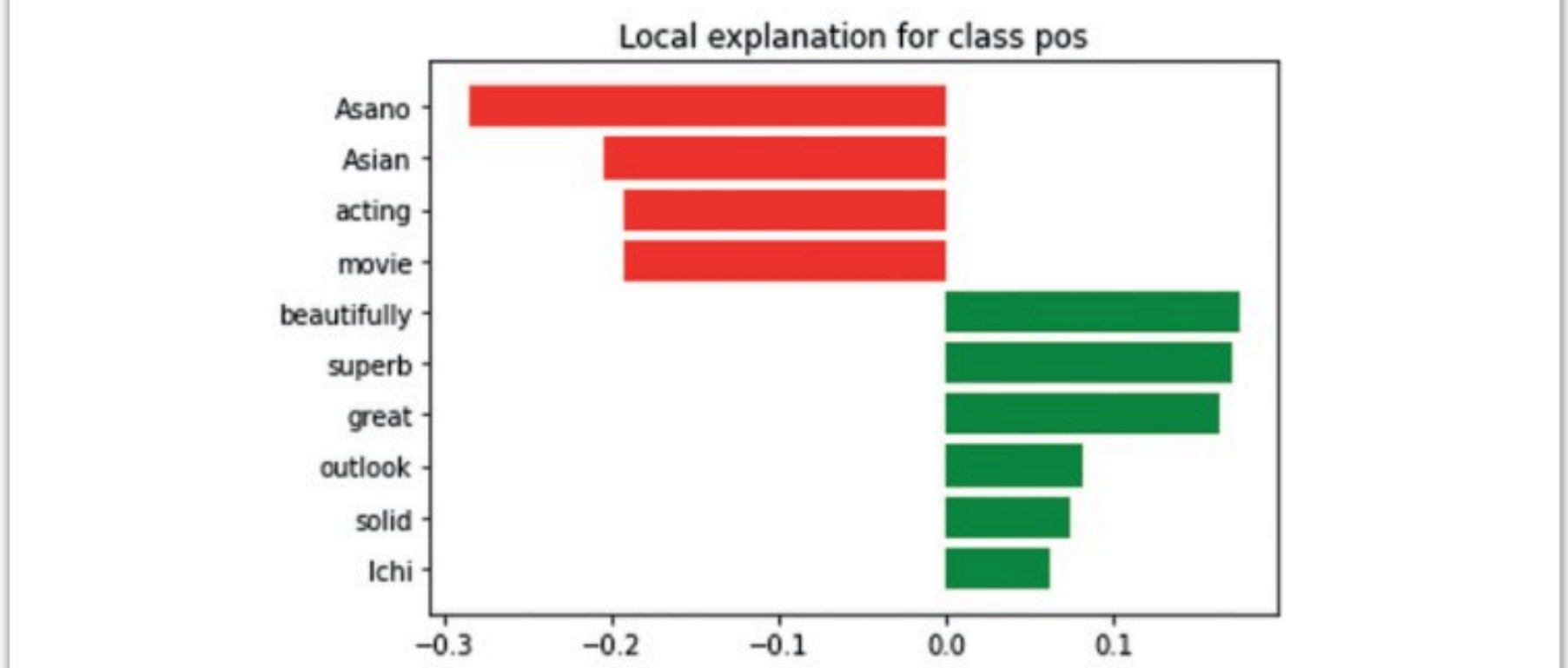
```
"I personally thought the movie was pretty good, very good acting by
Tadanobu Asano of Ichi the Killer fame. I really can't say much about the
story, but there were parts that confused me a little too much, and overall
I thought the movie was just too lengthy. Other than that however, the
movie contained superb acting great fighting and a lot of the locations
were beautifully shot, great effects, and a lot of sword play. Another
solid effort by Tadanobu Asano in my opinion. Well I really can't say
anymore about the movie, but if you're only outlook on Asian cinema is
Crouching Tiger Hidden Dragon or House of Flying Daggers, I would suggest
you trying to rent it, but if you're a die-hard Asian cinema fan I would
say this has to be in your collection very good Japanese film."
```

```
# get the correct labels of this example.
print('Correct Label:', class_names[Y_test[4600]])

# get the prediction probabilities for this example.
print('Prediction Probabilities for neg, pos:',
      prediction_pipeline_v1.predict_proba([mistake_example]))
```

```
Correct Label: pos
Prediction Probabilities for neg, pos: [[0.8367931 0.1632069]]
```

على الرغم من أن هذا التقييم إيجابي بشكل واضح، إلا أنّ نموذج التنبُّؤ قدّم تنبؤًا سلبيًا مؤكدًا للغاية باحتمالية وصلت إلى 83%. يمكن الآن استخدام المُفسِّر لتوضيح السبب وراء اتخاذ نموذج التنبُّؤ مثل هذا القرار الخاطئ:

```
# explain the prediction for this example.
exp = explainer_v1.explain_instance(mistake_example, prediction_pipeline_
v1.predict_proba, num_features=10)

# visualize the explanation.
fig = exp.as_pyplot_figure()
```



شكل 3.11: الكلمات التي أثرت على القرار الخاطئ

على الرغم من أن نموذج التنبُّؤ يستنبط التأثير الإيجابي لبعض الكلمات على نحوٍ صحيح مثل:

beautifully (بشكل جميل)، وgreat (رائع)، وsuperb (مدهش)، إلا أنّه يتّخذ في النهاية قرارًا سلبيًا استنادًا إلى العديد من الكلمات التي يبدو أنها لا تعبِّر بشكل واضح عن المشاعر السلبية مثل: Asano (أسانو)، وAsian (آسيوي)، وmovie (فيلم)، وacting (تمثيل).

وهذا يوضِّح العيوب الكبيرة في المنطِق الذي يستخدمه نموذج التنبُّؤ لتصنيف المفردات الواردة في نصوص التقييمات المُقدمة. القسم التالي يوضِّح كيف أن تحسين هذا المنطق يمكن أن يطور من أداء نموذج التنبُّؤ إلى حدٍ كبير.

تحسين البرمجة الاتجاهية للنصوص

Improving Text Vectorization

استخدم الإصدار الأول لخط أنابيب التنبؤ أداة CountVectorizer لحساب عدد المرات التي تظهر فيها كل كلمة في كل تقييم. تتجاهل هذه المنهجية حقيقتين أساسيتين حول اللغات البشرية:

- قد يتغير معنى الكلمة وأهميّتها حسب الكلمات المُستخدَمة معها.
- تكرار الكلمة في المُستنَد لا يُعدُّ دومًا تمثيلًا دقيقًا لأهميّتها. على سبيل المثال، على الرغم من أن تكرار كلمة great (رائع) مرتين قد يمثل مؤشرًا إيجابيًا في مستند يحتوي على 100 كلمة، إلا أنه يمثل مؤشرًا أقل أهمية بكثير في مستند يحتوي على 1000 كلمة.

سيشرح هذا الجزء كيفية تحسين البرمجة الاتجاهية للنصوص لأخذ هاتين الحقيقتين في عين الاعتبار. يستدعي المقطع البرمجي التالي ثلاثة مكتبات مختلفة بلغة البايثون، ستُستخدم لتحقيق ذلك:

- **nltk وجينسم (Gensim)**: تُستَخدم هاتان المكتبتان الشّهيرتان في مهام معالجة اللغات الطبيعية المُتنوّعة.
- **re**: تُستَخدم هذه المكتبة في البحث عن النّصوص، ومعالجتها باستخدام التعبيرات النمطية.

التعبير النمطي (Regular Expression):

التعبير النمطي هو نمط نص يُستخدَم لمطابقة ولمعالجة سلاسل النصوص وتقديم طريقة موجزة ومرنة لتحديد أنماط النصوص، كما تُستَخدم على نطاق واسع في معالجة النصوص وتحليل البيانات.

```
%%capture

!pip install nltk # install nltk
!pip install gensim # install gensim

import nltk # import nltk
nltk.download('punkt') # install nltk's tokenization tool, used to split a text into sentences.

import re  # import re

from gensim.models.phrases import Phrases, ENGLISH_CONNECTOR_WORDS # import tools
from the gensim library.
```

تحديد العبارات Detecting Phrases

يمكن استخدام الدالة الآتية لتقسيم مستند محدد إلى قائمة من الجُمل المُقسَّمة، حيث يمكن تمثيل كل جملة مُقسَّمة بقائمة من الكلمات:

التقسيم (Tokenization):

يقصد به: عملية تقسيم البيانات النصية إلى أجزاء مثل كلمات، وجُمل، ورموز، وعناصر أخرى يطلق عليها **الرموز** (Tokens).

```
# convert a given doc to a list of tokenized sentences.
def tokenize_doc(doc:str):
    return [re.findall(r'\b\w+\b',
            sent.lower()) for sent in nltk.sent_tokenize(doc)]
```

دالة ( )sent_tokenize تُقسِّم المُستنَد إلى قائمة من الجُمل.

دالة ( )sent_tokenize من مكتبة nltk تُقسِّم المُستنَد إلى قائمة من الجُمل.

بعد ذلك، يتم كتابة كل جملة بأحرف صغيرة وتغذيتها إلى دالة ( )findall من مكتبة re لتقوم بتحديد تكرارات التعبيرات النمطية '\b\w+\b'. ستختبرها على السلسلة النصية الموجودة في متغير raw_text. في هذا السياق:

- \w تتطابق مع كل الرموز الأبجدية الرقمية (0-9، A-Z، a-z) والشَرطة السفلية.
- \w+ تُستَخدم للبحث عن واحد أو أكثر من رموز \w. لذلك، في السلسلة النصية hello123_world (مرحبًا 123_ العالم)، النمط \w+ سيتطابق مع الكلمات hello (مرحبًا) و 123 وworld (العالم).
- \b تمثل **الفاصل** (Boundry) بين رمز \w ورمز ليس \w، وكذلك في بداية أو نهاية السلسلة النصية المُعطاة. على سبيل المثال: سيتطابق النمط \bcat\b مع الكلمة cat (القطة) في السلسلة النصية The cat is cute (القطة لطيفة)، ولكنه لن يتطابق مع الكلمة cat (القطة) في السلسلة النصية The category is pets (فئة الحيوانات الأليفة).

أدناه مثالًا على التقسيم باستخدام الدالة tokenize_doc( ).

```
raw_text='The movie was too long. I fell asleep after the first 2 hours.'
tokenized_sentences=tokenize_doc(raw_text)
tokenized_sentences
```

```
[['the', 'movie', 'was', 'too', 'long'],
 ['i', 'fell', 'asleep', 'after', 'the', 'first', '2', 'hours']]
```

يمكن الآن تجميع الدالة tokenize_doc( ) مع أداة العبارات من مكتبة **جينسم** (Gensim) لإنشاء نموذج العبارة، وهو نموذج يمكنه التعرّف على العبارات المكونة من عدة كلمات في جملة معطاة. يستخدِم المقطع البرمجي التالي بيانات التدريب IMDB الخاصة بـ (X_train_text) لبناء مثل هذا النموذج:

```
sentences=[] # list of all the tokenized sentences across all the docs in this dataset

for doc in X_train_text: # for each doc in this dataset
    sentences+=tokenize_doc(doc) # get the list of tokenized sentences in this doc

# build  a phrase model on the given data
imdb_phrase_model = Phrases(sentences,❶
                            connector_words=ENGLISH_CONNECTOR_WORDS,❷
                            scoring='npmi',❸
                            threshold=0.25).freeze()❹
```

كما هو موضَّح بالأعلى، تَستقبل الدالة Phrases( ) أربعة متغيرات:

❶ قائمة الجُمل المُقسَّمة من مجموعة النصوص المُعطاة.

❷ قائمة بالكلمات الإنجليزية الشائعة التي تظهر بصورة متكررة في العبارات (مثل: the، و of)، وليس لها أي قيمة موجبة أو سالبة، ولكن يمكنها إضفاء المشاعر حسب السياق، ولذلك يتم التعامل معها بصورة مختلفة.

❸ تُستَخدم دالة تسجيل النقاط لتحديد ما إذا كان تضمين مجموعة من الكلمات في العبارة نفسها واجبًا. المقطع البرمجي بالأعلى يَستخدِم مقياس **المعلومات النقطية المشتركة المُعايَر** (Normalized Pointwise Mutual Information – NPMI) لهذا الغرض. يستند هذا المقياس على تكرار توارد الكلمات في العبارة المُرشحة وتكون قيمته بين 1- ويرمز إلى **الاستقلالية الكاملة** (Complete Independence)، و 1+ ويرمز إلى **التوارد الكامل** (Complete Co-occurrence).

❹ في حدود دالة تسجيل النقاط يتم تجاهل العبارات ذات النقاط الأقل. ومن الناحية العملية، يمكن ضبط هذه الحدود لتحديد القيمة التي تُعطي أفضل النتائج في التطبيقات النهائية مثل: النمذجة التنبؤية.

تُحوِّل دالة freeze( ) نموذج العبارة إلى تنسيق غير قابل للتغيير أي **مُجمَّد** (Frozen) لكنّه أكثر سرعة.

عند تطبيقها على الجملتين المُقسَّمتين بالمثال المُوضح بالأعلى، سيُحقق نموذج العبارة النتائج التالية:

```
imdb_phrase_model[tokenized_sentences[0]]
```

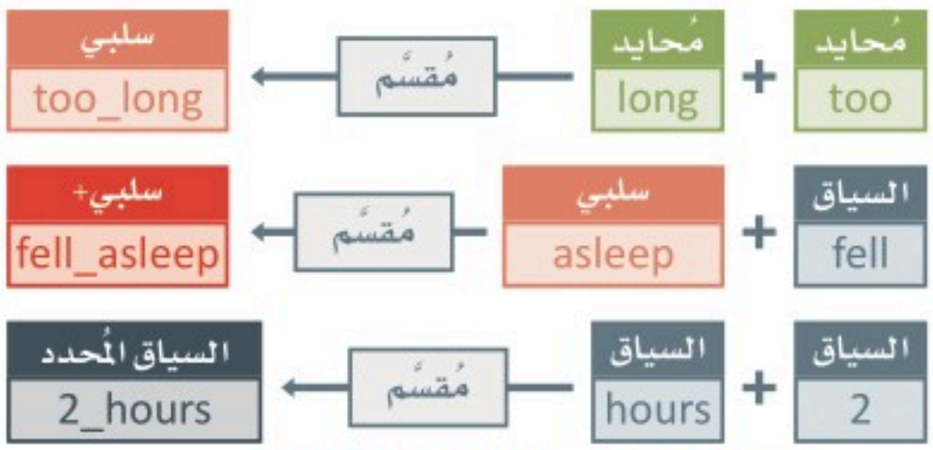
```
['the', 'movie', 'was', 'too_long']
```

```
imdb_phrase_model[tokenized_sentences[1]]
```

```
['i', 'fell_asleep', 'after', 'the', 'first', '2_hours']
```

يحدِّد نموذج العبارة ثلاثة عبارات على النحو التالي: fell_asleep (سقط نائمًا) وtoo_long (طويل جدًا)، و2_hours (2-ساعة) وجميعها تحمل معلومات أكثر من كلماتها المفردة.

على سبيل المثال، تحمل عبارة too_long (طويل جدًا) مشاعر سلبية واضحة، على الرغم من أن كلمتي too (جدًا) وlong (طويل) لا تعبران عن ذلك منفردتين، وبالمثل، فعلى الرغم من أن كلمة asleep (نائم) في مراجعة الفيلم تمثل دلالة سلبية، فالعبارة fell_asleep (سقط نائمًا) توصل رسالة أكثر وضوحًا. وأخيرًا، تستنبط من 2_hours (2-ساعة) سياقًا أكثر تحديدًا من الكلمتين 2 وhours كلٌ على حدة.



شكل 3.12: المشاعر الإيجابية والسلبية قبل التقسيم وبعده

تستخدِم الدالة التالية إمكانية تحديد العبارات بهذا الشكل لتفسير العبارات في وثيقة مُعطاه:

```
def annotate_phrases(doc:str, phrase_model):

    sentences=tokenize_doc(doc)# split the document into tokenized sentences.
    tokens=[] # list of all the words and phrases found in the doc
    for sentence in sentences: # for each sentence
        # use the phrase model to get tokens and append them to the list.
        tokens+=phrase_model[sentence]
    return ' '.join(tokens) # join all the tokens together to create a new annotated document.
```

يستخدِم المقطع البرمجي التالي دالة annotate_phrases( ) لتفسير كلٍ من تقييمات التدريب والاختبار من مجموعة بيانات IMDb.

```
# annotate all the test and train reviews.
X_train_text_annotated=[annotate_phrases(doc,imdb_phrase_model) for doc in X_
train_text]
X_test_text_annotated=[annotate_phrases(text,imdb_phrase_model)for text in X_
test_text]
```

```
# an example of an annotated document from the imdb training data
X_train_text_annotated[0]
```

```
'i_grew up b 1965 watching and loving the thunderbirds all my_mates at school watched
we played thunderbirds before school during lunch and after school we all wanted to
be virgil or scott no_one wanted to be alan counting down from 5 became an art_form
i took my children to see the movie hoping they would get_a_glimpse of what i_loved
as a child how bitterly disappointing the only high_point was the snappy theme_tune
not that it could compare with the original score of the thunderbirds thankfully
early saturday_mornings one television_channel still plays reruns of the series
gerry_anderson and his_wife created jonatha frakes should hand in his directors chair
his version was completely hopeless a waste of film utter_rubbish a cgi remake may_be
acceptable but replacing marionettes with homo_sapiens subsp sapiens was a huge error
of judgment'
```

استخدام مقياس تكرار المصطلح-تكرار المُستنَد العكسي في البرمجة الاتجاهية للنصوص
Using TF-IDF for Text Vectorization

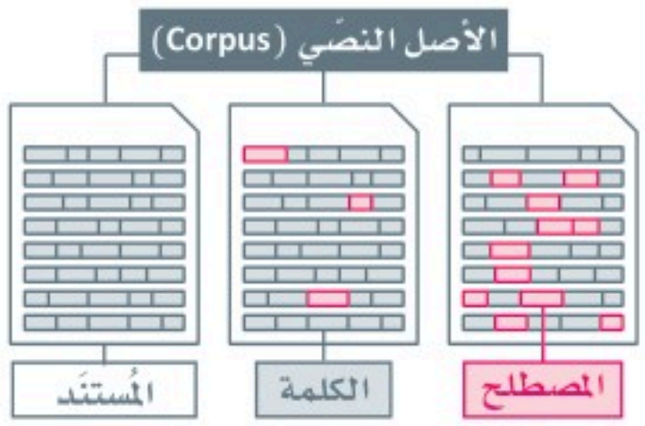
تكرار الكلمة في المُستنَد لا يُعدُّ دومًا تمثيلًا دقيقًا لأهميتها. الطريقة المُثلى لتمثيل التكرار هي المقياس الشهير **لتكرار المصطلح - تكرار المُستنَد العكسي** (TF-IDF). يستخدِم هذا المقياس صيغة رياضية بسيطة لتحديد أهمية الرموز مثل: الكلمات أو العبارات في المُستنَد بناءً على عاملين:

- تكرار الرمز في المُستنَد، بقياس عدد مرات ظهوره في المُستنَد مقسومًا على إجمالي عدد الرموز في جميع المُستنَدات.
- تكرار المُستنَد العكسي للرمز، المحسوب بقسمة إجمالي عدد المُستنَدات في مجموعة البيانات على عدد المُستنَدات التي تحتوي على الرمز.

العامل الأول يتجنب المبالغة في تقدير أهمية المصطلحات التي تظهر في الوثائق الأطول، أمّا العامل الثاني فيستبعد المصطلحات التي تظهر في كثيرٍ من المُستنَدات، مما يساعد على إثبات حقيقة أن بعض الكلمات هي أكثرُ شيوعًا من غيرها.

تكرار المصطلح - تكرار المستند العكسي Term Frequency Inverse Document Frequency (TF-IDF):

تكرار المصطلح - تكرار المُستنَد العكسي هو طريقة تُستخدم لتحديد أهمية الرموز في المُستنَد.



شكل 3.13: الكلمات والمصطلحات الواردة في المستند

$$\text{تكرار المُستنَد العكسي} = \frac{\text{عدد المُستنَدات في الأصل النصّي}}{\text{عدد المُستنَدات التي تحتوي على المصطلح}}$$

$$\text{تكرار المصطلح} = \frac{\text{عدد مرات ظهور المصطلح في المُستنَد}}{\text{عدد الكلمات في المُستنَد}}$$

تكرار المصطلح × تكرار المُستنَد العكسي = القيمة

أداة TfidfVectorizer

توفر مكتبة سكليرن (Sklearn) أداة تدعم هذا النوع من البرمجة الاتجاهية **لتكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF). يمكن استخدام أداة TfidfVectorizer لتمثيل عبارة باستخدام المتَّجهات.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Train a TF-IDF model with the IMDb training dataset
vectorizer_tf = TfidfVectorizer(min_df=10)
vectorizer_tf.fit(X_train_text_annotated)
X_train_tf = vectorizer_tf.transform(X_train_text_annotated)
```

يمكن الآن إدخال أداة التمثيل بالمتَّجَهات في مُصنَّف بايز الساذج لبناء خط أنابيب نموذج تنبُّؤ جديد وتطبيقه على بيانات اختبار IMDb:

```
# train a new Naive Bayes Classifier on the newly vectorized data.
model_tf =MultinomialNB()
model_tf.fit(X_train_v2, Y_train)

# create a new prediction pipeline.
prediction_pipeline_tf = make_pipeline(vectorizer_tf, model_tf)

# get predictions using the new pipeline.
predictions_tf = prediction_pipeline_tf.predict(X_test_text_annotated)

# print the achieved accuracy.
accuracy_score(Y_test, predictions_tf)
```

```
0.8858
```

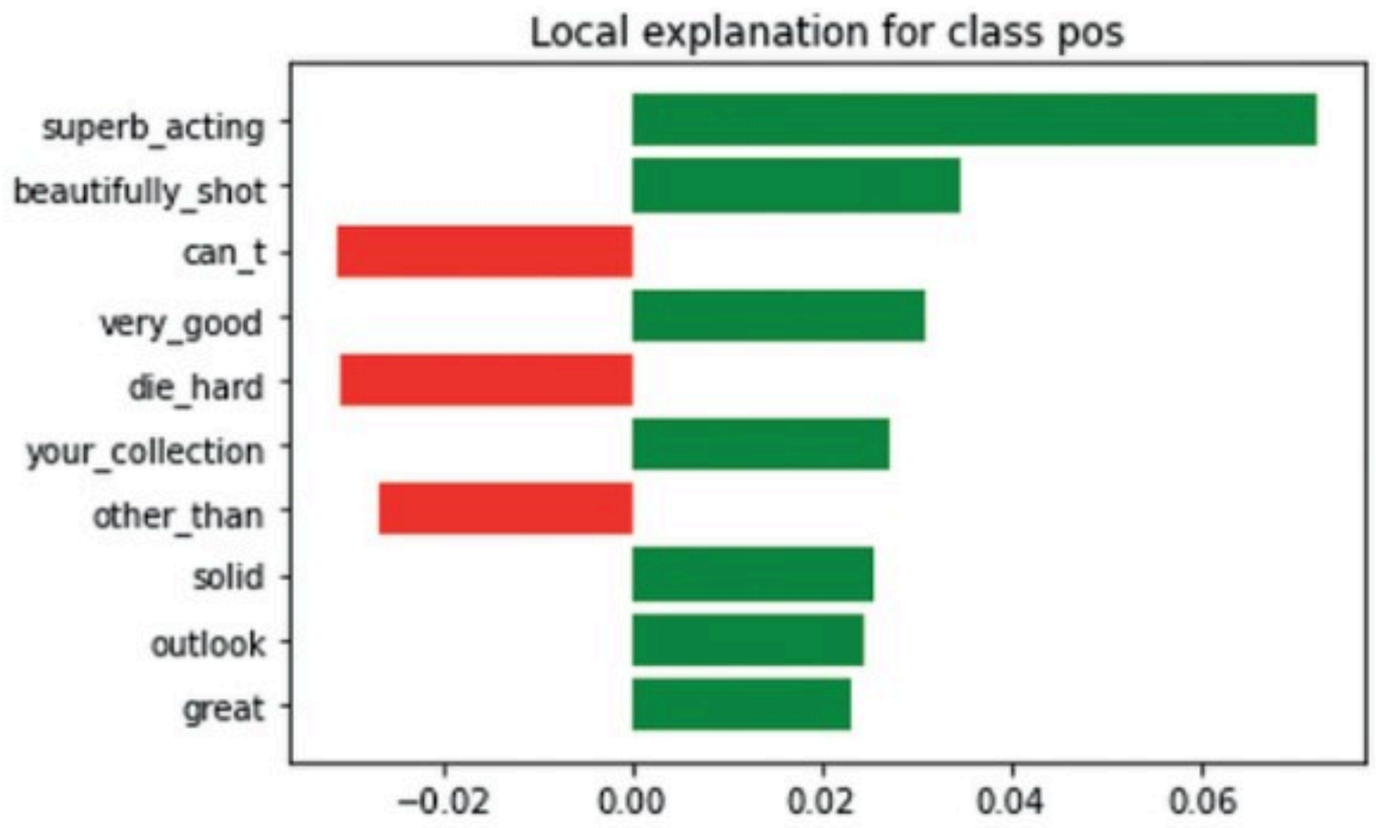
يحقق خط الأنابيب الجديد دقة تصل إلى 88.58%، وهو تحسُّن كبير بالمقارنة مع الدقة السابقة التي وصلت إلى 84.68%. يمكن الآن استخدام النموذج المُحسَّن لإعادة النظر في مثال الاختبار الذي تم تصنيفه بشكل خاطئ بواسطة النموذج الأول:

```
# get the review example that confused the previous algorithm
mistake_example_annotated=X_test_text_annotated[4600]

print('\nReview:',mistake_example_annotated)

# get the correct labels of this example.
print('\nCorrect Label:', class_names[Y_test[4600]])

# get the prediction probabilities for this example.
print('\nPrediction Probabilities for neg, pos:',prediction_pipeline_
tf.predict_proba([mistake_example_annotated]))
```

```
Review: i_personally thought the movie was_pretty good very_good acting by tadanobu_
asano of ichi_the_killer fame i really can_t say much about the story but there_were
parts that confused me a little_too much and overall i_thought the movie was just too
lengthy other_than that however the movie contained superb_acting great fighting and
a lot of the locations were beautifully_shot great effects and a lot of sword play
another solid effort by tadanobu_asano in my_opinion well i really can_t say anymore
about the movie but if_you re only outlook on asian_cinema is crouching_tiger hidden_
dragon or house of flying_daggers i_would suggest_you trying to rent_it but if_you re
a die_hard asian_cinema fan i_would say this has to be in your_collection very_good
japanese film

Correct Label: pos

Prediction Probabilities for neg, pos: [[0.32116538 0.67883462]]
```

يتنبأ خط الأنابيب الجديد بشكل صحيح بالقيمة الإيجابية لهذا التقييم. يَستخدِم المقطع البرمجي التالي مُفسِّر **النموذج المحايد المحلي القابل للتفسير والشرح** (LIME) لتفسير المنطق وراء هذا التنبؤ:

```
# create an explainer.
explainer_tf = LimeTextExplainer(class_names=class_names)

# explain the prediction  of the second pipeline for this example.
exp = explainer_tf.explain_instance(mistake_example_annotated, prediction_
pipeline_tf.predict_proba, num_features=10)

# visualize the results.
fig = exp.as_pyplot_figure()
```



شكل 3.14: تأثير الكلمة في مزيج تكرار المصطلح- تكرار المُستنَد العكسي ومصنِّف بايز الساذج

تؤكد النتائج أن خط الأنابيب الجديد يتبع منطقًا أكثر ذكاءً. فهو يُحدد بشكل صحيح المشاعر الإيجابية للعبارات مثل: beautifully_shot (لقطة _ جميلة)، و superb_acting (تمثيل_رائع)، وvery good (جيد جدًا)، ولا يمكن تضليله باستخدام الكلمات التي جعلت خط الأنابيب الأول يتنبأ بنتائج خاطئة.

يمكن تحسين أداء خط الأنابيب لنموذج التنبُّؤ بطرائق متعددة، بإستبدال مصنف بايز البسيط بطرائق أكثر تطورًا مع ضبط متغيراتها لزيادة احتمالاتها. وثمَّة خيار آخر يتلخص في استخدام تقنيات البرمجة الاتجاهية البديلة التي لا تستند إلى تكرار الرمز، مثل تضمين الكلمات والنصوص، وسيُستعرض ذلك في الدرس التالي.

تمرينات

1

حدِّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. في التعلُّم الموجَّه تُستخدم مجموعات البيانات المُعنوَنة لتدريب النموذج.	○	○
2. البرمجة الاتجاهية هي تقنية لتحويل البيانات من تنسيق متَّجَه رقمي إلى بيانات أولية.	○	○
3. تتطلب المصفوفة المتباعدة ذاكرة أقل بكثير من المصفوفة الكثيفة.	○	○
4. تُستخدم خوارزمية مُصنَّف بايز الساذج لبناء خط أنابيب التنبؤ.	○	○
5. تكرار الكلمة في المُستنَد يُعدُّ التمثيل الدقيق الوحيد لأهمية هذه الكلمة.	○	○

2 اشرح لماذا تتطلب المصفوفة الكثيفة مساحة من الذاكرة أكبر من المصفوفة المتباعدة.

3 حلّل كيف يُستخدَم العاملان الرّياضيّان في تكرار المصطلح- تكرار المُستنَد العكسي (TF-IDF) لتحديد أهمية الكلمة في النص.

4 لديك X_train_text وهي عبارة عن مصفوفة numPy تتضمن مستندًا واحدًا في كل صف. لديك كذلك مصفوفة ثانية Y_train تتضمن قيم المُستنَدات في X_train_text. أكمل المقطع البرمجي التالي بحيث يمكن استخدام تكرار المصطلح- تكرار المُستنَد العكسي (TF-IDF) لتمثيل البيانات بالمتَّجَهات، وتدريب نموذج تصنيف MultinomialNB على الإصدار المُمَثَّل بالمتَّجَهات، ثم تجميع أداة التمثيل بالمتَّجَهات ونموذج التصنيف في خط أنابيب تنبؤ واحد:

```
from _____.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import _____

vectorizer = _____(min_df=10)

vectorizer.fit(_____) # fits the vectorizer on the training data

X_train = vectorizer._____(X_train_text) # uses the fitted vectorizer to vectorize the data
model_MNB=MultinomialNB() # a Naive Bayes Classifier

model_MNB.fit(X_train, _____) # fits the classifier on the vectorized training data

prediction_pipeline = make_pipeline(_____, _____)
```

5 أكمل المقطع البرمجي التالي بحيث يمكنه بِناء مُفسِّر نصوص النموذج المحايد المحلي القابل للتفسير والشرح (LIME) لخط أنابيب التنبؤ الذي قمت ببنائه في التمرين السابق، واستخدِم المُفسِّر لتفسير التنبؤ على مثال لنصٍ آخر.

```
from _____ import LimeTextExplainer
text_example="I really enjoyed this movie, the actors were excellent"
class_names=['neg','pos'] # creates a local explainer for explaining individual predictions

explainer = _____(class_names=class_names) # explains the prediction for this example

exp = explainer._____(text_example.lower(),prediction_pipeline._____,

                      _____=10) # focuses the explainer on the 10 most influential features

print(exp._____) # prints the words with the highest influence on the prediction
```

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الثاني

التعلُّم غير الموجَّه

استخدام التعلُّم غير الموجَّه لفهم النصوص

Using Unsupervised Learning to Understand Text

التعلُّم غير الموجَّه هو نوع من تعلُّم الآلة، يستخدِم فيه النموذج بيانات غير مُعَنونة، حيثُ يُقدِّم له مجموعة من الأمثلة التي يتولى البحث فيها عن الأنماط والعلاقات بين البيانات من تلقاء نفسه. وفي سياق فهم النص، يمكن استخدام التعلُّم غير الموجَّه في تحديد الهياكل والأنماط الكامنة ضمن مجموعة بيانات المُستنَدات النصية. هناك العديد من التقنيات المختلفة التي يمكن استخدامها في التعلُّم غير الموجَّه للبيانات النصية، بما في ذلك **خوارزميات التجميع** (Clustering Algorithms)، و**تقنيات تقليص الأبعاد** (Dimensionality Reduction Techniques)، و**النماذج التوليدية** (Generative Models). تُستخدم خوارزميات التجميع لضم المُستنَدات المتشابهة معًا، بينما تُستخدم تقنيات تقليص الأبعاد لتقليص أبعاد البيانات وتحديد الخصائص الهامة. ومن ناحية أخرى، تُستخدم النماذج التوليدية لتعلُّم التوزيع الأساسي للبيانات وتوليد نص جديد مشابه لمجموعة البيانات الأصلية.

خوارزميات التجميع Clustering Algorithms

يمكن لخوارزميات التجميع تجميع العملاء المتشابهين استنادًا إلى السلوكيات أو الديموغرافيا، أو سجل المشتريات؛ لأغراض التسويق المُستهدَف وزيادة معدلات الاحتفاظ بالعملاء.

تقنيات تقليص الأبعاد

Dimensionality Reduction Techniques

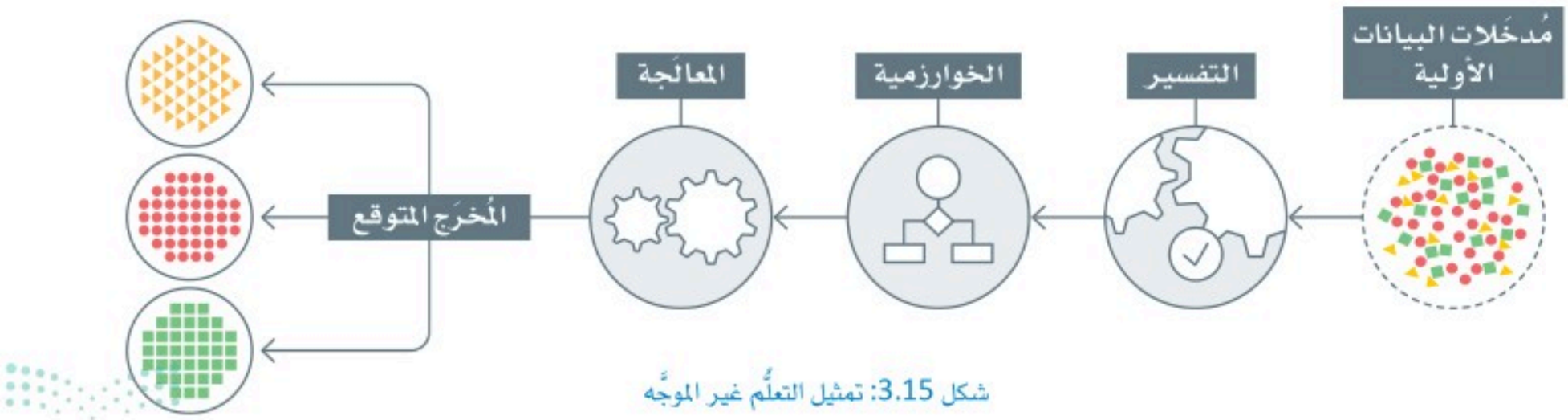
تُستخدم تقنيات تقليص الأبعاد في ضغط الصورة لتقليل عدد وحدات البيكسل فيها، مما يساعد على تقليص حجم البيانات اللازمة لتمثيلها مع الحفاظ على خصائصها الرئيسة.

النماذج التوليدية Generative Models

تُستخدم النماذج التوليدية في تطبيقات الكشف عن الاختلاف؛ حيث تُحدِّد الاختلافات في البيانات بتعلُّم الأنماط الطبيعية للبيانات باستخدام النموذج التوليدي.

التعلُّم غير الموجَّه (Unsupervised Learning):
في التعلُّم غير الموجَّه، يُزوَّد النموذج بكميات كبيرة من البيانات غير المُعنوَنة ويتوجب عليه البحث عن الأنماط في البيانات غير المُتراكبة من خلال الملاحظة والتجميع.

تقليص الأبعاد (Dimensionality Reduction):
تقنية تقليص الأبعاد هي إحدى تقنيات تعلُّم الآلة وتحليل البيانات المُستخدَمة لتقليص عدد الخصائص (الأبعاد) في مجموعة البيانات مع الاحتفاظ بأكبر قدر ممكن من المعلومات.



شكل 3.15: تمثيل التعلُّم غير الموجَّه

وإحدى المزايا الرئيسة لاستخدام التعلُّم غير الموجَّه هي أنه يمكن استخدامه للكشف عن الأنماط والعلاقات التي قد لا تبدو واضحة على الفور للمراقب البشري. وقد يكون هذا مفيدًا بشكل خاص في فهم مجموعات البيانات الكبيرة المكونة من النصوص غير المُتراكبة، حيث يكون التحليل اليدوي غير عملي. في هذه الوحدة، ستَستخدم مجموعة بيانات متوافرة للعامّة من المقالات الإخبارية من **هيئة الإذاعة البريطانية** (BBC) بواسطة **جرين وكوننجهام**، (Greene & Cunningham، 2006) لتوضيح بعض التقنيات الرئيسة للتعلُّم غير الموجَّه. يُستخدم المقطع البرمجي التالي لتحميل مجموعة البيانات، المُنظّمة في خمسة مجلدات إخبارية مختلفة تمثل مقالات من أقسام إخبارية مختلفة، هي: الأعمال التجارية، والسياسة، والرياضة، والتقنية، والترفيه. لن تستخدِم القيم الخمسة في توجيه أي من الخوارزميات المُستخدَمة في هذه الوحدة. وبدلًا من ذلك، ستُستخدم فقط لأغراض التصوير والمصادقة. يتضمن كل مجلد إخباري مئات الملفات النصية، وكل ملف يتضمن محتوى مقالة واحدة محددة. وقد حُمِّلت مجموعة البيانات بالفعل إلى **مفكرة جوبيتر** (Jupyter Notebook) وستقوم لبنة التعليمات البرمجية بفتح واستخراج كل المُستنَدات والقيم المطلوبة في تركيبتين لبيانات القوائم، على التوالي.

العنقود (Cluster):

العنقود هو مجموعة من الأشياء المتشابهة. وفي تعلُّم الآلة، يشير التجميع (Clustering) إلى عملية تجميع البيانات غير المُعنوَنة في عناقيد متجانسة.

شكل 3.16: تمثيل عنقود

BBC open dataset

https://www.kaggle.com/datasets/shivamkushwaha/bbc-full-text-document-classification

D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006. All rights, including copyright, in the content of the original articles are owned by the BBC.

```
# used to list all the files and subfolders in a given folder
from os import listdir
# used for generating random number
import random shuffling lists

bbc_docs=[] # holds the text of the articles
bbc_labels=[] # holds the news section for each article

for folder in listdir('bbc'): # for each news-section folder
    for file in listdir('bbc/'+folder): # for each text file in this folder

        # open the text file, use encoding='utf8' because articles may include non-ascii characters
        with open('bbc/'+folder+'/'+file,encoding='utf8',errors='ignore') as f:
            bbc_docs.append(f.read()) # read the text of the article and append to the docs list
        # use the name of the folder (news section) as a label for this doc
        bbc_labels.append(folder)
# shuffle the docs and labels lists in parallel
merged = list(zip(bbc_docs, bbc_labels)) # link the two lists
random.shuffle(merged) # shuffle them in parallel (with the same random order)
bbc_docs, bbc_labels = zip(*merged) # separate them again into individual lists.
```

تجميع المُستنَدات Document Clustering

الآن بعد تحميل مجموعة البيانات فإن الخطوة التالية هي تجربة عدة طرائق غير موجَّهة، ومنها: التجميع الذي يُعدّ الطريقة غير الموجَّهة الأكثر شهرة في هذا النطاق. وبالنّظر إلى مجموعة من المُستنَدات غير المُعنوَنة، سيكون الهدف هو تجميع الوثائق المتشابهة معًا، وفي الوقت نفسه الفصل بين الوثائق غير المتشابهة.

تجميع المُستنَدات (Document Clustering):
تجميع المُستنَدات هو طريقة تُستخدم لتجميع المُستنَدات النصيّة في عناقيد بناءً على تشابه محتواها.

جدول 3.2: العوامل التي تُحدد جودة النتائج

1	**طريقة تمثيل البيانات بالمتَّجَهات:** على الرغم من أن تقنية **تكرار المصطلح- تكرار المُستنَد العكسي** (TF-IDF) أثبتت كفاءتها وفعاليتها في هذا المجال، إلا أنّك ستتعرف في هذه الوحدة على مزيد من البدائل الأكثر تطورًا وتعقيدًا.
2	**التعريف الدقيق للتشابه بين مستند وآخر:** بالنسبة للبيانات النصيّة المُمثلة بالمتَّجَهات، تكون مقاييس المسافة الإقليدية وجيب التمام هما الأكثر شيوعًا، وسيُستخدم الأول في الأمثلة المشروحة في هذه الوحدة.
3	**عدد العناقيد المُختارة:** يوفر **التجميع التكتلي** (Agglomerative Clustering - AC) طريقة واضحة لتحديد العدد المناسب من العناقيد ضمن مجموعة محددة من البيانات، وهو التحدي الرئيس الذي يواجه مهام التجميع.

تحديد عدد العناقيد
Selecting the Number of Clusters

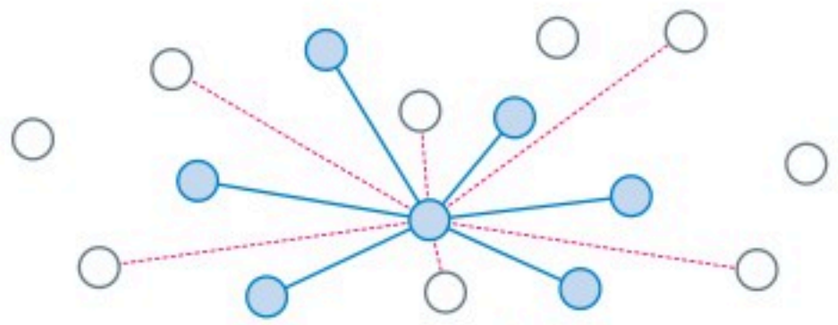
تحديد العدد الصحيح للعناقيد هو خطوة ضرورية ضمن مهام التجميع. للأسف، تعتمد الغالبية العظمى من خوارزميات التجميع على المُستخدِم في تحديد عدد العناقيد الصحيحة ضمن المُدخَلات. ربما يكون للعدد المحدد تأثيرًا كبيرًا على جودة النتائج وقابليتها للتفسير، ولكن هناك العديد من المقاييس أو المؤشرات التي يمكن استخدامها لتحديد عدد العناقيد.

- إحدى الطرائق الشائعة هي استخدام مقياس **التراص** (Compactness). يمكن القيام بذلك عن طريق حساب مجموع المسافات بين النقاط ضمن كل عنقود، وتحديد عدد العناقيد الذي يقلل من هذا المجموع إلى الحد الأدنى.
- هناك طريقة أخرى تتلخص في مقياس **الفصل** (Separation) بين العناقيد، مثل متوسط المسافة بين النقاط في العناقيد المختلفة، وبناء عليه، يتم تحديد عدد العناقيد الذي يرفع من هذا المتوسط.

وبشكل عملي، غالبًا ما تتعارض المنهجيات المذكورة بالأعلى مع بعضها من حيث التوصية بأرقام مختلفة، ويمثّل هذا تحدّيًا مشتركًا عند التعامل مع البيانات النصية بشكلٍ خاص، فعادةً ما يصعُب تمييز تركيبها.

المسافة الإقليدية (Euclidean Distance):
المسافة الإقليدية هي مسافة الخط المستقيم بين نقطتين في فضاء متعدد الأبعاد. وتُحسب بالجذر التربيعي لمجموع مربعات الفروقات بين الأبعاد المناظرة للنقاط. تُستخدم المسافة الإقليدية في التجميع لقياس التشابه بين نقطتي بيانات.

مسافة جيب التمام (Cosine Distance):
تُستخدم مسافة جيب التمام لقياس التشابه في جيب التمام بين نقطتي البيانات. فهي تحسب جيب تمام الزاوية بين متَّجَهين يمثلان نقاط البيانات، وتُستخدَم عادةً في تجميع البيانات النصيّة. وتقع قيمة جيب التمام بين -1 و 1؛ حيث تشير القيمة -1 إلى الاتجاه العكسي، بينما تشير القيمة 1 إلى الاتجاه نفسه.



شكل 3.17: آلة حساب المسافات بين النقاط

في التعلُّم غير الموجَّه، يشير عدد العناقيد إلى عدد المجموعات أو التصنيفات التي تنقسم إليها البيانات بواسطة الخوارزمية. ويُعدُّ تحديد عدد العناقيد الصحيح أمرًا مهمًا؛ لأنه يؤثر على دقة النتائج وقابليتها للتفسير. إذا كان عدد العناقيد كبيرًا للغاية، فإنّ المجموعات ستكون محدّدةً جدًا ودون معنى. في حين أنه إذا كان عدد العناقيد منخفضًا للغاية، فإنّ المجموعات ستكون ممتدة على نطاق واسع جدًا، ولن تستنبط التركيب الأساسي للبيانات. من الضروري تحقيق التوازن بين توفير عددٍ كافٍ من العناقيد لاستنباط أنماط ذات معنى، وألا تكون كثيرة في الوقت نفسه بالقدر الذي يجعل النتائج مُعقدة للغاية وغير مفهومة.

التجميع الهرمي (Hierarchical Clustering):

التجميع الهرمي هو خوارزمية التجميع المُستخدَمة لتجميع البيانات في عناقيد بناءً على التشابه. في التجميع الهرمي، تُنظّم نقاط البيانات في تركيب يشبه الشجرة، حيث تكون كل عُقدة بمثابة عنقود، وتكون العُقدة الأم هي نقطة التقاء العُقد المتفرعة منها.

يُستخدَم المقطع البرمجي التالي لاستيراد مكتبات محددة تُستخدَم في التجميع الهرمي من بدايته حتى نهايته:

```
# used for tfi-df vectorization, as seen in the previous unit
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import AgglomerativeClustering # used for agglomerative clustering

# used to visualize and support hierarchical clustering tasks
import scipy.cluster.hierarchy as hierarchy

# set the color palette to be used by the 'hierarchy' tool.
hierarchy.set_link_color_palette
(['blue','green','red','yellow','brown','purple','orange','pink','black'])

import matplotlib.pyplot as plt # used for general visualizations
```

البرمجة الاتجاهية للنصوص Text Vectorization

تتطلب العديد من طرائق التعلُّم غير الموجَّه تمثيل النصّ الأوليّ بالمتَّجهات في تنسيق رقميّ، كما تمّ عرضه في الوحدة السابقة، ويستخدِم المقطع البرمجي التالي أداة TfidfVectorizer التي اُستخدمت في الدرس السابق لهذا الغرض:

```
vectorizer = TfidfVectorizer(min_df=10) # apply tf-idf vectorization, ignore words that
appear in more than 10 docs.

text_tfidf=vectorizer.fit_transform(bbc_docs) # fit and transform in one line

text_tfidf
```

```
<2225x5867 sparse matrix of type '<class 'numpy.float64'>'
        with 392379 stored elements in Compressed Sparse Row format>
```

الآن تحوَّلت بيانات النص إلى تنسيق رقمي متباعد كما اُستخدمت في الدرس السابق.

يَستخدِم المقطع البرمجي التالي أداة TSENVisualizer من مكتبة yellowbrick لإسقاط وتصوير النصوص المُمثلة بالمتَّجهات في فضاء ثنائي الأبعاد:

```
%%capture
!pip install yellowbrick
from yellowbrick.text import TSNEVisualizer
```

تقليص الأبعاد Dimensionality Reduction

يكون تقليص الأبعاد مفيدًا في العديد من التطبيقات مثل:

- **تصوير البيانات عالية الأبعاد:** من الصعب تصوير البيانات في فضاء عالي الأبعاد، ولذلك تُقلّص الأبعاد ليسهل تصوير البيانات وفهمها في هذه الحالة.
- **تبسيط النموذج:** النموذج ذو الأبعاد الأقل يكون أبسط وأسهل فهمًا، ويستغرق وقتًا أقل في عملية التدريب.
- **تحسين أداء النموذج:** يُساعد تقليص الأبعاد في التخلص من التشويش وتكرار البيانات، مما يُحسّن أداء النموذج.

تضمين المجاور العشوائي الموزع على شكل T t-Distributed Stochastic Neighbor Embedding (T-SNE):

خوارزمية تضمين المجاور العشوائي الموزَّع على شكل T (T-SNE) هي خوارزمية تعلُّم الآلة غير الموجَّه المُستخدَمة لتقليص الأبعاد.

جدول 3.3: تقنيات تقليص الأبعاد

التقنية	الوصف	مثال التطبيق العملي
تحديد الخصائص (Feature Selection)	تحديد الخصائص يتضمن تحديد مجموعة فرعية من الخصائص الرئيسة.	تحتوي مجموعات البيانات الطبية على مئات من أعمدة البيانات ذات الصلة بحالة المريض. يمكن لعدد قليل من هذه الخصائص مساعدة النموذج في التشخيص السليم لحالة المريض، بينما تكون السمات الأخرى غير مرتبطة بالتشخيص وقد تُشتت النموذج، وتحديد الخصائص يتجاهل كل الخصائص بإستثناء الأكثر تميزًا منها.
تحويل الخصائص (Feature Transformation)	يتضمن تحويل الخصائص تجميع الخصائص الأصلية أو تحويلها لإنشاء مجموعة جديدة من الخصائص، واستبدال الخصائص الرئيسة إذا لم تكن هناك حاجة إليها.	إذا توقَّع النموذج إقامة المريض في المستشفى، يمكن إنشاء خصائص إضافية للنموذج باستخدام الخصائص الحالية لسجلات الحالة الطبية للمريض. على سبيل المثال، حساب عدد الفحوصات المخبرية المطلوبة على مدار الأسبوع الماضي، أو عدد الزيارات على مدار الشهر الماضي. وهناك مثال آخر، وهو: حساب مساحة المستطيل بإستخدام ارتفاعه وعرضه.
التعلُّم المتشعِّب (Manifold Learning)	تقنيات التعلُّم المتشعِّب، مثل **تضمين المجاور العشوائي الموزَّع على شكل T (T-SNE) والتقريب والإسقاط المتشعِّب المنتظم** (Uniform Manifold Approximation and Projection -UMAP) هي تقنيات التعلُّم غير الموجَّه التي تهدف إلى الحفاظ على تركيب البيانات في الفضاء منخفض الأبعاد.	يمكن لهذه التقنيات تحويل صورة عالية الأبعاد إلى فضاء منخفض الأبعاد مع الحفاظ على الخصائص والتركيب الأساسيين لها. ونظرًا لأن هذا يقلص من المساحة المطلوبة، فإنه يمكن تخزين وإرسال هذا التمثيل وإعادة بناء الصورة الأصلية مع خسارة أقل قدر من المعلومات.

إحدى الخصائص الرئيسة لتقنية **تضمين المجاور العشوائي الموزَّع على شكل T** (T-SNE) هي محاولة الحفاظ على التركيب المحلي للبيانات قدر الإمكان، حتى تتقارب نقاط البيانات الشبيهة في التمثيل منخفض الأبعاد، ويتحقق ذلك بتقليص التباعد بين التوزيعين المحتملين: توزيع البيانات عالية الأبعاد، وتوزيع البيانات منخفضة الأبعاد.

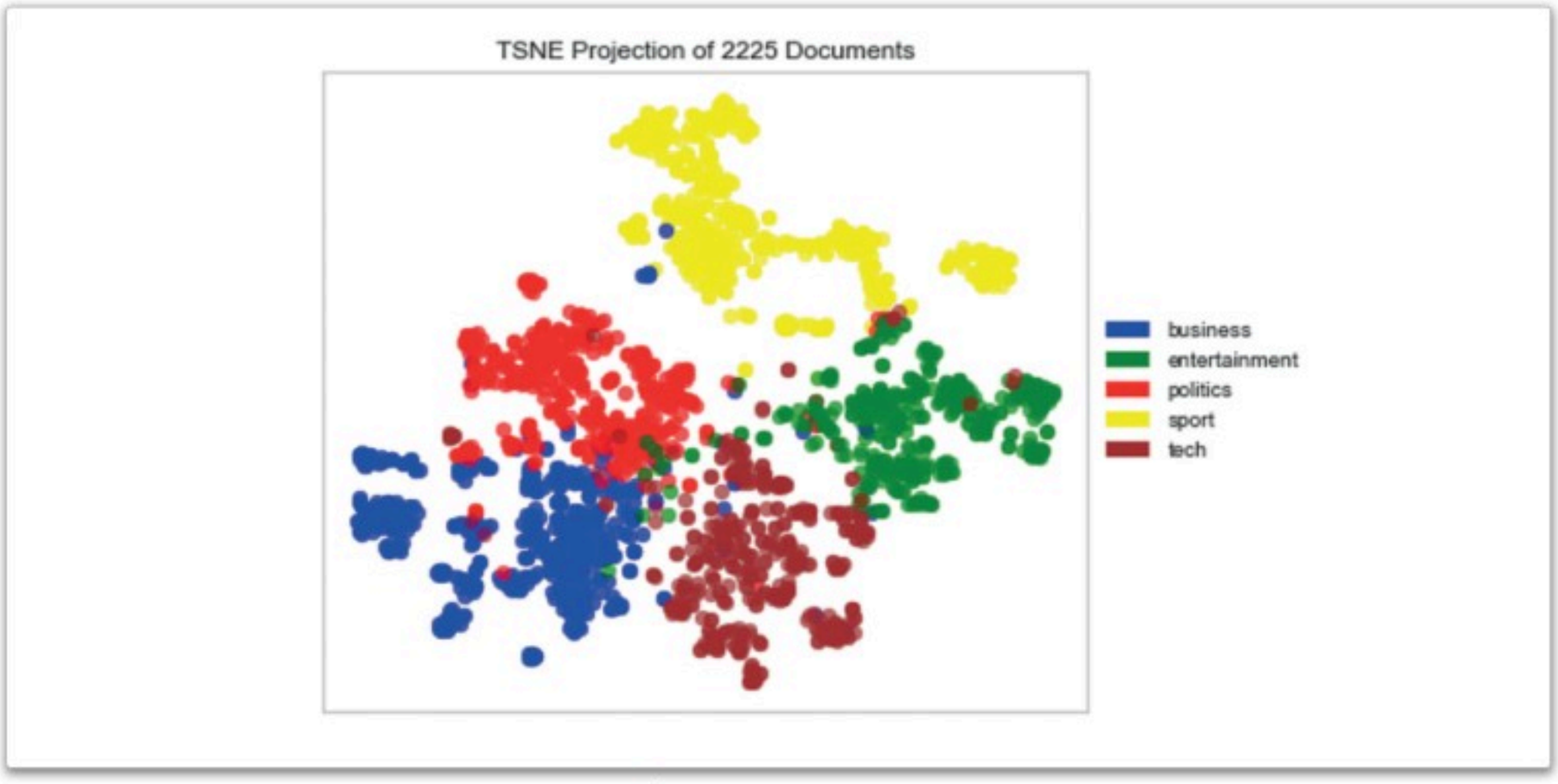
مجموعة بيانات هيئة الإذاعة البريطانية المُمثلة بالمتَّجَهات تُصنَّف بالتأكيد كبيانات عالية الأبعاد، لأنها تتضمن بُعدًا مستقلًا أي **عمودًا** (Column) لكل كلمة فريدة تظهر في البيانات. يُحسب العدد الإجمالي للأبعاد كما يلي:

```
print('Number of unique words in the BBC documents vectors:',
                    len(vectorizer.get_feature_names_out()))
```

```
Number of unique words in the BBC documents vectors: 5867
```

يُستخدَم المقطع البرمجي التالي لإسقاط 5,867 بُعدًا في محورين فقط وهما محوري X و Y في الرسم البياني. يُستخدَم المقطع البرمجي التالي لتصميم مُخطَّط الانتشار حيث يمثل كل لون أحد الأقسام الإخبارية الخمسة.

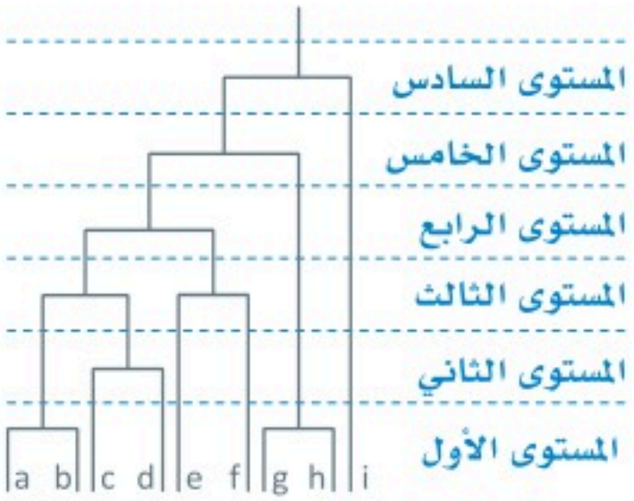
```
tsne = TSNEVisualizer(colors=['blue','green','red','yellow','brown'])
tsne.fit(text_tfidf,bbc_labels)
tsne.show();
```



شكل 3.18: إسقاط تضمين المجاور العشوائي الموزَّع على شكل T (T-SNE)

يَستخدِم هذا التصور قيمة ground-truth (بيانات الحقيقة المعتمدة) من **القسم الإخباري** (News Section) في كل مستند للكشف عن انتشار كل قيمة في إسقاط فضاء البرمجة الاتجاهية ثنائي الأبعاد. يوضِّح الشكل أنه على الرغم من ظهور بعض الشوائب في فراغات مُحدَّدة من فضاء البيانات، إلا أن الأقسام الإخبارية الخمسة منفصلة بشكل جيد. وسنستعرض لاحقًا البرمجة الاتجاهية المُحسَّنة للحد من هذه الشوائب.

التجميع التكتلي Agglomerative Clustering (AC)



شكل 3.19: التجميع التكتلي (AC)

التجميع التكتلي (AC) هو الطريقة الأكثر انتشارًا وفعاليةً في هذا الفضاء، فمن خلالها يمكن التغلّب على هذا التحدي بتوفير طريقة واضحة لتحديد العدد المناسب من العناقيد. يستند **التجميع التكتلي** (AC) إلى منهجية التصميم من أسفل إلى أعلى، حيث تبدأ بحساب المسافة بين كل أزواج نقاط البيانات، ثم اختيار النقطتين الأقرب ودمجهما في عنقود واحد. تتكرر هذه العملية حتى تُدمج كل نقاط البيانات في عنقود واحد، أو حتى الوصول إلى العدد المطلوب من العناقيد.

دالة linkage( ) *fx*

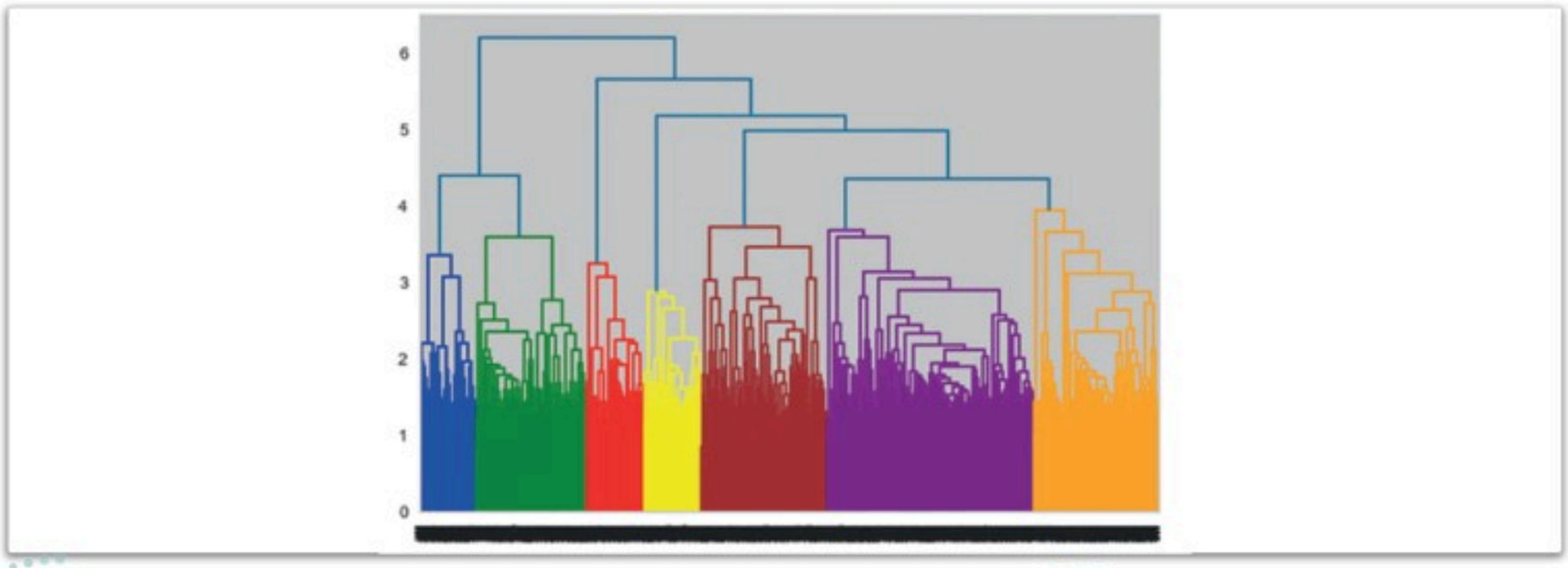
تُنفِذ لغة البايثون **التجميع التكتلي** (AC) باستخدام دالة linkage( ).

يجب توفير متغيرين لدالة linkage( ):

- البيانات النصيّة المُمثلة بالمتَّجَهات، ويمكن استخدام دالة toarray( ) لتحويل البيانات إلى تنسيق كثيف يمكن لهذه الدالة أن تتعامل معه.
- مقياس المسافة الذي يجب استخدامه لتحديد العناقيد التي ستُدمج أثناء عملية التجميع التكتلي. تتوفر عدة خيارات من مقاييس المسافة للاختيار من بينها وفقًا لمتطلبات وتفضيلات المُستخدِم، مثل المسافة **الإقليدية** (Euclidian)، ومسافة **مانهاتن** (Manhattan)... إلخ، ولكن في هذا المشروع ستستخدِم طريقة وارد (**ward**) القياسية.

يستخدِم المقطع البرمجي التالي دالة linkage( ) من الأداة **الهرمية** (Hierarchy) الواردة بالأعلى لتطبيق هذه العملية على بيانات هيئة الإذاعة البريطانية المُمثَّلة بالمتَّجَهات:

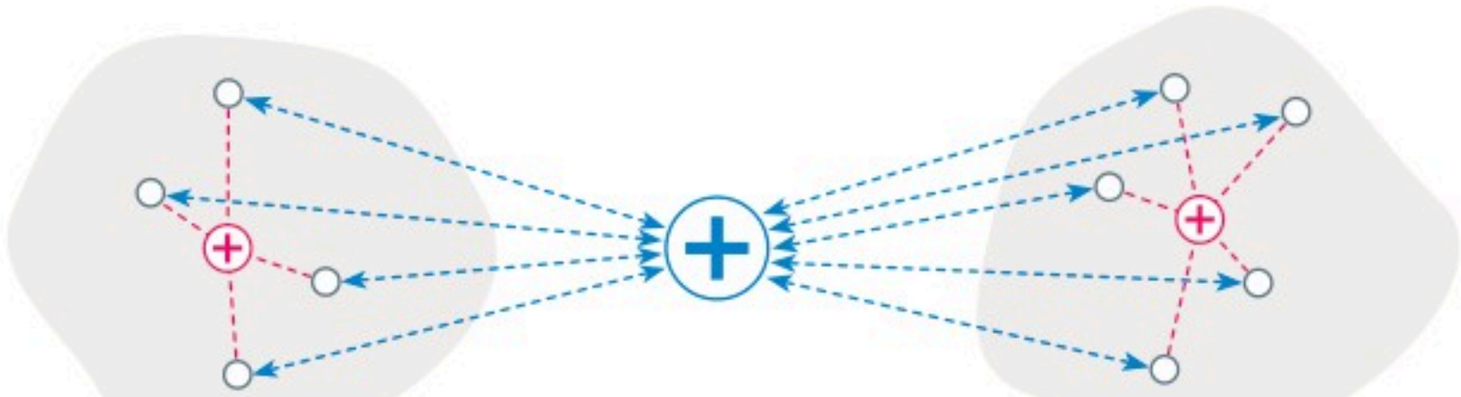
```
plt.figure() # create a new empty figure

# iteratively merge points and clusters until all points belong to a single cluster
# return the linkage of the produced tree
linkage_tfidf=hierarchy.linkage(text_tfidf.toarray(),method='ward')

# visualize the linkage
hierarchy.dendrogram(linkage_tfidf)

# show the figure
plt.show()
```



شكل 3.20: الرسم الشجري الهرمي لبيانات هيئة الإذاعة البريطانية

مسافة وارد Ward Distance

يَستخدِم المثال أعلاه طريقة وارد (Ward) القياسية لقياس المسافة للمتغير الثاني. تستند مسافة وارد (Ward) إلى مفهوم التباين داخل العنقود، وهو مجموع المسافات بين النقاط في العنقود. في كل تكرار، تُقيِّم الطريقة كل عملية دمج ممكنة بحساب التباين داخل العنقود قبل عمليّة الدمج وبعدها، ثم تبدأ عملية الدمج التي تحقِّق أقل ارتفاع في التباين. أظهرت مسافة وارد (Ward) نتائج جيدة في معالجة البيانات النصيّة، بالرغم من وجود العديد من الخيارات الأخرى.



شكل 3.21: مثال على طريقة وارد (Ward)

الرسم الشجري في الشكل 3.20 يعرض طريقة واضحة لتحديد عدد العناقيد. في هذا المثال، تقترح المكتبة استخدام 7 عناقيد، مع تمييز كل عنقود بلون مختلف. قد يتبنى المُستخدِم هذا المُقترح أو يَستخدِم الرسم الشجري لاختيار رقم مختلف. على سبيل المثال، دُمِّج اللونين الأزرق والأخضر في آخر خطوة مع مجموعة العناقيد لكل الألوان الأخرى. وهكذا، سيؤدي اختيار 6 عناقيد إلى دمج اللونين الأرجواني والبرتقالي، بينما اختيار 5 عناقيد سيؤدي إلى دمج اللونين الأزرق والأخضر.

الرسم الشجري (Dendrogram):

الرسم الشجري هو رسم تخطيطي تفرعي يوضِّح العلاقة الهرمية بين البيانات، ويأتي عادة في صورة أحد مُخرَجات التجميع الهرمي.

يتبنى المقطع البرمجي التالي مقترحات الأداة ويَستخدِم أداة التجميع التكتلي من مكتبة **سكليرن** (Sklearn) لتقسيم المُخطَّط الشجري بعد إنشاء العناقيد السبع:

```
AC_tfidf=AgglomerativeClustering(linkage='ward',n_clusters=7) # prepare the tool, set the number of clusters.

AC_tfidf.fit(text_tfidf.toarray()) # apply the tool to the vectorized BBC data.

pred_tfidf=AC_tfidf.labels_ # get the cluster labels.

pred_tfidf
```

```
array([6, 2, 4, ..., 6, 3, 5], dtype=int64)
```

لاحظ أن قيمة ground-truth (بيانات الحقيقة المعتمدة) من **القسم الإخباري** (News Section) في كل مستند لم تُستخدَم على الإطلاق في هذه العملية. وبدلًا من ذلك، عولجت عملية التجميع استنادًا إلى نص محتوى كل وثيقة على حده. إنَّ قيم بيانات الحقيقة المعتمدة مفيدة في التطبيق العملي، فهي تتيح التحقق من صحة نتائج التجميع. وقيم بيانات الحقيقة المعتمدة الحالية موجودة في قائمة bbc_labels (قيم_ هيئة الإذاعة البريطانية).

يَستخدِم المقطع البرمجي التالي قيم بيانات الحقيقة المعتمدة وثلاثة دوال مختلفة لتسجيل النقاط من مكتبة **سكليرن** (Sklearn) لتقييم جودة تجميع البيانات:

- تكون قيم **مؤشر التجانس** (Homogeneity Score) بين 0 و 1 ويمكن زيادة هذه القيم عندما تكون كل النقاط في كل عنقود لها قيمة بيانات الحقيقة المعتمدة. وبالمثل، يحتوي كل عنقود على نقاط البيانات وحيدة التصنيف.
- تكون قيمة **مؤشر راند المُعدل** (Adjusted Rand Score) بين 0.5- و 1.0 ويمكن زيادة هذه القيم عندما تقع كل نقاط البيانات ذات القيم نفسها في العنقود نفسه وكل نقاط البيانات ذات القيم المختلفة في عناقيد مختلفة.
- تكون قيمة **مؤشر الاكتمال** (Completeness Score) بين 0 و 1 ويمكن زيادة هذه القيمة بتعيين كل نقاط البيانات من تصنيف مُحدد في العنقود نفسه.

```
from sklearn.metrics import homogeneity_score,adjusted_rand_
score,completeness_score

print('\nHomogeneity score:',homogeneity_score(bbc_labels,pred_tfidf))
print('\nAdjusted Rand score:',adjusted_rand_score(bbc_labels,pred_tfidf))
print('\nCompleteness score:',completeness_score(bbc_labels,pred_tfidf))
```

```
Homogeneity score: 0.6224333236569846

Adjusted Rand score: 0.4630492696176891

Completeness score: 0.5430590192420555
```

المؤشر أقرب إلى 1 وهذا يعني أن مجموعة النصوص في العنقود تنتمي إلى قيمة واحدة.

المؤشر أقرب إلى 1 وهذا يعني إنشاء روابط أفضل بين العناقيد والقيم؛ كلٌّ على حده.

لاستكمال تحليل البيانات، يُعاد تجميع البيانات باستخدام 5 عناقيد، بالتساوي مع العدد الحقيقي لقيم ground-truth (بيانات الحقيقة المعتمدة):

```
AC_tfidf=AgglomerativeClustering(linkage='ward',n_clusters=5)
AC_tfidf.fit(text_tfidf.toarray())
pred_tfidf=AC_tfidf.labels_

print('\nHomogeneity score:',homogeneity_score(bbc_labels,pred_tfidf))
print('\nAdjusted Rand score:',adjusted_rand_score(bbc_labels,pred_tfidf))
print('\nCompleteness score:',completeness_score(bbc_labels,pred_tfidf))
```

```
Homogeneity score: 0.528836079209762

Adjusted Rand score: 0.456284128836283 83

Completeness score: 0.6075627851312266
```

نظرًا لقدرة التجميع الهرمي على إيجاد العدد الحقيقي من القيم، وتوفير مؤشر اكتمال أكثر دقة، ستحصل على عملية تجميع أفضل من حيث تمثيل البيانات.

على الرغم من أن نتائج المؤشر تُظهر أن التجميع التكتلي باستخدام البرمجة الاتجاهية **لتكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF) تحقق نتائج معقولة، إلا أنّه لا يزال بالإمكان تحسين دقة عملية التجميع. سيوضِّح القسم التالي كيف يمكن أن نحقق نتائج مبهرة باستخدام تقنيات البرمجة الاتجاهية المُستنِدة على الشبكات العصبية.

البرمجة الاتجاهية للكلمات باستخدام الشبكات العصبية
Word Vectorization with Neural Networks

البرمجة الاتجاهية **لتكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF) تستند إلى حساب تكرار الكلمات ومعالجتها عبر المُستنَدات في مجموعة البيانات. بالرغم من أن هذا يحقق نتائج جيدة، إلا أنّ القيود الكبيرة تعيب الطرائق المستنِدة إلى التكرار. فهي تتجاهل تمامًا العلاقة الدلالية بين الكلمات. على سبيل المثال، على الرغم من أن كلمتي trip (نزهة) وjourney (رحلة) مترادفتان، إلا أنّ البرمجة الاتّجاهيّة المُستنِدة إلى التّكرار ستتعامل معهما باعتبارهما كلمتان منفصلتان تمامًا ولهما خصائص مستقلة. وبالمثل، بالرغم من أن كلمتَي apple (تفاحة) و fruit (فاكهة) مترابطتان دلاليًا؛ لأن التفاح نوع من الفاكهة إلا أنّ ذلك لن يؤخذ بعين الاعتبار أيضًا.

تؤثر هذه القيود كثيرًا على التطبيقات التي تستخدِم هذا النوع من البرمجة الاتجاهية. فكّر في الجملتين التاليتين:

- I have a very high fever، so I have to visit a doctor (لديّ حمّى شديدة، ويجب عليّ زيارة الطبيب).
- My body temperature has risen significantly، so I need to see a healthcare professional (ارتفعت درجة حرارة جسمي كثيرًا، ويجب عليّ زيارة أخصائي الرعاية الصحية).

بالرغم من أن الجملتين تصفان الحالة نفسها إلا أنهما لا تتشاركان أي كلمات دلالية. ولذلك، ستفشل خوارزميات التجميع المُستنِدة إلى **تكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF) أو أي برمجة اتجاهية (تستند إلى التكرار) في رؤية التشابه بين الكلمات، ومن المحتمل ألا تضعها في نفس العنقود.

نموذج الكلمة إلى المتَّجَه Word2Vec

يمكن معالجة هذه القيود بالطرائق التي تأخذ بعين الاعتبار التشابه الدلالي بين الكلمات. إحدى الطرائق الشهيرة المُتبعة في هذا الصدد هي **نموذج الكلمة إلى المتَّجَه** (Word2Vec) التي تَستخدِم بُنية تَستند إلى الشبكات العصبية. يَستند **نموذج الكلمة إلى المتَّجَه** (Word2Vec) إلى فكرة أن الكلمات المتشابهة دلاليًا تُحاط بكلمات مماثلة في السياق نفسه. ولذلك، نجد الشبكات العصبية تستخدِم التضمين الخفي لكل كلمة للتنبؤ بالسياق، مع ضرورة إنشاء الروابط بين الكلمات والتضمينات الشبيهة. عمليًا، يخضع **نموذج الكلمة إلى المتَّجَه** (Word2Vec) للتدريب المُسبَق على ملايين المُستنَدات لتعلُّم التضمين عالي الدقة للكلمات. يمكن تحميل النماذج المُدرَّبة مسبَقًا واستخدامها في التطبيقات المُستندة إلى النصوص. يَستخدِم المقطع البرمجي التالي مكتبة **جينسم** (Gensim) لتحميل نموذج شهير مُدرَّب مسبَقًا على مجموعة كبيرة جدًا من **أخبار قوقل** (Google News):

الكلمات المُستبعَدة (Stopwords):
الكلمات المُستبعَدة هي كلمات شائعة في اللغات تُستبعد عادةً أثناء المعالجة المُسبَقة للنصوص ضمن مهام **معالجة اللغات الطبيعية** (NPL) مثل البرمجة الاتجاهية للكلمات. هذه الكلمات تشمل أدوات التعريف، وحروف العطف، وحروف الجر، والكلمات التي لا تكون مفيدة لتحديد معنى النصّ، أو سياقه.

التضمين (Embedding):
التضمين يُعبِّر عن الكلمات أو الرموز في فضاء المتَّجَه المستمر حيث ترتبط الكلمات المتشابهة دلاليًا مع النقاط القريبة.

```
import gensim.downloader as api
model_wv = api.load('word2vec-google-news-300')
fox_emb=model_wv['fox']
print(len(fox_emb))
```

```
300
```

هذا النموذج يربط كل كلمة بتضمين مكوّن من 300 بُعد.

الأبعاد العشرة الأولى للتضمين العددي لكلمة fox (ثعلب) موضحة بالأسفل:

```
fox_emb[:10]
```

```
array([-0.08203125, -0.01379395, -0.3125    , -0.04125977,  0.05493164,
       -0.12988281, -0.10107422, -0.00164795,  0.15917969,  0.12402344],
      dtype=float32)
```

يَستخدِم النموذج تضمينات الكلمات لتقييم درجة التشابه. فكِّر في المثال التالي حيث تُظهر المقارنة بين كلمة car (السيارة) والكلمات الأخرى درجة التشابة من خلال تناقص قيم التشابة. علمًا بأن قيم التشابه تقع دومًا بين 0 و 1.

```
pairs = [
    ('car', 'minivan'),
    ('car', 'bicycle'),
    ('car', 'airplane'),
    ('car', 'street'),
    ('car', 'apple'),
]
for w1, w2 in pairs:
    print(w1, w2, model_wv.similarity(w1, w2))
```

```
car minivan 0.69070363
car bicycle 0.5364484
car airplane 0.42435578
car street 0.33141237
car apple 0.12830706
```

يُمكن استخدام المقطع البرمجي التالي للعثور على الكلمات الخمسة المشابهة لإحدى الكلمات:

```
print(model_wv.most_similar(positive=['apple'], topn=5))
```

```
[('apples', 0.720359742641449), ('pear', 0.6450697183609009),
('fruit', 0.6410146355628967), ('berry', 0.6302295327186584), ('pears',
0.613396167755127)]
```

يُمكن استخدام التصوير في التحقق من صحة تضمينات هذا النموذج المُدرَّب مُسبقًا، ويُمكن تحقيق ذلك عبر:

- تحديد نماذج الكلمات من مجموعة بيانات هيئة الإذاعة البريطانية.
- استخدام **تضمين المجاور العشوائي الموزَّع على شكل T** (T-SNE) لتخفيض التضمين ذي الـ 300 بُعدٍ لكل كلمة إلى نقطة ثنائية الأبعاد.
- تصوير النقاط في مُخطَّط الانتشار في الفضاء ثنائي الأبعاد.

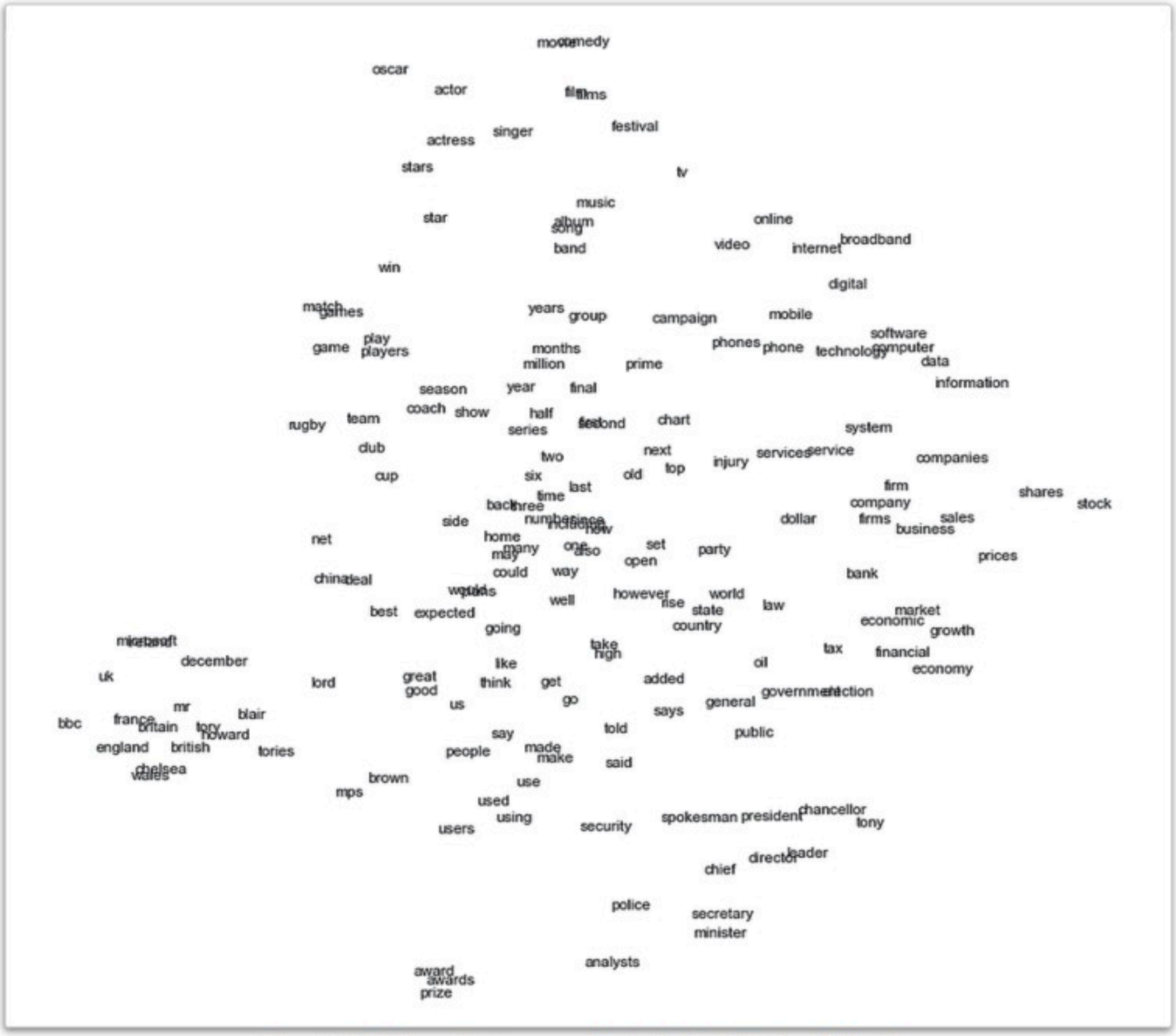
```
%%capture
import nltk # import the nltk library for nlp.
import re # import the re library for regular expressions.
import numpy as np # used for numeric computations
from collections import Counter # used to count the frequency of elements in a given list
from sklearn.manifold import TSNE # Tool used for Dimensionality Reduction.

# download the 'stopwords' tool from the nltk library. It includes very common words for different
languages
nltk.download('stopwords')

from nltk.corpus import stopwords # import the 'stopwords' tool.

stop=set(stopwords.words('english')) # load the set of english stopwords.
```

تُستخدَم الدالة الآتية لاحقًا لتحديد عينة من الكلمات التمثيلية من مجموعة بيانات هيئة الإذاعة البريطانية. يُحدِّد المقطع البرمجي الكلمات الخمسين الأكثر تكرارًا على وجه التحديد من الأقسام الإخبارية الخمسة لهيئة الإذاعة البريطانية مع استثناء **الكلمات المُستبعَدة** (Stopwords) وهي الكلمات الإنجليزية الشائعة جدًا والكلمات التي لم تُضمَّن في **نموذج الكلمة إلى المتَّجه** (Word2Vec) المُدرَّب مسبقًا.

بعض الكلمات الإنجليزية الشائعة التي تُعدُّ كلمات مُستبعَدة (Stopwords) هي a (أ) و the (ال) و is (يكون) و are (يكونون).

```
def get_sample(bbc_docs:list,
               bbc_labels:list
               ):

    word_sample=set() # a sample of words from the BBC dataset

    # for each BBC news section
    for label in ['business', 'entertainment', 'politics', 'sport', 'tech']:

        # get all the words in this news section, ignore stopwords.
        # for each BBC doc and for each word in the BBC doc
        # if the word belongs to the label and is not a stopword and is included in the Word2Vec model
        label_words=[word for i in range(len(bbc_docs))
                          for word in re.findall(r'\b\w\w+\b',bbc_docs[i].lower())
                          if bbc_labels[i]==label and
                             word not in stop and
                             word in model_wv]

        cnt=Counter(label_words) # count the frequency of each word in this news section.

        # get the top 50 most frequent words in this section.
        top50=[word for word,freq in cnt.most_common(50)]
        # add the top50 words to the word sample.
        word_sample.update(top50)


    word_sample=list(word_sample) # convert the set to a list.
    return word_sample

word_sample=get_sample(bbc_docs,bbc_labels)
```

وأخيرًا، ستَستخدِم طريقة **تضمين المجاور العشوائي الموزَّع على شكل T** (T-SNE) لتخفيض التضمينات ذات الـ 300 بُعدٍ للكلمات في العينة ضمن النقاط ثنائية الأبعاد. بعدها، تُمثَّل النقاط في مُخطَّط انتشار بسيط.



شكل 3.22: تمثيل الكلمات الأكثر تكرارًا من مجموعة بيانات هيئة الإذاعة البريطانية

يُثبت المُخطَّط أن تضمينات **نموذج الكلمة إلى المتَّجه** (Word2Vec) تستنبط الارتباطات الدلالية بين الكلمات، كما يتضح من مجموعات الكلمات الواضحة مثل:

- economy (الاقتصاد)، economic (الاقتصادية)، business (الأعمال)، financial (المالية)، sales (المبيعات)، bank (المصرف)، firm (الشركة)، firms (الشركات).
- Internet (الإنترنت)، mobile (الهاتف المحمول)، phones (الهواتف)، phone (الهاتف)، broadband (النطاق العريض)، online (متصل)، digital (رقمي).
- actor (ممثل)، actress (ممثلة)، film (فيلم)، comedy (كوميدي)، films (أفلام)، festival (مهرجان)، band (فرقة)، movie (فيلم).
- game (لعبة)، team (فريق)، match (مباراة)، players (لاعبون)، coach (مدرِّب)، injury (إصابة)، club (نادي)، rugby (الرجبي).

البرمجة الاتجاهية للجُمل باستخدام التعلُّم العميق
Sentence Vectorization with Deep Learning

على الرغم من إمكانية استخدام **نموذج الكلمة إلى المتَّجَه** (Word2Vec) في نمذجة الكلمات الفردية، يتطلب التجميع البرمجة الاتجاهية للنص بأكمله. إحدى الطرائق الأكثر شهرة لتحقيق ذلك هي تمثيلات **ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) المُستنِدة إلى منهجية التعلُّم العميق.

تمثيلات الترميز ثنائية الاتجاه من المحولات
Bidirectional Encoder Representations from Transformers (BERT)

تمثيلات الترميز ثنائية الاتجاه من المحولات (BERT) هي نموذج تمثيل لغوي قوي طورته شركة قوقل، ويعدُّ التدريب المُسبَق والضبط الدقيق عاملان رئيسان وراء قدرة **تمثيلات الترميز ثنائية الاتجاه من المحولات** (BERT) على تطبيق نقل التعلُّم، أي القدرة على الاحتفاظ بالمعلومات حول مشكلة ما والاستفادة منها في حلِّ مشكلة أخرى، ويتمّ التدريب المُسبَق عبر تغذية النموذج بكمية هائلة من البيانات غير المُعنوَنة لعدة مهام، مثل التنبؤ اللغوي المُقنَّع (إخفاء الكلمات العشوائية في مُدخَلات النصوص والمُهِمَّة هي التنبؤ بهذه الكلمات). يُهيِّئ نموذج **تمثيلات الترميز ثنائية الاتجاه من المحولات** (BERT) المتغيرات المُدرَّبة مُسبقًا للضبط الدقيق، كما تُستخدَم مجموعات البيانات المُعنوَنة من المهام النهائية لضبط دقة عمل النموذج، ويكون لكل مُهِمَّة نهائية نماذج دقيقة منفصلة، برغم أنها مُهيَّئة بالمتغيرات المُدرَّبة نفسها مُسبقًا. على سبيل المثال، تختلف عملية الضبط الدقيق لنموذج تحليل المشاعر عن نموذج الإجابة على الأسئلة. ومن المهم معرفة أن الفروقات في بنية النماذج تصبح ضئيلة أو منعدمة بعد خطوة ضبط الدقة.

تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات SBERT

تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT) هي الإصدار المُعدَّل من **تمثيلات الترميز ثنائية الاتجاه من المحولات** (BERT). تُدرَّب **تمثيلات الترميز ثنائية الاتجاه من المحولات** (BERT) مثل **نموذج الكلمة إلى المتَّجَه** (Word2Vec) للتنبؤ بالكلمات بناءً على سياق الجُمل الواردة بها. ومن ناحية أخرى، تُدرَّب **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) للتنبؤ بما إذا كانت جملتان متشابهتين دلاليًا. تُستخدَم **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) لإنشاء تضمينات لأجزاء النصوص الأطول من الجُمل، مثل الفقرات، أو النصوص القصيرة، أو المقالات في مجموعة بيانات هيئة الإذاعة البريطانية محل الدراسة في هذه الوحدة. بالرغم من أن النماذج الثلاث تستند جميعها إلى الشبكات العصبية، إلا أن **تمثيلات الترميز ثنائية الاتجاه من المحولات** (BERT) و**تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) تنفذان بُنية مختلفة بشكل كبير وأكثر تعقيدًا من **نموذج الكلمة إلى المتَّجَه** (Word2Vec).

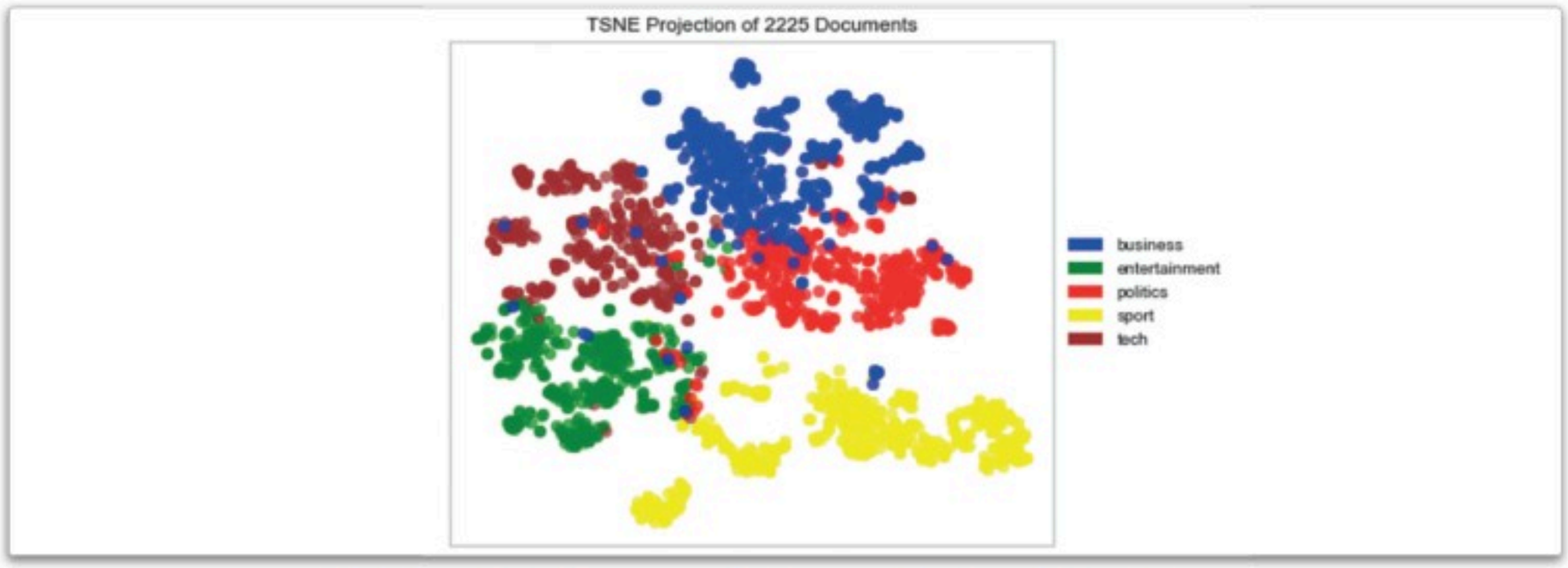
مكتبة الجُمل والمحولات Sentence_transformers Library

تُطبق مكتبة **الجُمل والمحولات** (sentence_transformers) الوظائف الكاملة لنموذج تمثيلات **ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT). تأتي المكتبة بالعديد من نماذج **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) المُدرَّبة مُسبقًا؛ كلٌ منها مُدرَّب على مجموعة بيانات مختلفة ولتحقيق أهداف مختلفة. يعمل المقطع البرمجي التالي على تحميل أحد النماذج العامة الشهيرة المُدرَّبة مُسبقًا، ويستخدمها لإنشاء تضمينات للمستندات في مجموعة بيانات هيئة الإذاعة البريطانية:

```
%%capture
!pip install sentence_transformers
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2') # load the pre-trained model.

text_emb = model.encode(bbc_docs) # embed the BBC documents.
```

لقد استخدمت في وقت سابق في هذه الوحدة أداة تضمين المجاور العشوائي الموزع على شكل T والتي هي (TSNEVisualizer)، لتصوير المُستندات المُمثلة بالمتُّجَهات المُنتجة باستخدام أداة تكرار المصطلح-تكرار المستند العكسي (TF-IDF). يمكن الآن استخدامها للتضمينات المُنتَجة بواسطة **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT):

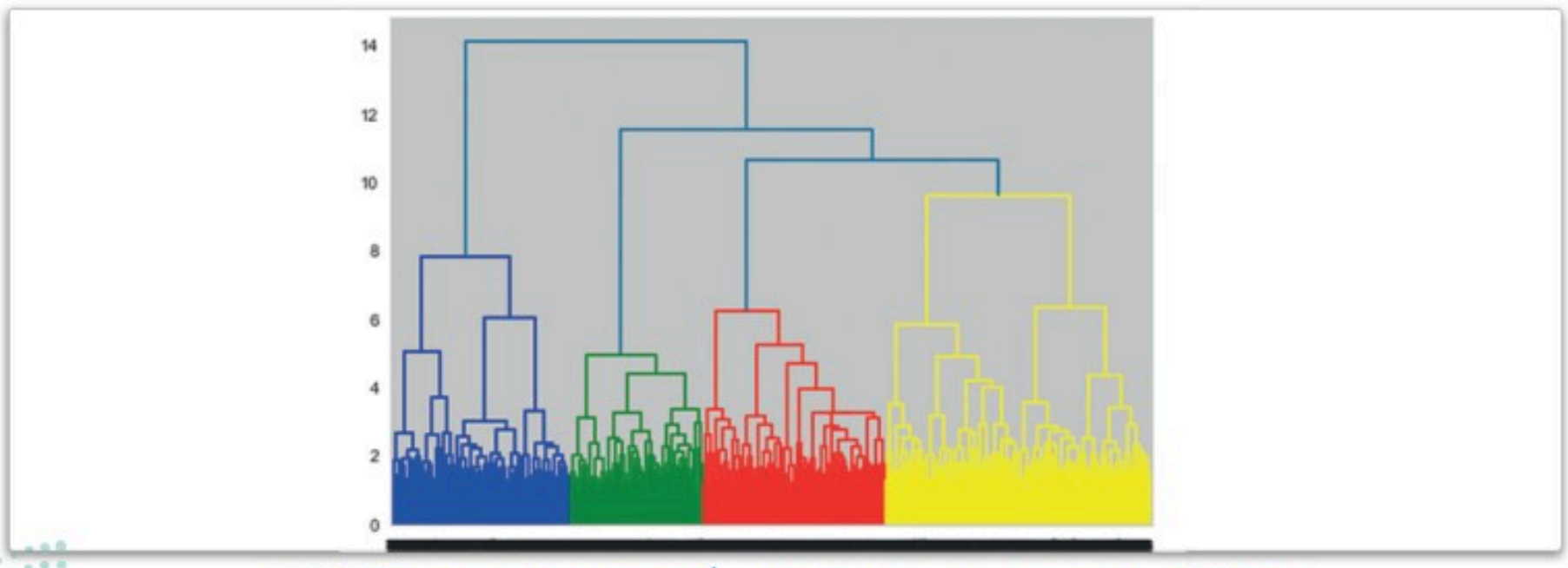
```
tsne = TSNEVisualizer(colors=['blue','green','red','yellow','brown'])
tsne.fit(text_emb,bbc_labels)
tsne.show();
```



شكل 3.23: إسقاط تضمين المجاور العشوائي الموزَّع على شكل T (T-SNE) للتضمينات المُنتجة بواسطة تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT)

يوضِّح الشكل أن **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) تؤدي إلى فصل أكثر وضوحًا للأقسام الإخبارية المختلفة مع عدد أقل من الشوائب من **تكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF). الخطوة التالية هي استخدام التضمينات لتدريب خوارزمية التجميع التكتلي:

```
plt.figure() # create a new figure.

# iteratively merge points and clusters until all points belong to a single cluster. Return the the linkage of
the produced tree.
linkage_emb=hierarchy.linkage(text_emb, method='ward')

hierarchy.dendrogram(linkage_emb) # visualize the linkage.
plt.show() # show the figure.
```



شكل 3.24: الرسم الشجري الهرمي لتمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT)

كما هو موضّح في الشكل 3.24، فإن أداة الرسم الشجري تشير إلى 4 عناقيد، كل واحد منها مُميز بلون مختلف. يَستخدِم المقطع البرمجي التالي هذا المقترح لحساب العناقيد وحساب مقاييس التقييم:

```
AC_emb=AgglomerativeClustering(linkage='ward',n_clusters=4)
AC_emb.fit(text_emb)
pred_emb=AC_emb.labels_

print('\nHomogeneity score:',homogeneity_score(bbc_labels,pred_emb))
print('\nAdjusted Rand score:',adjusted_rand_score(bbc_labels,pred_emb))
print('\nCompleteness score:',completeness_score(bbc_labels,pred_emb))
```

```
Homogeneity score: 0.6741395570357063

Adjusted Rand score: 0.6919474005627763

Completeness score: 0.7965514907905805
```

إذا كانت البيانات قد تم إعادة تجميعها باستخدام العدد الصحيح من 5 عناقيد، فالعنقود الأصفر المُحدد بالشكل أعلاه سينقسم إلى اثنين، وستكون النتائج على النحو التالي:

```
AC_emb=AgglomerativeClustering(linkage='ward',n_clusters=5)
AC_emb.fit(text_emb)
pred_emb=AC_emb.labels_

print('\nHomogeneity score:',homogeneity_score(bbc_labels,pred_emb))
print('\nAdjusted Rand score:',adjusted_rand_score(bbc_labels,pred_emb))
print('\nCompleteness score:',completeness_score(bbc_labels,pred_emb))
```

```
Homogeneity score: 0.7865655030556284

Adjusted Rand score: 0.8197670431956582

Completeness score: 0.7887580797775077
```

تُظهِر النتائج أن استخدام **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) في البرمجة الاتجاهية للنصوص يَنتج عنه نتائج تجميع مُحسَّنة بالمقارنة مع **تكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF). إذا كان عدد العناقيد هو 5 **لتكرار المصطلح-تكرار المُستنَد العكسي** (TF-IDF) (القيمة الصحيحة) و4 عناقيد **لتمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT)، فإن المقاييس الثلاثة **لتمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) لا تزال هي الأعلى بفارق كبير. ثم تزداد الفجوة إذا كان العدد 5 لكلٍ من الطريقتين.

وهذا يُعدُّ دليلًا على إمكانات الشبكات العصبية، التي تسمح لها بُنيتها المتطورة بفهم الأنماط الدلالية المُعقدة في البيانات النصيّة.

تمرينات

1

حدّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. في التعلُّم غير الموجَّه تُستخدم مجموعات البيانات المُعنوَنة لتدريب النموذج.	○	○
2. يتطلب التعلُّم غير الموجَّه البرمجة الاتجاهية للبيانات.	○	○
3. تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT) تُعدُّ أفضل من تكرار المصطلح-تكرار المستند العكسي (TF-IDF) للبرمجة الاتجاهية للكلمات.	○	○
4. يَتبع التجميع التكتلي منهجية التصميم من أعلى إلى أسفل لتحديد العناقيد.	○	○
5. تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT) مُدرَّبة للتنبؤ بما إذا كانت جملتان مختلفتين دلاليًا.	○	○

2 استعرِض بعض التطبيقات التي يُستخدم فيها تقليص الأبعاد، وصِف التقنيات المُستخدَمة فيه.

3 اشرح وظائف البرمجة الاتجاهية لمقياس تكرار المصطلح-تكرار المستند العكسي (TF-IDF).

4 لديك مصفوفة numPy تدعى 'Docs' تتضمن مستندًا نصيًّا واحدًا في كل صف. لديك كذلك مصفوفة labels تتضمن قيم كل مستند في Docs. أكمل المقطع البرمجي التالي بحيث تستخدِم نموذج تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT) المُدرَّب مُسبقًا لحساب تضمينات كل الوثائق في Docs، ثم استخدم أداة TSNEVisualizer تضمين المجاور العشوائي الموزَّع على شكل T لتصوير التضمينات في الفضاء ثنائي الأبعاد، باستخدام لون مختلف لكل واحد من القيم الأربعة المحتملة:

```
from sentence_transformers import _________________

from ____________ import TSNEVisualizer model = _____________('all-MiniLM-
L6-v2') # loads the pre-trained model.

docs_emb = model._____________(Docs) # embeds the docs

tsne = _________________(_________________=['blue','green','red','yellow'])

tsne._____________(_____________,_____________)

tsne.show();
```

5 أكمل المقطع البرمجي التالي بحيث تَستخدِم نموذج الكلمة إلى المتَّجَه (Word2Vec) لاستبدال كل كلمة في إحدى الجُمل بأخرى تكون أكثر شبهًا بها:

```
import gensim.downloader as _____________
import re

model_wv = _____________._____________('word2vec-google-news-300')
old_sentence='My name is John and I like basketball.'
new_sentence=''

for word in re._____________(r'\b\w\w+\b',old_sentence.lower()):

        replacement=model_wv._____________(positive=['apple'], _____________=1)[0]

        new_sentence+= _____________
sentence=new_sentence.strip()
```

رابط الدرس الرقمي

www.ien.edu.sa

الدرس الثالث

توليد النص

توليد اللغات الطبيعية Natural Language Generation (NLG)

توليد اللغات الطبيعية (NLG) هو أحد فروع **معالجة اللغات الطبيعية** (NLP) التي تركِّز على توليد النصوص البشرية باستخدام خوارزميات الحاسب. الهدف من **توليد اللغات الطبيعية** (NLG) هو توليد اللغات المكتوبة أو المنطوقة بصورة طبيعية ومفهومة للبشر دون الحاجة إلى تدخل بشري. توجد العديد من المنهجيات المختلفة لتوليد اللغات الطبيعية مثل: المنهجيات المُستنِدة إلى القوالب، والمُستنِدة إلى القواعد، والمُستنِدة إلى تعلُّم الآلة.

معالجة اللغات الطبيعية (Natural Language Processing-NLP):

معالجة اللغات الطبيعية (NLP) هو أحد فروع الذكاء الاصطناعي الذي يمنح أجهزة الحاسب القدرة على محاكاة اللغات البشرية الطبيعية.

توليد اللغات الطبيعية (Natural Language Generation-NLG):

توليد اللغات الطبيعية (NLG) هي عملية توليد النصوص البشرية باستخدام **الذكاء الاصطناعي** (AI).

شكل 3.25: مُخطَّط فِنْ (Venn) لمعالجة اللغات الطبيعية (NLP)

جدول 3.4: تأثير توليد اللغات الطبيعية

يُستخدم **توليد اللغات الطبيعية** (NLG) لتوليد المقالات والتقارير الإخبارية، والمحتوى المكتوب آليًا مما يوفّر الوقت، ويساعد الأشخاص في التركيز على المهام الإبداعية أو المهام عالية المستوى.	
يمكن الاستفادة من ذلك في تحسين كفاءة وفعالية روبوت الدردشة لخدمة العملاء وتمكينه من تقديم ردود طبيعية ومفيدة لأسئلتهم واستفساراتهم.	
يمكن الاستفادة من **توليد اللغات الطبيعية** (NLG) في تحسين إمكانية الوصول لذوي الإعاقة أو لذوي الحواجز اللغوية، بتمكينهم من التواصل مع الآلات بطريقة طبيعية وبديهية تناسبهم.	

هناك أربع أنواع من **توليد اللغات الطبيعية** (NLG):

توليد اللغات الطبيعية المبني على القوالب Template-Based NLG

يتضمن توليد اللغات الطبيعية المبنيّ على القوالب استخدام قوالب مُحدَّدة مُسبقًا تحدد بنية ومحتوى النص المتولِّد. تُزَوّد هذه القوالب بمعلومات مُحدَّدة لتوليد النص النهائي. تُعدُّ هذه المنهجية بسيطة نسبيًا وتحقق فعالية في توليد النصوص للمهام المُحدَّدة والمُعرَّفة جيدًا. من ناحية أخرى، قد تواجه صعوبة مع المهام المفتوحة أو المهام التي تتطلب درجة عالية من التباين في النص المُوَلَّد. على سبيل المثال، قالب تقرير حالة الطقس ربما يبدو كما يلي: Today in **[city]**, it is **[temperature]** degrees with **[weather condition]**. (اليوم في [المدينة]، درجة الحرارة هي [**درجة الحرارة**] مئوية و[**حالة الطقس**].).

توليد اللغات الطبيعية المبني على القواعد Rule-Based NLG

يَستخدِم توليد اللغات الطبيعية المبنيّ على القواعد مجموعة من القواعد المُحدَّدة مُسبقًا لتوليد النص. قد تحدِّد هذه القواعد طريقة تجميع الكلمات والعبارات لتشكيل الجُمل، أو كيفية اختيار الكلمات وفقًا للسياق المُستخدَمة فيه. عادةً تُستخدم هذه القواعد لتصميم روبوت الدردشة لخدمة العملاء. قد يكون من السهل تطبيق الأنظمة المبنية على القواعد. وفي بعض الأحيان قد تتسم بالجمود ولا تُولِّد مُخرَجات تبدو طبيعية.

توليد اللغات الطبيعية المبني على الاختيار Selection-Based NLG

يتضمن توليد اللغات الطبيعية المبنيّ على الاختيار تحديد مجموعة فرعية من الجُمل أو الفقرات لإنشاء ملخّص للنصّ الأصلي الأكبر حجمًا. بالرغم من أن هذه المنهجية لا تُولِّد نصوصًا جديدة، إلا أنها مُطبقَّة عمليًا على نطاق واسع؛ وذلك لأنّها تأخذ العينات من مجموعة من الجُمل المكتوبة بواسطة البشر، يمكن الحد من مخاطرة توليد النصوص غير المُتنبئ بها أو ضعيفة البنية. على سبيل المثال، مُولِّد تقرير الطقس المبنيّ على الاختيار قد يضم قاعدة بيانات من العبارات مثل: It is hot outside (الطقس حار بالخارج)، و The temperature is rising (درجة الحرارة ترتفع)، و Expect sunny skies (تنبؤات بطقس مُشمس).

توليد اللغات الطبيعية المبني على تعلُّم الآلة Machine Learning-Based NLG

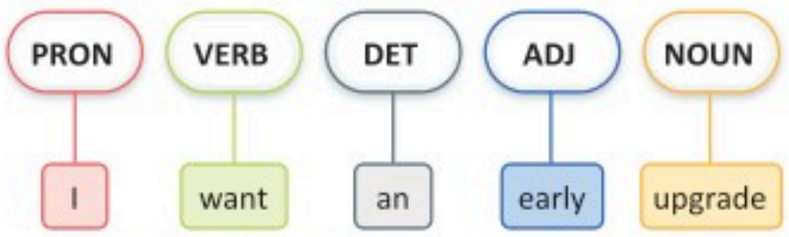
يتضمن توليد اللغات الطبيعية المبنيّ على تعلُّم الآلة تدريب نموذج تعلُّم الآلة على مجموعة كبيرة من بيانات النصوص البشرية. يتعلّم النموذج أنماط النصّ وبنيته، ومن ثَمّ يمكنه توليد النص الجديد الذي يشبه النص البشري في الأسلوب والمحتوى. قد تكون المنهجية أكثر فعالية في المهام التي تتطلب درجة عالية من التباين في النص المُولَّد. وقد تتطلب المنهجية مجموعات أكبر من بيانات التدريب والموارد الحسابية.

استخدام توليد اللغات الطبيعية المبني على القوالب Using Template-Based NLG

توليد اللغات الطبيعية المبنيّ على القوالب بسيط نسبيًا وقد يكون فعالًا في توليد النصوص للمهام المُحدَّدة والمُعرَّفة مثل إنشاء التقارير أو توصيف البيانات. إحدى مميزات توليد اللغات الطبيعية المبنيّ على القوالب هو سهولة التطبيق والصيانة. يُصمِّم الأشخاص القوالب، دون الحاجة إلى خوارزميات تعلُّم الآلة المُعقدَة أو مجموعات كبيرة من بيانات التدريب. وهذا يجعل توليد اللغات الطبيعية المبنيّ على القوالب هو الخيار المناسب للمهام التي تكون ذات بنية ومحتوى نصّ محدّدين، دون الحاجة إلى إجراء تغييرات كبيرة. تَستنِد قوالب **توليد اللغات الطبيعية** (NLG) إلى أي بُنية لغوية مُحدَّدة مُسبقًا. إحدى الممارسات الشائعة هي إنشاء القوالب التي تتطلب كلمات بوسوم محددة كجزء من الكلام لإدراجها في الفراغات المُحدَّدة ضمن الجملة.

وسوم أقسام الكلام Part of Speech (POS) Tags

وسوم **أقسام الكلام** (Part Of Speech)، التي تُعرَّف كذلك باسم وسوم POS هي قيم تُخصَّص للكلمات في النص للإشارة إلى البناء النحوي للكلمات، أو جزء الكلام في الجملة. على سبيل المثال، قد تكون الكلمة اسمًا أو فعلًا أو صفةً أو ظرفًا، إلخ، وتُستخدَم وسوم أقسام الكلام في **معالجة اللغات الطبيعية** (NLP) لتحليل بنية النصّ وفهم معناه.



شكل 3.26: مثال على عملية وسم أقسام الكلام

تحليل بِناء الجُمل Syntax Analysis

يُستخدم تحليل بِناء الجُمل عادةً إلى جانب وسوم **أقسام الكلام** (POS) في توليد اللغات الطبيعية المبنيّ على القوالب لضمان قدرة القوالب على توليد النصوص الواقعية. يتضمن تحليل بِناء الجُمل التعرّف على أجزاء الكلام في الجُمل، والعلاقات بينها لتحديد البناء النحوي للجُملة. تتضمن الجُملة أنواعًا مختلفة من عناصر بناء الجُملة، مثل:

- **الفعل** (Predicate) هو قسم الجُملة الذي يحتوي على الفعل. وهو عادةً يعبر عمّا يقوم به الفاعل أو عمّا يحدث.
- **الفاعل** (Subject) هو قسم الجُملة الذي يُنفّذ الفعل.
- **المفعول به** (Direct Object) هو اسم أو ضمير يشير إلى الشخص أو الشيء الذي يتأثر مباشرةً بالفعل.

يَستخدِم المقطع البرمجي التالي مكتبة **ووندروردز** (Wonderwords) التي تتبع منهجية بِناء الجُمل لعرض بعض الأمثلة على توليد اللغات الطبيعية المبنيّ على القوالب.

```
%%capture

!pip install wonderwords
# used to generate template-based randomized sentences
from wonderwords.random_sentence import RandomSentence

# make a new generator with specific words
generator=RandomSentence(
                            # specify some nouns
                            nouns=["lion", "rabbit", "horse","table"],
                            verbs=["eat","run","laugh"], # specify some verbs.
                            adjectives=['angry','small']) # specify some adjectives.

# generates a sentence with the following template: [subject (noun)] [predicate (verb)]
generator.bare_bone_sentence()
```

```
'The table runs.'
```

```
# generates a sentence with the following template:
# the [(adjective)] [subject (noun)] [predicate (verb)] [direct object (noun)]
generator.sentence()
```

```
'The small lion runs rabbit.'
```

توضح الأمثلة بالأعلى أنه، بينما يُستخدَم توليد اللغات الطبيعية المبنيّ على القوالب لتوليد الجُمل وفق بُنية مُحدَّدة ومُعتمدة مُسبقًا، إلا أنّ هذه الجُمل قد لا تكون ذات مغزىً عملي. وعلى الرغم من إمكانية تحسين دقة النتائج إلى حدٍ كبير بتحديد قوالب متطورة ووضع المزيد من القيود على استخدام المفردات، إلا أن هذه المنهجية غير عملية لتوليد النصوص الواقعية على نطاقٍ واسع. فبدلًا من إنشاء القوالب المُحدَّدة مُسبقًا، تُستخدَم المنهجية الأخرى لتوليد اللغات الطبيعية القائمة على القوالب البنيةَ والمفرداتِ نفسها المُكوِّنة لأي جملة حقيقية كقالب ديناميكي متغير. تتبنى دالة paraphrase( ) هذه المنهجية.

دالة Paraphrase( )

تُقسِّم الدالة في البداية النص المُكوَّن من فقرة إلى مجموعة من الجُمل. ثم تحاول استبدال كل كلمة في الجُملة بكلمة أخرى متشابهة دلاليًا. يُقيَّم التشابه الدلالي بواسطة **نموذج الكلمة إلى المتَّجه** (Word2Vec) الذي درسته في الدرس السابق. قد يوصي نموذج الكلمة إلى المتَّجه (Word2Vec) باستبدال الكلمة في الجملة بكلمة أخرى مشابهة لها، مثل: استبدال apple (تفاحة) بـ apples (تفاح)، ولتجنب مثل هذه الحالات تُستخدم دالة مكتبة fuzzywuzzy الشهيرة لتقييم تشابه المُفردات بين الكلمة الأصلية والكلمة البديلة.

الدالة نفسها مُوضحَّة بالأسفل:

```
def paraphrase(text:str, # text to be paraphrased
               stop:set, # set of stopwords
               model_wv,# Word2Vec Model
               lexical_sim_ubound:float, # upper bound on lexical similarity
               semantic_sim_lbound:float # lower bound on semantic similarity
              ):

    words=word_tokenize(text) # tokenizes the text to words

    new_words=[] # new words that will replace the old ones.

    for word in words: # for every word in the text

        word_l=word.lower() # lower-case the word.

        # if the word is a stopword or is not included in the Word2Vec model, do not try to replace it.
        if word_l in stop or word_l not in model_wv:
            new_words.append(word) # append the original word

        else: # otherwise

            # get the 10 most similar words, as per the Word2Vec model.
            # returned words are sorted from most to least similar to the original.
            # semantic similarity is always between 0 and 1.
            replacement_words=model_wv.most_similar(positive=[word_l],
topn=10)
            # for each candidate replacement word
            for rword, sem_sim in replacement_words:
                # get the lexical similarity between the candidate and the original word.
                # the partial_ratio function returns values between 0 and 100.
                # it compares the shorter of the two words with all equal-sized substrings
                # of the original word.
                lex_sim=fuzz.partial_ratio(word_l,rword)

                # if the lexical sim is less than the bound, stop and use this candidate.
                if lex_sim<lexical_sim_ubound:
                    break
```

fuzz تشير إلى مكتبة fuzzywuzzy

```
            # quality check: if the chosen candidate is not semantically similar enough to
            # the original, then just use the original word.
            if sem_sim<semantic_sim_lbound:
                new_words.append(word)
            else: # use the candidate.
                new_words.append(rword)

    return ' '.join(new_words) # re-join the new words into a single string and return.
```

المُخرَج هو إصدار مُعاد صياغته من النص المُدخَل.

يُستخدَم المقطع البرمجي التالي لاستيراد كل الأدوات اللازمة لدعم دالة ( )paraphrase وفي المربع الأبيض أدناه، تحصل على مُخرَج طريقة **إعادة الصياغة** (Paraphrase) للنص المُسند إلى المتغير text:

```
%%capture

import gensim.downloader as api # used to download and load a pre-trained Word2Vec model
model_wv = api.load('word2vec-google-news-300')

import nltk
# used to split a piece of text into words. Maintains punctuations as separate tokens
from nltk import word_tokenize
nltk.download('stopwords') # downloads the stopwords tool of the nltk library
# used to get list of very common words in different languages
from nltk.corpus import stopwords
stop=set(stopwords.words('english')) # gets the list of english stopwords

!pip install fuzzywuzzy[speedup]
from fuzzywuzzy import fuzz

text='We had dinner at this restaurant yesterday. It is very close to my
house. All my friends were there, we had a great time. The location is
excellent and the steaks were delicious. I will definitely return soon, highly
recommended!'
# parameters: target text, stopwords, Word2Vec model, upper bound on lexical similarity, lower bound
on semantic similarity
paraphrase(text, stop, model_wv, 80, 0.5)
```

```
'We had brunch at this eatery Monday. It is very close to my bungalow. All
my acquaintances were there, we had a terrific day. The locale is terrific
and the tenderloin were delicious. I will certainly rejoin quickly, hugely
advised!'
```

كما في المنهجيات الأخرى المُستنِدة إلى القوالب، يمكن تحسين النتائج بإضافة المزيد من القيود لتصحيح بعض البدائل الأقل وضوحًا والمذكورة في الأعلى. ومع ذلك، يوضِّح المثال أعلاه أنه يُمكن باستخدام هذه الدالة البسيطة توليد نصوص واقعية.

استخدام توليد اللغات الطبيعية المبني على الاختيار

Using Selection-Based NLG

في هذا القسم، ستستعرض منهجية عملية لاختيار نموذج من الجُمل الفرعية من وثيقة مُحدَّدة. هذه المنهجية تُجسِد استخدام ومزايا توليد اللغات الطبيعية المبنيّ على الاختيار يستند إلى لبنتين رئيستين:

- **نموذج الكلمة إلى المتَّجَه** (Word2Vec) المُستخدَم لتحديد أزواج الكلمات المتشابهة دلاليًا.
- مكتبة Networkx الشهيرة ضمن لغة البايثون المُستخدَمة لإنشاء ومعالجة أنواع مختلفة من بيانات الشبكة.

النَّص المُدخَل الذي سيُستخدم في هذا الفصل هو مقالة إخبارية نُشرت بعد المباراة النهائية لكأس العالم 2022.

```
# reads the input document that we want to summarize
with open('article.txt',encoding='utf8',errors='ignore') as f: text=f.read()
text[:100] # shows the first 100 characters of the article
```

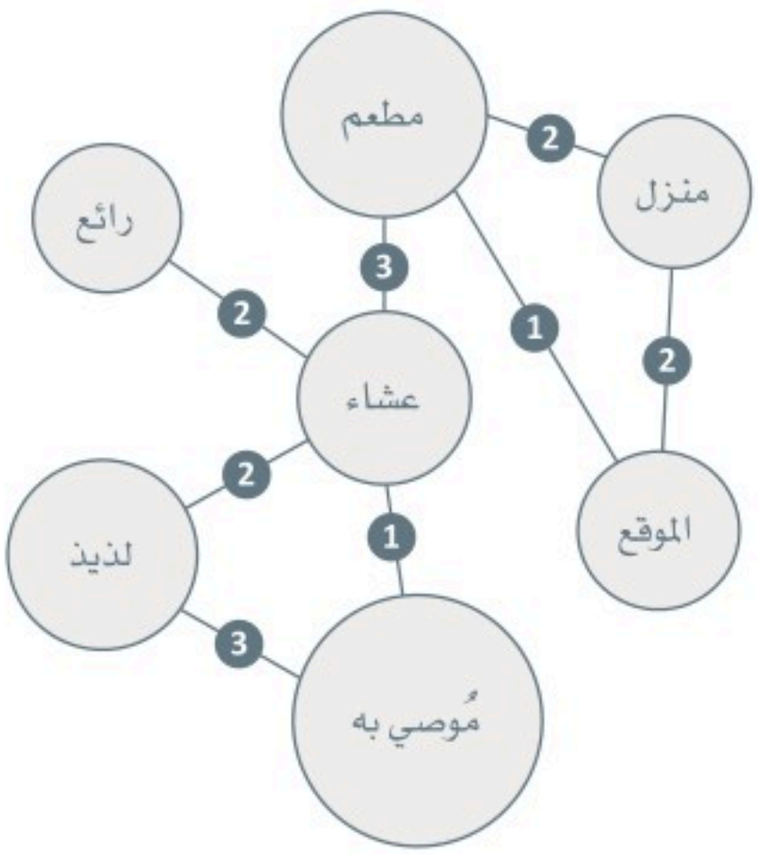
```
'It was a consecration, the spiritual overtones entirely appropriate.
Lionel Messi not only emulated '
```

في البداية، يُرمَّز النص باستخدام مكتبة re والتعبير النمطي نفسه المُستخدَم في الوحدات السابقة:

```
import re # used for regular expressions

# tokenize the document, ignore stopwords, focus only on words included in the Word2Vec model.
tokenized_doc=[word for word in re.findall(r'\b\w\w+\b',text.lower()) if word
not in stop and word in model_wv]

# get the vocabulary (set of unique words).
vocab=set(tokenized_doc)
```

مكتبة Networkx

يمكن الآن نمذجة مفردات المُستنَد في **مُخطَّط موزون** (Weighted Graph). تُوفر مكتبة Networkx في لغة البايثون مجموعة واسعة من الأدوات لإنشاء وتحليل المُخطَّطات. في توليد اللغات الطبيعية المبنيّ على الاختيار، يُساعد تمثيل مفردات الوثيقة في مُخطَّط موزون في تحديد العلاقات بين الكلمات وتسهيل اختيار العبارات والجُمل ذات الصلة. في المُخطَّط الموزون، تُمثل كل عُقدة كلمةً أو مفهومًا، وتُمثل الحواف بين العُقد العلاقات بين هذه المفاهيم. تُعبر الأوزان على الحواف عن قوة هذه العلاقات، مما يسمح لنظام توليد اللغات الطبيعية بتحديد المفاهيم الأقوى ارتباطًا. عند توليد النصوص، يُستخدم المُخطَّط الموزون للبحث عن العبارات والجُمل استنادًا إلى العلاقات بين الكلمات. على سبيل المثال، قد يَستخدِم النظام المُخطَّط للبحث عن الكلمات والعبارات الأكثر ارتباطًا لوصف كيان مُحدَّد ثم استخدام هذه الكلمات لتحديد الجُملة الأكثر ملاءمةً من قاعدة بيانات النظام.



شكل 3.27: مثال على مُخطَّط موزون لـ Networkx

دالة Build_graph( )

تُستخدم دالة Build_graph( ) مكتبة NetworkX لإنشاء مُخطَّط يتضمن:

- عُقدة واحدة لكل كلمة ضمن مفردات محددة.
- حافة بين كل كلمتين. الوزن على الحافة يساوي التشابه الدلالي بين الكلمات، المحسوب بواسطة أداة Doc2Vec وهي أداة معالجة اللغات الطبيعية المُخصصة لتمثيل النصّ كمتَّجَه وهي تعميم لمنهجية **نموذج الكلمة إلى المتَّجَه** (Word2Vec).

تَرسم الدالة مخطّطًا ذا عُقدة واحدة لكل كلمة في المفردات المُحدَّدة. توجد كذلك حافة بين عُقدتين إذا كان تشابه **نموذج الكلمة إلى المتَّجَه** (Word2Vec) أكبر من الحد المُعطى.

```
# tool used to create combinations (e.g. pairs, triplets) of the elements in a list
from itertools import combinations
import networkx as nx # python library for processing graphs

def build_graph(vocab:set, # set of unique words
                model_wv # Word2Vec model
               ):
    # gets all possible pairs of words in the doc
    pairs=combinations(vocab,2)

    G=nx.Graph() # makes a new graph

    for w1,w2 in pairs: # for every pair of words w1, w2
        sim=model_wv.similarity(w1, w2) # gets the similarity between the two words
        G.add_edge(w1,w2,weight=sim)

    return G

# creates a graph for the vocabulary of the World Cup document
G=build_graph(vocab,model_wv)
# prints the weight of the edge (semantic similarity) between the two words
G['referee']['goalkeeper']
```

```
{'weight': 0.40646762}
```

وبالنظر إلى ذلك المُخطَّط المبني على الكلمة، يمكن تمثيل مجموعة من الكلمات المتشابهة دلاليًا في صورة عناقيد من العُقد المتصلة معًا بواسطة حواف عالية الوزن. يُطلق على عناقيد العُقد كذلك **المجتمَعات** (Communities). مُخرَج المُخطَّط هو مجموعة بسيطة من الرؤوس والحواف الموزونة. لم تُجرى عملية التجميع حتى الآن لإنشاء المجتمَعات. في الشكل 3.28 تُستخدم ألوان مختلفة لتمييز المجتمَعات في المُخطَّط المذكور بالمثال السابق.

شكل 3.28: المجتمَعات في المُخطَّط

خوارزمية لوفان Louvain Algorithm

تتضمن مكتبة Networkx العديد من الخوارزميات لتحليل المُخطَّط والبحث عن المجتمَعات. واحدة من الخيارات الأكثر فعالية هي خوارزمية لوفان التي تعمل عبر تحريك العُقد بين المجتمَعات حتى تجد بُنية المجتمع التي تمثل الربط الأفضل في الشبكة الضمنية.

fx دالة Get_communities( )

تَستخدِم الدالة الآتية خوارزمية لوفان للبحث عن المجتمَعات في المُخطَّط المبنيّ على الكلمات. تَحسب الدالة كذلك مؤشر الأهمية لكل مجتمع على حده، ثم تكون المُخرَجات في صورة قاموسين:

- word_to_community الذي يربط الكلمة بالمجتمع.
- community_scores الذي يربط المجتمع بدرجة الأهمية.

الدرجة تساوي مجموع تكرار الكلمات في المجتمع. على سبيل المثال، إذا كان المجتمع يتضمن ثلاثة كلمات تظهر 5 و8 و6 مرات في النصّ، فإنّ مؤشّر المجتمع حينئذٍ يساوي 19. ومن ناحية المفهوم، يمثل المؤشر جزءًا من النصّ الذي يضُمُّه المجتمع.

```
from networkx.algorithms.community import louvain_communities
from collections import Counter # used to count the frequency of elements in a list

def get_communities( G, # the input graph
                     tokenized_doc:list): # the list of words in a tokenized document

     # gets the communities in the graph
    communities=louvain_communities(G, weight='weight')
    word_cnt=Counter(tokenized_doc)# counts the frequency of each word in the doc

    word_to_community={}# maps each word to its community

    community_scores={}# maps each community to a frequency score

    for comm in communities: # for each community
         # convert it from a set to a tuple so that it can be used as a dictionary key.
        comm=tuple(comm)
        score=0  # initialize the community score to 0.

        for word in comm: # for each word in the community

            word_to_community[word]=comm  # map the word to the community

            score+=word_cnt[word]  # add the frequency of the word to the community's score.

        community_scores[comm]=score  # map the community to the score.

    return word_to_community, community_scores
```

```
word_to_community, community_scores = get_communities(G,tokenized_doc)
word_to_community['player'][:10] # prints 10 words from the community of the word 'team'
```

```
('champion',
 'stretch',
 'finished',
 'fifth',
 'playing',
 'scoring',
 'scorer',
 'opening',
 'team',
 'win')
```

الآن بعد ربط كل الكلمات بالمجتمع، وربط المجتمع بمؤشر الأهمية، ستكون الخطوة التالية هي استخدام هذه المعلومات لتقييم أهمية كل جملة في المُستنَد الأصلي. دالة ( )evaluate_sentences مُصمَّمة لهذا الغرض.

fx دالة ( )Evaluate_sentences

تبدأ الدالة بتقسيم المُستنَد إلى جُمل، ثم حساب مؤشر الأهمية لكل جُملة، استنادًا إلى الكلمات التي تتضمنها. تكتسب كل كلمة مؤشر الأهمية من المجتمع الذي تنتمي إليه.

على سبيل المثال، لديك جملة مكونة من خمسة كلمات w1، w2، w3، w4، w5. الكلمتان w1 وw2 تنتميان إلى مجتمع بمؤشر قيمته 25، والكلمتان w3 وw4 تنتميان إلى مجتمع بمؤشر قيمته 30، والكلمة w5 تنتمي إلى مجتمع بمؤشر قيمته 15. مجموع مؤشرات الجُمل هو 25+25+30+30+15=125. تَستخدِم الدالة بعد ذلك هذه المؤشرات لتصنيف الجُمل في ترتيب تنازلي، من الأكثر إلى الأقل أهمية.

```
from nltk import sent_tokenize # used to split a document into sentences

def evaluate_sentences(doc:str, # original document
                       word_to_community:dict,# maps each word to its community
                       community_scores:dict, # maps each community to a score
                       model_wv): # Word2Vec model

    # splits the text into sentences
    sentences=sent_tokenize(doc)
    scored_sentences=[]# stores (sentence, score) tuples

    for raw_sent in sentences: # for each sentence

        # get all the words in the sentence, ignore stopwords and focus only on words that are in the
Word2Vec model.
        sentence_words=[word
        for word in re.findall(r'\b\w\w+\b',raw_sent.lower()) # tokenizes
            if word not in stop and  # ignores stopwords
```

```
                word in model_wv] # ignores words that are not in the Word2Vec model

        sentence_score=0 # the score of the sentence

        for word in sentence_words: # for each word in the sentence

            word_comm=word_to_community[word] # get the community of this word
            sentence_score+=community_scores[word_comm] # add the score of this
community to the sentence score.

        scored_sentences.append((sentence_score,raw_sent)) # stores this sentence and
its total score

    # scores the sentences by their score, in descending order
    scored_sentences=sorted(scored_sentences,key=lambda x:x[0],reverse=True)

    return scored_sentences


scored_sentences=evaluate_sentences(text,word_to_community,community_
scores,model_wv)
len(scored_sentences)
```

```
61
```

يتضمن المُستنَد الأصلي إجمالي 61 جُملة، ويُستخدَم المقطع البرمجي التالي للعثور على الجُمل الثلاثة الأكثر أهمية من بين هذه الجُمل:

```
for i in range(3):
    print(scored_sentences[i],'\n')
```

```
(3368, 'Lionel Messi not only emulated the deity of Argentinian football,
Diego Maradona, by leading the nation to World Cup glory; he finally
plugged the burning gap on his CV, winning the one title that has eluded
him – at the fifth time of asking, surely the last time.')

(2880, 'He scored twice in 97 seconds to force extra-time; the first a
penalty, the second a sublime side-on volley and there was a point towards
the end of regulation time when he appeared hell-bent on making sure that
the additional period would not be needed.')

(2528, 'It will go down as surely the finest World Cup final of all time,
the most pulsating, one of the greatest games in history because of how
Kylian Mbappé hauled France up off the canvas towards the end of normal
time.')
```

```
print(scored_sentences[-1]) # prints the last sentence with the lowest score
print()
print(scored_sentences[30]) # prints a sentence at the middle of the scoring scale
```

```
(0, 'By then it was 2-0.')

(882, 'Di María won the opening penalty, exploding away from Ousmane
Dembélé before being caught and Messi did the rest.')
```

النتائج تؤكد أن هذه المنهجية تُحدِّد بنجاح الجُمل الأساسية التي تستنبط النقاط الرئيسة في المُستنَد الأصلي، مع تعيين مؤشرات أقل للجُمل الأقل دلالةً. تُطبَّق المنهجية نفسها كما هي لتوليد ملخّص لأي وثيقة مُحدَّدة.

استخدام توليد اللغات الطبيعية المبني على القواعد لإنشاء روبوت الدردشة
Using Rule-Based NLG to Create a Chatbot

في هذا القسم، ستُصمِّم **روبوت دردشة** (Chatbot) وفق المسار المُحدَّد الموصي به بالجمع بين قواعد المعرِفة الرئيسة للأسئلة والأجوبة والنموذج العصبي **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT)، ويشير هذا إلى أن نقل التعلُّم المُستخدَم في **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) له البنية نفسها كما في **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) all-MiniLM-L6-v2 وسوف يهيَّأ بشكل دقيق لمُهِمَّة أخرى غير تحليل المشاعر، وهي: توليد اللغات الطبيعية.

1. تحميل نموذج تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات المُدرَّب مُسبقًا
Load the Pre-Trained SBERT Model

الخطوة الأولى هي تحميل نموذج **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) المُدرَّب مُسبقًا:

```
%%capture
from sentence_transformers import SentenceTransformer, util
model_sbert = SentenceTransformer('all-MiniLM-L6-v2')
```

2. إنشاء قاعدة معرِفة بسيطة Create a Simple Knowledge Base

الخطوة الثانية هي إنشاء قاعدة معرِفة بسيطة لتحديد النص البرمجي المكون من الأسئلة والأجوبة التي يستخدمها روبوت الدردشة. يتضمن النص البرمجي 4 أسئلة (السؤال 1 إلى 4) والأجوبة على كل سؤال (الإجابة 1 إلى 4). كل إجابة مكونة من مجموعة من الخيارات كل خيار يتكون من قيمتين فقط، تُمثِّل القيمة الثانية السؤال التالي الذي يستخدمه روبوت الدردشة. إذا كان هذا هو السؤال الأخير، ستكون القيمة الثانية خالية. هذه الخيارات تمثل الإجابات الصحيحة المحتملة على الأسئلة المعنية بها. على سبيل المثال، الإجابة على السؤال الثاني لها خياران محتملان [None،"Python"] and [None،"Java"](["جافا"،لا يوجد] و ["البايثون"،لا يوجد]). كل خيار مُكون من قيمتين:

- النص الحقيقي للإجابة المقبولة مثل: Java (جافا) أو Courses on Marketing (دورات تدريبية في التسويق).
- مُعرِّف يشير إلى السؤال التالي الذي سيطرحه روبوت الدردشة عند تحديد هذا الخيار. على سبيل المثال، إذا حدَّد المُستخدِم خيار ["3"،"Courses on Engineering"] (["دورات تدريبية في الهندسة"، "3"]) كإجابة على السؤال الأول، يكون السؤال التالي الذي سيطرحه روبوت الدردشة هو السؤال الثالث.

يمكن توسيع قاعدة المعرِفة البسيطة لتشمل مستويات أكثر من الأسئلة والأجوبة، وتجعل روبوت الدردشة أكثر ذكاءً.

```
QA={
        "Q1":"What type of courses are you interested in?",
        "A1":[["Courses in Computer Programming","2"],
              ["Courses in Engineering","3"],
              ["Courses in Marketing","4"]],

        "Q2":"What type of Programming Languages are you interested in?",
        "A2":[["Java",None],["Python",None]],

        "Q3":"What type of Engineering are you interested in?",
        "A3":[["Mechanical Engineering",None],["Electrical Engineering",None]],

        "Q4":"What type of Marketing are you interested in?",
        "A4":[["Social Media Marketing",None],["Search Engine
Optimization",None]]
}
```

دالة Chat( ) *fx*

في النهاية، تُستخدَم دالة Chat( ) لمعالجة قاعدة المعرِفة وتنفيذ روبوت الدردشة. بعد طرح السؤال، يقرأ روبوت الدردشة رد المُستخدِم.

- إن كان الرد مشابهًا دلاليًا لأحد خيارات الإجابات المقبولة لهذا السؤال، يُحدَّد ذلك الخيار وينتقل روبوت الدردشة إلى السؤال التالي.
- إن لم يتشابه الرد مع أيٍ من الخيارات، يُطلب من المُستخدِم إعادة صياغة الرد.

تُستخدَم دالة **تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات** (SBERT) لتقييم مؤشر التشابه الدلالي بين الرد وكل الخيارات المُرشَّحة. يُعدُّ الخيار متشابهًا إذا كان المؤشر أعلى من مُتغير الحد الأدنى sim_lbound.

```
import numpy as np # used for processing numeric data

def chat(QA:dict, # the Question-Answer script of the chatbot
         model_sbert, # a pre-trained SBERT model
         sim_lbound:float): # lower bound on the similarity between the user's response and the
closest candidate answer

    qa_id='1' # the QA id

    while True: # an infinite loop, will break in specific conditions

        print('>>',QA['Q'+qa_id]) # prints the question for this qa_id
        candidates=QA["A"+qa_id] # gets the candidate answers for this qa_id

        print(flush=True) # used only for formatting purposes
        response=input() # reads the user's response

        # embed the response
        response_embeddings = model_sbert.encode([response], convert_to_
tensor=True)
        # embed each candidate answer. x is the text, y is the qa_id. Only embed x.
```

```
        candidate_embeddings = model_sbert.encode([x for x,y in candidates],
convert_to_tensor=True)

        # gets the similarity score for each candidate
        similarity_scores = util.cos_sim(response_embeddings, candidate_
embeddings)

        # finds the index of the closest answer.
        # np.argmax(L) finds the index of the highest number in a list L
        winner_index=np.argmax(similarity_scores[0])

        # if the score of the winner is less than the bound, ask again.
        if similarity_scores[0][winner_index]<sim_lbound:
            print('>> Apologies, I could not understand you. Please rephrase
your response.')
            continue

        # gets the winner (best candidate answer)
        winner=candidates[winner_index]

        # prints the winner's text
        print('\n>> You have selected:',winner[0])
        print()

        qa_id=winner[1] # gets the qa_id for this winner

        if qa_id==None: # no more questions to ask, exit the loop
            print('>> Thank you, I just emailed you a list of courses.')
            break
```

أنظر إلى التفاعلين التاليين بين روبوت الدردشة والمُستخدِم:

التفاعل الأول

```
chat(QA,model_sbert, 0.5)
```

```
>> What type of courses are you interested in?
marketing courses
>> You have selected: Courses on Marketing
>> What type of Marketing are you interested in?
seo
>> You have selected: Search Engine Optimization
>> Thank you, I just emailed you a list of courses.
```

في التفاعل الأول، يفهم روبوت الدردشة أن المُستخدِم يبحث عن دورات تدريبية في التسويق. وكذلك، روبوت الدردشة ذكي بالقدر الكافي ليفهم أن المصطلح SEO يشبه دلاليًا مصطلح Search Engine Optimization (تحسين محركات البحث) مما يؤدي إلى إنهاء المناقشة بنجاح.

التفاعل الثاني

```
chat(QA,model_sbert, 0.5)
```

```
>> What type of courses are you interested in?

cooking classes
>> Apologies, I could not understand you. Please rephrase your response.
>> What type of courses are you interested in?

software courses

>> You have selected: Courses on Computer Programming

>> What type of Programming Languages are you interested in?

C++

>> You have selected: Java
>> Thank you, I just emailed you a list of courses.
```

في التفاعل الثاني، يفهم روبوت الدردشة أن Cooking Classes (دروس الطهي) لا تشبه دلاليًا الخيارات الموجودة في قاعدة المعرِفة. وهو ذكي بالقدر الكافي ليفهم أن Software courses (الدورات التدريبية في البرمجة) يجب أن ترتبط بخيار Courses on Computer Programming (الدورات التدريبية في برمجة الحاسب). الجزء الأخير من التفاعل يسلط الضوء على نقاط الضعف: يربط روبوت الدردشة بين رد المُستخدِم C++ و Java. على الرغم من أن لغتي البرمجة مرتبطتان بالفعل ويمكن القول بأنهما أكثر ارتباطًا من لغتي البايثون و C++، إلا أن الرد المناسب يجب أن يُوضِّح أن روبوت الدردشة لا يتمتع بالدراية الكافية للتوصية بالدورات التدريبية في لغة C++. إحدى الطرائق لمعالجة هذا القصور هي استخدام التشابه بين المفردات بدلًا من التشابه الدلالي للمقارنة بين الردود والخيارات ذات الصلة ببعض الأسئلة.

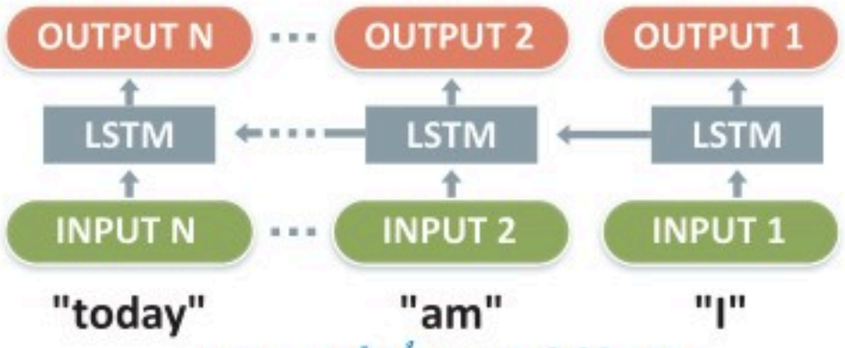
استخدام تعلُّم الآلة لتوليد نص واقعي

Using Machine Learning to Generate Realistic Text

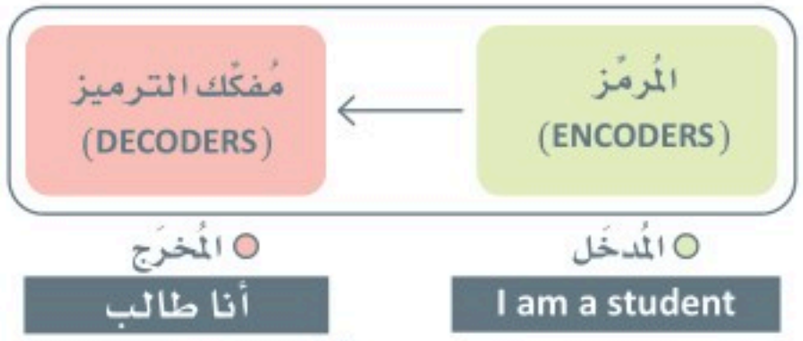
الطرائق الموضحة في الأقسام السابقة تَستخدِم القوالب، والقواعد، أو تقنيات التحديد لتوليد النصوص للتطبيقات المختلفة. في هذا القسم، ستتعرَّف على أحدث تقنيات تعلُّم الآلة المُستخدَمة في **توليد اللغات الطبيعية** (NLG).

جدول 3.5: تقنيات تعلُّم الآلة المُتقدمة المُستخدَمة في توليد اللغات الطبيعية

التقنية	الوصف
شبكة الذاكرة المُطولَة قصيرة المدى (Long Short-Term Memory – LSTM)	تتكون شبكة **الذاكرة المُطولَة قصيرة المدى** (LSTM) من خلايا **ذاكرة** (Memory Cells) مرتبطة ببعض. عند إدخال سلسلة من البيانات إلى الشبكة، تتولى معالجة كل عنصر في السلسلة واحدًا تلو الآخر، وتُحدِّث الشبكة خلايا الذاكرة لتوليد مُخرَج لكل عنصر على حده. شبكات **الذاكرة المُطولَة قصيرة المدى** (LSTM) تناسب مهام **توليد اللغات الطبيعية** (NLG) لقدرتها على الاحتفاظ بالمعلومات من سلاسل البيانات (مثل التعرّف على الكلام أو الكتابة اليدوية) ومعالجة تعقيد اللغات الطبيعية.
النماذج المبنية على المحولات (Transformer-Based Models)	النماذج المبنية على المحولات هي تلك التي تفهم اللغات البشريّة وتولّدها، وتَستنِد هذه النماذج في عملها إلى تقنية **الانتباه الذاتي** (Self-Attention) التي تمكِّنها من فهم العلاقات بين الكلمات المختلفة في الجُمل.



شكل 3.29: الذاكرة المُطوّلة قصيرة المدى



شكل 3.30: المُحوّل

المُحوِّلات Transformers

المُحوِّلات مناسبة لمهام توليد اللغات الطبيعية لقدرتها على معالجة البيانات المُدخَلة المُتسلسلة بكفاءة. في نموذج المُحوِّلات، تُمرَّر البيانات المُدخَلة عبر المُرمِّز الذي يُحوّل المُدخَلات إلى تمثيل مستمر، ثم يُمرَّر التمثيل عبر مُفكِّك الترميز الذي يُولِّد التسلسل المُخرَج. إحدى الخصائص الرئيسة لهذه النماذج هي استخدام آليات الانتباه التي تسمح للنموذج بالتركيز على الأجزاء المُهِمَّة من التسلسل في حين تتجاهل الأجزاء الأقل دلالةً. أظهرت نماذج المُحوِّلات كفاءة في توليد النص عالي الدقة للعديد من مهام توليد اللغات الطبيعية بما في ذلك ترجمة الآلة، والتلخيص، والإجابة على الأسئلة.

نموذج الإصدار الثاني من المُحوِّل التوليدي مُسبَق التدريب GPT-2 Model

في هذا القسم، ستستخدِم **الإصدار الثاني من المُحوِّل التوليدي مُسبَق التدريب** (GPT-2) وهو نموذج لغوي قوي طورته شركة **أوبن أي آي** (OpenAI) لتوليد النصوص المُستنِدة إلى النص التلقيني المُدخَل بواسطة المُستخدِم. **الإصدار الثاني من المُحوِّل التوليدي مُسبق التدريب** (Generative Pre-training Transformer 2 - GPT-2) مُدرَّب على مجموعة بيانات تضم أكثر من ثمان ملايين صفحة ويب ويتميز بالقدرة على إنشاء النصوص البشرية بعدَّة لغات وأساليب. بُنية **الإصدار الثاني من المُحوِّل التوليدي مُسبق التدريب** (GPT-2) المبنية على المُحوِّل تسمح بتحديد **التبعيَّات** (Dependencies) بعيدة المدى وتوليد النصوص المُتَّسقة، وهو مُدرَّب للتنبؤ بالكلمة التالية وفقًا لكل الكلمات السابقة ضمن النص، وبالتالي، يمكن استخدام النموذج لتوليد نصوص طويلة جدًا عبر التنبؤ المستمر وإضافة المزيد من الكلمات.

```
%%capture
!pip install transformers
!pip install torch
import torch # an open-source machine learning library for neural networks, required for GPT2.
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# initialize a tokenizer and a generator based on a pre-trained GPT2 model.

# used to:
# -encode the text provided by the user into tokens
# -translate (decode) the output of the generator back to text
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

# used to generate new tokens based on the inputted text
generator = GPT2LMHeadModel.from_pretrained('gpt2')
```

يُقدَّم النص التالي كأساس يستند إليه **الإصدار الثاني من المُحوِّل التوليدي مُسبق التدريب** (GPT-2):

```
text='We had dinner at this restaurant yesterday. It is very close to my
house. All my friends were there, we had a great time. The location is
```

```
excellent and the steaks were delicious. I will definitely return soon, highly
recommended!'

# encodes the given text into tokens
encoded_text = tokenizer.encode(text, return_tensors='pt')

# use the generator to generate more tokens.
# do_sample=True prevents GPT-2 from just predicting the most likely word at every step.
generated_tokens = generator.generate(encoded_text,
                                      max_length=200) # max number of new tokens to
generate
#decode the generates tokens to convert them to words
# skip_special_tokens=True is used to avoid special tokens such as '>' or '-' characters.
print(tokenizer.decode(generated_tokens[0], skip_special_tokens=True))
```

```
We had dinner at this restaurant yesterday. It is very close to my house.
All my friends were there, we had a great time. The location is excellent
and the steaks were delicious.I will definitely return soon, highly
recommended!

I've been coming here for a while now and I've been coming here for a while
now and I've been coming here for a while now and I've been coming here for
a while now and I've been coming here for a while now and I've been coming
here for a while now and I've been coming here for a while now and I've
been coming here for a while now and I've been coming here for a while now
and I've been coming here for a while now and I've been coming here for a
while now and I've been coming here for a while now and I've been coming
here for a while now and I've been coming here for a while now and I've
been coming here for a while now and
```

```
# use the generator to generate more tokens.
# do_sample=True prevents GPT-2 from just predicting the most likely word at every step.
generated_tokens = generator.generate(encoded_text,
                                      max_length=200, # max number of new tokens to
generate
                                      do_sample=True)
print(tokenizer.decode(generated_tokens[0],skip_special_tokens=True))
```

```
We had dinner at this restaurant yesterday. It is very close to my house.
All my friends were there, we had a great time. The location is excellent
and the steaks were delicious.I will definitely return soon, highly
recommended!

If you just found this place helpful. If you like to watch videos or
go to the pool while you're there, go for it! Good service - I'm from
Colorado and love to get in and out of this place. The food was amazing!
Also, we were happy to see the waitstaff with their great hands - I went
for dinner. I ordered a small side salad (with garlic on top), and had a
slice of tuna instead. When I was eating, I was able to get up and eat my
salad while waiting for my friend to pick up the plate, so I had a great
time too. Staff was welcoming and accommodating. Parking is cheap in this
neighborhood, and it is in the neighborhood that it needs to
```

يحقّق هذا مُخرَجات أكثر تنوعًا، مع الحفاظ على دقة وسلامة النص المولَّد، حيث يستخدِم النص مفردات غنية وهو سليم نحويًا. يسمح **الإصدار الثاني من المُحوّل التوليدي مُسبق التدريب** (GPT-2) بتخصيص المُخرَج بشكل أفضل. يتضح ذلك عند استخدام مُتغير temperature (درجة الحرارة) الذي يسمح للنموذج بتقبل المزيد من المخاطر بل وأحيانًا اختيار بعض الكلمات الأقل احتمالًا. القيم الأعلى لهذا المُتغير تؤدي إلى نصوص أكثر تنوعًا، مثل:

```
# Generate tokens with higher diversity
generated_tokens = generator.generate(
    encoded_text, max_length=200, do_sample=True, temperature=2.0)

print(tokenizer.decode(generated_tokens[0], skip_special_tokens=True))
```

```
We had dinner at this restaurant yesterday. It is very close to my house.
All my friends were there, we had a great time. The location is excellent
and the steaks were delicious.I will definitely return soon, highly
recommended!

Worth a 5 I thought a steak at a large butcher was the end story!! We were
lucky. The price was cheap!! That night though as soon as dinner was on
my turn that price cut completely out. At the tail area they only have
french fries or kiwifet - no gravy - they get a hard egg the other day too
they call kawif at 3 PM it will be better this summer if I stay more late
with friends. When asked it takes 2 or 3 weeks so far to cook that in this
house. Once I found a place it was great. Everything I am waiting is just
perfect as usual....great prices especially at one where a single bite
would suffice or make more as this only runs on the regular hours
```

ومع ذلك، إذا كانت درجة الحرارة مرتفعة للغاية، فإنّ النموذج سيتجاهل الإرشادات الأساسية التي تظهر في **المُدخَل الأولي** (Original Seed) ويُولِّد مُخرَجًا أقل واقعية وليس له معنى:

```
# Too high temperature leads to divergence in the meaning of the tokens
generated_tokens = generator.generate(
    encoded_text, max_length=200, do_sample=True, temperature=4.0)

print(tokenizer.decode(generated_tokens[0], skip_special_tokens=True))
```

```
We had dinner at this restaurant yesterday. It is very close to my house.
All my friends were there, we had a great time. The location is excellent
and the steaks were delicious.I will definitely return soon, highly
recommended! It has the nicest ambagas of '98 that I like; most Mexican.
And really nice steak house; amazing Mexican atmosphere to this very
particular piece of house I just fell away before its due date, no surprise
my 5yo one fell in right last July so it took forever at any number on
it being 6 (with it taking two or sometimes 3 month), I really have found
comfort/affability on many more restaurants when ordering.If you try at
it they tell ya all about 2 and three places will NOT come out before they
close them/curry. Also at home i would leave everything until 1 hour but
sometimes wait two nights waiting for 2+ then when 2 times you leave you
wait in until 6 in such that it works to
```

تمرينات

1

حدِّد الجملة الصحيحة والجملة الخاطئة فيما يلي:	صحيحة	خاطئة
1. توليد اللغات الطبيعية المبنيّ على تعلُّم الآلة يتطلب مجموعات كبيرة من بيانات التدريب والموارد الحسابية.	○	○
2. الفعل هو نوع من وسوم أقسام الكلام (POS).	○	○
3. في تحليل بِناء الجُمل لتوليد اللغات الطبيعية المبنيّ على القوالب، يُستخدَم التحليل بصورة منفصلة عن وسوم أقسام الكلام (POS).	○	○
4. المجتمَعات هي عناقيد العُقد التي تُمثِّل الكلمات المختلفة دلاليًا.	○	○
5. يصبح روبوت الدردشة أكثر ذكاءً كلما ازداد عدد مستويات الأسئلة والأجوبة المُضافة إلى قاعدة المعرِفة.	○	○

2 قارن بين المنهجيات المختلفة لتوليد اللغات الطبيعية (NLG).

3 حدِّد ثلاث تطبيقات مختلفة لتوليد اللغات الطبيعية (NLG).

4 أكمل المقطع البرمجي التالي حتى تقبل الدالة build_graph() مفردات مُحدَّدة من الكلمات ونموذج الكلمة إلى المتَّجَه (Word2Vec) المُدرَّب لرسم مُخطَّط ذي عُقدة واحدة لكل كلمة في المفردات المُحدَّدة. يجب أن يحتوي المُخطَّط على حافة بين عُقدتين إذا كان تشابه نموذج الكلمة إلى المتَّجَه (Word2Vec) أكبر من مستوى التشابه المُعطى، ويجب ألا تكون هناك أوزان على الحواف.

```
from ____________ import combinations # tool used to create combinations

import networkx as nx # python library for processing graphs

def build_graph(vocab:set, # set of unique words

                model_wv, # Word2Vec model

                similarity_threshold:float

               ):

    pairs=combinations(vocab,_________________) # gets all possible pairs of words in the vocabulary

    G=nx._________________ # makes a new graph

    for w1,w2 in pairs: # for every pair of words w1,w2

        sim=model_wv._____________________(w1, w2)# gets the similarity between the two words

        if _________________:

            G._____________________(w1,w2)

    return G
```

5 أكمل المقطع البرمجي التالي حتى تَستخدِم الدالة get_max_sim() نموذج تمثيلات ترميز الجُمل ثنائية الاتجاه من المحولات (SBERT) للمقارنة بين جُملة مُحدَّدة my_sentence وكل الجُمل الواردة في قائمة أخرى من الجُمل L. يجب أن تُعيد الدالة الجُملة ذات مُؤشر التشابه الأعلى من L1 إلى my_sentence.

```
from sentence_transformers import _________________, util

from ____________ import combinations # tool used to create combinations

model_sbert = ___________________ ('all-MiniLM-L6-v2')

def get_max_sim(L1,my_sentence):

    # embeds my_sentence

    my_embedding = model_sbert,_____________([my_sentence], convert_to_tensor=True)

    # embeds the sentences from L2

    L_embeddings = model_sbert._____________(L, convert_to_tensor=True)

    similarity_scores = _____________.cos_sim(_________________, __________)

    winner_index=np.argmax(similarity_scores[0])

    return _____________
```

المشروع

تصنيف النص هو عملية مكونة من خطوتين تشمل:

الخطوة الأولى: استخدام مجموعة من نصوص التدريب ذات القيم (التصنيفات) المعروفة لتدريب نموذج التصنيف.

الخطوة الثانية: استخدام نموذج التدريب للتنبؤ بالقيم لكل نصّ في مجموعة بيانات الاختبار. القيم في مجموعة بيانات الاختبار إما غير معروفة أو مخبَّأة وتُستخدم لاحقًا في عملية التحقق.

يجب تمثيل النصوص في كل من مجموعات بيانات التدريب والاختبار بالمتَّجَهات قبل استخدامها. تُستخدَم أدوات CountVectorizer أو TfidfVectorizer من مكتبة **سكليرن** (Sklearn) في البرمجة الاتجاهية.

تُقدِّم مكتبة **سكليرن** (Sklearn) في لغة البايثون قائمة طويلة من نماذج التصنيف. مثل:

```
GradientBoostingClassifier( ) <
DecisionTreeClassifier( ) <
RandomForestClassifier( ) <
```

مهمتك هي استخدام مجموعة بيانات التدريب IMDB المُستخدَمة في هذا الدرس لتدريب النموذج الذي يحقق أعلى درجة من الدقة على مجموعة بيانات الاختبار IMDB (imdb_data/imdb_ test.csv). يمكنك تحقيق ذلك عبر:

1. استبدال المُصنِّف MultinomialNB بنماذج تصنيف أخرى من مكتبة **سكليرن** (Sklearn) مثل الموضحة بالأعلى.
2. إعادة تشغيل المفكرة التفاعلية لديك بعد الاستبدال، لحساب دقة كل نموذج جديد بعد تجربته.
3. إنشاء تقرير للمقارنة بين دقة كل النماذج التي جرَّبتها وتحديد النموذج الذي حقق نتائجَ دقيقة.

ماذا تعلّمت

- تصنيف النص باستخدام نماذج التعلُّم غير الموجَّه.
- تحليل النص باستخدام نماذج التعلُّم الموجَّه.
- استخدام نماذج تعلُّم الآلة لتوليد اللغات الطبيعية.
- برمجة روبوت دردشة بسيط.

المصطلحات الرئيسة

وسوم أقسام الكلام	Part of Speech (POS) Tags
تحليل المشاعر	Sentiment Analysis
التعلُّم الموجَّه	Supervised Learning
تحليل بِناء الجُمل	Syntax Analysis
التقسيم	Tokenization
التعلُّم المنقول	Transfer Learning
التعلُّم غير الموجَّه	Unsupervised Learning
البرمجة الاتجاهية	Vectorization

مُتنبِّئات الصندوق الأسود	Black-Box predictors
روبوت الدردشة	Chatbot
عنقود	Cluster
الرسم الشجري	Dendrogram
تقليص الأبعاد	Dimensionality Reduction
تجميع المُستنَدات	Document Clustering
توليد اللغات الطبيعية	Natural Language Generation
معالجة اللغات الطبيعية	Natural Language Processing