علوم الحاسب

COMPUTER SCIENCE

كتاب الطالب

12

المسار التكنولوجي

الفصل الدراسي الأول

2021 – 2022

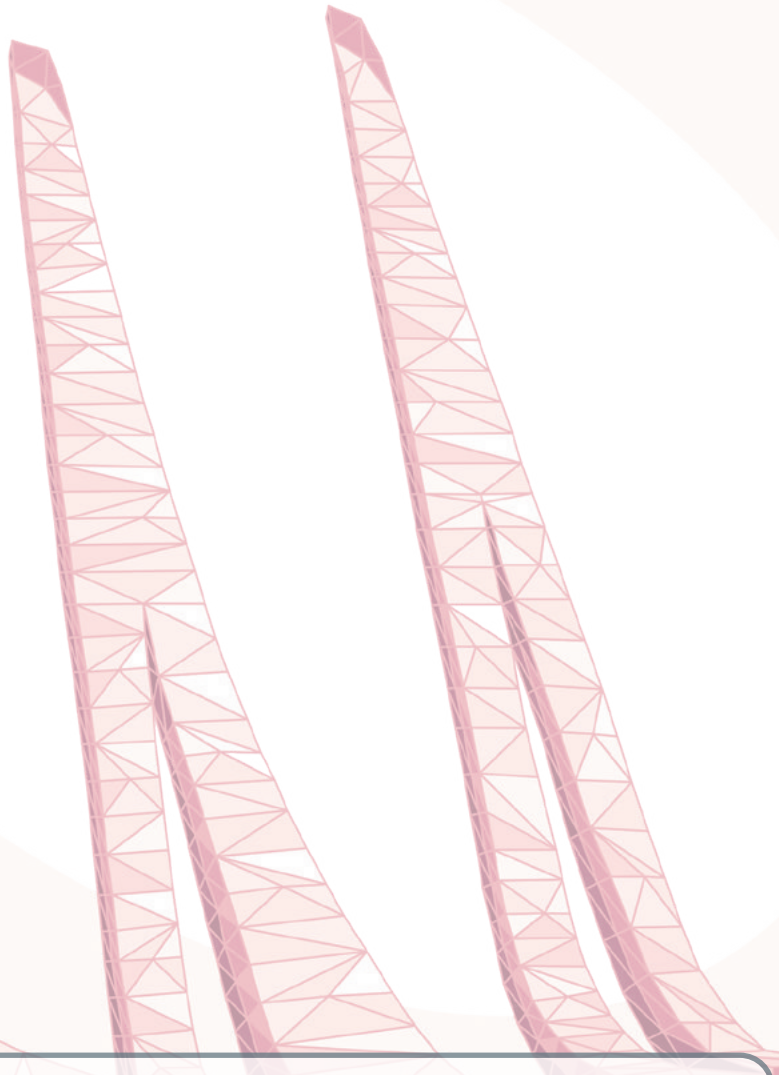
الطبعة الأولى

binarylogic

علوم الحاسب

COMPUTER SCIENCE

كتاب الطالب



الاسم .....

الشعبة .....

حضرة صاحب السمو الشيخ تميم بن حمد آل ثاني
أمير دولة قطر

النشيد الوطني

قَسَمًا بِمَنْ رَفَعَ السَّمَاءْ قَسَمًا بِمَنْ نَشَرَ الضِّيَاءْ
قَطَرٌ سَتَبْقَى حُرَّةً تَسْمُو بِرُوحِ الأَوْفِيَاءْ
سِيرُوا عَلَى نَهْجِ الأُلَى وَعَلَى ضِيَاءِ الأَنْبِيَاءْ
قَطَرٌ بِقَلْبِي سِيرَةٌ عِزٌّ وَأَمْجَادُ الإِبَاءْ
قَطَرُ الرِّجَالِ الأَوَّلِين حُمَاتُنَا يَوْمَ النِّدَاءْ
وَحَمَائِمٌ يَوْمَ السَّلاَمْ جَوَارِحٌ يَوْمَ الفِدَاءْ

مفاتيح رموز الكتاب

برامج أخرى:
قسم في نهاية الوحدة يعرض بعض الأدوات والبرامج البديلة.

المصطلحات:
قسم يوضح ما تعلمته والمفردات الجديدة التي يحتويها الدرس.

مشروع الوحدة:
نشاط في نهاية كل وحدة يدمج المهارات التي يتم تدريسها في الوحدة.

ماذا تعلمت:
قسم يركز على النقاط المهمة التي يحتاج الطلاب إلى مراجعتها.

تمرين عملي

تمرين نظري

نصيحة ذكية:
معلومات مفيدة.

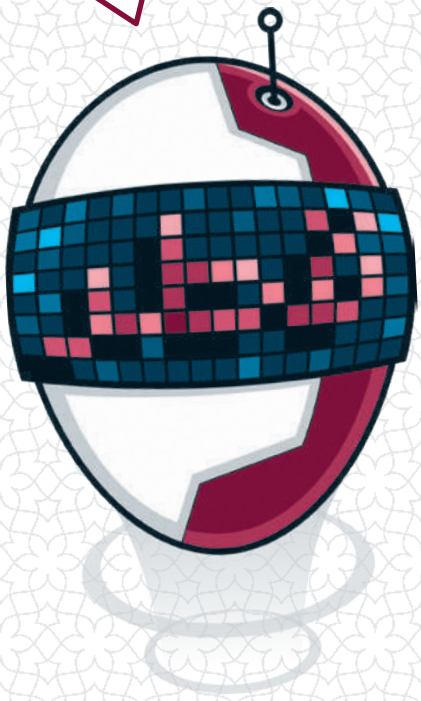
كن آمنًا:
معلومات لحماية نفسك.

لمحة تاريخية:
أحداث حقيقة من الماضي.

أهلا بك!

تعال معي لنستكشف عالم تكنولوجيا المعلومات

انتقل إلى حاسوبك واتبعني!



وزارة التعليم والتعليم العالي
إدارة المناهج الدراسية ومصادر التعلم

المراجعة والتدقيق
فِرَق من:
كلية الهندسة - جامعة قطر
إدارة التوجيه التربوي
الميدان التربوي

الإشراف العلمي والتربوي
إدارة المناهج الدراسية ومصادر التعلم
قسم المواد الدراسية

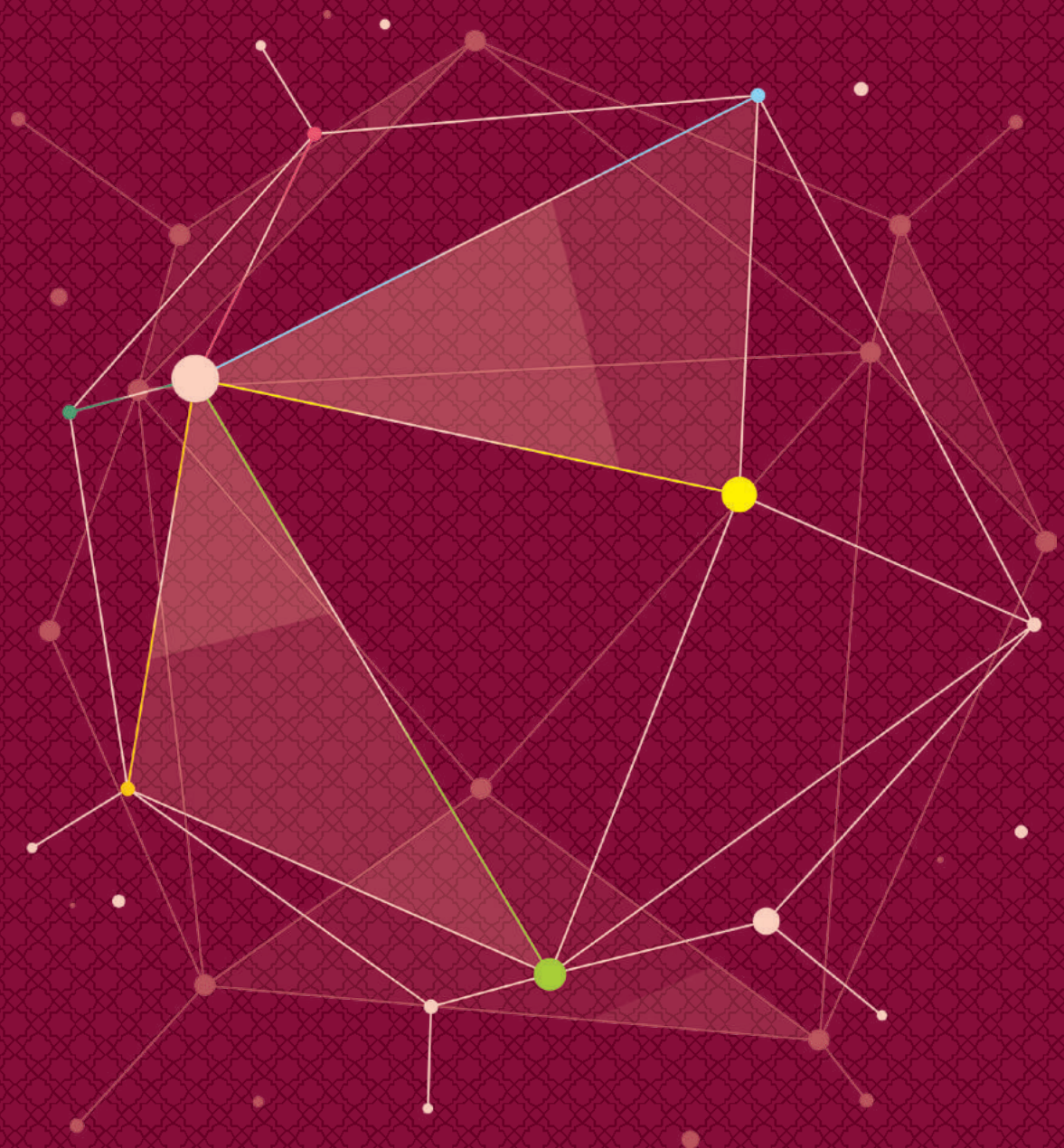
الكفايات الأساسية للمنهج التعليمي الوطني لدولة قطر

- التفكير الإبداعي والتفكير الناقد
- الكفاية اللغوية
- الكفاية العددية
- التواصل
- التعاون والمشاركة
- التقصي والبحث
- حل المشكلات

1. هياكل البيانات المتقدمة

تعرفنا سابقًا مفهوم هياكل البيانات ودورها واستعمالاتها في البرمجة، وتطرقنا باستفاضة لبعض الهياكل. في هذه الوحدة سنواصل استعراض أهم هياكل البيانات المركبة (غير البسيطة) التي يمكن تصنيفها الى هياكل بيانات خطية مثل المُكدّس Stack والطابور Queue والقوائم المرتبطة Linked Lists وأخرى غير خطية كالشجرة Tree والمخططات Graphs.

كل هذه المعارف ستتخللها مهارات عملية باستخدام لغة البرمجة بايثون وذلك من خلال تعلم حل المسائل البرمجية باستعمال الهياكل التي تم ذكرها وتعلم كيفية توظيفها في عملية البرمجة.



ماذا سنتعلمْ؟

في هذه الوحدة سنتعلم:

> التعرف على المُكدّس Stack في البرمجة.

> شرح استخدامات المكدس في البرمجة.

> التعرف على أهم العمليات التي يمكن اجراءها على المكدس.

> كتابة برامج لحل بعض المسائل باستخدام المكدس.

> التعرف على الطابور Queue.

> التعرف على أهم العمليات التي يمكن اجراءها على الطابور.

> كتابة برامج لحل بعض المسائل البرمجية باستخدام الطابور.

> التعرف على القائمة المرتبطة Linked list.

> توضيح العمليات الأساسية على البيانات داخل القائمة المرتبطة.

> التعرف على الشجرة الثنائية Binary tree.

> توضيح استخدام الأشجار.

> توضيح هياكل الشجرة الثنائية.

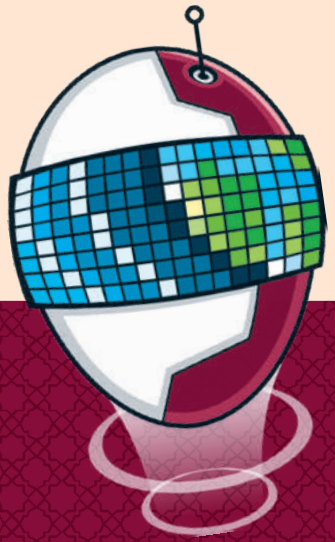
> إنشاء برنامج لإدارة الشجرة الثنائية.

> التعرف على المخطط Graph.

> توضيح استخدامات المخططات.

> إنشاء برنامج ينشئ مخطط.

> استخدام البرمجة بلغة Python لاستكشاف هياكل بيانات معقدة.



مواضيع الوحدة

> المُكدّس والطابور Stack & Queue

> القائمة المرتبطة Linked list

> هياكل بيانات الأشجار

> هياكل بيانات المخططات

الأدوات

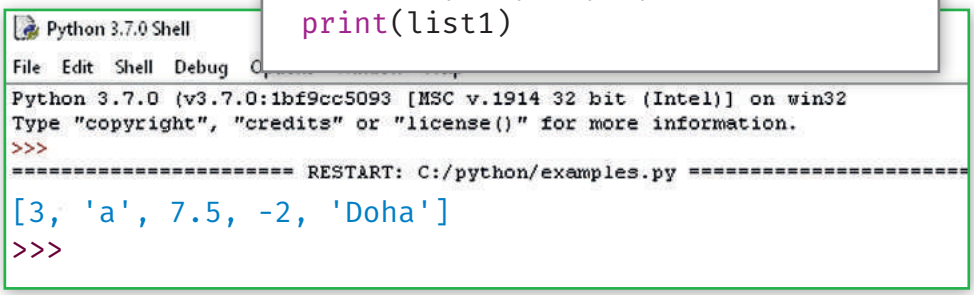
> Python

هل تذكر؟

هياكل البيانات هي وسيلة لتخزين وتنظيم البيانات في الذاكرة بحيث يمكن استخدامها بكفاءة، ولها أمثلة متعددة، أبرزها القوائم والصفوف والمصفوفات والقواميس، وغيرها.

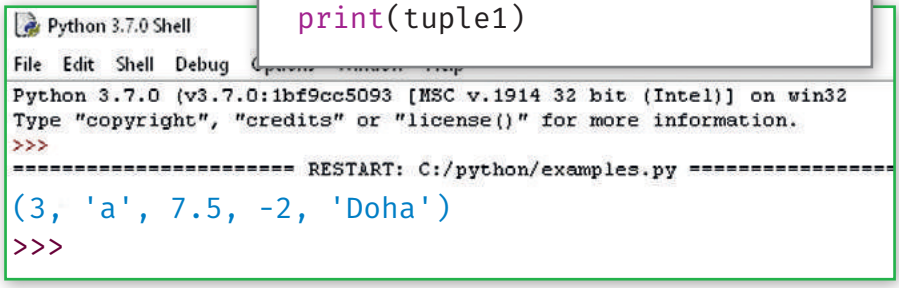
القائمة (List) في البايثون عبارة عن سلسلة من العناصر من نفس النوع أو من أنواع مختلفة مثل النصوص والأعداد الصحيحة والأعداد العشرية وغيرها.

```
list1=[3,'a',7.5,-2,'Doha']
print(list1)
```



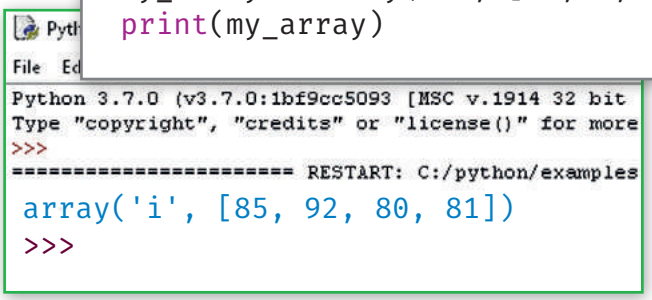
يعتبر الصف (tuple) مجموعة مرتبة من البيانات بحيث يمكن أن تكون القيم المخزنة داخله من أي نوع ولكن لا يمكن تغييرها.

```
tuple1=(3,"a",7.5,-2,"Doha")
print(tuple1)
```



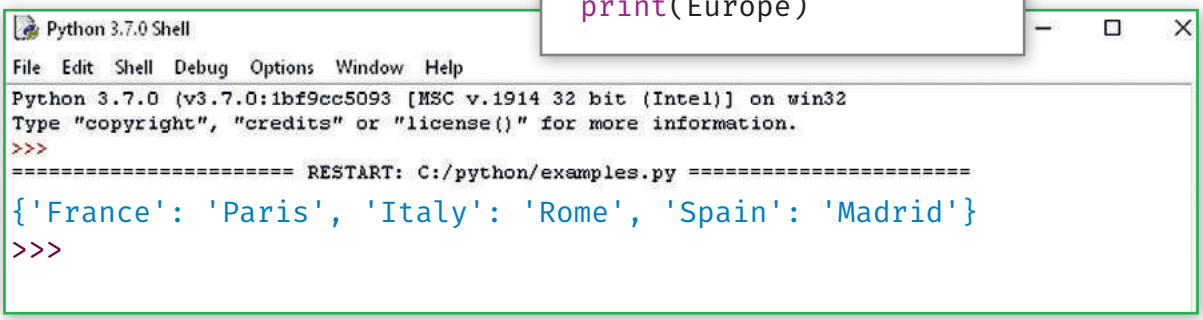
تستخدم المصفوفة لتخزين بيانات من نفس النوع، لذا يمكننا اعتبار المصفوفة كمجموعة من القيم من نفس النوع. يستخدم Python القائمة List لتمثيل كل مصفوفة. أيضًا يمكن أن تكون المصفوفات ذات بعد واحد (1D) أو ثنائية الأبعاد (2D) أو حتى عدد N من الأبعاد.

```
from array import *
my_array = array('i', [85,92,80,81])
print(my_array)
```



القاموس هو هيكل بيانات يقوم بتخزين البيانات على شكل أزواج، كل زوج عبارة عن جزئين هما المفتاح والقيمة.

```
Europe= {
    "France" : "Paris",
    "Italy" : "Rome",
    "Spain" : "Madrid",
    }
print(Europe)
```



هياكل البيانات Data Structures

تقوم أجهزة الحاسوب بتخزين البيانات ومعالجتها بسرعة ودقة فائقتين، لذلك من الضروري للغاية أن يتم تخزين البيانات بكفاءة لكي يتم الوصول إليها بسرعة.

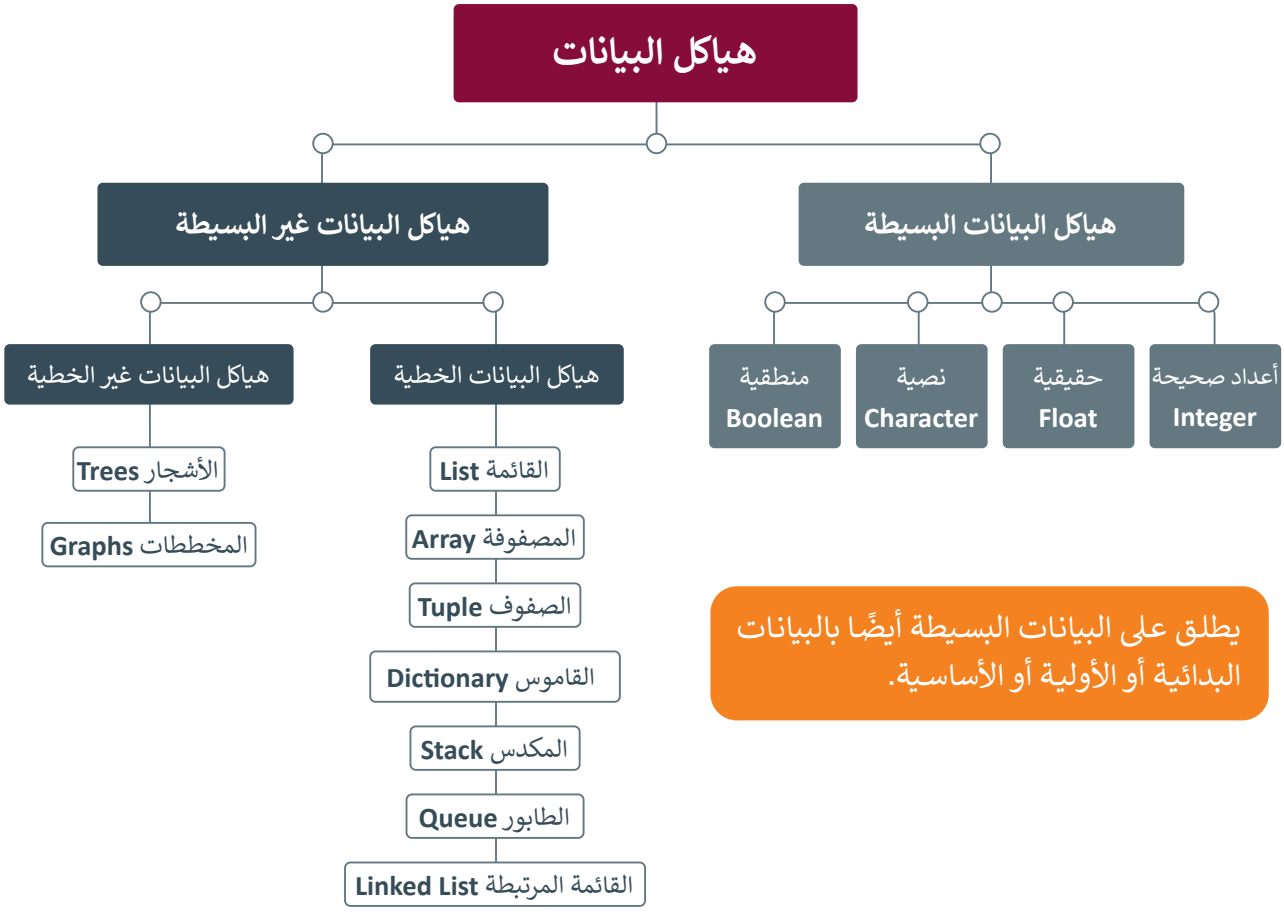
هيكلة البيانات هي تقنية لتخزين البيانات في الذاكرة وتنظيم عملية الوصول إليها لاستخدامها بكفاءة وسهولة.

يمكن تصنيف هذه الهياكل إلى:

هياكل البيانات البسيطة **Primitive Data Structures**

هياكل البيانات غير البسيطة **Non-Primitive Data Structures**

المخطط التالي يظهر تصنيفات هياكل البيانات المختلفة.



يطلق على البيانات البسيطة أيضًا بالبيانات البدائية أو الأولية أو الأساسية.

هياكل البيانات البسيطة Primitive Data Structures

تتضمن هذه الهياكل قيمًا بسيطة من البيانات. تخبر أنواع البيانات البسيطة مترجم لغة البرمجة بنوع البيانات التي يمكن للمتغير تخزينها داخله.

من هياكل البيانات البسيطة في **Python**:

1. الأعداد **Numbers** (تستخدم لتمثيل البيانات الرقمية)

> الأعداد الصحيحة **Integers**.

> الأعداد الحقيقية **Floating point**.

2. النصوص Strings (تستخدم لتمثيل حروف أو رموز أو أرقام).

3. المتغيرات المنطقية **Boolean** (تستخدم لتمثيل حالة فريدة: صواب أو خطأ).

هياكل البيانات غير البسيطة Non-Primitive Data Structures

هياكل البيانات غير البسيطة هي هياكل تختص بتخزين مجموعة من القيم بحيث يتم إنشاؤها بواسطة المبرمج وليست معرفة ضمنياً في **Python** مثل هياكل البيانات البسيطة.

يمكن تقسيم هياكل البيانات غير البسيطة إلى فئتين:

> هياكل البيانات الخطية أو المتسلسلة.

> وهياكل البيانات غير الخطية.

تخزن هياكل البيانات الخطية عناصر البيانات بتسلسل معين، بينما لا تحتوي هياكل البيانات غير الخطية على ارتباط تسلسلي بين عناصر البيانات، ولكن يمكن ربط أي زوج أو مجموعة من عناصر البيانات ببعضها البعض.

يتم استخدام أنواع مختلفة من هياكل البيانات في تطبيقات ومهام الحاسوب المختلفة بناءً على متطلبات المشروع أو المهمة وقيود الذاكرة.

في هذا الدرس سنتعرف على بعض هياكل البيانات الجديدة مثل المُكدّس والطابور، وهما من أكثر الهياكل شيوعًا في حياتنا اليومية.

المُكدِّس Stack

لكي ننشئ المُكدّس من الكتب علينا وضع الكتب الواحد فوق الآخر، وعند رغبتنا باستخدام كتاب، فإننا سنلتقط الكتاب الموجود أعلى المُكدّس، وللوصول إلى العناصر الأخرى في المُكدّس سنضطر إلى إزالة العناصر الموجودة في الأعلى.

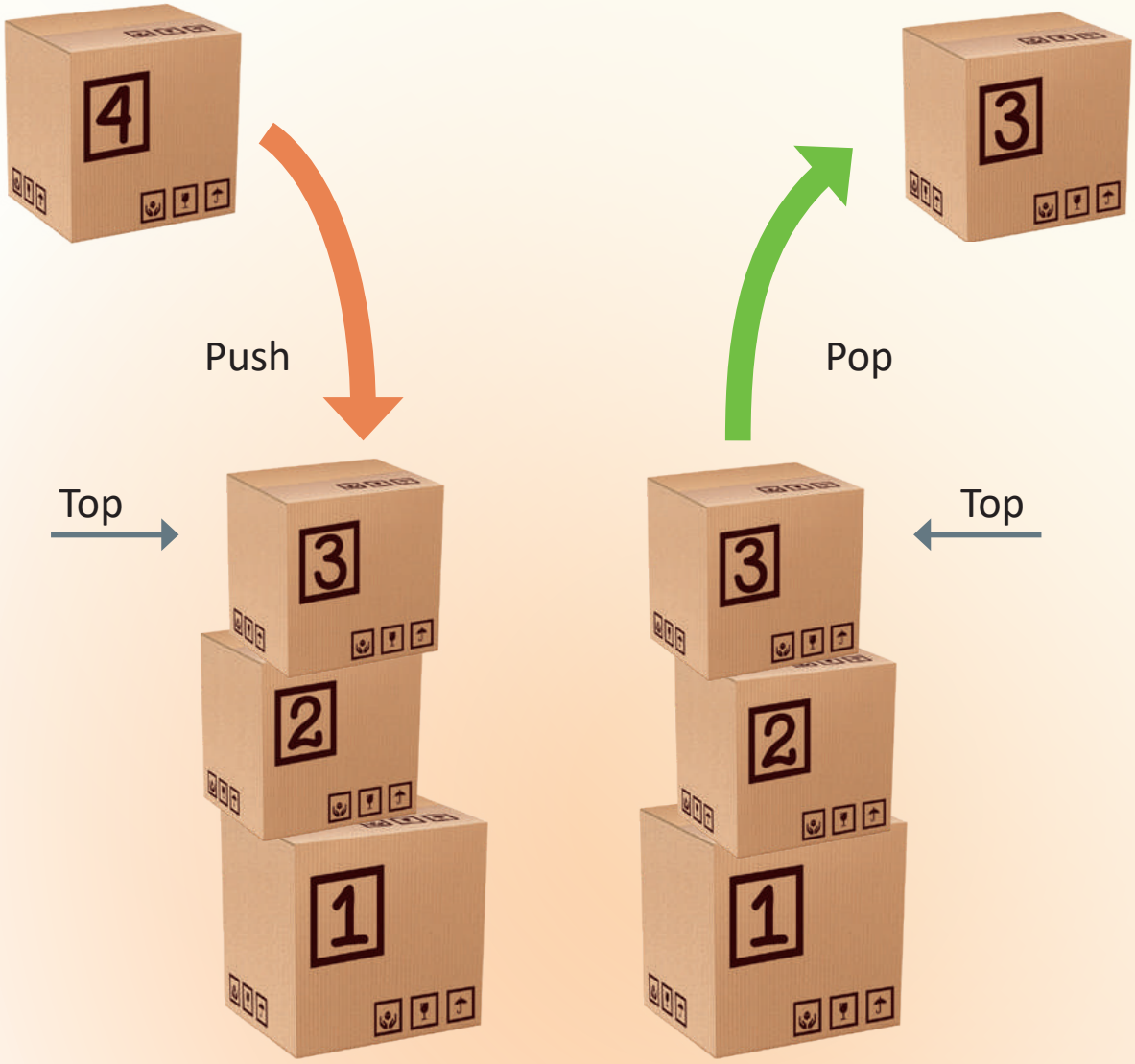
المُكدّس: هو هيكل بيانات يتبع قاعدة **Last In First Out (LIFO)** أي أن آخر من يدخل أول من يخرج، وأطلق عليه إسم المُكدّس لتكدس عناصره على التوالي.

العنصر الذي يتم إضافته مؤخرًا (آخر عنصر) يتم الوصول إليه أولًا، هذا المبدأ يُسمى (آخر من يدخل أول من يخرج) Last-In First-Out (LIFO).

- قد يكون حجم المُكدّس ثابتا أو قابلاً للتغيير.
- يتم تمثيل المكدسات في لغة Python باستخدام القوائم List.

هناك عمليتان رئيستان على المكدس:

- **إضافة (Push):** تستخدم هذه العملية لادخال عنصر إلى أعلى المكدس.
- **إزالة (Pop):** تستخدم هذه العملية لحذف العنصر من أعلى (قمة) المكدس.



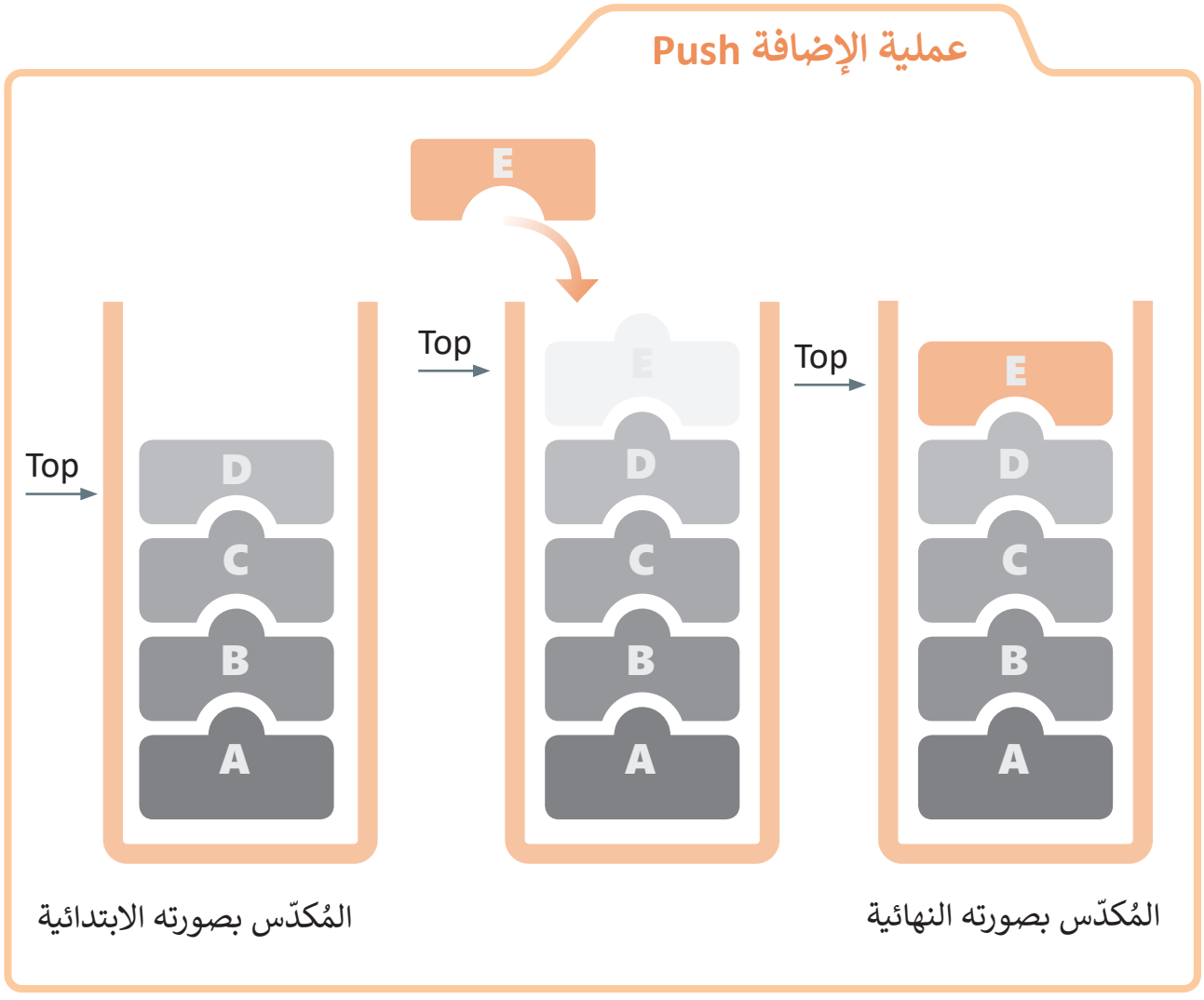
عملية الإضافة Push

يُطلق على إجراء إضافة عنصر جديد إلى المُكدّس **Push**.

يستخدم المُكدّس مؤشرًا يسمى Top، ويشير إلى العنصر في أعلى المُكدّس.

عند إضافة عنصر جديد في المُكدّس:

- تزيد قيمة المؤشر بمقدار 1.
- تتم إضافة العنصر الجديد إلى أعلى المُكدّس.



تجاوز سعة المُكدّس Stack Overflow

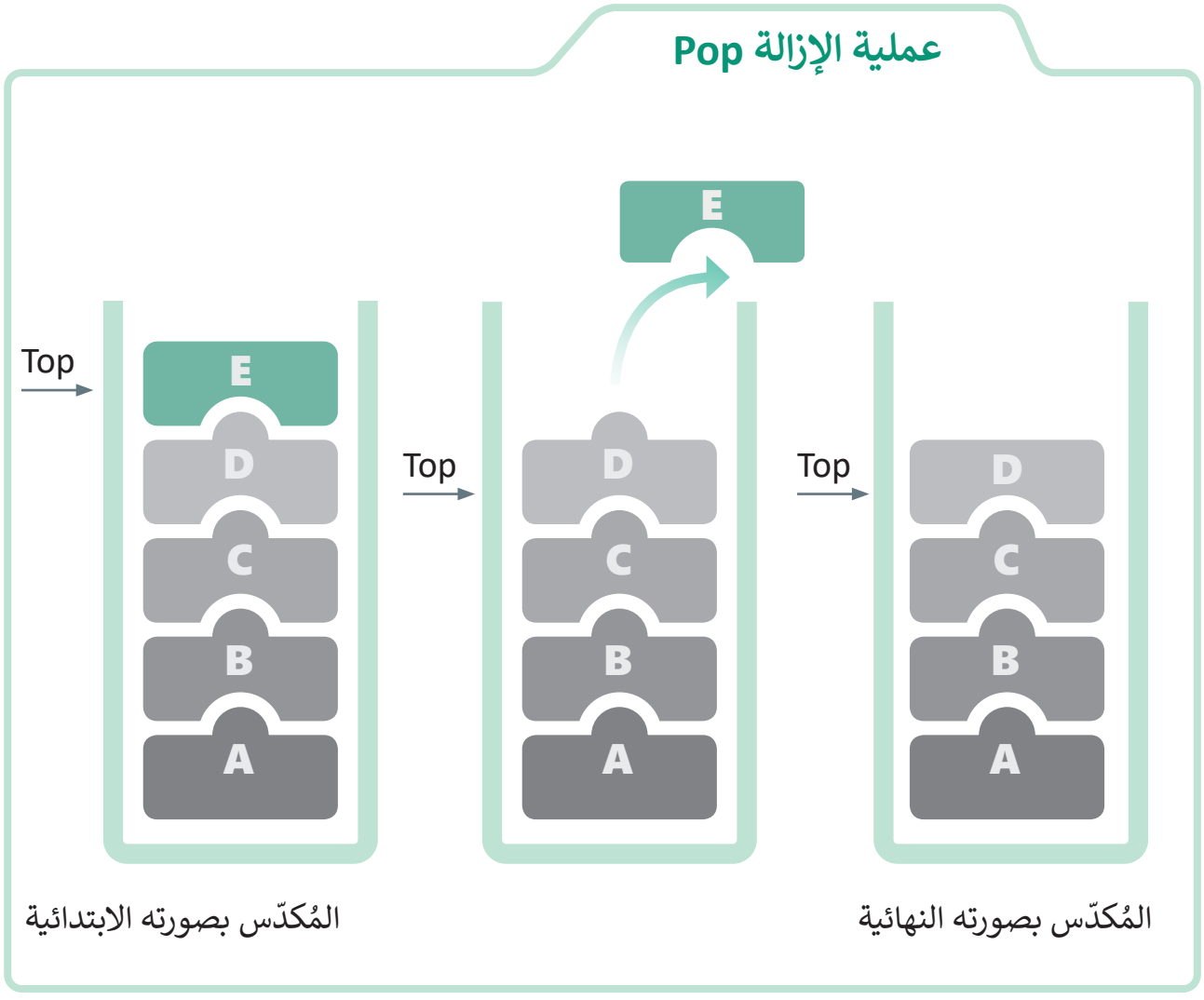
للمُكدّس سعة محددة تعتمد على ذاكرة الحاسوب، إذا كانت تلك السعة ممتلئة فإن إضافة عنصر جديد سيتسبب في تجاوز سعة المُكدّس **Stack Overflow**، وعليه ينبغي التحقق من امتلاء المُكدّس قبل إضافة أي عنصر.

عملية الإزالة Pop

يطلق على عملية إزالة عنصر من المُكدّس اسم **Pop**.

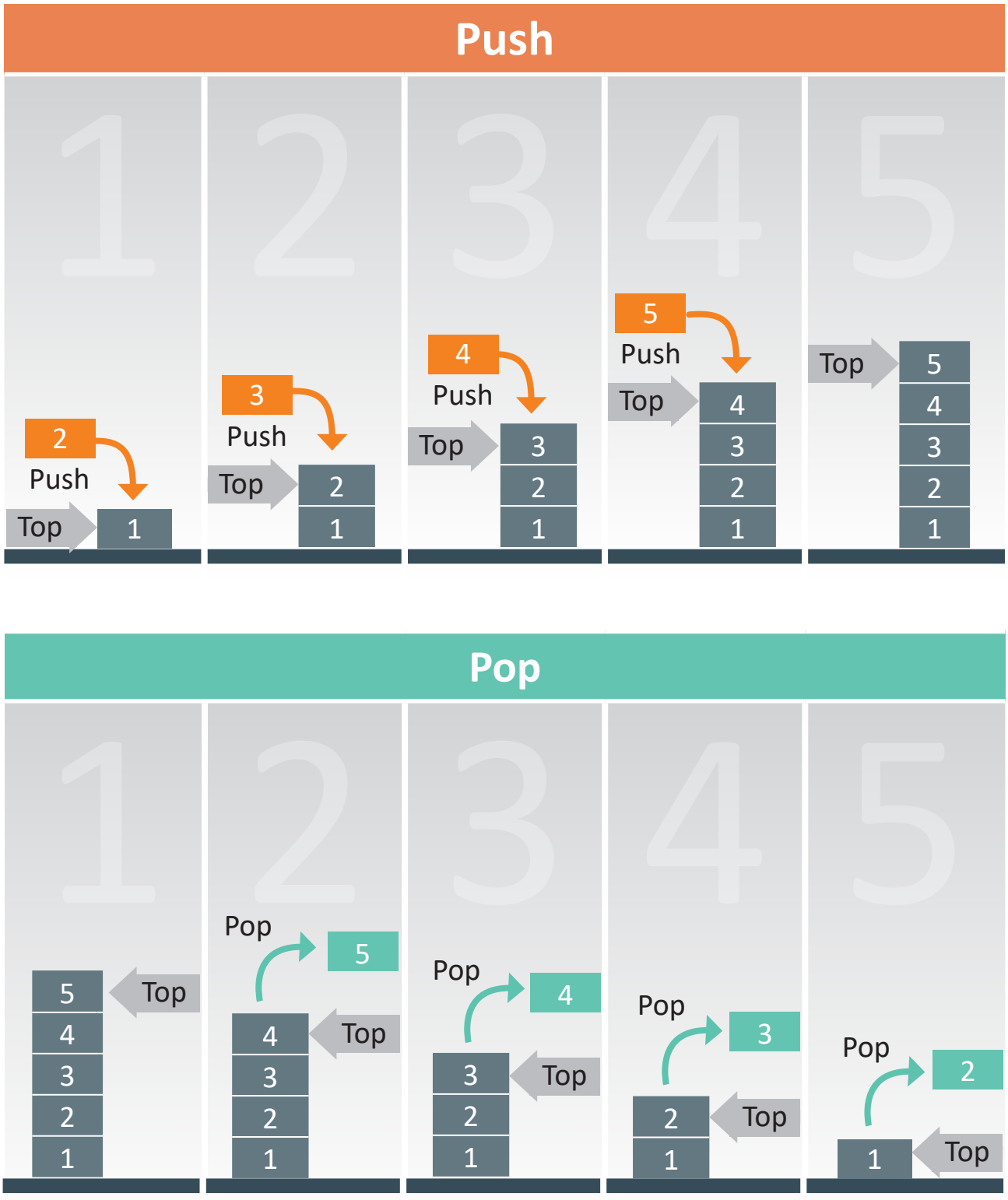
عند إزالة عنصر من المُكدّس:

- تنقص قيمة المؤشر بمقدار 1.
- يتم إزالة العنصر الموجود في أعلى المُكدّس.



تجاوز الحد الأدنى للمُكدّس Stack Underflow

إذا أردنا إزالة عنصر من المُكدّس، يجب التحقق من احتواء المكدس على عنصر واحد على الأقل؛ فإذا كان المُكدّس فارغًا فسوف نتسبب في تجاوز الحد الأدنى للمُكدّس (**Stack Underflow**).


Push
1
2
Push
Top
1
2
3
Push
Top
2
1
3
4
Push
Top
3
2
1
4
5
Push
Top
4
3
2
1
5
Top
5
4
3
2
1
Pop
1
5
Top
5
4
3
2
1
2
Pop
5
4
Top
4
3
2
1
3
Pop
4
3
Top
3
2
1
4
Pop
3
2
Top
2
1
5
Pop
2
1
Top
1

تمثيل المُكدّسات بلغة Python

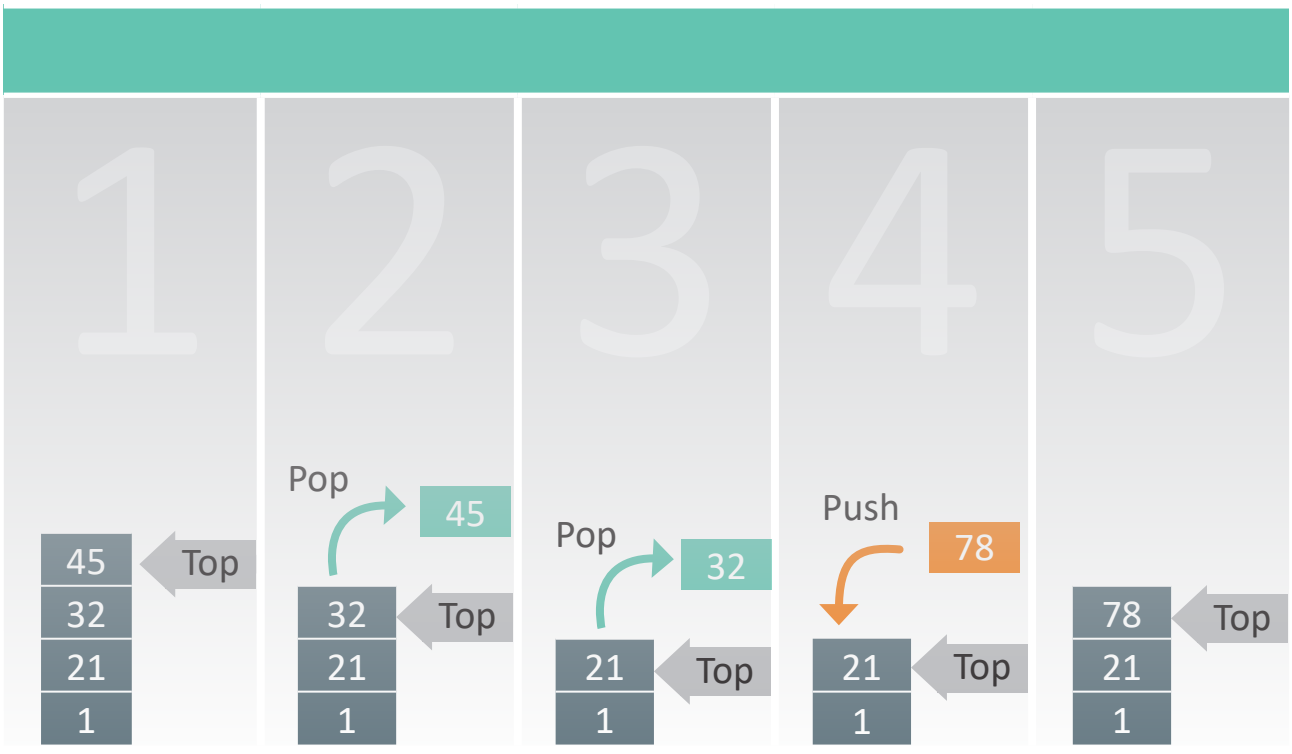
يتم تمثيل المُكدّس بلغة **Python** باستخدام القائمة **List**، والتي بدورها تقدم بعض العمليات الجاهزة لاستخدامها مع المُكدّسات.

عمليات المكدسات	
العملية	الوصف
listName.append(x)	إضافة العنصر x إلى نهاية القائمة التي تمثل المكدس.
listName.pop()	إزالة أخر عنصر من القائمة التي تمثل المكدس.

لنستعرض الأمثلة التالية على تمثيل المُكدّس في **Python**

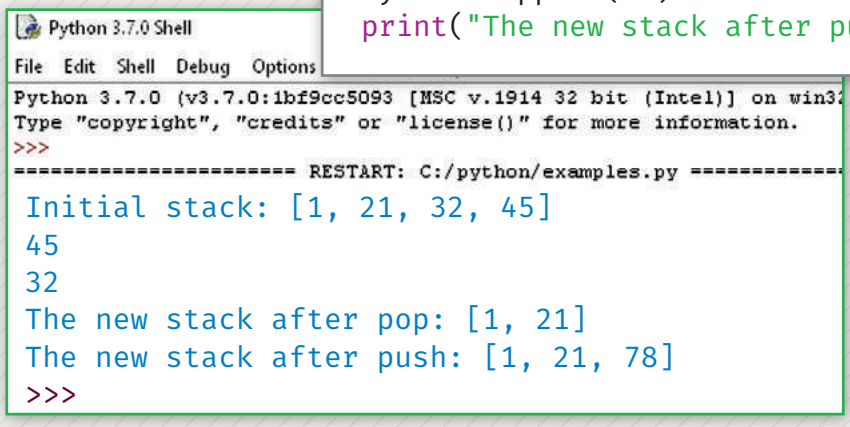
مثال: 1

1. سننشئ المُكدّس لحفظ مجموعة من الأعداد (1,21,32,45).
2. سنستخدم عملية **pop** مرتين لحذف آخر عنصرين من المُكدّس.
3. سنستخدم عملية **append** لإضافة عنصر جديد إلى المُكدّس.



نستخدم دالة
print(myStack.pop())
لعرض القيمة التي ترجعها
الدالة **myStack.Pop()**

```
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)
```

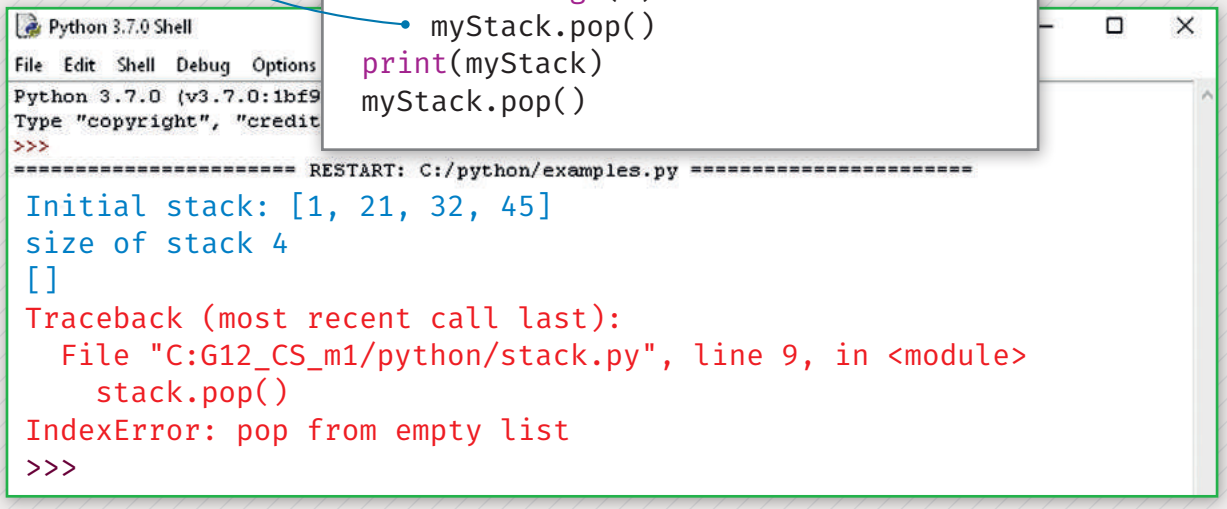


مثال: 2

لنرى ما الذي سيحدث إذا حاولنا إزالة عنصر من مكدس فارغ، ولكن أولًا علينا إفراغ المُكدّس.

تستخدم هذه الجملة لحذف جميع عناصر المُكدّس.

```
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()
```



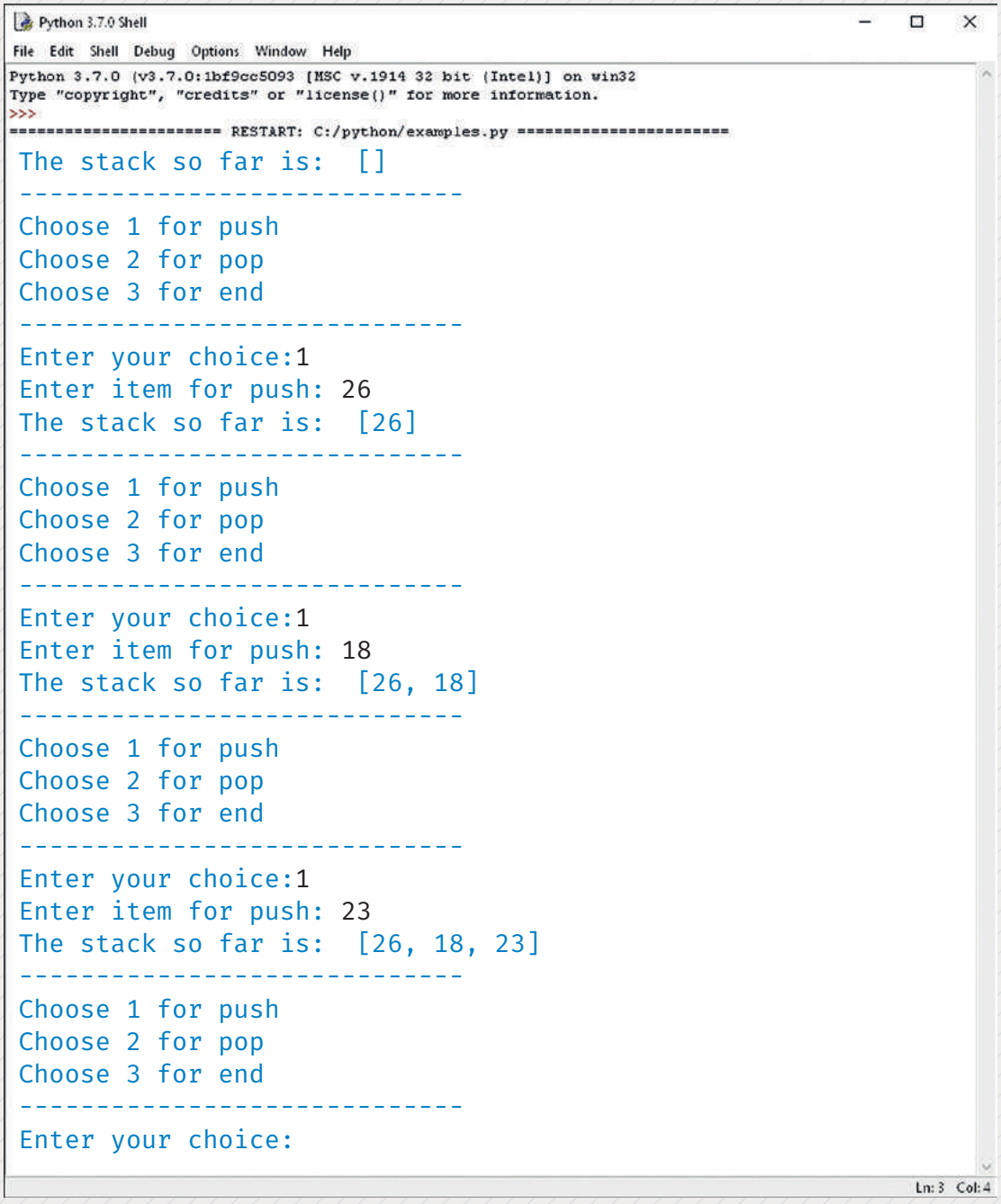
مثال: 3

في البرنامج التالي سيقوم المستخدم بإنشاء مُكدّس ثم يقوم بإضافة عنصر ثم إزالته، وسيعرض البرنامج قائمة تسأل المستخدم عن الإجراء الذي يود القيام به في كل مرة.

```
def push(stack,item):
    stack.append(item)
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 item for push: "))
        push(newStack,x)
    elif choice==2:
        if not isEmpty(newStack):
            print ("The pop item is:",pop(newStack))
        else:
            print ("The stack is empty")
    else:
        print ("End of program")
        break;
```

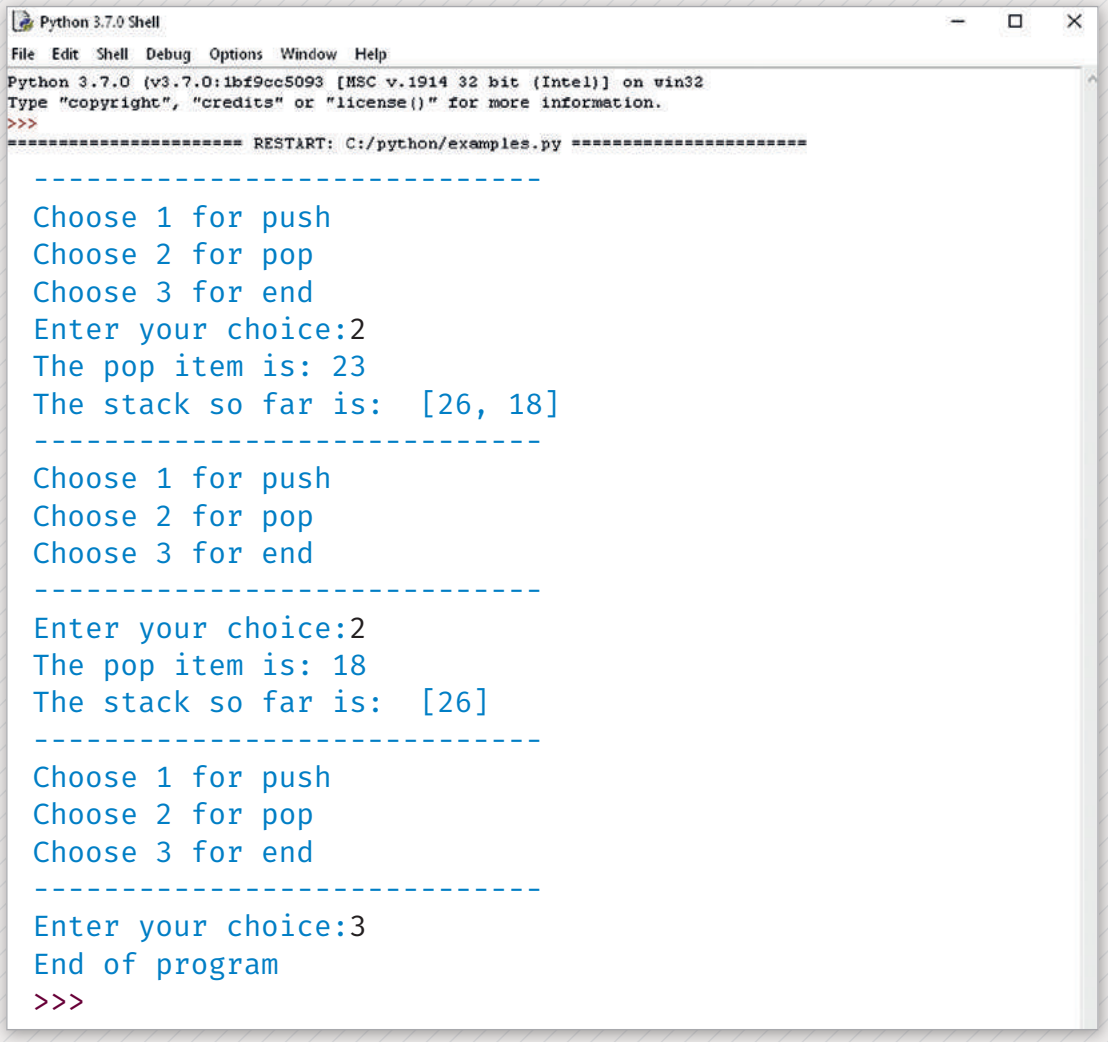
نلاحظ استخدام الأمر **break** وهو هنا للخروج من البرنامج ومقاطعة التكرار اللانهائي لجملة **while** عندما يقوم المستخدم باختيار الرقم 3

سنقوم بتنفيذ البرنامج السابق كالتالي: سننشئ مُكدّس من ثلاثة أعداد، ولإضافة العناصر إلى المُكدّس يجب الضغط على الرقم 1 من قائمة البرنامج.



لنقم الآن بإزالة عنصرين من المُكدّس ثم نخرج من البرنامج.

علينا الضغط على رقم 2 من قائمة البرنامج لإزالة عنصر، أما للخروج من البرنامج فعلينا الضغط على رقم 3 من القائمة.

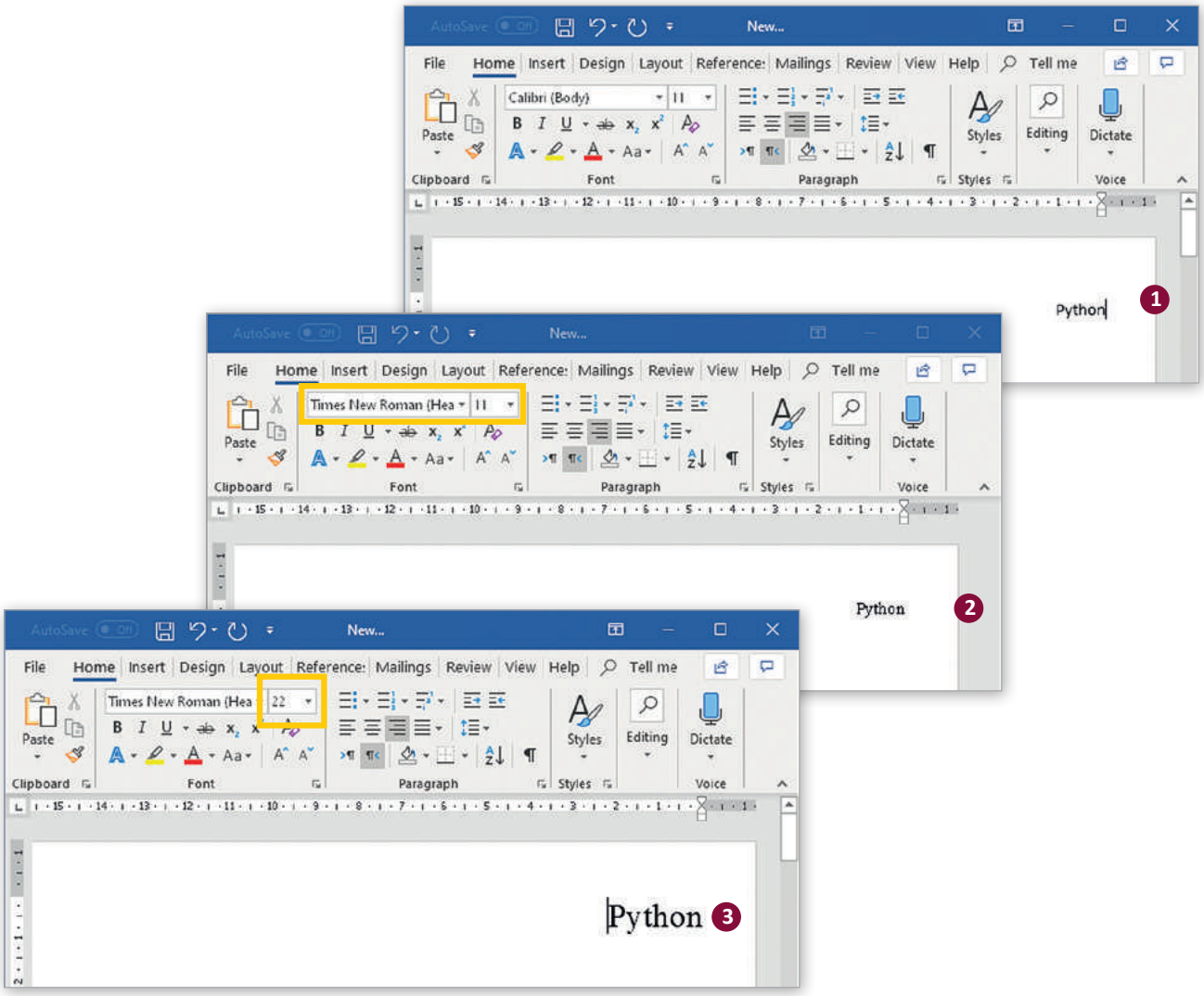


أمر التراجع undo

أحد التطبيقات على المُكدّس هو أمر التراجع (**undo**)، فإجراء التراجع في جميع التطبيقات في حقيقة الأمر يتم عن طريق استخدام المُكدّس، فنحن نضيف الحدث باستخدام **push** في مُكدّس التراجع وإذا أردنا التراجع فإننا نستخدم **pop** من المُكدّس السابق.

مُكدّس الأحداث التي قمنا بها

ترتيب العملية	الحدث	العملية
3	تغيير الحجم	push
2	تغيير نوع الخط	push
1	كتابة النص	push

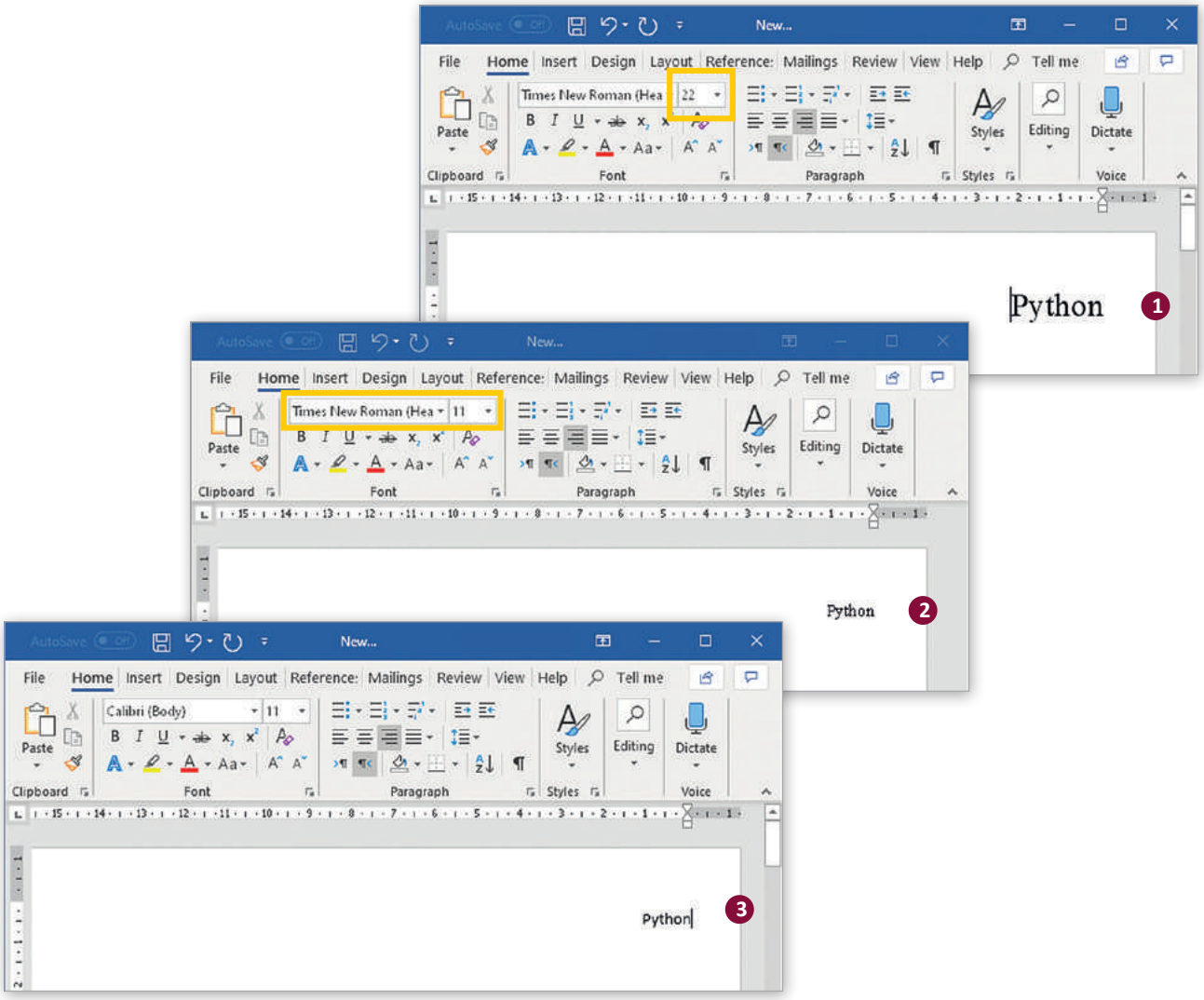


لنفترض أننا ضغطنا على زر التراجع **Undo**، فإن أول إجراء سيتم التراجع عنه هو تغيير حجم الخط، وإذا ضغطنا مرةً أخرى زر التراجع، فسيتم التراجع عن إجراء تغيير نوع الخط.

بضغطنا زر التراجع يتم تنشيط عملية **pop** (الإزالة من المُكدّس).

مكدس التراجع (undo stack) هي تاريخ الحركات التي قمنا بها.

مكدس التراجع		
ترتيب العملية	الحدث	العملية
1	تغيير حجم الخط	pop
2	تغيير نوع الخط	pop
3	كتابة النص	pop



الطابور Queue

نشاهد في حياتنا اليومية العديد من الطوابير، ولعل من أكثرها شيوعًا هو طابور انتظار السيارات عند إشارة المرور، وما نلاحظه أنه بمجرد تحول إشارة المرور إلى اللون الأخضر فإن السيارة التي تدخل أولاً في طابور السيارات هي التي تخرج أولاً.

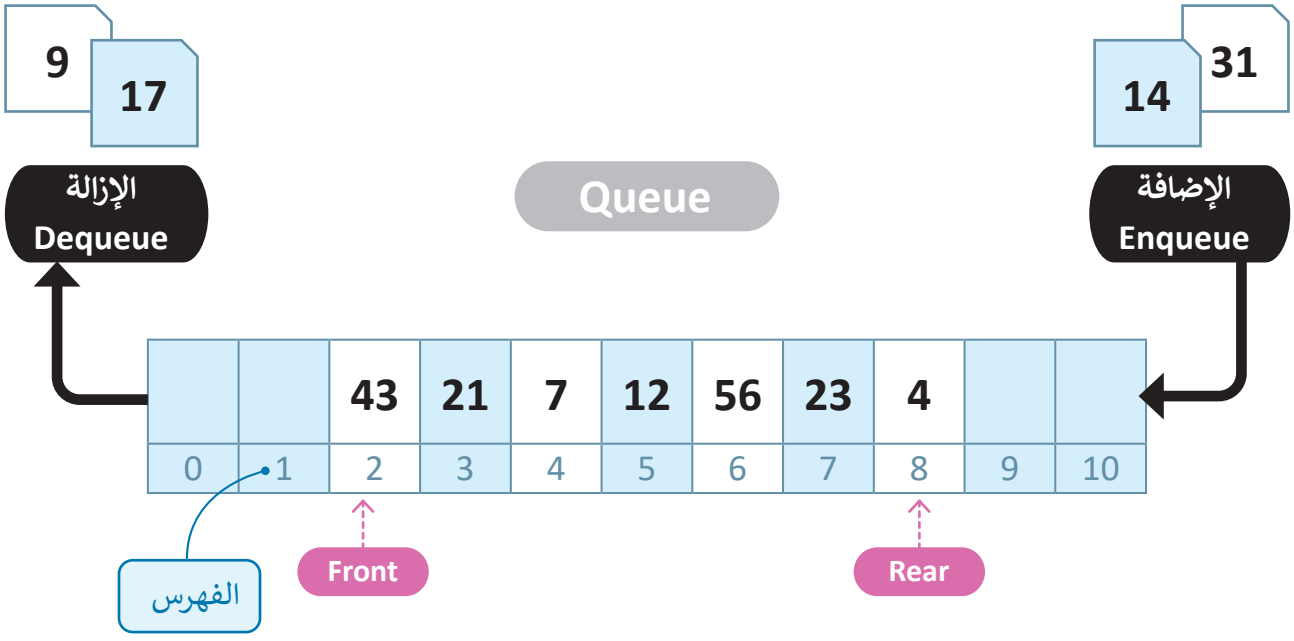
الطابور هو هيكل بيانات يتبع قاعدة **First In First Out (FIFO)** أي أن أول من يدخل هو أول من يخرج، وهذا يعني أن كل عنصر في قائمة الطابور يتم التعامل معه حسب ترتيب وصوله.

أهم مظاهر الاختلاف بين المكدس والطابور هو أنه في المكدس يتم الإضافة والحذف من نفس الطرف بينما في الطابور يتم الإضافة من طرف والحذف من طرف آخر، ولذلك في المكدس عند الحذف يتم حذف آخر عنصر تم إضافته بينما في الطابور يتم حذف أول عنصر تمت إضافته.

توجد عمليتان رئيستان خاصتان بالطابور:

- **إضافة (Enqueue):** وهي العملية التي يتم فيها إضافة عنصر إلى آخر الطابور.
- **إزالة (Dequeue):** وهي العملية التي يتم فيها إزالة عنصر من مقدمة الطابور.

في قائمة الانتظار الخاصة بالطابور، نستخدم مؤشرين: المؤشر الأمامي ويسمى **Front** ويشير إلى موضع أول عنصر في الطابور، والمؤشر الخلفي **Rear**، والذي يشير إلى موضع العنصر الأخير في الطابور.



المؤشر Pointer

المؤشر هو متغير يشير إلى عنوان عنصر من عناصر الطابور.

للطابور مؤشرين:

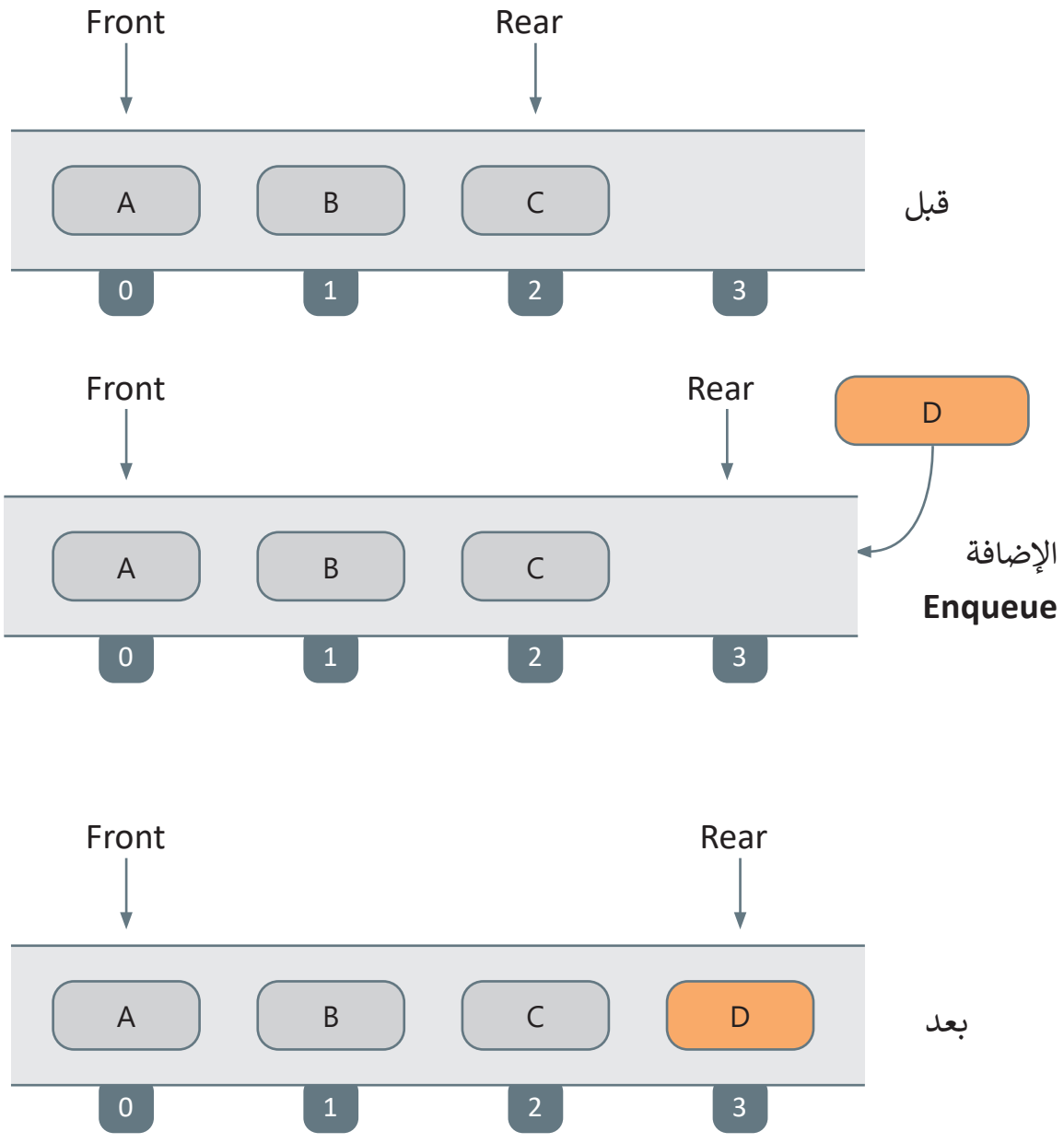
- المؤشر الأول (**Front**): يشير إلى العنصر الأول في الطابور (أصغر قيمة).
- المؤشر الثاني (**Rear**): يشير إلى آخر عنصر في الطابور (أكبر قيمة).

في حال كان الطابور فارغاً فإن قيمة المؤشر Front تتساوى مع قيمة المؤشر Rear وهي صفر.

عملية الإضافة للطابور Enqueue

يطلق على عملية إضافة عنصر جديد إلى الطابور **Enqueue**.

لإضافة عنصر جديد إلى الطابور، يتم زيادة قيمة المؤشر **Rear** بمقدار 1، حيث يشير إلى موضع العنصر الجديد الذي قمنا بإدخاله وتبقى قيمة المؤشر **Front** كما هي لا تتغير.

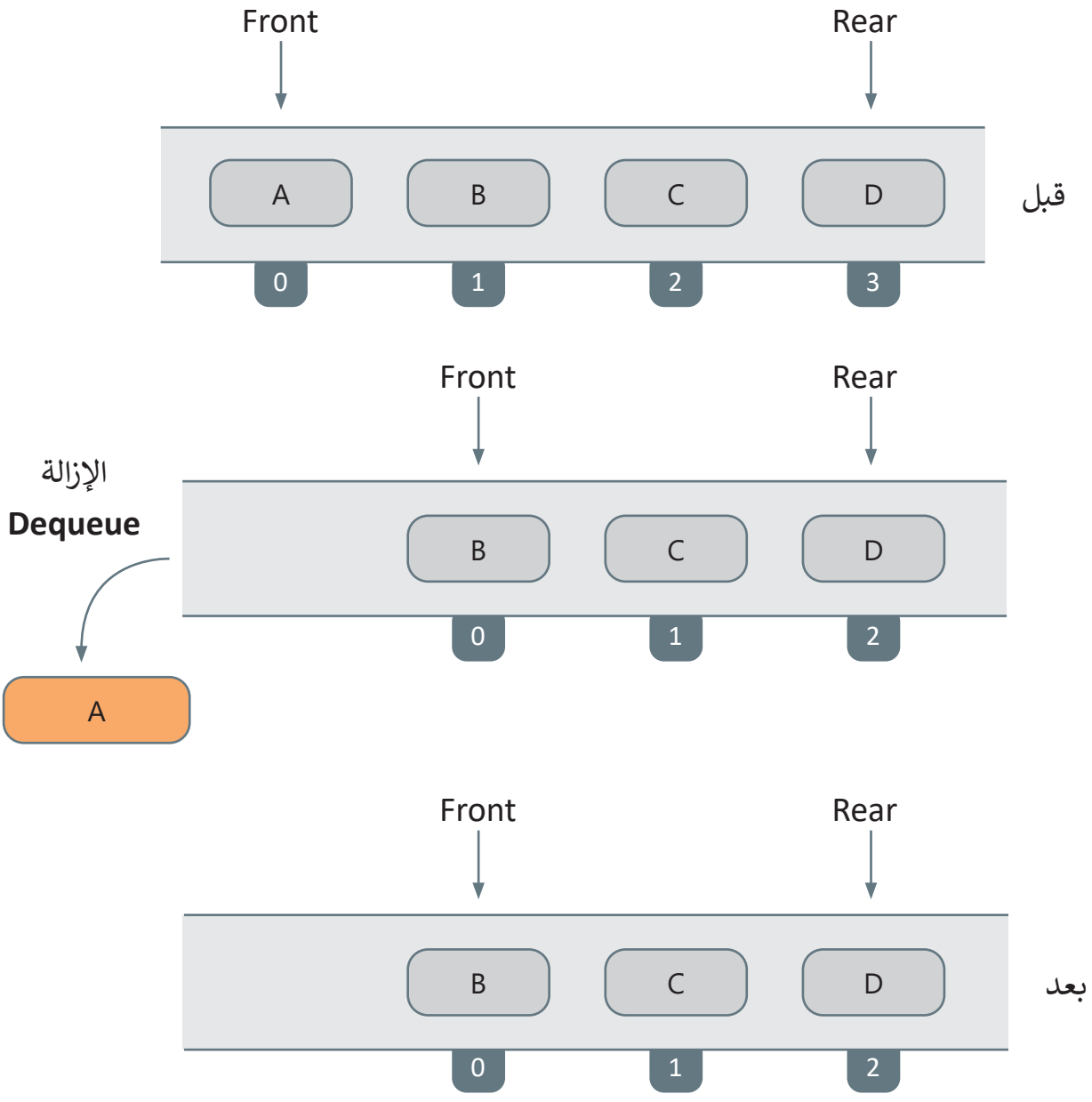


لا يمكننا إضافة أو إزالة عنصر من منتصف الطابور.

عملية الإزالة من الطابور Dequeue

يطلق على عملية إزالة عنصر من الطابور **Dequeue**.

يتم إخراج العنصر الذي يشير المؤشر الأمامي **Front** إليه ليشير إلى العنصر التالي.



يجب التحقق من وجود مساحة كافية في الطابور لإضافة عنصر جديد، كما يجب التأكد من وجود عنصر واحد على الأقل ليتم إزالته.

تمثيل الطابور باستخدام لغة Python

في لغة البرمجة بايثون يمكن تمثيل الطابور بعدة طرق منها القوائم **Lists** يرجع ذلك لأن القائمة تمثل مجموعة من العناصر الخطية وأيضا لإمكانية اضافة عنصر في نهاية القائمة وإمكانية حذف عنصر من بدايتها.

فيما يلي الصيغ العامة لبعض العمليات التي يمكن إجراؤها على الطابور:

عمليات الطابور	
العملية	**الوصف**
listName.append(x)	إضافة العنصر x إلى نهاية القائمة التي تمثل الطابور.
listName.pop(0)	إزالة أول عنصر من القائمة التي تمثل الطابور، حيث يشير (0) إلى أول عنصر.

ربما لاحظت أن دالة listName.pop() الخاصة بالمكدس ليس لها وسائط **(Arguments)**، ولكن بالنسبة للطابور نستخدم الدالة listName.pop(0)، فيمكنك ملاحظة وجود الرقم صفر داخل الأقواس.

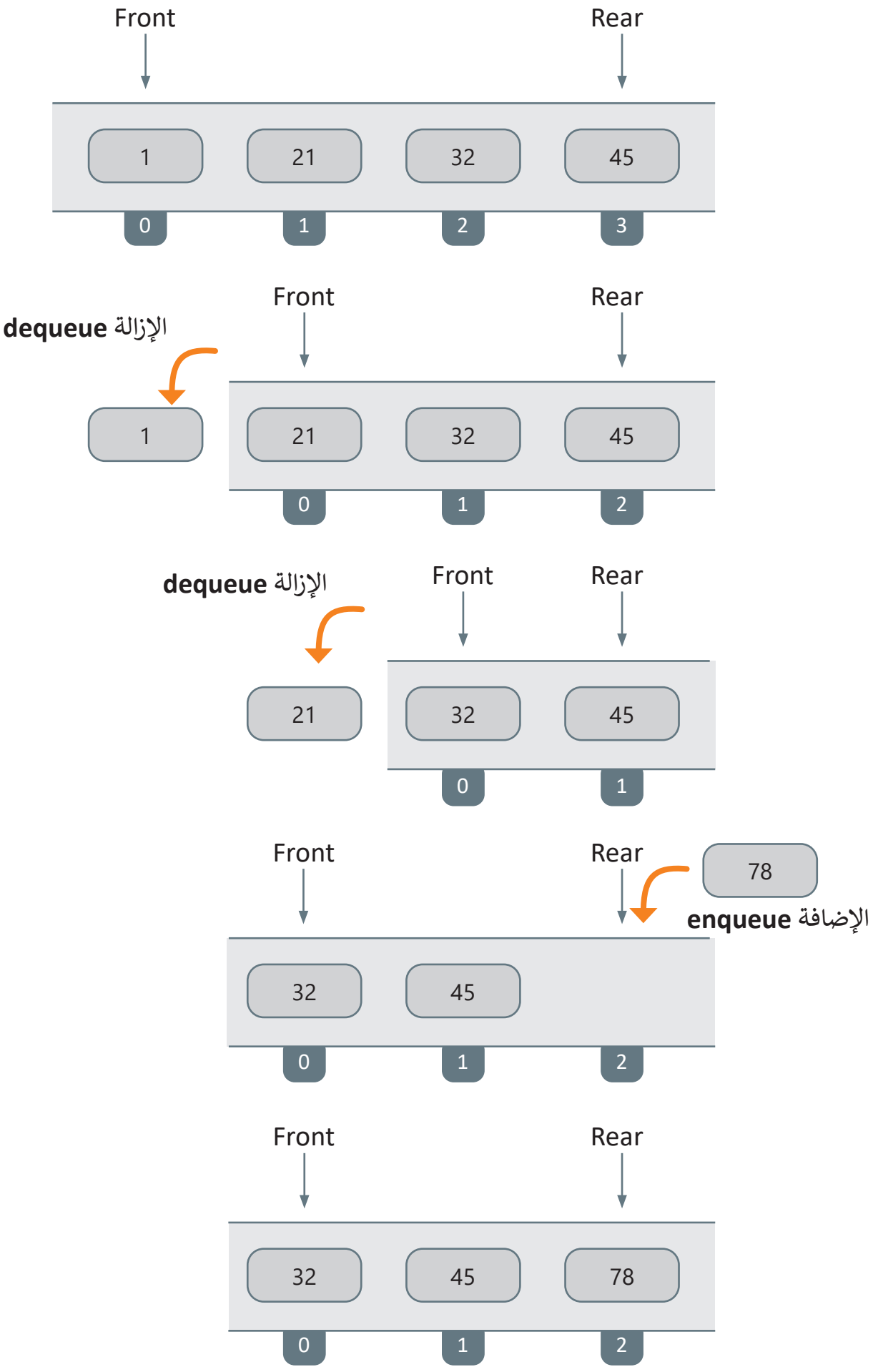
لنتعرف على الفرق بين الدالتين.

مقارنة عملية listName.pop() وعملية listName.pop(0)	
العملية	**الوصف**
listName.pop()	إذا كان وسيط الدالة فارغًا، ستتم إزالة آخر عنصر من نهاية القائمة التي تمثل المكدس.
listName.pop(0)	إذا كان وسيط الدالة صفرًا، ستتم إزالة أول عنصر من عناصر القائمة التي تمثل الطابور.

لنستعرض الأمثلة التالية على كيفية تمثيل الطابور بلغة **Python**

مثال: 4

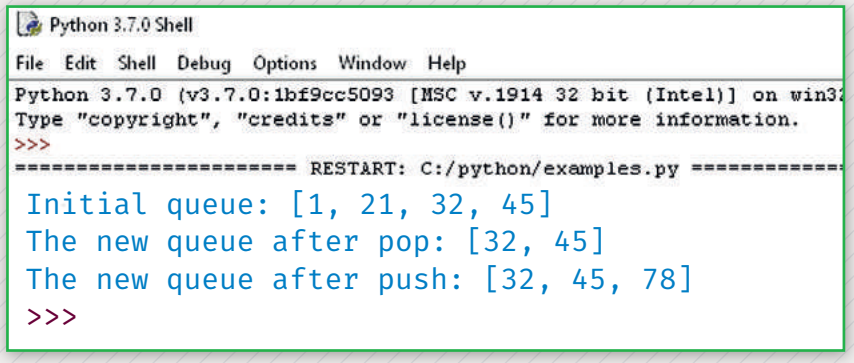
1. سننشئ طابورًا لمجموعة من الأعداد (1,21,32,45).
2. ستستخدم عملية **pop** مرتين لإزالة أول عنصرين من الطابور
3. سنستخدم عملية **append** لإضافة عنصر جديد إلى الطابور.


Front
Rear
1
21
32
45
0
1
2
3
dequeue الإزالة
Front
Rear
1
21
32
45
0
1
2
dequeue الإزالة
Front
Rear
21
32
45
0
1
Front
Rear
78
enqueue الإضافة
32
45
0
1
2
Front
Rear
32
45
78
0
1
2

لنبرمج الخطوات السابقة في Python.

سنستخدم القائمة لتطبيق مفهوم هيكل الطابور كما سبق وفعلنا مع المكدس.

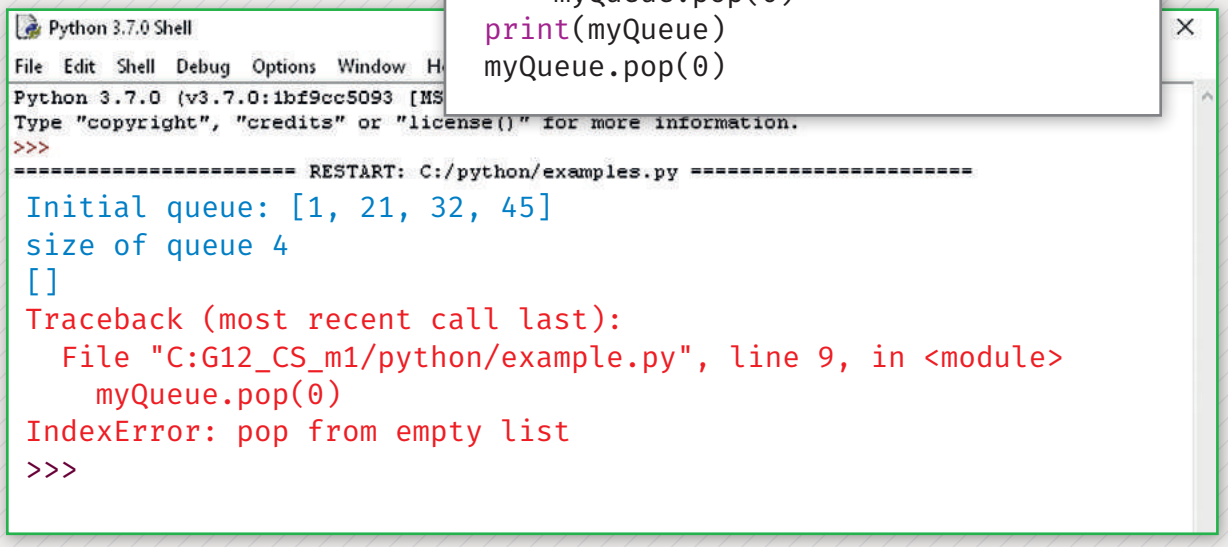
```
myQueue=[1,21,32,45]
print("Initial queue:", myQueue)
myQueue.pop(0)
myQueue.pop(0)
print("The new queue after pop:", myQueue)
myQueue.append(78)
print("The new queue after push:", myQueue)
```



مثال: 5

لنرى ماذا سيحدث إذا حاولنا إزالة عنصر من طابور فارغ، لكن أولًا علينا إفراغ الطابور.

```
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)
```



نلاحظ ظهور خطأ وسببه أننا قمنا بكتابة أمر حذف عنصر من طابور فارغ، ولذلك يجب التأكد دائمًا من وجود عناصر داخل الطابور قبل القيام بعملية الحذف.

تطبيقات الطابور في الحاسوب

أحد الأمثلة الشائعة على الطابور في علوم الحاسوب هو طابور الطباعة. على سبيل المثال، لدينا مختبر حاسوب يحتوي على 30 جهاز حاسوب متصل بطابعة واحدة. عندما يرغب الطلبة في الطباعة فإن مهام الطباعة الخاصة بهم تنشئ طابورًا. يتم وضع المهام في طابور لتتم معالجتها بطريقة **First-In First-Out (FIFO)**. سيتم طباعة المهام بناءً على الترتيب الزمني لإرسالها فالمهمة التي أرسلت أولاً سيتم طباعتها قبل التي أرسلت بعدها. المهمة التي في نهاية الطابور لن يتم طباعتها حتى تتم طباعة جميع المهام التي سبقتها. عندما تكمل الطابعة مهمة ستنظر في الطابور لمعرفة ما إذا كانت هناك أي مهام متبقية للمعالجة.

تمثيل المُكدّس والطابور باستخدام Queue Library

بخلاف الطريقة السابقة التي استخدمنا فيها القائمة لبناء كل من هيكل المُكدّس وهيكل الطابور، توفر **Python** مكتبة **Library** تستخدم في تمثيل هياكل الطابور أو المُكدّس.

تتضمن مكتبة **Queue** القياسية بعض الدوال الجاهزة للاستخدام التي يمكن استخدامها مع المُكدّس أو الطابور.

العمليات في مكتبة Queue القياسية	
العملية	**الوصف**
queueName=queue.Queue()	انشاء طابور جديد تحت مسمى queueName.
queueName.put(x)	إضافة العنصر x إلى الطابور.
queueName.qsize()	إرجاع حجم الطابور.
queueName.get()	جلب وإزالة أول عنصر من الطابور وآخر عنصر من المُكدّس.
queueName.full()	ترجع True إذا كان المُكدّس ممتلئ وFalse إذا كان المُكدّس فارغ ويمكن تطبيقها على الطابور أيضا.
queueName.empty()	ترجع True إذا كان المكدس فارغ وFalse إذا كان المُكدّس ممتلئ ويمكن تطبيقها على الطابور أيضا.

مثال: 6

لنستخدم مكتبة queue القياسية لإنشاء طابور.

سنقوم في هذا المثال بـ:

- استدعاء المكتبة queue
- إنشاء طابور فارغ myQueue
- إضافة العناصر a,b,c,d,e للطابور myQueue
- طباعة عناصر الطابور

لا بد من استدعاء الوحدة القياسية queue قبل الشروع في العمليات التي يمكن اجراءها على الطابور.

```
import queue

myQueue = queue.Queue()

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

#print the items of the queue
for item in list(myQueue.queue):
    print(item)
```

طباعة عناصر الطابور.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  W
Python 3.7.0 (v3.7.0:1bf9cc
Type "copyright", "credits"
>>>
========== RES
a
b
c
d
e
>>>
```

مثال: 7

لننشئ طابورًا يتم إدخال خمس قيم فيه من قبل المستخدم أثناء تنفيذ البرنامج ثم طباعة هذه القيم وطباعة حجم الطابور في النهاية.

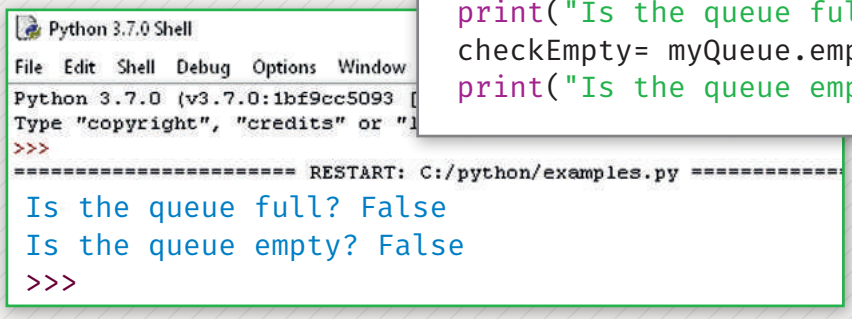
```
import queue

myQueue = queue.Queue()

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

#print the items of the queue
for item in list(myQueue.queue):
    print(item)

print ("Queue size is :",myQueue.qsize())
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  H
Python 3.7.0 (v3.7.0:1bf9cc5093 [MS
Type "copyright", "credits" or "lic
>>>
========== RESTART: C:
enter queue item: 5
enter queue item: f
enter queue item: 12
enter queue item: b
enter queue item: 23
5
f
12
b
23
Queue size is :5
>>>
```

مثال: 8

سنقوم بانشاء برنامج لنتحقق ما إذا كان الطابور فارغًا أم ممتلئًا.

```
import queue

myQueue = queue.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)
```



كما ذكرنا سابقًا، تتضمن مكتبة **Queue** القياسية بعض الدوال الجاهزة للاستخدام التي يمكن استخدامها مع المكدس أو الطابور. يوضح الجدول التالي وظائف المكتبة التي نستخدمها مع المكدس.

العمليات في مكتبة Queue القياسية	
العملية	**الوصف**
stackName=queue.LifoQueue()	إنشاء مكدس جديدة تحت مسمى stackName.
stackName.get()	ازالة اخرعنصر من المكدسة stackName

مثال: 9

لنستخدم مكتبة **queue** القياسية لإنشاء مكدس فارغ.

إنشاء مكدس فارغ باسم **myStack**

```
import queue

myStack = queue.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)

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

طباعة وإزالة العناصر من المكدس.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  H
Python 3.7.0 (v3.7.0:1bf9cc5093 [MS
Type "copyright", "credits" or "lic
>>>
========== RESTART: C:/python/examples.py ==========
e
d
c
b
a
Is the stack empty? True
>>>
```

تذكر أن العمليات في المكدس تسير وفق قاعدة LIFO.

عند استخدام الدالة get مع الطابور ستكون عملية الجلب والطباعة وفق القاعدة FIFO.

مثال:10

عندما يرسل المستخدمون مهام الطباعة تتم إضافتها إلى طابور الطباعة. تستخدم الطابعة هذا الطابور لمعرفة ما هو ملف الطباعة التالي.

> لنفترض أن طاقة استيعاب الطابعة هي 7 ملفات فقط في نفس الوقت ونحن بحاجة لطباعة 10 ملفات من الملف A إلى الملف J.

> اكتب برنامجًا يجسد طابور الطباعة من بداية مهمة الطباعة الأولى (A) حتى اكتمال جميع مهام الطباعة.

يمكنك الاستعانة بالخوارزمية الآتية:

- إنشاء طابور مهام الطباعة

- إدخال الملفات من A إلى G داخل طابور مهام الطباعة

- إخراج الملف A

- إدخال الملف H

- إخراج الملف B

- إدخال الملف I

- إخراج الملف C

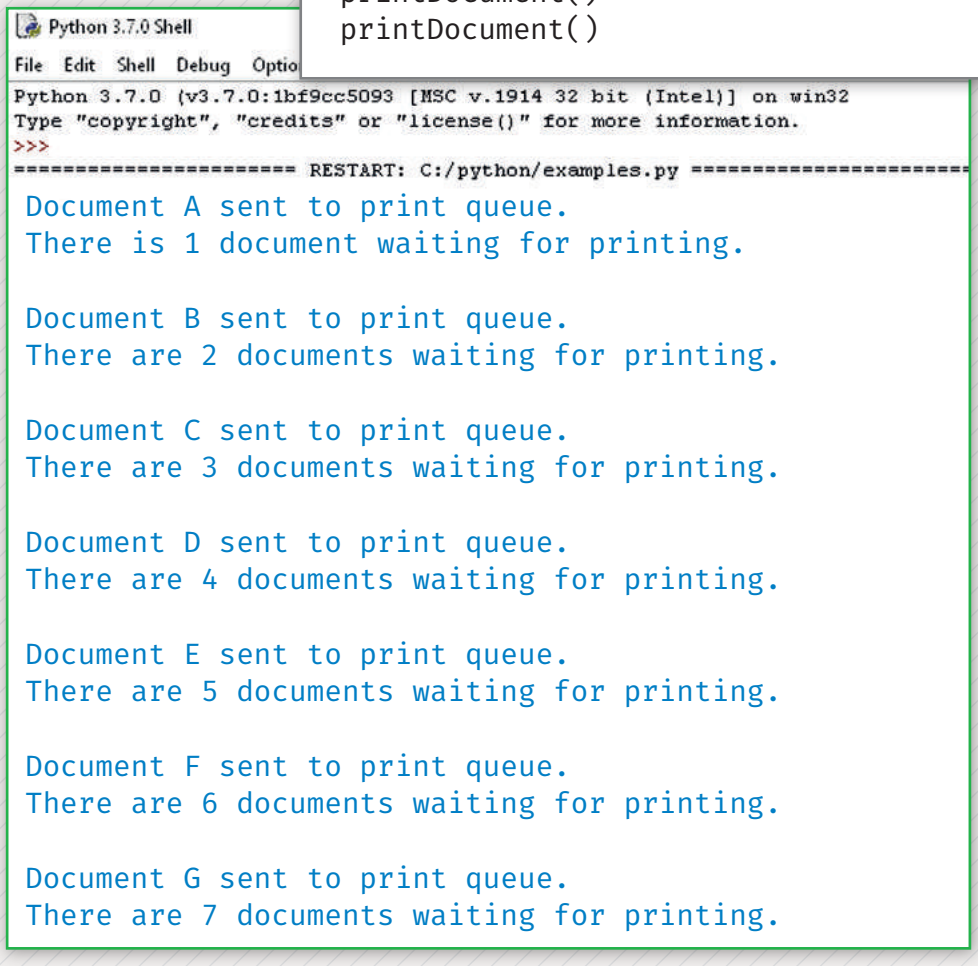
- إدخال الملف J

- إخراج الملفات التي تمت طباعتها (D-E-F-G-H-I-J) الواحد تلو الآخر.

> أضف المقطع البرمجي الذي يمكن من التأكد من أن طابور مهام الطباعة فارغا.

```
#import the queue library
import queue
#import the time library to use the sleep function
import time

#initiliase variables and the queue
printDocument = ""
printQueueSize = 0
printQueueMaxSize = 7
printQueue = queue.Queue(printQueueMaxSize)

# add a document to the print 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 0.5 second
    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 1 second
    print ("OK -", printDocument, "is printed.")
    printQueueSizeMessage()

# print a message with the queue size
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()
```

```
# main program
# send documents to 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()
printDocument()
printDocument()
```



```
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.
>>>
```

1

اختر الإجابة الصحيحة مما يلي.

1. يتبع المكدس قاعدة	.LIFO rule
	.FIFO rule
	.FIKO rule
2. يتبع الطابور قاعدة	.LIFO rule
	.FIFO rule
	.FIKO rule
3. لإضافة عنصر جديد في الطابور نستخدم	عملية push
	عملية dequeue
	عملية enqueue
4. لإزالة عنصر من المكدس نستخدم	عملية push
	عملية pop
	عملية enqueue
5. مؤشر Rear يشير إلى	أول عنصر في الطابور
	آخر عنصر في الطابور
	العنصر الأوسط في الطابور
6. مؤشر Top يشير إلى	أول عنصر في الطابور
	آخر عنصر في الطابور
	آخر عنصر تمت إضافته في المكدس

2

اكتب مثالًا على استخدام المكدس في حياتنا اليومية.

_____

_____

_____

_____

_____

3

اكتب مثالًا على استخدام الطابور في حياتنا اليومية.

_____

_____

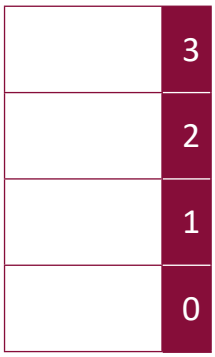
_____

_____

_____

4

أكمل تعبئة المكدس التي تم إنشاؤه بالمقطع البرمجي التالي موضحًا موضع المؤشر Top.

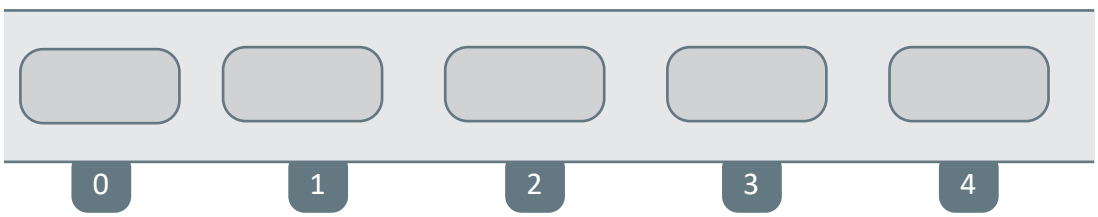



```
import queue

myStack = queue.LifoQueue()

myStack.put("a")
myStack.put("H")
myStack.put("o")
myStack.put("D")
```

5

أكمل تعبئة الطابور الذي تم إنشاؤه بالمقطع البرمجي التالي موضحًا موضع مؤشري Front وRear.

```
import queue

q = queue.Queue()

q.put(22)
q.put("k")
q.put(35)
q.put(42)
q.put("l")
```



6

لدينا مكدس بستة أماكن فارغة.

> سنضيف الحروف التالية C و E و B و A و D في المواضع من 0 إلى 4.

> قم بتعبئة المكدس مبينًا موضع المؤشر Top.

> في ورقة خارجية، نفذ العمليات التالية:

pop <

pop <

pop <

push X <

push K <

pop <

	5
	4
	3
	2
	1
	0

> وضح الناتج النهائي بعد تنفيذ العمليات أعلاه مبينًا موضع المؤشر top.

	5
	4
	3
	2
	1
	0

> اكتب برنامجًا ينشيء المكدس الموضح أعلاه، ثم نفذ العمليات المذكورة باستخدام مكتبة queue القياسية.

7

لديك تسلسل الأعداد التالي: 4، 8، 2، 5، 9، 13.

> ما هي العملية المستخدمة لإضافة العناصر أعلاه إلى الطابور.

_____

> أكمل الطابور بعد إضافة العناصر.

0	1	2	3	4	5

> ما هي العملية المستخدمة لإزالة العناصر من الطابور؟

_____

> كم مرة يجب تنفيذ العملية السابقة لإزالة العنصر الذي يحمل القيمة 5؟

_____

8

أنشئ الطابور السابق باستخدام البرمجة بلغة Python.

> قم بإزالة جميع العناصر من الطابور.

> تحقق مما إذا كان الطابور فارغًا في النهاية.

9

ترغب إدارة أحد البنوك في تنفيذ برنامج لإدارة خدمة العملاء. يستقبل البرنامج من موظف البنك أرقام العملاء الموجودين في الانتظار ويخزنهم في طابور مكون من 10 خانات، يقوم الموظف نفسه بعرض رقم العميل على الشاشة عندما يأتي دوره وإنهاء البرنامج عند الانتهاء من خدمة جميع العملاء.

قم بإعداد برنامج يقوم بـ :

> إدخال أرقام العملاء من المستخدم.

> إذا تم إعطاء كلمة Next فسيخرج البرنامج رقم العميل التالي ويعرضه على الشاشة، إذا لم يكن هناك عملاء في الانتظار فسيعرض الرسالة "No one waits".

> إذا تم إعطاء الكلمة END فلن يسمح البرنامج بإدخال أرقام أخرى باستثناء قيمة Next.

> في أي حالة أخرى يتم إدخال رقم العميل في الطابور إلا إذا كان الطابور ممتلئ، يتم عرض الرسالة "No more space".

> يتوقف البرنامج عندما يتم إعطاء الكلمة END ويكون الطابور فارغ.

هياكل البيانات الثابتة والديناميكية

كما أشرنا مسبقًا بأن هياكل البيانات هي طريقة لتخزين وتنظيم البيانات بكفاءة.

قمنا سابقًا بتصنيف هياكل البيانات إلى بسيطة وغير بسيطة، كما يمكن تصنيفها أيضًا إلى:

> ثابتة **Static**.

> ديناميكية (متغيرة) **Dynamic**.

هياكل البيانات الثابتة

تتميز هياكل البيانات الثابتة بحجمها الثابت، فيتم تخصيص مواقع متجاورة في الذاكرة لعناصر البيانات، ومن أكثر الأمثلة شيوعًا على هذا النوع هو المصفوفات.

هياكل البيانات الديناميكية (المتغيرة)

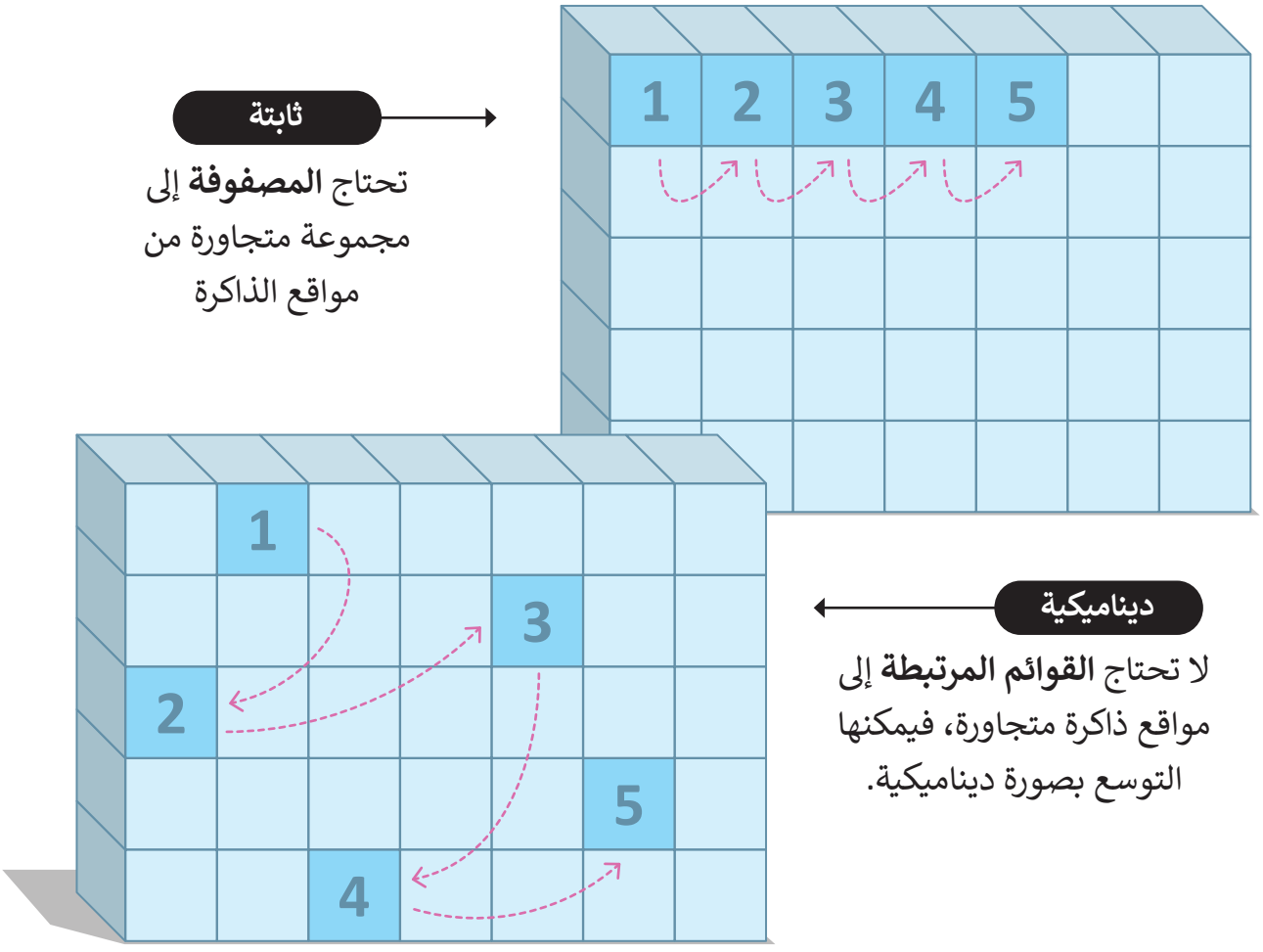
في هذه الهياكل يمكن تغيير الحجم أثناء تنفيذ البرنامج اعتمادًا على العمليات التي يتم إجراؤها عليها. تم تصميم هياكل البيانات الديناميكية لتوفير المرونة في تغيير حجم هياكل البيانات أثناء وقت التشغيل.

من أكثر طرق تمثيل هياكل البيانات الديناميكية شيوعًا القائمة المرتبطة.

هياكل البيانات الثابتة وهياكل البيانات الديناميكية		
	الثابتة	الديناميكية
حجم الذاكرة المخصصة لها	ثابت.	يمكن أن يتغير أثناء تنفيذ البرنامج.
طريقة التخزين في الذاكرة	يتم تخزين العناصر في مواقع متجاورة في الذاكرة الرئيسة.	يتم تخزين العناصر في مواقع عشوائية في الذاكرة الرئيسة.
سرعة الوصول إلى البيانات	أسرع في الوصول.	أبطأ في الوصول.

تخصيص الذاكرة

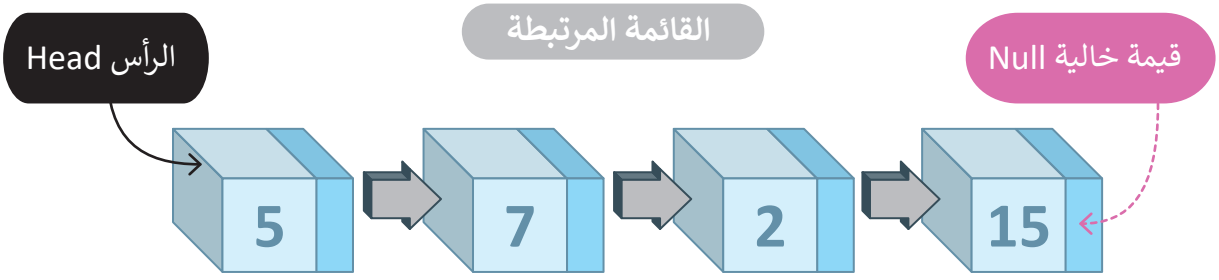
تنتمي القائمة المرتبطة إلى هياكل البيانات الديناميكية، وهذا الأمر يعني أن عُقَدْ القائمة المرتبطة **Nodes** لا يتم تخزينها في مواقع ذاكرة متجاورة مثل بيانات المصفوفات، ولهذا السبب نحتاج إلى استخدام مؤشرات أخرى خاصة للعُقَدْ.



القائمة المرتبطة Linked List

هي أحد هياكل البيانات الخطية، وهي من أكثر هياكل البيانات شيوعًا في عالم البرمجة. تشبه القائمة المرتبطة سلسلة من العُقَد المتتالية (**chain of nodes**)، وكل عقدة تحوي على حقلين : حقل البيانات المخزنة فيها وحقل يحوي عنوان العقدة التالية ويستثنى من ذلك العقدة الأخيرة التي لا يحمل حقل العنوان فيها أي بيانات.

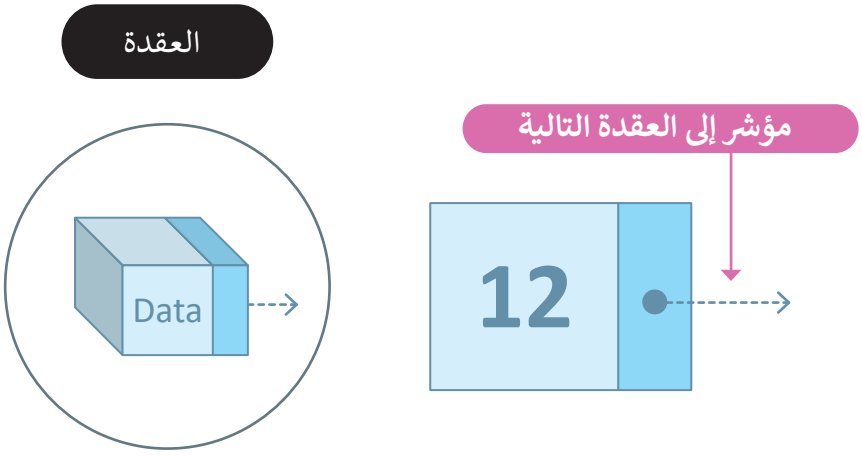
تتمثل إحدى مزايا القائمة المرتبطة في إمكانية زيادة حجمها أو تقليصه تلقائيًا عن طريق إضافة أو حذف العُقَد (**nodes**).



العُقدة Node

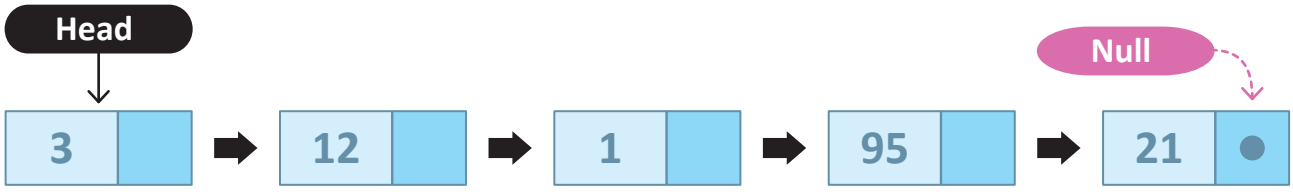
تتكون كل عقدة في القائمة المرتبطة من قسمين.

- يحتوي القسم الأول على البيانات.
- يحتوي القسم الثاني على مؤشر يشير إلى عنوان العقدة التالية.



لنستعرض مثالًا على قائمة مرتبطة خاصة بالأعداد.

لدينا هنا قائمة بخمس عقد.



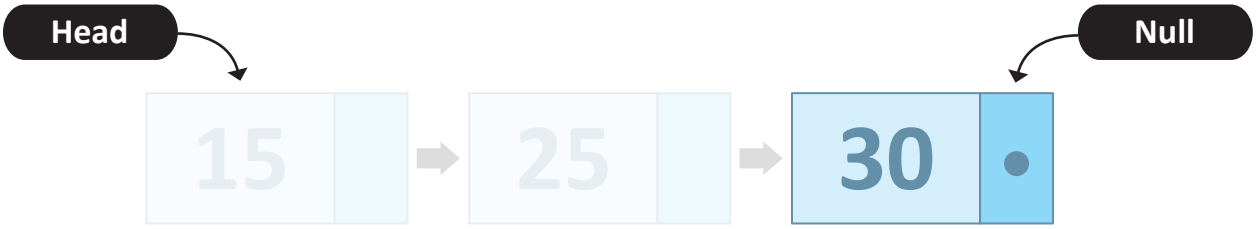
Null تعني عدم وجود قيمة أو عدم تعريفها أو أن العقدة فارغة. يستخدم الكثيرون العدد 0 للإشارة إلى القيمة الخالية، إلا أن 0 يمثل نهاية رقم معين ويمكن أن تكون قيمة حقيقية.

ليس للعقد في القائمة أي أسماء، فما نعرفه عن العقدة هو فقط العنوان (موقع الذاكرة) الذي يتم تخزينه. للوصول إلى أي عقدة في القائمة، نحتاج فقط إلى معرفة عنوان العقدة الأولى، ثم نتبع سلسلة العقد للوصول إلى العقدة التي نريدها.

لقراءة محتوى عقدة ما؛ يجب المرور عبر جميع العقد السابقة لها.

الاختلافات بين القائمة والقائمة المرتبطة		
	القائمة	**القائمة المرتبطة**
تخزين البيانات	تخزن في مواقع متجاورة	تخزن في مواقع غير متجاورة
الهيكل	يمكن التعامل مع عناصرها من خلال الرقم التسلسلي **Index**	يمكن التعامل مع عناصرها من خلال المؤشر.
	يتم تخزين كل عنصر بعد الآخر	يتم تخزين العناصر على شكل عقد تتضمن البيانات وعنوان العنصر التالي (مؤشر)
استخدام الذاكرة	يتم تخزين البيانات فقط في الذاكرة	يتم تخزين البيانات والمؤشرات في الذاكرة
الوصول إلى البيانات	وصول عشوائي لأي عنصر من عناصر القائمة	وصول تسلسلي للعناصر
سرعة الوصول	وصول أسرع	وصول أبطأ
سرعة الإضافة والإزالة	إضافة وإزالة أبطأ	إضافة وإزالة أسرع

على سبيل المثال، إذا أردنا الوصول إلى العقدة الثالثة من القائمة لمعالجة بياناتها، يجب أن نبدأ من العقدة الأولى في القائمة، ثم من العقدة الأولى للثانية ومن الثانية للوصول إلى الثالثة.



- يتم تخزين عنوان العقدة الأولى في متغير خاص (مستقل) نسميه عادة الرأس **Head**.
- يشير مؤشر العقدة الأخيرة من القائمة إلى قيمة خالية **Null** ونمثلها بالرمز ●.
- عندما تكون القائمة فارغة، يشير مؤشر الرأس إلى قيمة خالية **Null**.

القائمة المرتبطة في Python

لا تقدم لغة برمجة **Python** نوع بيانات محدد مسبقًا للقوائم المرتبطة؛ لذلك علينا إنشاء نوع بيانات خاص بنا أو استخدام مكتبات Python إضافية توفر تمثيلًا لهذا النوع من البيانات.

في المثال التالي سننشئ قائمة مرتبطة بثلاث عقد تحتوي كل منها على يوم من أيام الأسبوع.

يمكن استخدام مفهوم الفئات **Class** لانشاء القوائم المرتبطة

الفئة class

الفئة هي بنية بيانات يحددها المستخدم والتي تحتفظ بمجموعتها الخاصة من البيانات مثل الخصائص والوظائف. يتم استخدام الفئات كقالب لإنشاء الكائنات. لإنشاء فئة نستخدم كلمة **class**. سنناقش كل التفاصيل حول الفئة واستخدامها في وحدة مقبلة.

مثال: 1

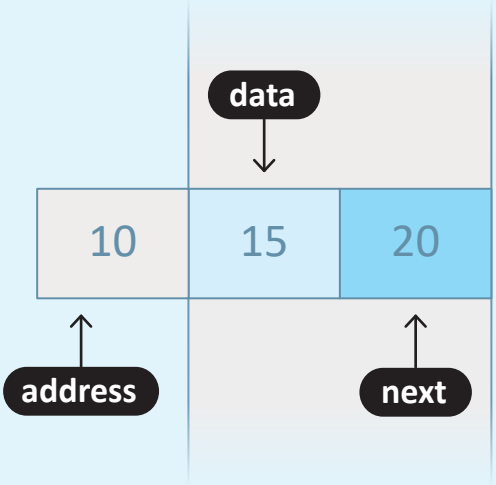
لنرى مثال رسومي. كما ذكرنا سابقًا، تتكون كل عقدة من البيانات ومؤشر يشير إلى العقدة التالية. كما يتم تخزين كل عقدة في عنوان في الذاكرة.

لنرى عقدة معينة كمثال.

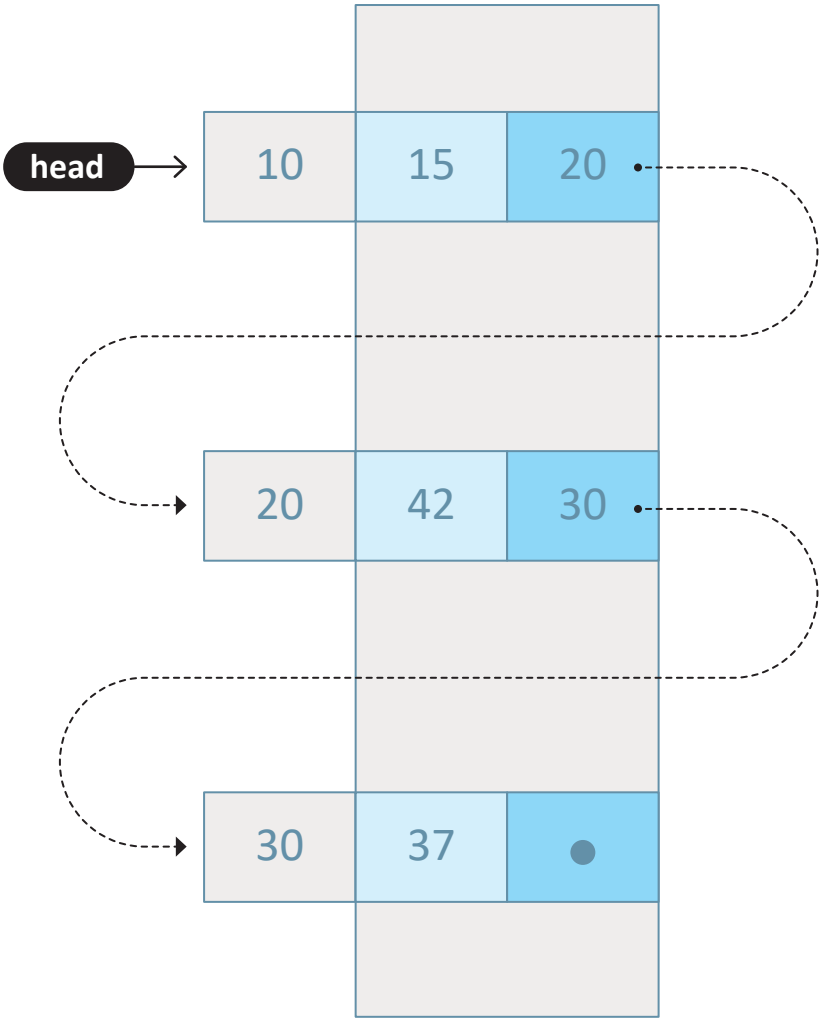
بيانات العقدة هي الرقم 15

العنوان المخزن في الذاكرة هو 10

العقدة التالية ستكون في العنوان 20



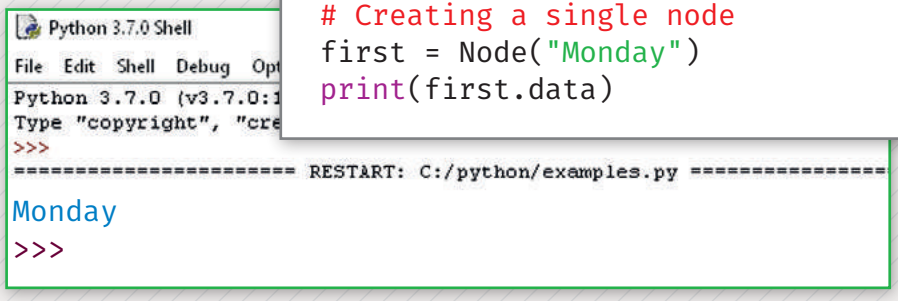
لنربط العقدة السابقة بالعقدة التي تليها وتحمل القيمة 42 وتشير بدورها إلى العقدة الثالثة والأخيرة في العنوان 30 وتحمل القيمة 37.



مثال: 2

سنقوم أولاً بإنشاء عقدة باستخدام الفئة **Class**.

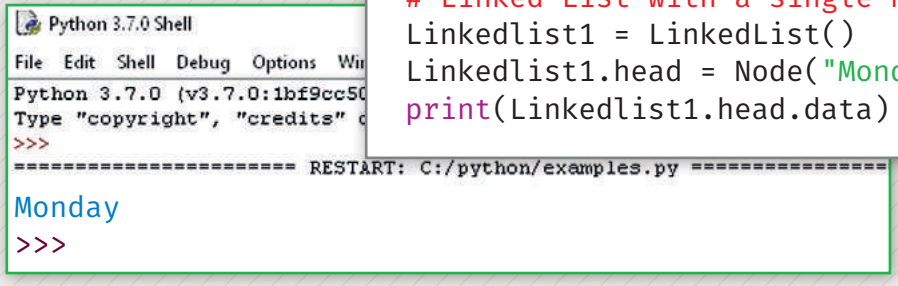
```
# A single node
class Node:
    def __init__(self, data, next=None):
        self.data = data # data of the node
        self.next = next # pointer to the nextNode

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



مثال: 3

الخطوة التالية هي إنشاء قائمة مرتبطة بعقدة واحدة، في هذه المرة سنستخدم مؤشر الرأس **head** للإشارة إلى العقدة الأولى.

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

# A Linked List with a single head node
class LinkedList:
  def __init__(self):
    self.head = None

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

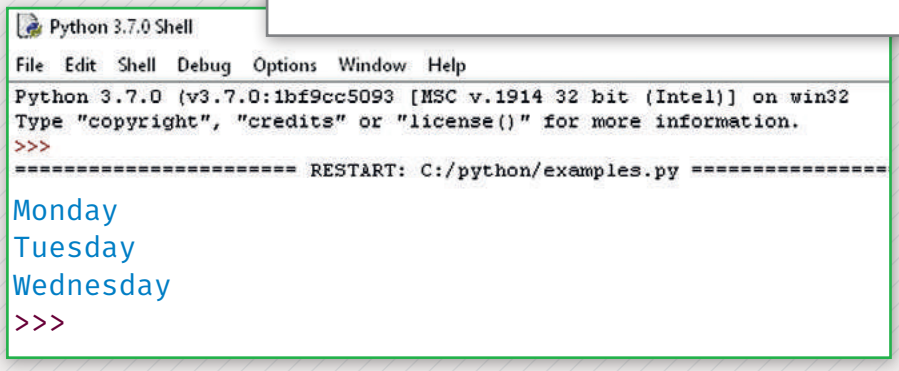


مثال: 4

الآن سنقوم بإضافة المزيد من العقد إلى قائمتنا المرتبطة.

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

# A Linked List with a single head node
class LinkedList:
  def __init__(self):
    self.head = None

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

تستخدم جملة while للإنتقال من العقدة الأولى إلى العقدة الأخيرة.

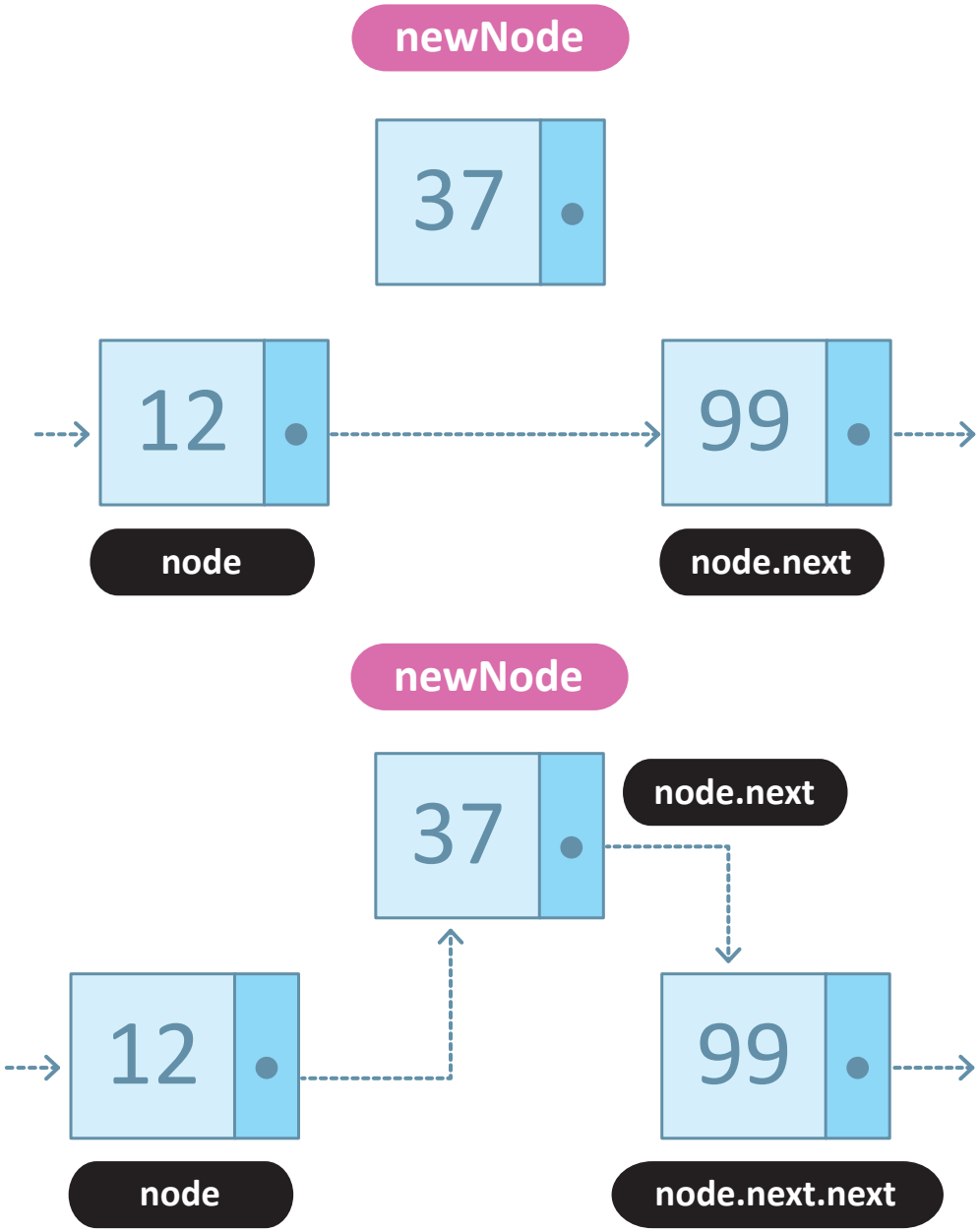


إضافة العقدة إلى قائمة مرتبطة

الإجراءات المطلوبة لإضافة العقدة الجديدة هي:

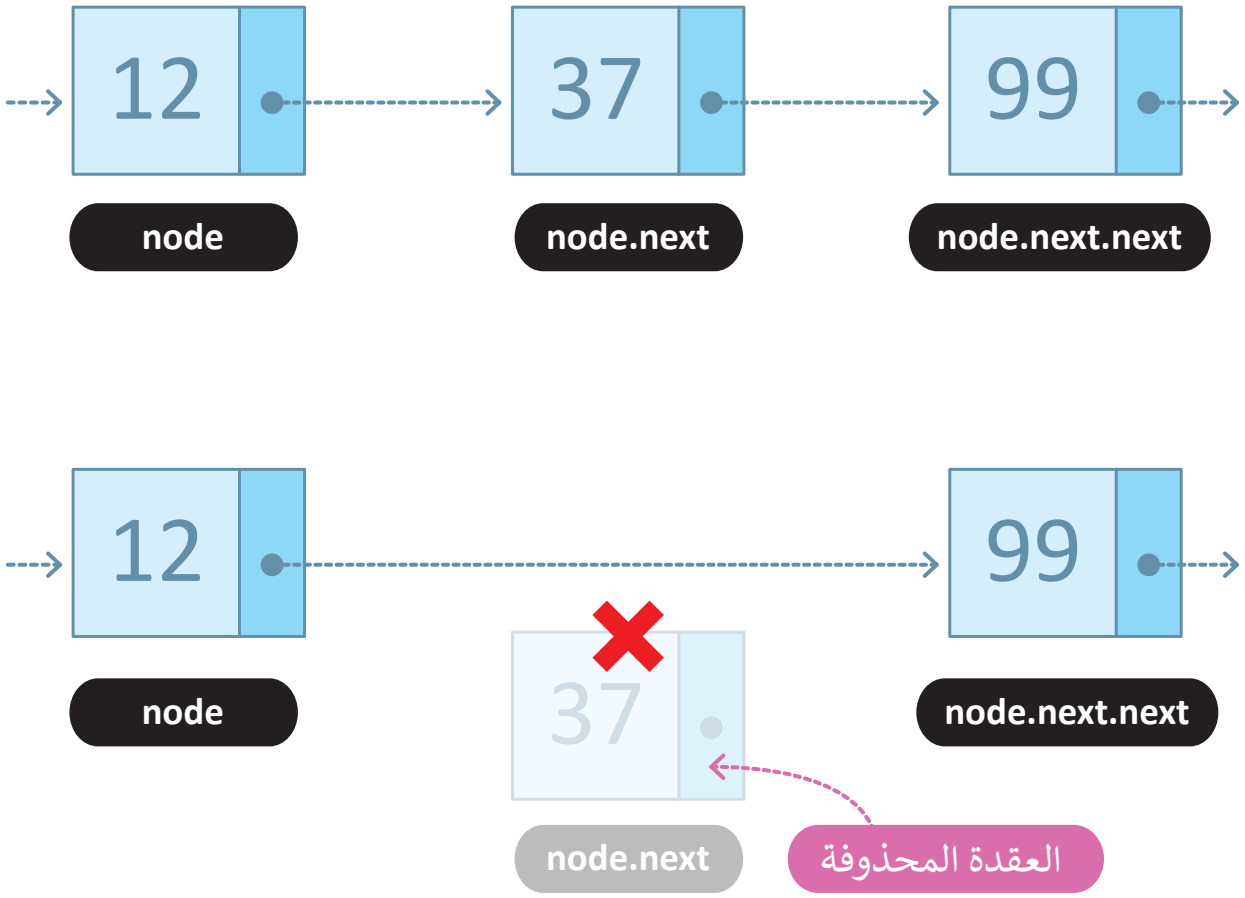
- مؤشر أول عقدة يجب أن يشير إلى عنوان العقدة الجديدة لتصبح هذه العقدة هي العقدة الثانية.
- يجب أن يشير مؤشر العقدة الجديدة (الثانية) إلى عنوان العقدة الثالثة.

بهذه الطريقة لا نضطر إلى إزاحة العناصر في حال إضافة عنصر جديد في الوسط وتقتصر العملية على تغيير قيم العناوين في العقد، ولذلك تعتبر الإضافة أسرع في حالة القوائم المرتبطة منها في حالة القوائم المتسلسلة.



حذف العقدة من القائمة المرتبطة

لحذف عقدة يتوجب علينا تغيير مؤشر العقدة السابقة لها لتشير إلى العقدة التالية للعقدة المحذوفة. العقدة المحذوفة (الثانية) هي "بيانات عديمة الفائدة" ويتم تخصيص مساحة الذاكرة التي تشغلها لاستخدامات أخرى.



إذا أردنا حذف العقدة الأولى من قائمة مرتبطة، فعلينا نقل الرأس إلى العقدة الثانية من القائمة.

تطبيقات القوائم المرتبطة في علم الحاسوب

- تخصيص الذاكرة الديناميكية من خلال نظام التشغيل.
- أداء العمليات الحسابية الخاصة بالأعداد الصحيحة الطويلة.
- الاحتفاظ بجميع التطبيقات التي تعمل على جهاز الحاسوب في قائمة مرتبطة حيث يمنح نظام التشغيل مقدارًا زمنيًا ثابتًا لتشغيل كل مهمة حتى يتم الانتهاء من جميع التطبيقات.

تطبيقات أخرى للقوائم المرتبطة

- من الأمثلة على القوائم المرتبطة الطريقة التي يعمل بها برنامج عارض الصور (**image viewer**)، حيث يتم ربط الصور السابقة والتالية والوصول إليها عن طريق أزرار "التالي" و"السابق".
- الحركة بين الصفحة السابقة والتالية في متصفح الويب، وهي طريقة أخرى لتخزين الصفحات التي قمنا بزيارتها في المتصفح حيث يمكننا الوصول إلى عنوان **URL** السابق والتالي عن طريق الضغط على زر الرجوع والتالي.
- ربط ملفات الصوت السابقة والتالية معًا في مشغل الصوتيات.

1 أكمل الفراغ:

1. تقع عقد القائمة المرتبطة في مواقع ذاكرة _____.
2. القائمة المرتبطة هي نوع من أنواع هياكل البيانات الـ _____.
3. تتكون عقدة القائمة المرتبطة من جزأين _____ و_____ .
4. يتم تخزين عنوان العقدة الأولى في متغير خاص نطلق عليه عادةً _____ .
5. يشير مؤشر العقدة الأخيرة من القائمة إلى _____.
6. عندما تكون القائمة فارغة، يشير _____ إلى قيمة خالية.
7. للوصول إلى أي عقدة في القائمة، نحتاج إلى معرفة عنوان العقدة _____ .

2 قارن بين هياكل البيانات الثابتة وهياكل البيانات الديناميكية، واذكر مثالًا لكل منهما.

_____

_____

_____

_____

3

اكتب مثالين على استخدامات القوائم المرتبطة.

_____

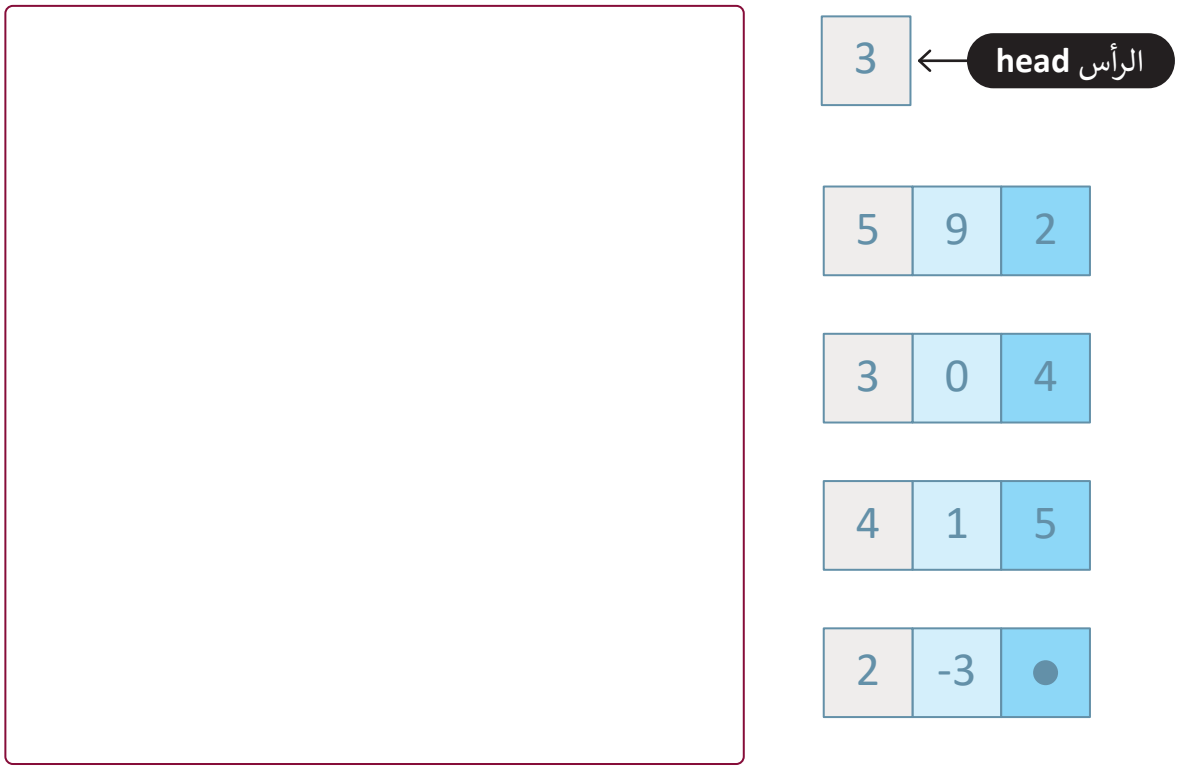
_____

_____

_____

4

بالنظر إلى العقد التالية، ارسم القائمة المرتبطة ثم اكتب القيم التي تحتويها هذه القائمة بشكل مرتب حسب التسلسل الصحيح.



5

لقد أنشأنا قائمة بالأعداد التالية: 5، 20، 45، 8، 1 بحيث يكون قيمة العقدة الأولى 5.

> ارسم عقد القائمة المرتبطة.

> صِف عملية إضافة العدد 7 بعد العدد 45.

_____

_____

_____

> ارسم القائمة الجديدة.

> صِف العملية المطلوبة لإزالة العقدة الثانية من القائمة.

_____

_____

_____

> ارسم القائمة المرتبطة النهائية.

6

بالنظر إلى الجدول أدناه، ارسم القائمة المرتبطة ثم اكتب القيم التي تحتويها هذه القائمة بشكل مرتب حسب التسلسل الصحيح.

العقدة	Head	2	3	4	5	6	7
القسم الأيمن من العقدة (البيانات)	(	b	a	+	2	/	)
القسم الأيسر من العقدة (المؤشر)	3	7	4	2	Null	5	6

> شغل البيئة البرمجية IDLE Python وقم بإنشاء القائمة المرتبطة السابقة، ثم عرض متوسط العددين الصحيحين a وb يتم إدخالها من طرف المستخدم.

7

قم بإنشاء برنامج في Python لتنفيذ القائمة المرتبطة التالية.

الدرس الثالث

هياكل بيانات الأشجار

تعرفنا في الدروس السابقة على بعض هياكل البيانات الخطية والتي تتميز بأن كل عنصر يلي العنصر الآخر، وذلك بشكل خطي، ولكن هل فكرت في الحالات التي لا تسير فيها الأمور بشكل خطي، حيث يمكن أن يتبع العنصر بأكثر من عنصر واحد.

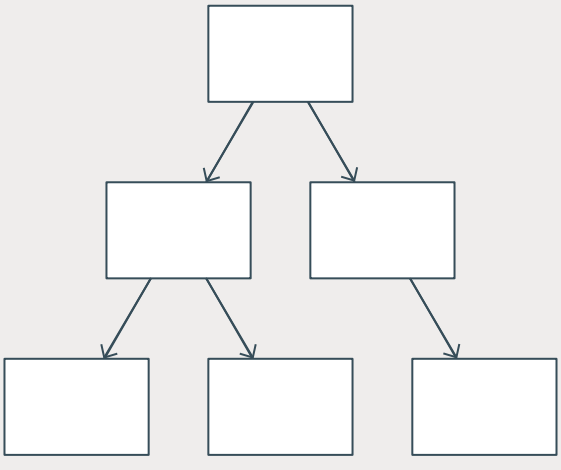
هياكل البيانات غير الخطية

هي هياكل تمتاز بإمكانية ربط عناصرها بأكثر من عنصر آخر في نفس الوقت.

يمكن ربط عنصر هياكل البيانات غير الخطية بأكثر من عنصر واحد. ومن الأمثلة على هياكل البيانات غير الخطية الأشجار **(Trees)** والمخططات **(Graphs)**.

هيكل بيانات خطي

هيكل بيانات غير خطي

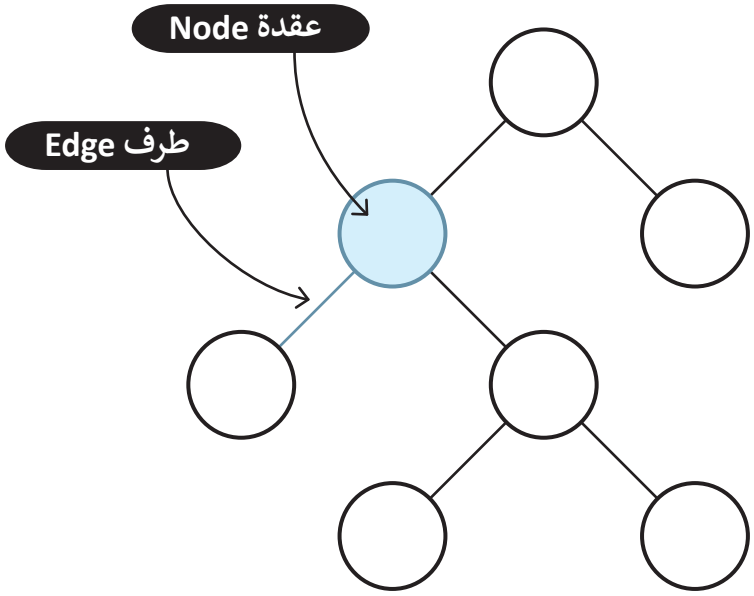


الفروقات بين هياكل البيانات الخطية وغير الخطية	
الخطية	غير الخطية
يتم ترتيب عناصر البيانات بشكل خطي حيث يتم إلحاق كل عنصر بعنصر سابق وعنصر تالي.	يتم ترتيب عناصر البيانات بشكل غير خطي بحيث يمكن ربط عنصر بأكثر من عنصر.
يوجد مستوى واحد لتمثيل البيانات.	توجد مستويات متعددة لتمثيل البيانات.
يتم التنفيذ بشكل أسهل.	يكون التنفيذ أكثر تعقيدًا.

الأشجار Trees

الأشجار هي هياكل بيانات غير خطية، وتتكون من مجموعة من العقد التي يتم ترتيبها بشكل هرمي بحيث يمكن لكل عقدة أن ترتبط بعقدة أو أكثر.

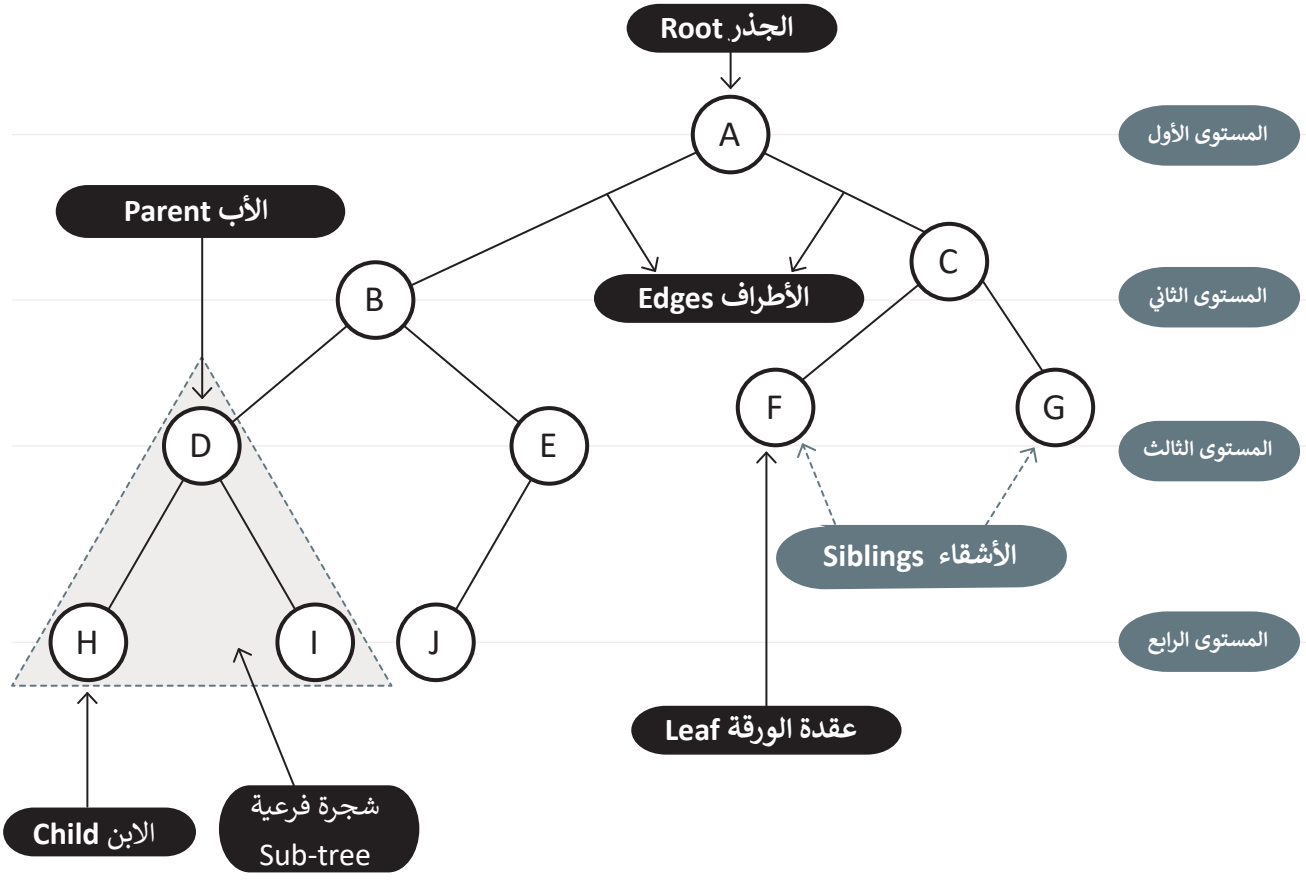
تربط الأطراف **Edges** العقد **Nodes** مع بعضها بصورة تشبه علاقة الأب والأبناء **(Parent-child Relationship)**.



تُستخدم الأشجار في العديد من مجالات علوم الحاسوب، بما في ذلك أنظمة التشغيل والرسومات وأنظمة قواعد البيانات والألعاب والذكاء الاصطناعي وشبكات الحاسوب.

المصطلحات المستخدمة في هيكل بيانات الشجرة

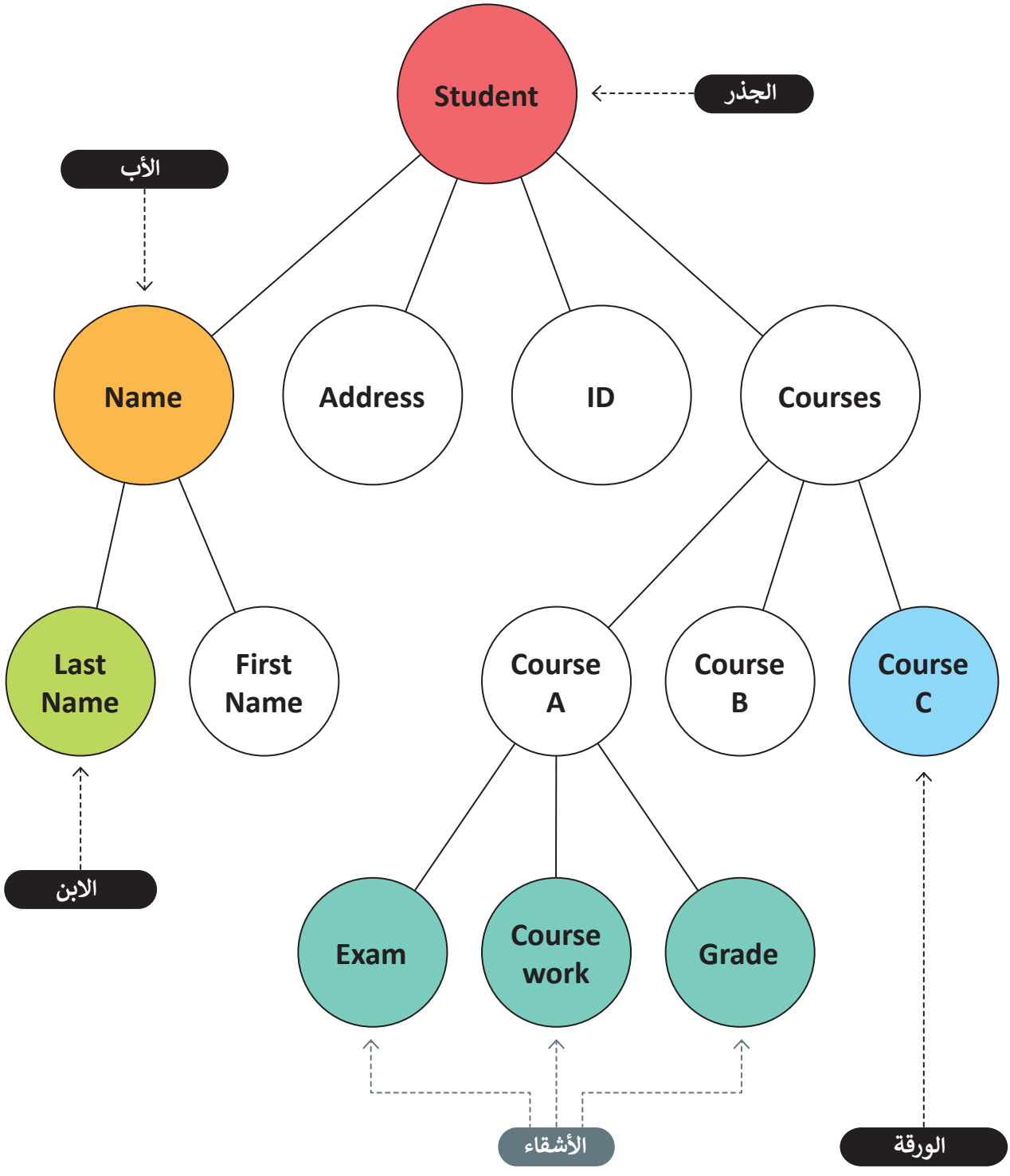
- الجذر (**Root**): هي العقدة الأولى والوحيدة في الشجرة التي ليس لها أب (**Parent**) وتكون في المستوى الأول من الشجرة.
- الابن (**Child**): هي العقدة المتصلة مباشرة بعقدة في مستوى أعلى.
- الأب (**Parent**): هي العقدة التي لديها ابن واحد أو أكثر في مستوى أدني.
- الورقة (**Leaf**): هي العقدة التي لا تحتوي على أي عقد فرعية (ابن).
- الأشقاء (**Siblings**): هي جميع العقد التابعة لنفس العقدة الأصلية (الأب).
- الأطراف (**Edges**): هي الروابط التي تربط بين عقد الشجرة.
- الشجرة الفرعية (**Sub-Tree**): هي الأشجار الصغيرة التي يمكن العثور عليها داخل الشجرة الكبيرة.



نصيحة ذكية

يمكن للعقدة أن تكون ابن وأب في نفس الوقت: ابن بالنسبة للعقدة السابقة وأب بالنسبة للعقدة اللاحقة.

لنستعرض مثالًا على هياكل بيانات الشجرة.

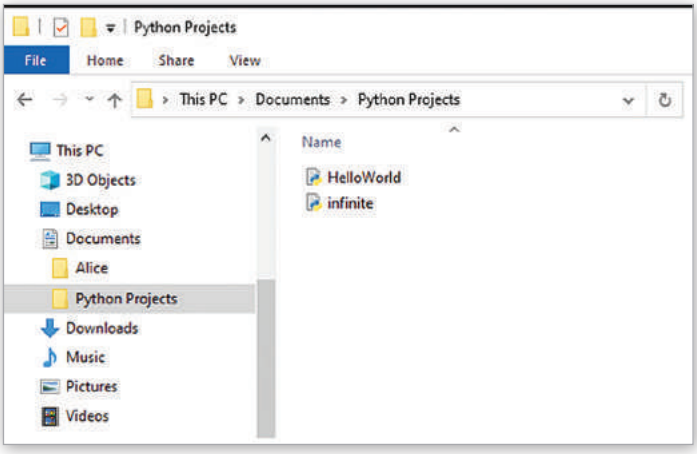


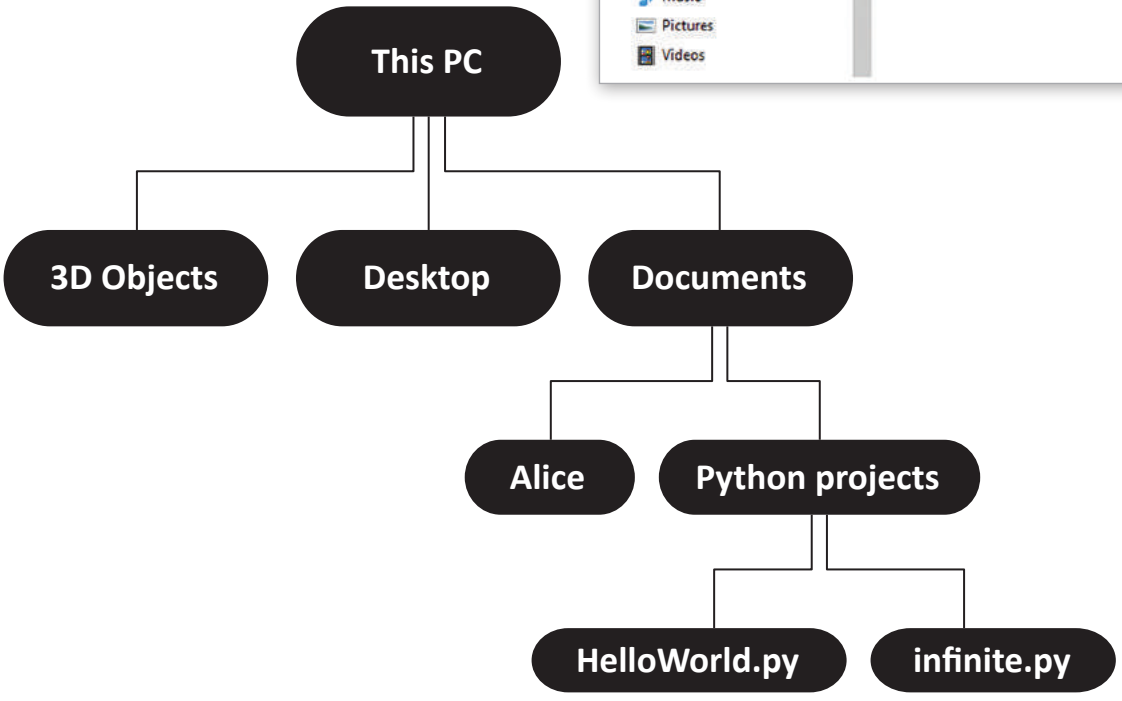
قد نجد شجرة بسيطة تتكون من عقدة واحدة، وهذه العقدة قد تكون جذرًا لهذه الشجرة البسيطة نظرًا لأنه ليس لها أب ولا ورقة ولا أبناء.

ميزات هيكل بيانات الشجرة

- تستخدم لتمثيل التسلسل الهرمي.
- تتسم بالمرونة، فمن السهل جدًا إضافة أو إزالة عنصر.
- تتسم بسهول البحث عن عنصر داخل الشجرة.
- تبين العلاقات الهياكل بين البيانات.

يعتبر تنظيم الملفات في نظام التشغيل من الأمثلة العملية على الشجرة، على سبيل المثال في الصورة أدناه؛ يوجد داخل مجلد "**Documents**" مجلد آخر يسمى **Projects Python** والذي يحتوي على ثلاثة ملفات أخرى.



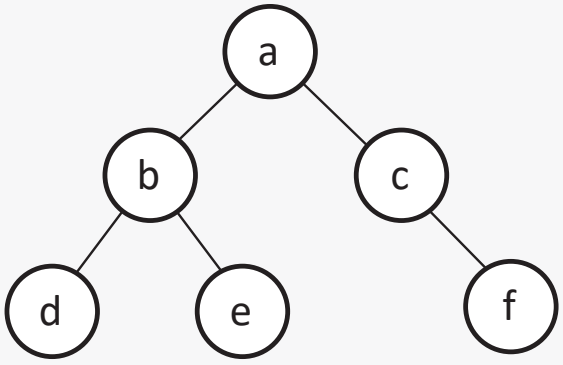


هيكل بيانات الشجرة في Python

لا تقدم لغة برمجة **Python** نوع بيانات محدد مسبقًا لهيكل بيانات الشجرة، ولكن يمكن إنشاؤه بسهولة برمجيًا باستخدام القوائم (**Lists**) والقواميس (**Dictionaries**).

مثال: 1

فيما يلي تمثيل لهياكل بيانات الشجرة باستخدام القاموس.



عقدة

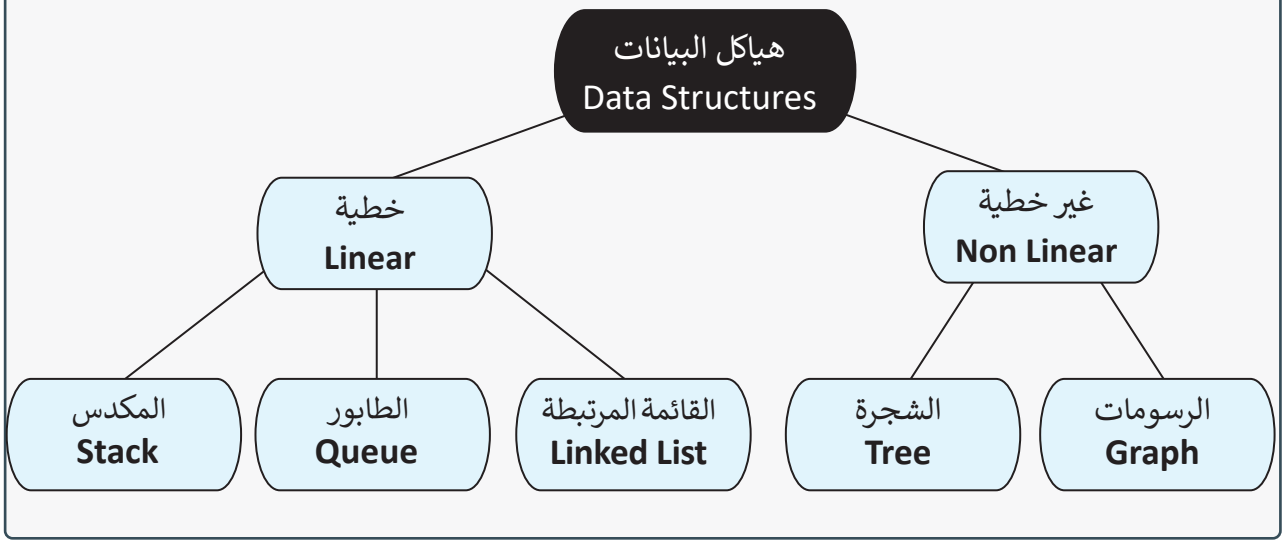
```
myTree = {
    "a": ["b", "c"],
    "b": ["d", "e"],
    "c": [None, "f"],
    "d": [None, None],
    "e": [None, None],
    "f": [None, None],
}
print(myTree)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.
Type "copyright", "credits" or "license
>>>
========== RESTART: C:/python/examples.py ==========
{'a': ['b', 'c'], 'b': ['d', 'e'], 'c': [None, 'f'],
'd': [None, None], 'e': [None, None], 'f': [None, None]}
>>>
```

إن مفاتيح القاموس هنا عبارة عن عقد الشجرة، ومقابل كل مفتاح توجد قائمة تحتوي على العقد المتصلة بها.

مثال: 2

في المثال التالي سننشئ شجرة كالتي في الصورة.



الأب

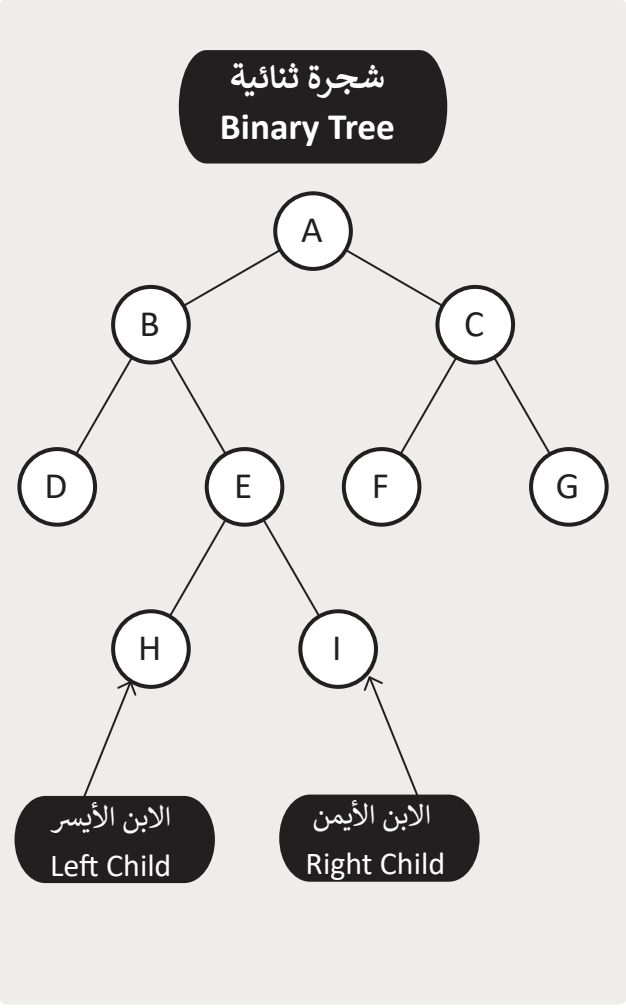
الابن

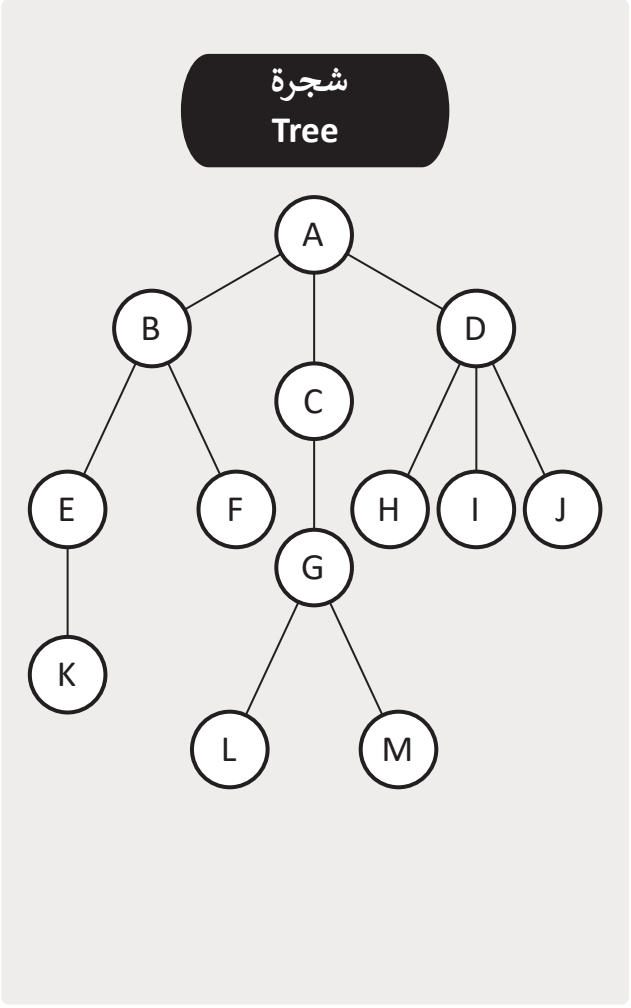
```
myTree = {"Data structures":["Linear","Non-linear"],
          "Linear":["Stack","Queue","Linked list"],
          "Non-linear":["Tree", "Graph"]}

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

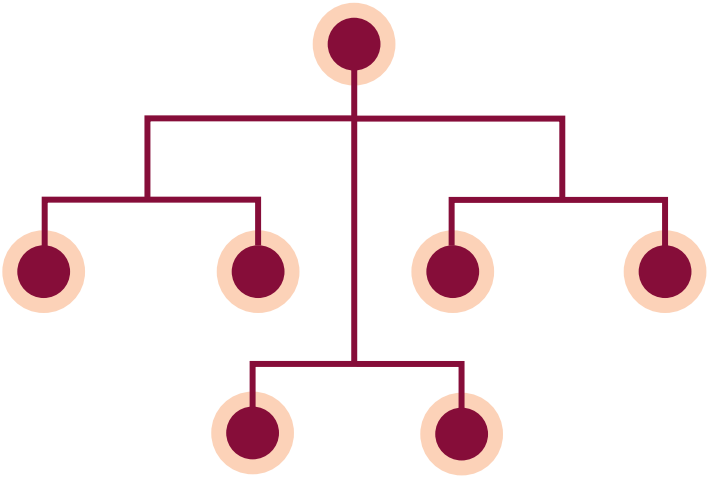
```
Python 3.7.0 Shel
File  Edit  Shell  D
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
Data structures has 2 nodes
  Linear
  Non-linear
Linear has 3 nodes
  Stack
  Queue
  Linked list
Non-linear has 2 nodes
  Tree
  Graph
>>>
```

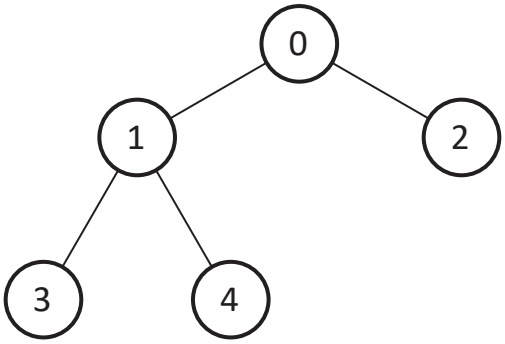
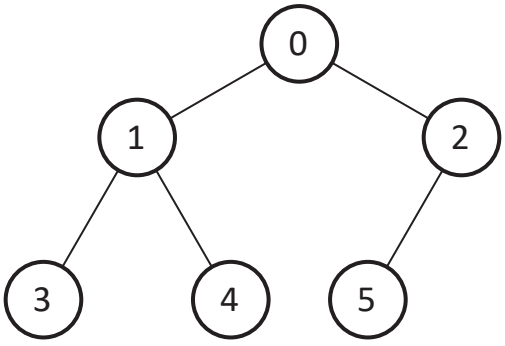
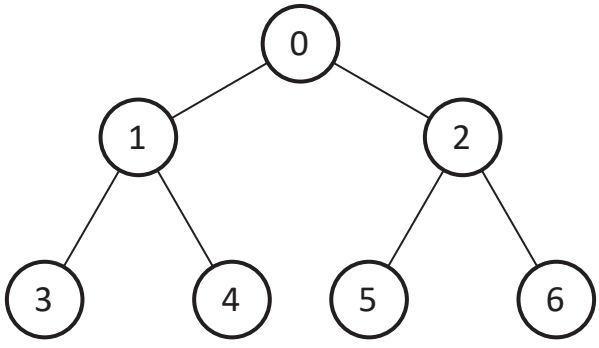
الشجرة الثنائية Binary Tree

هى حالة خاصة من هياكل بيانات الأشجار، وتتميز بوجود "ابنين" على الأكثر لكل عقدة، ويُطلق عليهما اسم الابن الأيمن (**Right Child**) والابن الأيسر (**Left Child**).







أنواع هيكل بيانات الشجرة الثنائية		
النوع	الوصف	رسم الهيكل
الشجرة الثنائية الممتلئة Full Binary Tree	يوجد لكل عقدة إما 0 أو 2 "أبناء"	
الشجرة الثنائية الكاملة Complete Binary Tree	تمتلئ جميع المستويات بالكامل باستثناء المستوى الأخير، والذي يحتوي على جميع الأوراق المتبقية وذلك على الجهة اليسرى ما أمكن ذلك.	
الشجرة الثنائية التامة Perfect Binary Tree	جميع العقد الداخلية لها ابنان وجميع الأوراق في نفس المستوى.	

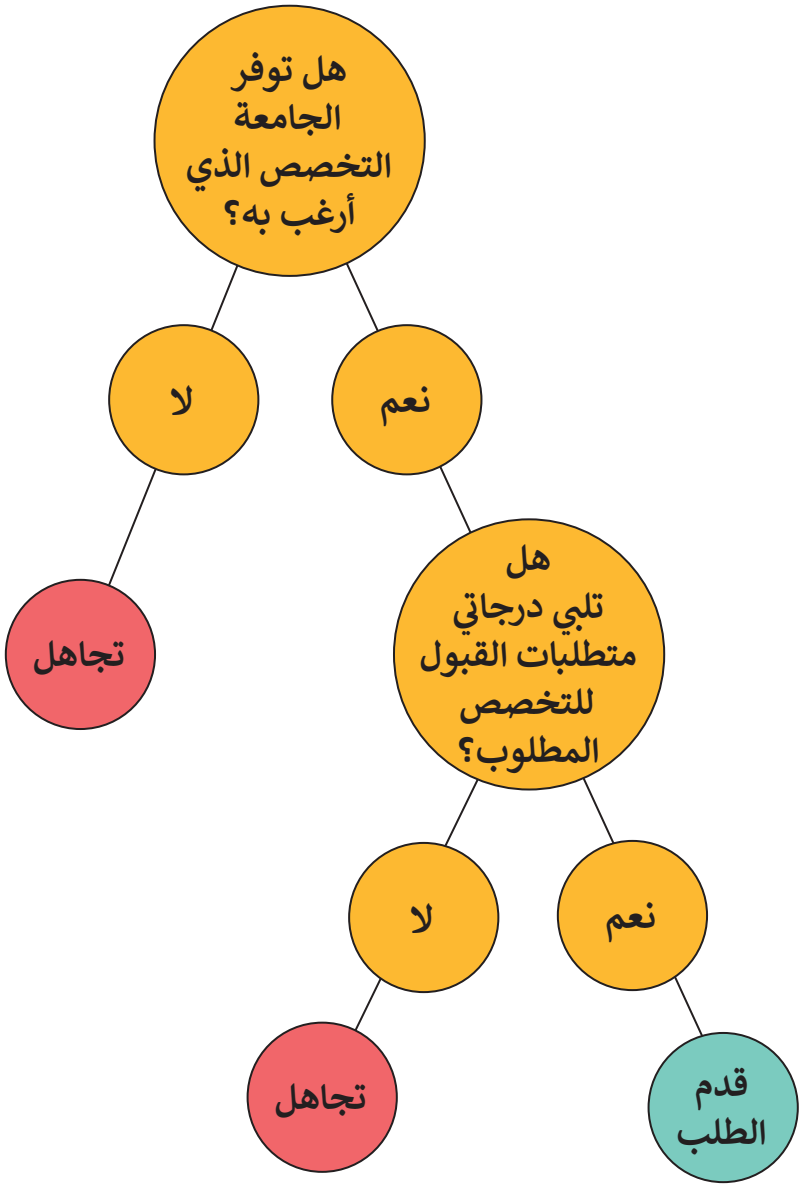
أمثلة على تطبيقات هياكل بيانات الأشجار

- تخزين البيانات الهرمية، مثل هياكل المجلدات.
- تعريف البيانات في لغة **HTML**.
- تنفيذ الفهرسة في قواعد البيانات.

شجرة القرار Decision Tree

تعتبر جملة القرار (**if a: else b**) من أكثر الجمل البرمجية تكرارًا واستخدامًا في برمجة **Python**. من خلال دمج هذه الجمل البرمجية، يمكننا بناء شجرة القرار (**Decision Tree**)، والتي تتشابه نوعا ما مع المخطط الانسيابي (**Flow Chart**)، مع ملاحظة أن المخططات الانسيابية تسمح بالتكرارات أيضًا.

يطلق مصطلح "تعلم شجرة القرار" (**Decision Tree Learning**) على تطبيق أشجار القرار في علم الذكاء الاصطناعي وتعلم الآلة. حيث تسمى العقد النهائية بالورقة أو بعقدة القرار وتحتوي عادة على الحلول الممكنة للمشكلة. ترتبط كل عقدة -باستثناء الأوراق- بشرط منطقي يتفرع منه احتمالات الإجابة بنعم أو لا وعمومًا يعتبر من السهل التعرف على استخدامات أشجار القرار وتصورها والتحقق منها كما في المثال التالي:



1

أكمل الجمل التالية بالكلمات المناسبة.

1. يطلق على العقدة الأولى للشجرة تسمية _____ .

2. _____ في الشجرة هي عقد غير متصلة بعقد فرعية أخرى.

3. في الشجرة Full Binary Tree يمكن أن يكون لكل أب _____ أو _____ من الأبناء.

4. في الشجرة الثنائية Perfect Binary Tree يجب أن تحتوي جميع العقد الداخلية على _____ أبناء.

5. في الشجرة الثنائية، يطلق على الأبناء _____ و _____.

6. تسمى جميع العقد الفرعية التي لها نفس العقدة الأب باسم _____.

7. الأشجار هي هياكل بيانات _____.

8. عند ترتيب عناصر البيانات بشكل متسلسل تسمى هياكل البيانات بـ _____.

9. تسمى الأشجار الصغيرة التي يمكن أن نجدها داخل شجرة _____.

10. يمكن تمثيل الأشجار في Python باستخدام _____ و _____.

2

اكتب الفروقات ما بين هياكل البيانات الخطية وغير الخطية.

_____

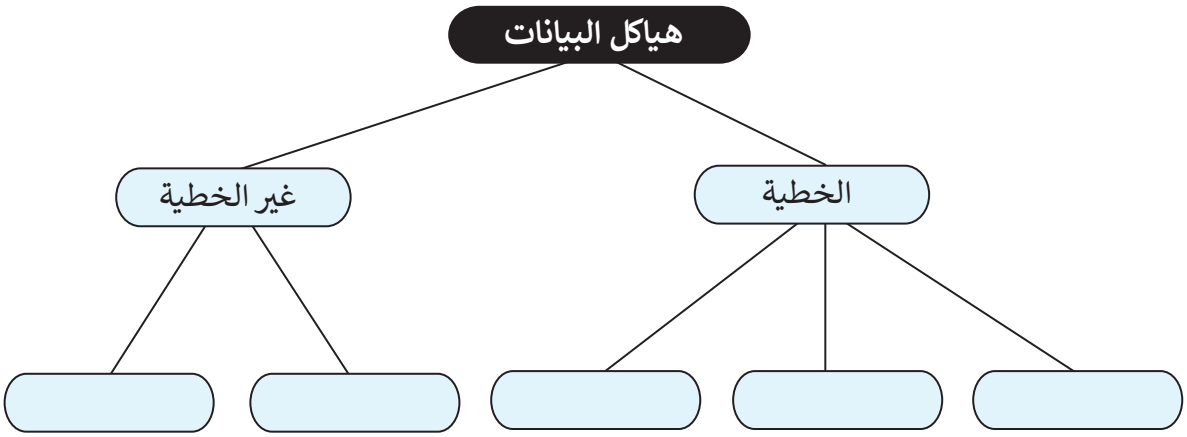
_____

_____

_____

3

أكمل تعبئة الشجرة بهياكل البيانات الخطية وغير الخطية.

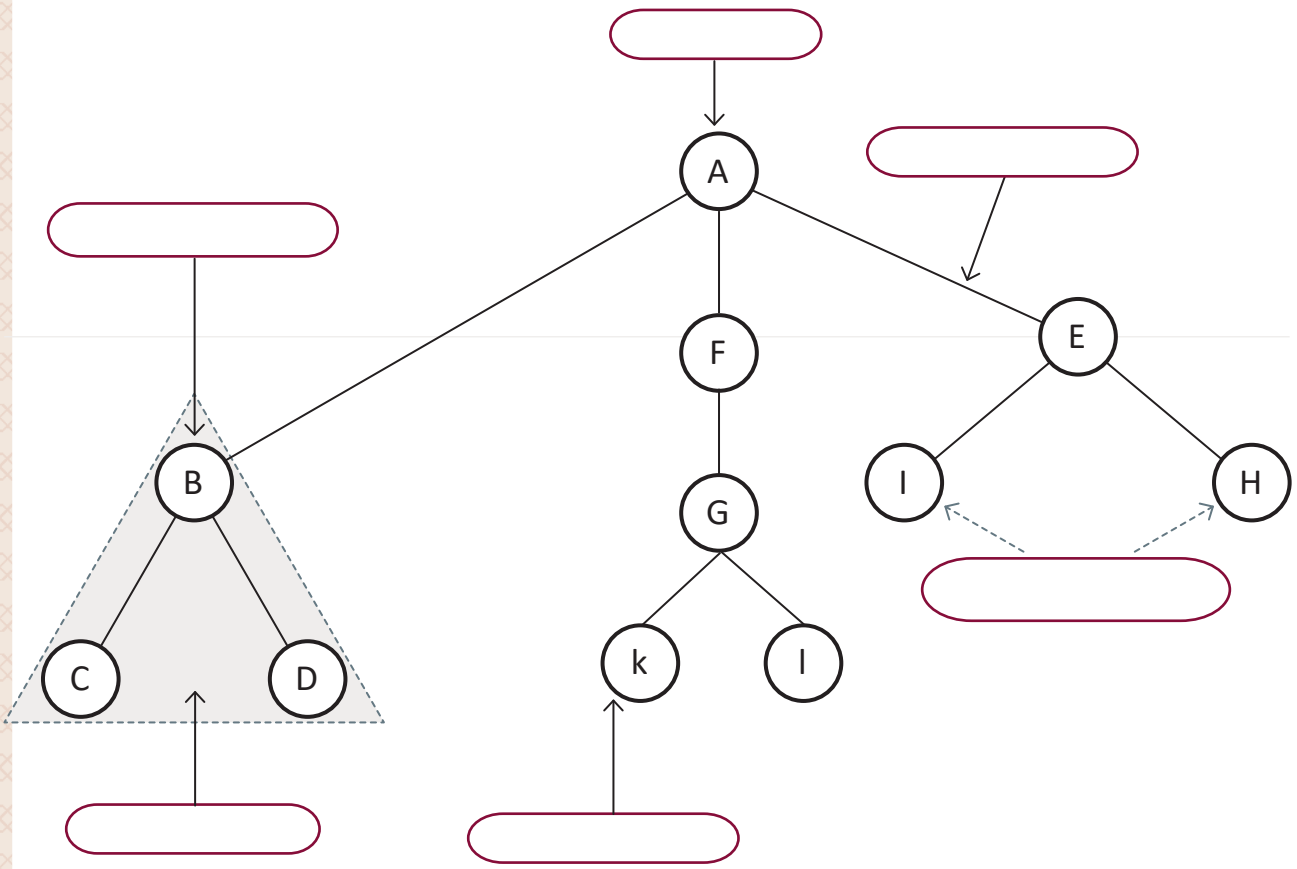


4

اكتب ثلاثة تطبيقات على استخدامات هيكل بيانات الشجرة.

5

إملأ الفراغات بأسماء الأجزاء الصحيحة الموجودة في الشجرة التالية.



6

في الصورة التالية يمكنك رؤية صفحة محتويات الكتاب، قم بإكمال تمثيل الشجرة الخاص به.

Book

C1

C1.1

C1.2

C2

C2.1

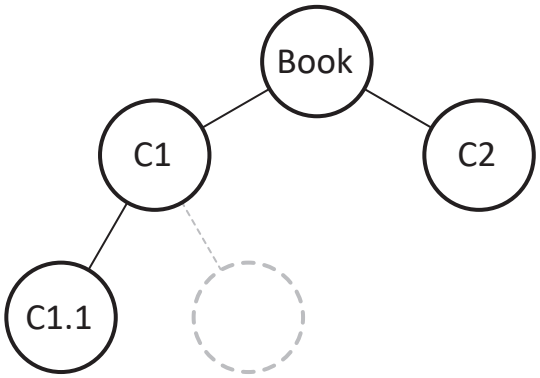
C2.1.1

C2.1.2

C2.2

C2.3

C3



> هل تعتبر شجرة ثنائية؟ علل إجابتك.

_____

_____

7 ارسم الشجرة التي ستنتج من المعلومات التالية:

> تحتوي العقدة A على العقد الأبناء B وC وD.

> العقد E وZ لهما نفس الأب وهو العقدة D.

> العقدة K لها أشقاء وهم العقد L وM ونفس أب العقدة C.

> العقدة P لها ابن كالعقدة O ونفس أب العقدة B.

8

ارسم الشجرة التي ستنتج من المعلومات التالية:

> العقدة A لها أبناءB وC.

> العقدة D وE لهما نفس الأب وهي العقدة B.

> العقدة F لها شقيق وهو العقدة G ولهما نفس الأب وهي العقدة C.

> العقدة H لديها عقدتين أبناء وهما I وK ولها أب وهي العقدة F.

> ما نوع الشجرة السابق؟ _____

> باستخدام القاموس في بايثون قم بكتابة البرنامج المناسب لتمثيل هذه الشجرة وطباعة الآباء والأبناء.

الدرس الرابع

هياكل بيانات المخططات

لقد رأينا أن أهم ميزة لهياكل البيانات غير الخطية هي أن بياناتها لا تتبع تسلسلًا كما هو الحال في المصفوفات أو القوائم المرتبطة، وأن عناصرها يمكن أن ترتبط بأكثر من عنصر.

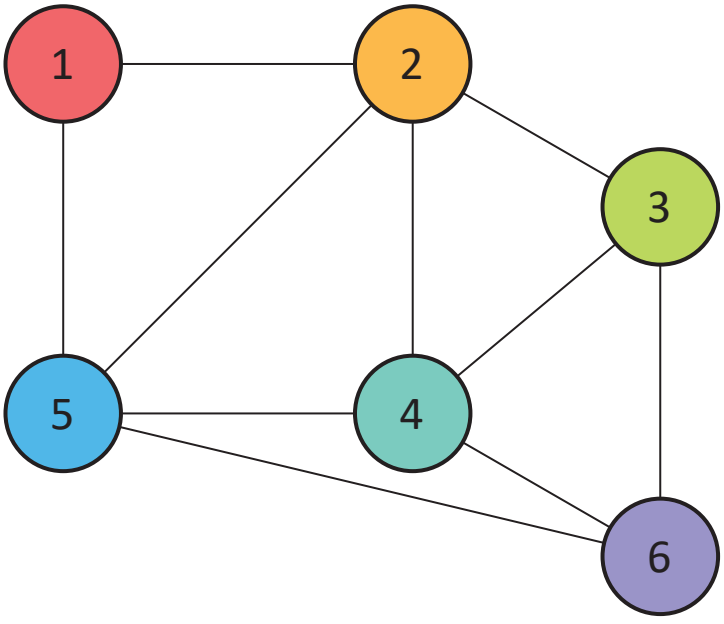
تبدأ الأشجار كما رأينا، بعقدة جذرية يمكن توصيلها بعقد أخرى، وتتبع الأشجار قواعد محددة بينما في أنواع معينة من الأشجار يتم تطبيق قواعد خاصة مثلما هو الحال في الأشجار الثنائية.

ولكن ماذا سيحدث إذا لم نتبع قواعد الأشجار؟ حسنًا، نحن هنا إذن لا نتحدث عن الأشجار ولكن عن هياكل بيانات ديناميكية جديدة وهي المخططات **Graphs**. في الواقع فإن الأشجار ليست سوى حالة خاصة من المخططات.

المخطط Graph

المخطط هو أحد أنواع هياكل البيانات التي تتكون من مجموعة من العقد (أو النقاط أو الرؤوس)، ومجموعة من الخطوط أو (الأضلاع أو الأطراف)، والتي تربط جميع العقد أو بعضها.

يعتبر المخطط أكثر هياكل البيانات عمومًا، حيث أن جميع هياكل البيانات السابقة يمكن اعتبارها كحالات خاصة من المخططات.



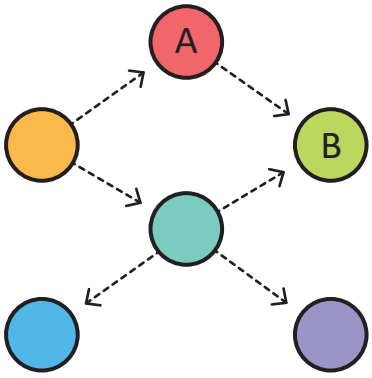
تعتبر الشجرة مخططًا ولكن العكس غير صحيح حيث لا تمثل جميع المخططات أشجارًا.

الفروقات بين الأشجار والمخططات	
الأشجار	المخططات
تترابط العقد فيما بينها في شكل هرمي.	تترابط العقد فيما بينها في شكل شبكة.
توجد في الأشجار عقدة فريدة تسمى الجذر.	لا توجد عقدة فريدة أو جذر.
ترتبط العقد بعلاقة أب وأبناء (Parent-Child).	لا توجد علاقة أب وأبناء بين العقد.
تعتبر الأشجار هياكل أكثر بساطة مقارنةً بالمخططات.	المخططات هي هياكل أكثر تعقيدًا.

أنواع المخططات

المخطط الموجه Directed Graph

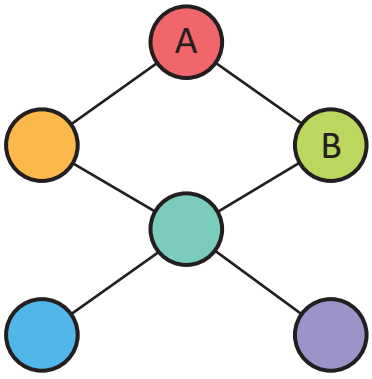
في المخطط الموجه، ترتبط العُقد بأضلاع موجهة باتجاه واحد فقط، أي أننا نستطيع الانتقال من العقدة **A** إلى العقدة **B** والعكس غير صحيح.



مخطط موجه

المخطط غير الموجه Undirected graphs

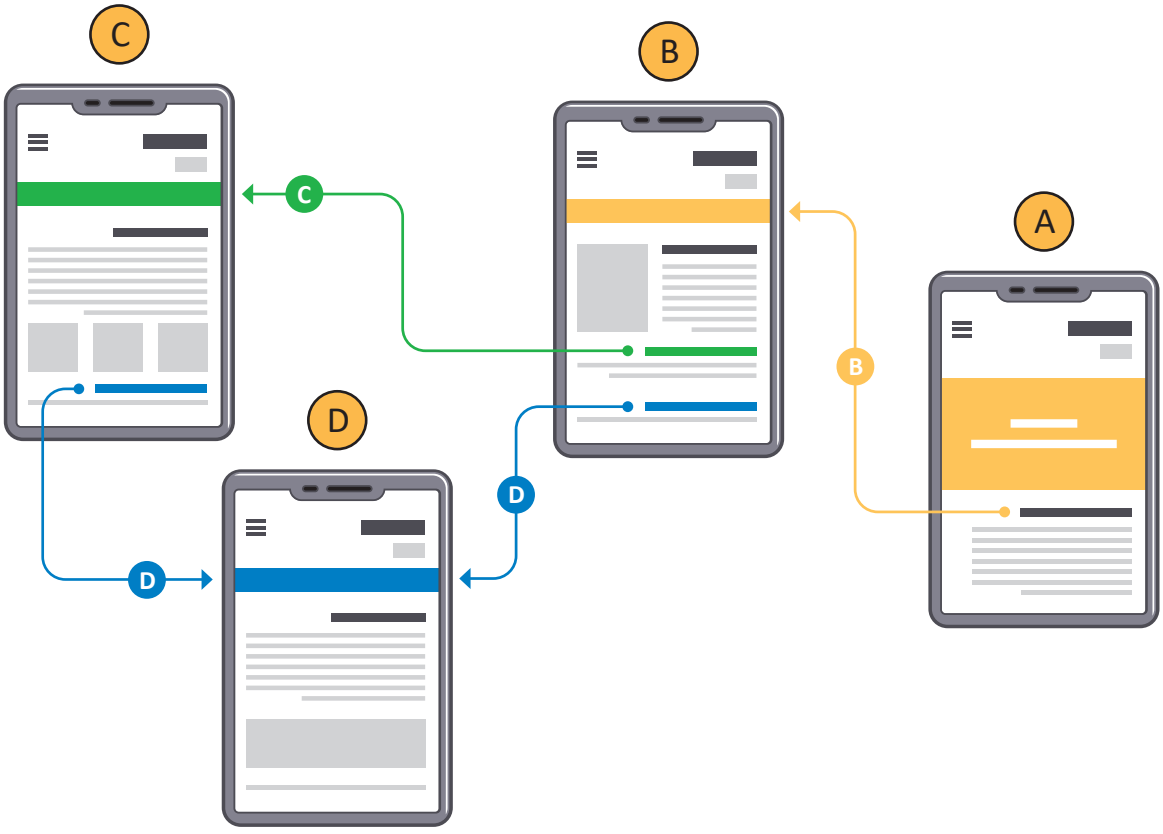
في المخطط غير الموجه، لا يكون للارتباطات أي اتجاه، أي أننا نستطيع الانتقال من العقدة **A** إلى العقدة **B** والعكس أيضا صحيح.



مخطط غير موجه

شبكة الويب العالمية World Wide Web

أكثر الأمثلة تمثيلًا للمخططات في حياتنا هو شبكة الويب العالمية، حيث يمكن اعتبارها بمثابة مخطط موجه، حيث تمثل العقد صفحات الويب وتمثل الأضلاع الموجهة الارتباطات التشعبية.

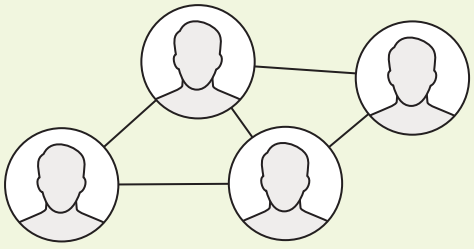


إن تنقيب هياكل الويب **(Web Structure Mining)** هو عملية اكتشاف المعرفة المفيدة من هياكل شبكة الويب العالمية من خلال الارتباطات التشعبية. يمكن للشخص أن يُكوِّن هياكل لمخطط هذه الارتباطات التشعبية والعلاقات الموجودة بين صفحات الويب المختلفة، ويمكن حساب ما يسمى بالأهمية النسبية **(Relative Importance)** لصفحة الويب من خلال الوصول لمثل هذا المخطط.

يستخدم محرك بحث Google نهجًا مشابهًا للعثور على الأهمية النسبية لصفحة الويب ويقدم نتيجة البحث للمستخدم وفقًا لهذه الأهمية، وتُعرف هذه الخوارزمية التي تستخدمها Google باسم خوارزميةPageRank ، وقد اخترعها مؤسسا جوجل Sergey Brin وLarry Page.

Facebook

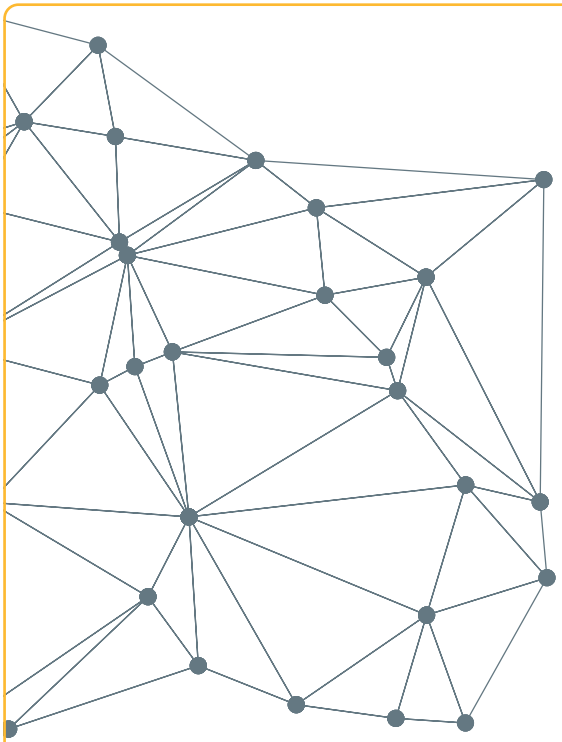
يعتبر الفيسبوك مثال آخر على هياكل المخطط الغير موجه، فعندما تضيف صديقًا يتوجب عليه قبول طلبك، فلا يمكن أن يكون هذا الشخص صديقك على الشبكة بدون الموافقة على صداقته. إن العلاقة هنا بين اثنين من المستخدمين (عقدتين) هي علاقة ثنائية الاتجاه، كما وأن خوارزمية اقتراح الأصدقاء في فيسبوك تستخدم نظرية المخطط (**Graph Theory**).



يدرس تحليل الشبكات الاجتماعية العلاقات الاجتماعية باستخدام المخطط أو نظرية الشبكة من علوم الحاسوب، حيث تمثل العقد المشاركين في الشبكة، بينما تمثل الخطوط العلاقات بين العقد.

خرائط Google

تستخدم خرائط **Google** وجميع التطبيقات المماثلة الأخرى المخططات غير الموجهة لإظهار أنظمة النقل وحساب أقصر مسار بين موقعين.



هل تعلم

تُعتبر الشبكة العصبية (**neural network**) أحد المفاهيم الأساسية للذكاء الاصطناعي، وهي عبارة عن مخطط يستخدم للتنبؤ بالمخططات الأخرى، فالعقد فيها تمثل الأماكن التي تتم فيها عمليات الحساب، بينما الأضلاع هي المسارات التي تتدفق من خلالها الإشارات من خلال العمليات الرياضية. تعتمد تقنيات الترجمة الآلية (**machine translation**) وتصنيف الصور (**image classification**) والتعرف على الأشياء (**object detection**) على الشبكات العصبية.

نظرًا لأن **Python** لا تقدم نوع بيانات معرف مسبقًا للأشجار، فهي لا تقدم نوع بيانات مُعرف مسبقًا للمخططات (تذكر أن الأشجار هي حالة خاصة من المخططات). لذلك، يمكننا أيضًا إنشاء المخططات بواسطة القوائم والقواميس.

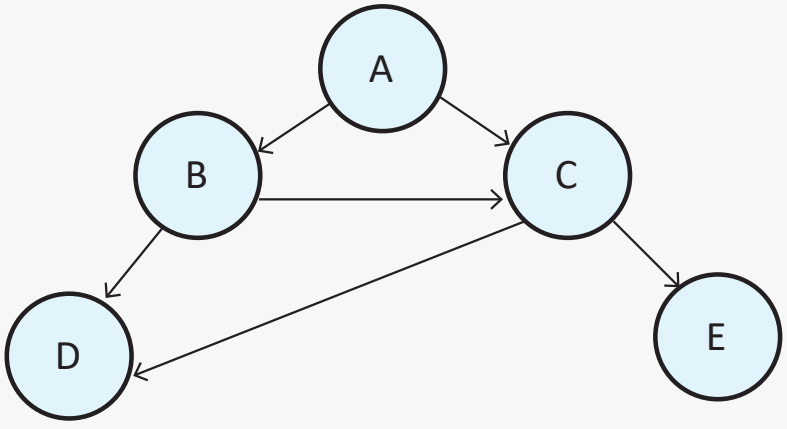
مثال: 1

في المثال التالي سنقوم بما يلي:

1. إنشاء مخططاً موجهاً كالمخطط الظاهر بالصورة.
2. إنشاء دالة لإضافة عقدة إلى المخطط.
3. إنشاء دالة تحتوي على جميع مسارات المخطط.

ثم البرنامج الرئيسي والذي يقوم بـ:

1. إنشاء المخطط.
2. طباعة المخطط
3. استدعاء دالة الإضافة.
4. طباعة جميع مسارات المخطط.



سنستخدم قاموسًا تكون مفاتيحه عُقد المخطط. ولكل مفتاح، ستكون القيمة المقابلة هي قائمة تحتوي على العقد المتصلة بواسطة ضلع مباشر من هذه العقدة.

هيا بنا ننشئ القاموس أولًا.

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

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Op
Python 3.7.0 (v3.7.0:
Type "copyright", "cr
>>>
========== RESTART: C:/python/examples.py ==========
{'a': ['b', 'c'], 'b': ['c', 'd'], 'c': ['d', 'e'],
 'd': [], 'e': []}
>>>
```

هذه الدالة تنشئ الأضلاع التي تربط عُقد المخطط.

```
# 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))
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
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')]
>>>
```

1

أكمل الجمل التالية بالكلمات المناسبة.

1. يصنف المخطط أنه من هياكل البيانات _____.

2. يمكن اعتبار الـ _____ من المخططات، ولكن ليس كل _____ يعتبر شجرة.

3. في المخططات تترابط العقد فيما بينها في شكل _____.

4. في الأشجار تترابط العقد فيما بينها في شكل _____.

5. تنقسم المخططات إلى _____ و _____.

6. Facebook نموذج على مخطط _____.

7. تعتبر شبكة الويب نموذج على مخطط _____.

8. المخططات في Python يمكن إنشاءها بواسطة _____ و _____.

2

اكتب مثالين على المخططات في حياتنا اليومية.

_____

_____

_____

3

ما وجه الاختلاف بين الأشجار والمخططات؟

_____

_____

_____

4

لماذا يعتبر Facebook مخطط غير موجه؟

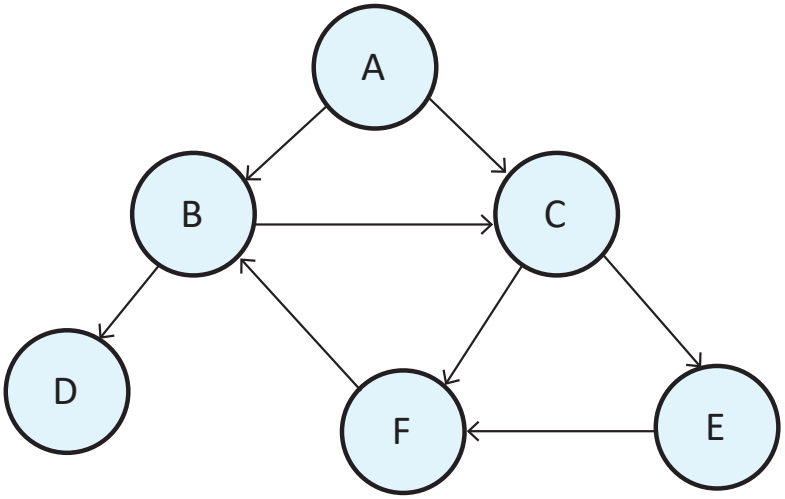
_____

_____

_____

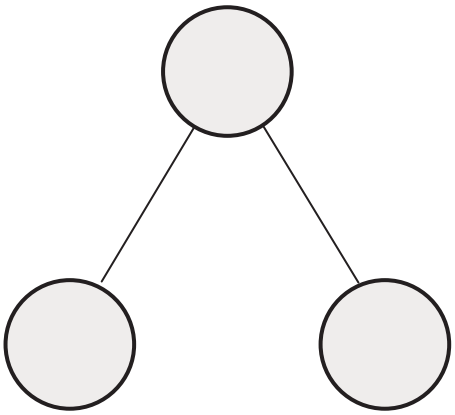
5

أنشئ قاموس في Python للمخطط التالي.

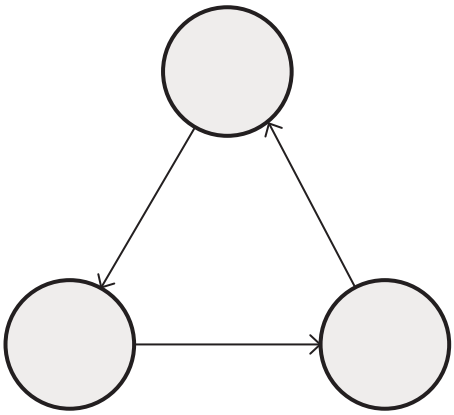


6

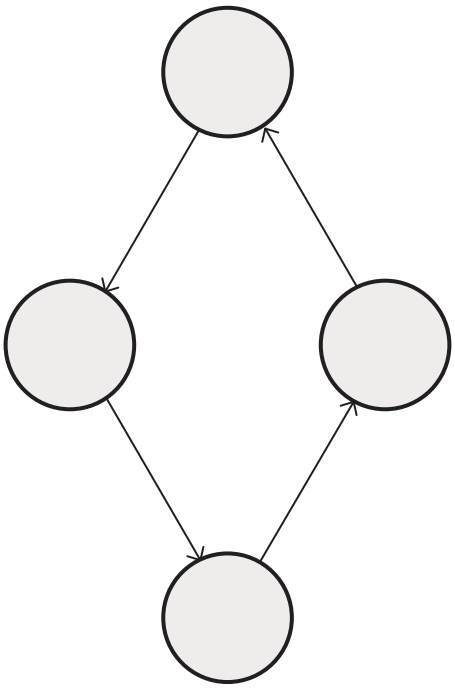
حدد في المخططات التالية أي منها موجه أو غير موجه؟



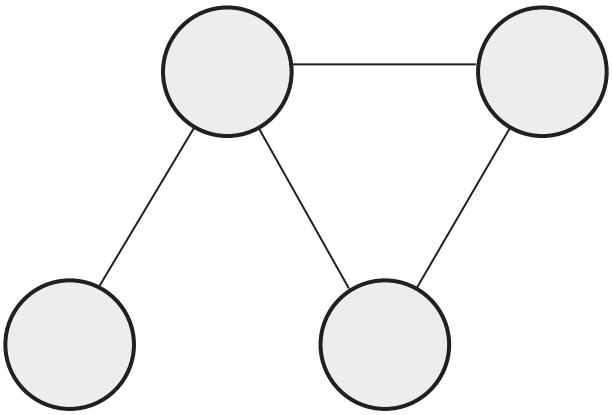
1



2



3



4

7

ارسم مخططًا يمثل مسارات إحدى الطرق وفقًا للمعلومات التالية.

> ترتبط المدينة A بالمدن B و E.

> ترتبط المدينة B بالمدن C و Z.

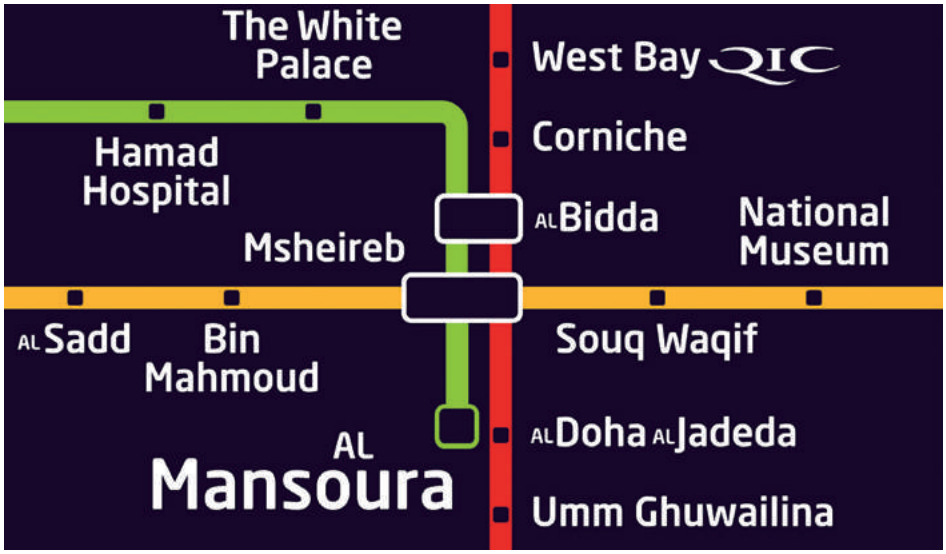
> ترتبط المدينة C بالمدن Z و D.

> ترتبط المدينة E بالمدن D و Z.

ما هو نوع المخطط السابق؟ _____

8

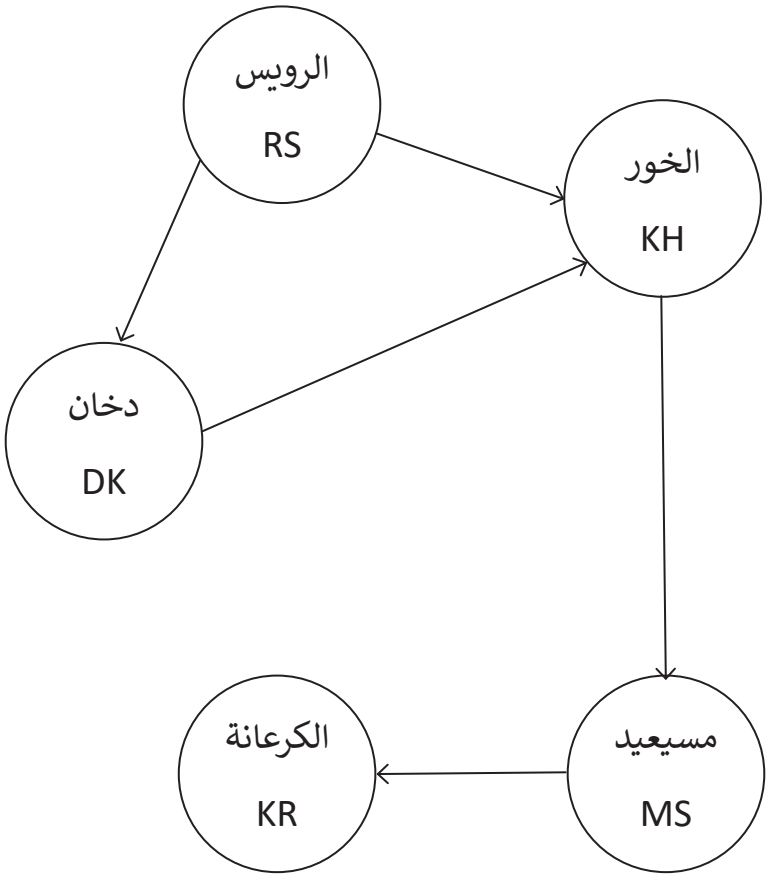
ارسم مخططًا يمثل أقسام خطوط مترو الدوحة كما يظهر في الصورة؟



ما نوع المخطط الذي قمت برسمه؟ _____

9

قررت الهيئة العامة للأشغال توصيل المدن التالية (الرويس - دخان – الكرعانة – مسيعيد – الخور) بطرق سريعة ومباشرة فقام أحد المهندسين برسم المخطط التالي:



> اعتمادًا على ما درست، ما هو نوع المخطط السابق؟ علل إجابتك.

> شغل البيئة البرمجية IDLE Python ثم قم بإنشاء قاموس تكون مفاتيحه أسماء المدن الموجودة بالمخطط السابق. ويقابل كل مدينة قائمة بالمدن المرتبطة بها بشكل مباشر

مشروع الوحدة

العنوان: المصرف

الوصف: يتم تقديـم الخدمة لعملاء البنك بناءً على طلب وصولهم إلى الفرع. لدى البنك صندوق واحد ومتوسط وقت الخدمة لكل عميل دقيقتان. لا يمكن أن يتجاوز الطابور 40 شخصًا في البنك.

الأدوات: لغة برمجة بايثون Python.

خطوات التنفيذ:

> سيحصل البرنامج على أحد قيم الاستيراد: "ENTRY" أو "NEXT".

> إذا أعطيت القيمة "ENTRY"، سيقوم بقراءة اسم العميل وبعد ذلك مباشرة يظهر عدد الأشخاص المنتظرين أمامه، إلى إذا كان الطابور ممتلئ، وسيظهر وقتها الرسالة "The store is full. Come another day".

> إذا تم تحديد قيمة "NEXT" فيجب عرض اسم العميل الذي ينتظر للخدمة.

> كرر العملية المذكورة أعلاه حتى لا يكون هناك عملاء بانتظار الخدمة.

> في النهاية سيتم عرض البرنامج على الشاشة

- عدد الأشخاص الذين تلقوا الخدمة.

- متوسط وقت انتظار العملاء.

ماذا تعلمت

تعلمت في هذه الوحدة:

- استخدام هياكل البيانات المتقدمة.
- المقارنة ما بين هياكل البيانات واستخداماتها.
- تمثيل هياكل البيانات المتقدمة باستخدام البرمجة بلغة Python.

المصطلحات:

الدرس 1	هيكل البيانات Data Structure	بسيطة Primitive	غير بسيطة Non Primitive
	مكدس Stack	طابور Queue	إضافة Push
	إزالة Pop	تجاوز الحد الأعلى Overflow	تجاوز الحد الأدنى Underflow
	مؤشر Pointer	إضافة للطابور Enqueue	إزالة من الطابور Dequeue
	مؤشر للمكدس Top	أمامي Front	خلفي Rear
الدرس 2	ثابت Static	ديناميكي Dynamic	قائمة مرتبطة Linked List
	عقدة Node	رأس Head	قيمة خالية Null
الدرس 3	خطي Linear	غير خطي Non-linear	شجرة Tree
	جذر Root	ابن Child	أب Parent
	ورقة Leaf	أشقاء Siblings	أطراف Edges
	شجرة فرعية Sub-tree	شجرة ثنائية Binary tree	شجرة القرار Decision Tree
الدرس 4	مخطط Graph	مخطط موجه Directed Graph	مخطط غير موجه Undirected Graph

2. أدوات التطوير البرمجي وتقنيات الاختبار

سيدرك الطلبة في هذه الوحدة أهمية تحويل لغات البرمجة عالية المستوى (**High Level Programming Languages**) إلى تعليمات قابلة للتنفيذ بلغة الآلة، وسيتعرفون على البرامج المستخدمة للقيام بهذه العملية. كما سيتمكن الطلبة من المقارنة بين مختلف أنواع تقنيات أدوات تطوير البرمجيات، واستخدام المكتبات البرمجية والدوال والإجراءات لتنظيم الكود البرمجي وإعادة استخدامه عند الحاجة.

ماذا سنتعلمْ؟

في هذه الوحِدة سنتعلم:

> لمحة تاريخية عن لغات البرمجة.
> ما هي لغة الآلة.
> ما هي لغة التجميع.
> لغات البرمجة عالية المستوى وميزاتها.
> تصنيفات لغات البرمجة.
> وظيفة كل من المترجم والمفسر.
> كيفية التعامل مع الأخطاء البرمجية في لغة البرمجة.
> التصنيفات المختلفة لأدوات تطوير البرمجيات.
> ما هو محرر التعليمات البرمجية وما مميزاته وتحديات استخدامه.
> ما هي بيئة التطوير المتكاملة وما مميزاتها وتحديات استخدامها.
> أمثلة لبعض الأداوت المتخصصة لمراحل تطوير البرمجيات المختلفة.
> كيفية الاستفادة من أدوات تطوير البرمجيات لتقديم حلول برمجية مختلفة.
> ما هو التصميم الهرمي وكيفية استخدامه لحل المشاكل المركبة.
> ما هي البرمجية التركيبية وما مميزاتها.
> ما هو البرنامج الفرعي وما خصائصه.
> الدوال والمعاملات والوسيطات.
> استخدام البرمجة التركيبة في تحسين المقاطع البرمجية.
> ما المكتبات البرمجية وما خصائصها.
> مكتبة Python القياسية وبعض وحداتها البرمجية.
> كيفية استخدام المكتبة واستيراد وحداتها البرمجية والدوال الموجودة بها.
> مدير الحزم Python PIP ودوره في استخدام المكتبات الخارجية.
> ما هو اختبار البرمجيات وما دور المختبر.
> أنواع الأخطاء البرمجية وكيفية ظهورها في المقاطع البرمجية.
> أهم استراتجيات الاختبار وأنواع البيانات المستخدمة فيها.
> توثيق عملية الاختبار والإصدارات المختلفة للبرامج.

مواضيع الوحدة

> لغات البرمجة عالية المستوى.
> أدوات تطوير البرمجيات.
> البرمجة التركيبية.
> المكتبات البرمجية.
> اختبار البرمجيات.

الأدوات

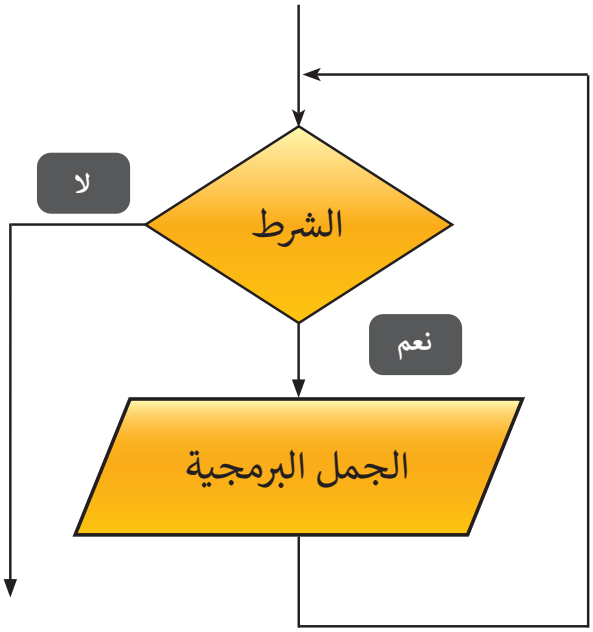
> Python 

هل تذكر؟

تكرار while

يستخدم تكرار **while** عندما لا يكون عدد مرات التكرار معروف سلفًا، حيث يستمر التكرار في العمل طالما كان شرط التكرار صحيحًا، وتستخدم المسافة البادئة في بداية الجمل البرمجية التي يحتويها مقطع التكرار تحت الأمر **while**، وذلك لتمييزها أثناء قراءة المقطع.

```
while condition:
    statements
```



tkinter module

مجموعة **tkinter module** هي مكتبات تحتوي على مقاطع برمجية (دوال) جاهزة يمكن استدعائها وتنفيذها في البرنامج دون الحاجة إلى كتابتها. سنتعرف مفهوم الدوال لاحقًا في هذه الوحدة.

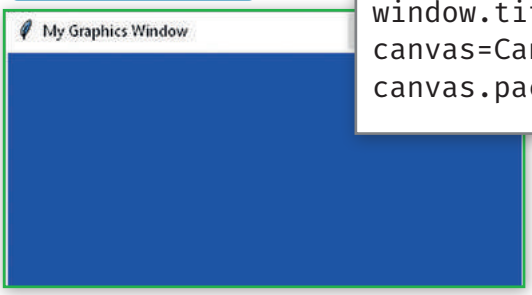
```
from tkinter import *
window=Tk()
window.title("My Graphics Window")
canvas=Canvas(bg="blue", width=400, height=200)
canvas.pack()
```

إنشاء لوحة رسومية **window**

عنوان اللوحة الرسومية.

لون خلفية اللوحة الرسومية.

حجم اللوحة الرسومية بالبكسل.



الدوال Functions

الدالة عبارة عن مجموعة من الجمل البرمجية المنظمة والقابلة لإعادة الاستخدام والتي يتم استخدامها لأداء وظيفة محددة، تتكون الدالة من ثلاثة أجزاء: الاسم والمعاملات وجسم الدالة. عندما نريد تعريف دالة، فإننا نستخدم الكلمة الرئيسة "**def**". وعندما نريد استخدامها، فإننا نكتب اسمها متبوعًا بأقواس.

خطوات كتابة الدالة	
.1	يبدأ تعريف الدالة البرمجية بكلمة **def** متبوعة باسم الدالة والأقواس ( )، تليها النقطتان (:).
.2	في حال تم استخدام المعاملات توضع بين الأقواس.
.3	تستخدم المسافة البادئة في بداية كل جملة برمجية داخل الدالة.
.4	ترجع الدالة قيمة عند استخدام كلمة **return**.
.5	يتم استدعاء الدالة من خلال البرنامج الرئيس أو من خلال أي دالة أخرى بكتابة اسم الدالة متبوعاً بقوسين يحتويان القيم أو الوسائط التي سيتم تمريرها لمعاملات الدالة بنفس الترتيب، وفي حال عدم وجود معاملات يتم وضع قوسين فارغين.

my_function: الاسم

fname: المعاملات

```
def my_function(fname):
    print("Name=" ,fname)

my_function("Ghalia")
my_function("Saad")
my_function("Deema")
```

المسافة البادئة مهمة جدًا. قبل جملة الطباعة **print** إذ تخبر مترجم اللغة البرمجية أنها جزء من الدالة.

جسم الدالة: هو مجموعة الجمل البرمجية الموجودة داخل الدالة.

تأخذ الدالة القيم من المعاملات كمدخلات، ويمكن أن تقوم بتنفيذ عدة عمليات على تلك القيم وفق الأوامر التي تحتويها لكي يتم إخراج النتائج.

```
def my_function(x):
    y=x*2
    return y

print(my_function(0))
print(my_function(1))
print(my_function(7))
print(my_function(10))
```

تُرجع الدالة قيمة باستخدام كلمة **return**

الدرس الأول

لغات البرمجة عالية المستوى

سنبدأ في هذا الدرس بتاريخ موجز عن لغات البرمجة وأنواع البرمجة، وسنتعرف على التقنيات المستخدمة لإنشاء البرمجيات بشكل صحيح، كما وسنتعلم كيفية عمل البيئات البرمجية.

لمحة تاريخية عن لغات البرمجة

لقد تغيرت الكثير من الأشياء منذ إنشاء أول حاسوب وحتى يومنا هذا، فقد تطورت مكونات الحواسيب وتقنياتها بشكل كبير، كما تقدمت إمكانات المعالجة، ورغم ذلك فإن مفاهيم تشغيل الحاسوب التي صاغها فون نيومان عام 1945 لم تتغير بشكل مطلق، فما زالت لغات البرمجة رغم تطورها وتحسيناتها المستمرة تتمتع بنفس الخصائص الأساسية الثابتة.

تم اختراع لغات البرمجة لغرض التواصل بين الإنسان والآلة.

لغة الآلة Machine Language

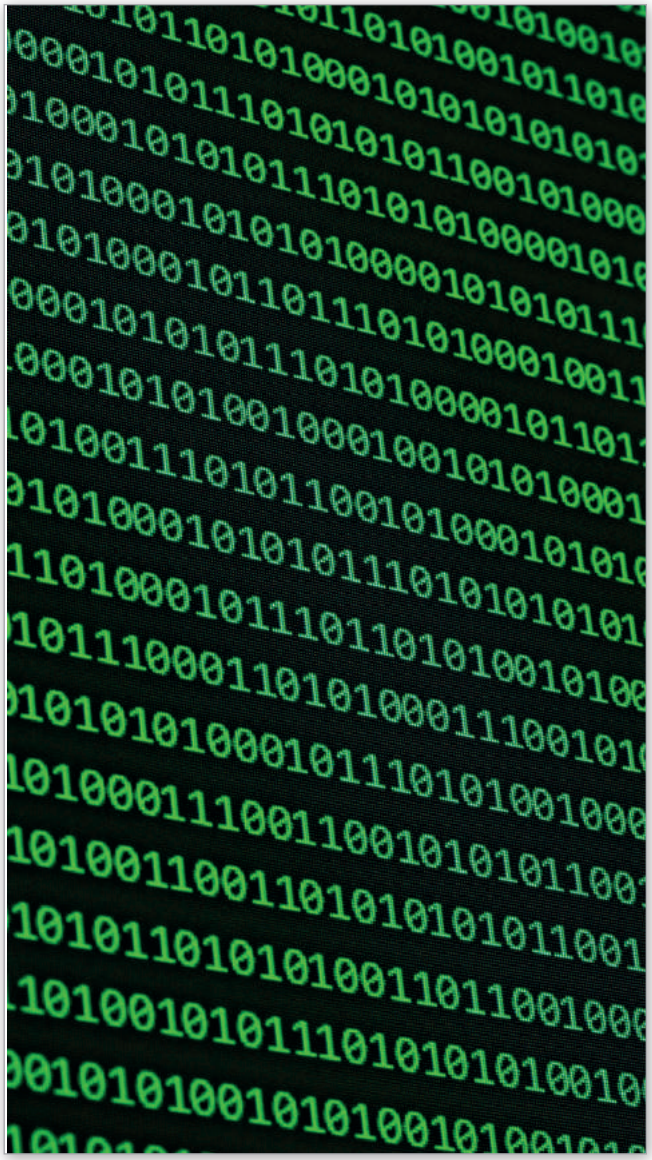
لكي يقوم الحاسوب بأداء أي وظيفة مطلوبة منه، يجب إعطاؤه التسلسلات المناسبة على شكل أرقام ثنائية مكونة من 0 و 1 مباشرة على شكل أوامر للحاسوب، ولكنها لن تكون مفهومة من البشر.

يعد استخدام مثل هذه اللغات صعب التطبيق والتنفيذ نظرًا لضرورة وجود معرفة عميقة بمكونات الحاسوب وبنيتها من قِبل المبرمج وخصوصًا لأن كل وحدة معالجة مركزية تختلف عن الأخرى في لغة الآلة الخاصة بها.

البرنامج بلغة الآلة عبارة عن تعليمات متسلسلة من وحدات البت الثنائية، وهي الأوامر الصادرة للمعالج لتنفيذ الوظائف الأساسية.

00111100

يتم تحويل أوامر البرمجة في النهاية إلى تسلسلات من الأوامر بلغة الآلة على شكل 0 و 1 ليتم تنفيذها بواسطة الحاسوب، حيث أن جميع لغات البرمجة ستتحول إلى برامج بلغة الآلة قابلة التنفيذ.



لمحة تاريخية

لبرمجة أول حاسوب ENIAC الشهير لكي يجري العمليات الحسابية المطلوبة، كان على شخص ما تغيير موضع مئات المفاتيح وضبط جميع الأسلاك على التوالي، وقد كانت هذه العملية معقدة للغاية وتستغرق وقتًا طويلاً، حيث تطلب إجراء عملية حسابية القيام بتغيير التركيبات السابقة للجهاز.

لغة التجميع Assembly Language

توجد بين لغة الآلة واللغات عالية المستوى لغة وسيطة يطلق عليها اسم لغة التجميع، وتسمى كذلك لغة البرمجة الرمزية.

تتشابه لغة التجميع مع لغة الآلة ولكنها أسهل نسبيًا في البرمجة حيث تسمح للمبرمج باستبدال الأرقام (0 ، 1) بالرموز.

يتم تحويل أوامر لغة التجميع المفهومة للإنسان، إلى التسلسلات المقابلة من 0 و 1 ليتمكن الحاسوب من فهمها وتنفيذها.

على سبيل المثال، كلمة **ADD** (المستخدمة للجمع) متبوعة برقمين، سهلة الفهم والحفظ للبشر، ولكن يتم ترجمتها في الحاسوب إلى سلسلة من وحدات البت ليتم تنفيذ العملية المطلوبة وهي جمع الرقمين. يتم تنفيذ عملية الترجمة هذه بواسطة برنامج خاص يسمى المجمع **(Assembler)**.

تتكون الأوامر بلغة التجميع من مقاطع رمزية تتوافق مع أوامر لغة الآلة.

مثال

لغة التجميع	لغة الآلة
ADD 7 20	101010101010

أوجه القصور في لغة التجميع:

من الواضح أن استخدام لغة التجميع هو عملية تطوير للتسلسلات الثنائية غير المفهومة، ولكن رغم ذلك فإنها تعتبر ضمن اللغات منخفضة المستوى.

- ترتبط ارتباطًا تامًا ببنية كل حاسوب.
- لا توفر أوامر لأداء وظائف أكثر تعقيدًا من الإضافات البسيطة والمضاعفات والمقارنات، مما يضطر المبرمج لكتابة برامج طويلة ومعقدة يصعب فهمها وتصحيحها.
- لا يمكن نقل أي برنامج من حاسوب إلى آخر.

في الجدول التالي ترى برنامجًا تم إنشاؤه بلغة الآلة وبلغة التجميع وبلغات البرمجة عالية المستوى لحساب مجموع الأرقام من 1 إلى 10.

لغة عالية المستوى	لغة التجميع	لغة الآلة

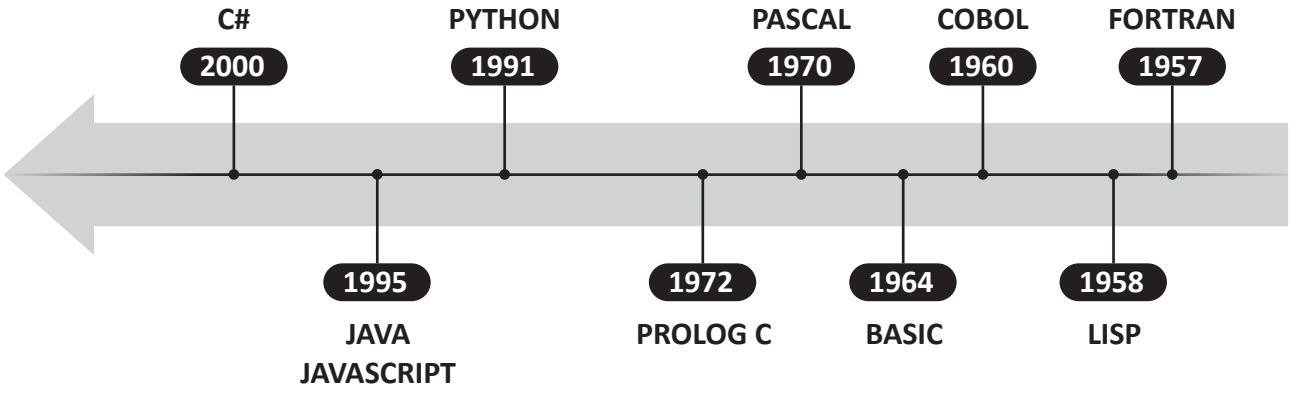
`sum = 0`
`for i in range (1,11):`
`  sum = sum + i`
`  print (sum)`
	`INDEX=$01`
	`SUM=$02`
	`LDA #10`
	`STA INDEX`
	`CLA`
	`ADD INDEX`
	`DEC INDEX`
	`BNE LOOP`
	`STA SUM`
	`BRK`
		`10001100 00000001`
		`00111100`
		`01010001 00000001`
		`01000011 00000001`
		`11000000 11111010`
		`10001100 00000010`
		`11111111`

لغات البرمجة عالية المستوى High Level Languages

أدت أوجه القصور الموجودة في لغة الآلة ولغة التجميع إلى تضافر الجهود لتحقيق تواصل أفضل بين الإنسان والآلة، مما نجم عنه ظهور أول لغة برمجة عالية المستوى في خمسينات القرن الماضي.

استخدمت لغات البرمجة عالية المستوى جملًا برمجية قريبة من لغة الإنسان، مما استدعى ترجمة البرامج الناتجة إلى لغة الآلة بواسطة الحاسوب نفسه وذلك من خلال برنامج خاص يسمى المترجم **Compiler**.

تطور لغات البرمجة زمنيًا



تطور لغات البرمجة عالية المستوى

يختار المطور لغة البرمجة التي تسمح بتطوير التطبيق بسهولة في بيئة معينة لتنفيذ حل برمجي، ولكن في نفس الوقت يختارها بناءً على معرفته الشخصية ومهاراته وتفضيلاته.

تحتوي كل لغة برمجة على مجموعة فريدة من الكلمات المحجوزة (**Reserved Words**) (الكلمات التي تفهمها هذه اللغة) ومجموعة من الصيغ **Syntax** يتم اتباعها من طرف المبرمج لكتابة التعليمات.

لمحة تاريخية

في عام 1970 تم للمرة الأولى إدخال تقنية تصميم البرنامج مع PASCAL والتي غيرت جذريًا طريقة تطوير البرمجيات ولغات البرمجة نفسها، وقد أطلق عليها اسم تقنية البرمجة التركيبية (Structured Programming Technique)، وهي تضمن سهولة كتابة البرامج وفهمها وسهولة تصحيحها وصيانتها.

لغة البرمجة	المطور	أصل التسمية	الخصائص
FORTRAN	IBM	FORmula TRANslation	مناسبة لحل المشكلات الرياضية والعلمية، ولكنها لا تناسب إدارة ملفات البيانات مثلًا.
LISP	MIT	LISt Processor	لغة خاصة بالذكاء الاصطناعي.
COBOL	CODASYL	Common Business Oriented Language	مناسبة لتطوير التطبيقات التجارية وتطبيقات الإدارة العامة.
BASIC	جامعة Dartmouth	Beginner's All Purpose Symbolic Instruction Code	تم تطويرها لتدريب المبتدئين على البرمجة.
PASCAL	البروفسور نيكولاس ويرث	سميت باسم عالم الرياضيات Blaise Pascal	تشتهر بتقديمها لتقنيات البرمجة المنظمة. وتعتمد تصميم البرامج بطريقة منهجية ودقيقة.
C	دينيس ريتشي ومختبرات بيل	سُميت بلغة C بعد لغة مسبقة كان اسمها B	تستخدم لتطوير نظام تشغيل **UNIX**، وهي ملائمة لنظم التشغيل.
JAVA	أنظمة SUN	سميت نسبة إلى كلمة القهوة (Java)	هي لغة برمجة من نوع **(Object Oriented)**، تستخدم لتطوير التطبيقات التي يمكن تشغيلها على مجموعة كبيرة من أجهزة الحاسوب أو أنظمة التشغيل المختلفة.

ميزات لغات البرمجة عالية المستوى

تتمتع لغات البرمجة عالية المستوى بالعديد من المزايا مقارنةً بلغة التجميع كما يلي:

- تستخدم منطق وصيغ برمجية مفهومة وقريبة من لغة الإنسان.
- مستقلة عن نوع الحاسوب، فيمكن استخدامها على أي جهاز بتعديلات طفيفة أو دون ذلك.
- يمكن للمطورين تعلم لغات البرمجة عالية المستوى بسهولة وبسرعة أكثر.
- عملية تصحيح وصيانة البرامج أسهل بكثير.

بشكل عام فإن لغات البرمجة عالية المستوى تقلل من وقت وتكلفة تطوير البرمجيات بشكل كبير مقارنةً بلغات البرمجة منخفضة المستوى.

لغات برمجة الجيل الرابع

بالنظر إلى لغات البرمجة عالية المستوى، نلاحظ أنه يوجد ما يسمى بلغات برمجة الجيل الرابع والتي عادةً ما يتم اختصارها بـ **4GL**. تعتبر لغات برمجة الجيل الرابع أقرب إلى لغة الإنسان من اللغات الأخرى عالية المستوى ويمكن الوصول إليها من قبل الأشخاص الذين ليس لديهم تدريب رسمي كمبرمجين لأنها تتطلب مقدارًا أقل من الترميز.

تعتبر لغات الجيل الرابع أكثر ملاءمة للمبرمجين وتعزز كفاءة البرمجة عن طريق استخدام كلمات وعبارات تشبه اللغة الإنجليزية، وكذلك استخدام الأيقونات، والتمثيلات الرمزية والواجهات الرسومية عند الحاجة. يكمن المفتاح الرئيسي في تحقيق الكفاءة عن طريق استخدام لغات الجيل الرابع في التوافق بين الأداة ومجال التطبيق.

يمكن لمستخدم حاسوب في لغات الجيل الرابع إجراء تغييرات على البرنامج من أجل تلبية حاجة جديدة، كما تكون لديه القدرة على حل المشاكل الصغيرة بنفسه. يمكن إجراء عمليات مشتركة متعددة باستخدام أمر واحد يدخله المبرمج.

على سبيل المثال، بالنسبة لقواعد البيانات، يمكن للمستخدم إنشاء استعلامات وتقارير باستخدام لغة **SQL** للإحصاءات والمشاريع العلمية حيث يمكن لمتخصص الرياضيات أو الباحث استخدام برامج مثل **SPSS** و **MATLAB** و **LabVIEW** لتحليل هذه البيانات.

تتوفر لمستخدم الحاسوب القدرة النسبية لإنشاء استعلام عن النظام أو تطوير التطبيقات التي تسترد المعلومات من قواعد البيانات وتحديد الطريقة التي سيتم بها عرض المعلومات بسهولة بشكل دقيق.

تصنيفات لغات البرمجة

هناك العديد من التصنيفات للغات البرمجة فيمكن تصنيف اللغات من حيث كتابة أوامر البرنامج إلى لغات البرمجة الإجرائية ولغات البرمجة الكائنية.

حيث تستخدم البرمجة الإجرائية مجموعة من التعليمات لإخبار الحاسوب بما يجب القيام به خطوة خطوة، ومن الأمثلة عليها لغة كوبول (**Cobol**) وفورتران (**Fortran**).

أما البرمجة الكائنية فيتم فيها تقسيم البرنامج إلى وحدات تسمى الكائنات (**Object**) ومن الأمثلة عليها لغة السي شارب (**C#**) ولغة البايثون.

هناك تصنيف آخر يعتمد على الوظيفة البرمجية للغة:

- لغات البرمجة العامة: نظريًا يمكن استخدام أي لغة عامة لحل أي مشكلة، ولكن من الناحية العملية، تم تصميم كل لغة لحل نوع معين من المشكلات، ويتم تقسيم هذه اللغات كما يلي:
  - اللغات ذات التوجه العلمي مثل **Fortran**.
  - اللغات الموجهة نحو الأعمال مثل **COBOL**.
  - اللغات متعددة المجالات مثل **BASIC** و **Pascal**.
  - لغات برمجة نظم التشغيل مثل **C**.
  - لغات الذكاء الاصطناعي مثل **LISP** و **PROLOG**.
  - لغات إدارة قواعد البيانات مثل **SQL**.
- لغات متخصصة مثل **LISP** يتم استخدامها في نوع معين من التطبيقات مثل الروبوتات أو الدارات المتكاملة.

كيف تفهم الحواسيب لغات البرمجة.

يتم تحويل أي برنامج مكتوب بأي لغة برمجة إلى لغة الآلة التي يمكن فهمها وتنفيذها من قبل الحاسوب، وذلك من خلال استخدام برامج ترجمة خاصة.

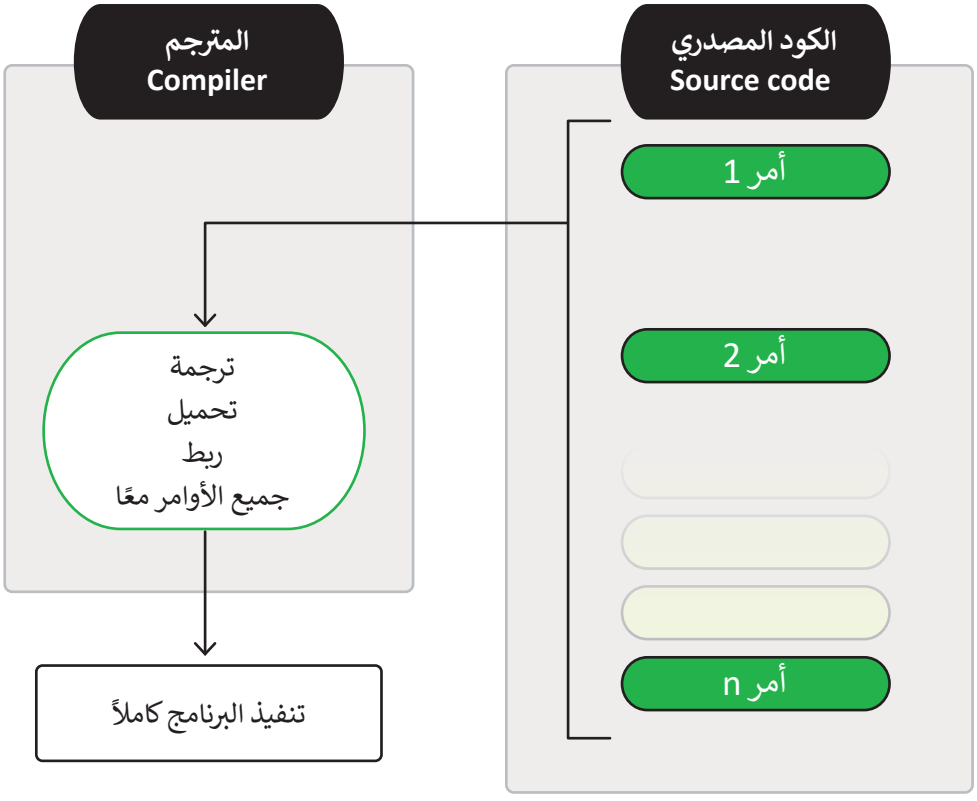
هناك طريقتان لتشغيل البرامج المكتوبة بلغة عالية المستوى، أكثرهما شيوعًا هي ترجمة البرنامج بواسطة المترجم **Compiler**، والأخرى هي تمرير البرنامج من خلال المفسر **Interpreter**.

لنستعرض كيفية تنفيذ هاتين الطريقتين المختلفتين.

ما هو المترجم Compiler؟

المترجم هو برنامج حاسوبي يُحوِّل كامل المقطع البرمجي المكتوب بلغة برمجة عالية المستوى إلى لغة الآلة، والتي يفهمها معالج الحاسوب (لغة النظام الثنائي 0 و 1).

عملية ترجمة وتنفيذ برنامج باستخدام المترجم **Compiler**:



البرنامج أو الكود المصدري هو البرنامج المكتوب بلغة برمجة عالية المستوى.

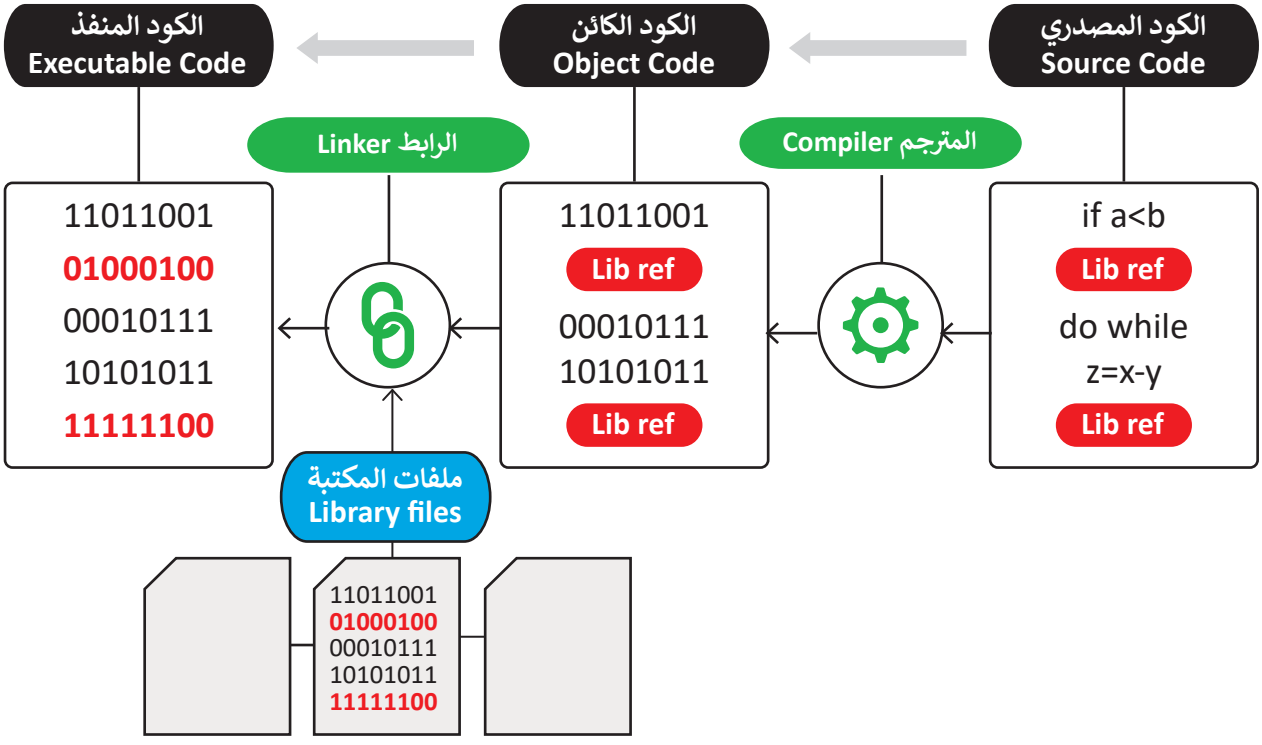
عملية ترجمة وربط البرنامج:

- يقبل المترجم **Compiler** برنامجًا مكتوبًا بلغة عالية المستوى كمدخل (الكود المصدري **Source Code**)، وينتج برنامجًا مكافئًا بلغة الآلة يسمى الكود الكائن **Object Code**.

- يتعذر على المترجم تحويل الجمل البرمجية التي تشير إلى مكتبات قياسية **Libraries** أو مصادر خارج الكود المصدري، ولذلك ستحتاج العملية إلى خطوة إضافية لربط وتحويل تلك الجمل.

- يتولى عملية الربط برنامج آخر يسمى "الرابط" **Linker** أو **Loader** ويقوم بربط ملف الكود الكائن **Object Code** بملفات المكتبة القياسية، وينتج الكود القابل للتنفيذ **Executable Code**، وهو البرنامج النهائي الذي ينفذه الحاسوب.

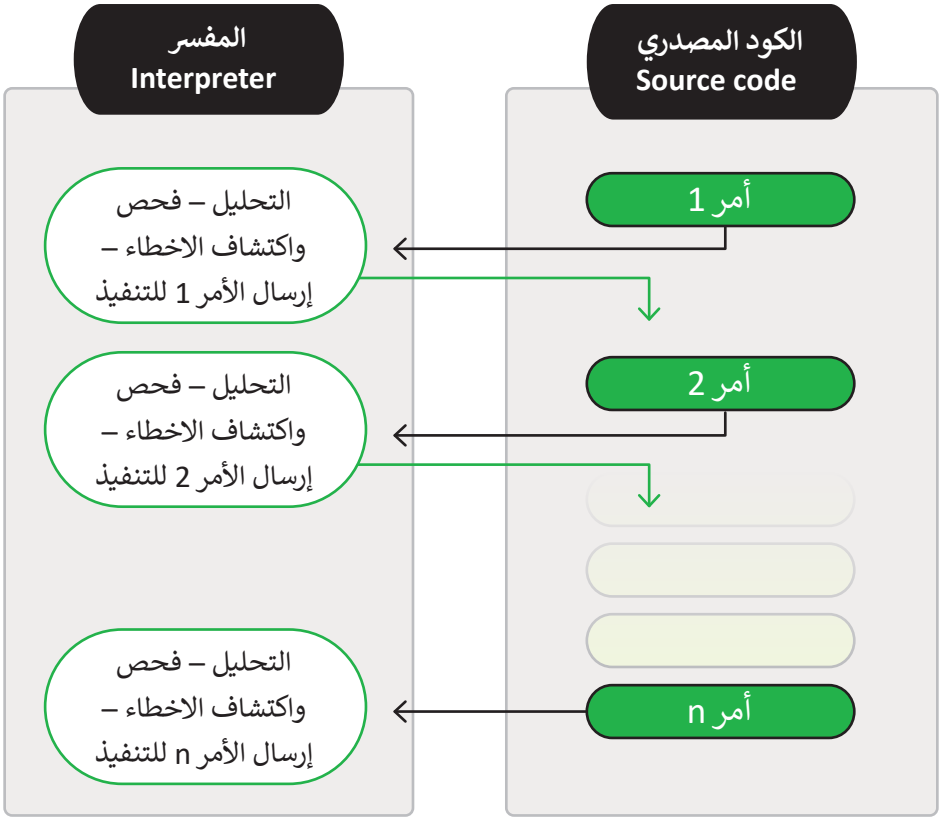
مراحل ترجمة وربط البرنامج



ما هو المُفسِّر Interpreter؟

هو برنامج حاسوبي يحول كل جملة برمجية من المقطع البرمجي المكتوب بلغة عالية المستوى إلى لغة الآلة ويرسلها للتنفيذ مباشرةً.

عملية ترجمة وتنفيذ البرنامج باستخدام المفسر:



يقوم كل من المترجم والمفسر بالمهمة ذاتها، وهي تحويل البرنامج المكتوب بلغة البرمجة عالية المستوى إلى لغة الآلة، ولكن بطريقتين مختلفتين، حيث يحول المترجم كامل البرنامج إلى لغة الآلة في صورة ملف (**exe**) جاهز للتنفيذ، بينما يحول المفسر كل جملة برمجية مكتوبة ويرسلها للتنفيذ.

استخدام اللغات للمترجم والمفسر

تستخدم معظم لغات البرمجة الحديثة المترجمات البرمجية **Compilers** لإنتاج التعليمات البرمجية بسرعة وبصورة محسنة، ولكن هناك بعض اللغات التي ما زالت تستخدم المفسرات **Interpreters** عند الحاجة لإنشاء برنامج بسيط دون الاهتمام بالسرعة بالمقام الأول.

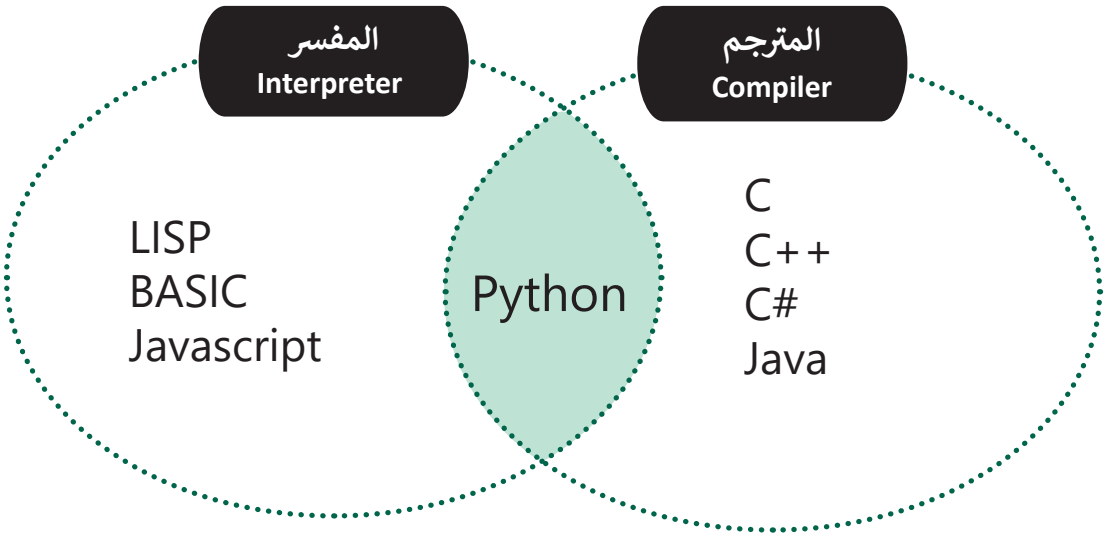
اللغات التي تعتمد على المترجمات Compilers

تستخدم لغات البرمجة C و **C++** و **C#** و **Java** المترجم لإنشاء برامج سريعة وموثوقة. يتم إنشاء الكود القابل للتنفيذ لكل نوع من أنواع أجهزة الحاسوب مما يعني أن على المطورين الإلمام بالمعرفة حول أجهزة الحاسوب التي يستخدمها المستخدم النهائي.

اللغات التي تعتمد على المفسرات Interpreters

تستخدم لغات **JavaScript** و**LISP** و**BASIC** القديمة المفسرات، مما يعني أن البرامج تعمل بصورة بطيئة ولكن يمكن تشغيل كودها المصدري على أي جهاز حاسوب يحتوي على مفسر للغة البرمجة خاصة، على سبيل المثال، يمكن تشغيل تطبيق الويب المكتوب بلغة **JavaScript** على جهاز حاسوب يعمل بنظام **Windows** أو جهاز لوحي يعمل بنظام **Android** من خلال متصفح الويب الذي يدمج المفسر الخاص بلغة **JavaScript**.

تعبر **Python** لغة مترجمة ومفسرة أيضًا، وما يحدث أن التطبيق يحدد طريقة البرمجة المستخدمة لتنفيذه، بينما لا تتغير الصيغة البرمجية.



المقارنة بين المترجم Compiler والمفسر Interpreter

	المترجم Compiler	المفسر Interpreter
الوظيفة الرئيسة	يحول كامل المقطع البرمجي المكتوب بلغة برمجة عالية المستوى إلى لغة الآلة، وينتج برنامجًا تنفيذيًا.	يحول المقطع البرمجي إلى لغة الآلة بحيث يترجم الجملة البرمجية ثم تُنفذ، وينتقل إلى الجملة التالية أثناء تشغيل البرنامج.
الإدخال	المترجم يأخذ الكود المصدري بأكمله كمدخل.	يأخذ المفسر أحد تعليمات الكود المصدري كمدخل في كل مرة.
الإخراج	يقوم المترجم بإنشاء وتخزين ملف كود كائن كمخرج.	لا يقوم المفسر بإنشاء ملف كود الكائن.
الذاكرة	يتطلب المزيد من الذاكرة بسبب إنشاء كود الكائن.	الذاكرة المطلوبة أقل.
عملية التنفيذ	تتم عملية الترجمة لكامل المقطع البرمجي قبل البدء بالتنفيذ.	تتم عملية الترجمة لكل جملة برمجية بالتوازي مع عملية التنفيذ.
التحقق من الأخطاء	يعرض المترجم جميع الأخطاء اللغوية والتحذيرات عند ترجمة البرنامج، فلا يمكنك تشغيل البرنامج إذا لم تقم بتصحيح تلك الأخطاء.	يقرأ المفسر جملة واحدة ويعرض أي خطأ موجود بها، ويتوجب تصحيح هذا الخطأ قبل الانتقال إلى الجملة التالية.

	المترجم Compiler	المفسر Interpreter
ربط الملفات	يحتاج إلى برنامج لربط الملف الكائن بملفات المكتبة القياسية لإنشاء الملف التنفيذي.	لا يحتاج إلى عملية الربط، ولا ينشئ ملفًا تنفيذيًا.
سهولة الاستخدام	لإجراء أي تغيير أو تطوير في البرنامج يجب الرجوع إلى الكود المصدري.	يمكن تغيير أو تطوير البرنامج بتعديل الكود المصدري في أي وقت مما يجعله أسهل في الاستخدام للمبتدئين.
السرعة	توفر الملف التنفيذي **exe** يجعل التنفيذ أكثر سرعة.	تكون عملية التنفيذ أبطأ لعدم توفر البرنامج التنفيذي مما يتطلب إعادة ترجمة كامل البرنامج في كل عملية تنفيذ.
لغات البرمجة المتعلقة	تستخدم **C** و **C ++** و **C #** و **Java** جميع المترجمات.	تستخدم **LISP** و **BASIC** و **Javascript** مفسراً.
الاعتماد على الأجهزة ونظم التشغيل	يعتمد البرنامج التنفيذي الذي تم إنشاؤه بواسطة المترجم على الأجهزة المستهدفة الفعلية. لا يمكن تشغيله على بنى **CPU** مختلفة أو أنظمة تشغيل مختلفة.	المفسر مستقل عن الأجهزة ونظام التشغيل. على سبيل المثال، يمكن أن يعمل مفسر **Python** على نظامي التشغيل **Windows** و **Linux** بنفس الكود المصدري ويعطي نفس النتائج.

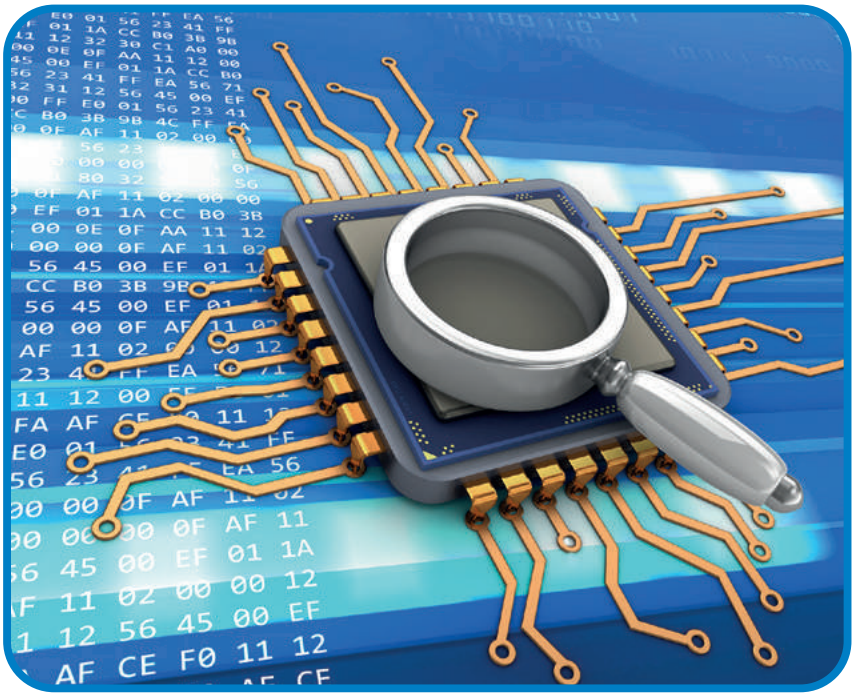
المترجم Compiler:

- إنشاء البرنامج.
- سيحلل المترجم ويعالج جميع الجمل البرمجية ويتأكد من صحتها، فإذا كان هناك خطأ فسوف يظهر هذا الخطأ.
- إذا لم يكن هناك خطأ سيقوم المترجم البرمجي بتحويل الكود المصدري إلى لغة الآلة.
- سيتم ربط ملفات التعليمات البرمجية المختلفة ببرنامج قابل للتنفيذ (يعرف باسم ملف **exe**).

المفسر Interpreter:

- إنشاء البرنامج.
- يقرأ المفسر جملة واحدة ويعرض أي خطأ لغوي بها، ويتوجب تصحيح هذا الخطأ قبل الانتقال إلى الجملة التالية.
- يتم تنفيذ جميع عبارات تعليمات الكود المصدري البرمجية سطرًا تلو الآخر أثناء تنفيذ البرنامج بواسطة المفسر.

تصحيح الأخطاء أثناء عملية الترجمة والربط Debugging



قد يحتوي الكود المصدري في نسخته الأولى في أغلب الأحيان على العديد من الأخطاء، والتي تنقسم إلى ثلاثة أنواع: أخطاء في منطق البرنامج، وتسمى أخطاء منطقية **Logic Errors**، والأخطاء التي تحدث أثناء تنفيذ البرنامج، وتسمى أخطاء وقت التشغيل **Runtime Errors** وأخطاء في بناء الجمل البرمجية وتسمى أخطاء لغوية **Syntax Errors**. تحدث الأخطاء المنطقية وأخطاء وقت التشغيل فقط عند تنفيذ البرنامج، بينما تحدث الأخطاء اللغوية أثناء الترجمة البرمجية. يتم تنفيذ البرنامج فقط في حالة كون البرنامج المصدري لا يحتوي على أخطاء لغوية **Syntax Errors**.

آلية تصحيح الأخطاء:

- في الخطوة الأولى يكشف المترجم أو المفسر عن الأخطاء اللغوية ويعرض الرسائل الدالة على هذا الخطأ وموقعه، ويمكن لبعضها تحديد سبب الخطأ.
- الخطوة التالية هي تصحيح الأخطاء الموجودة في البرنامج.
- أخيرًا، يتم اعتماد البرنامج المصحح لترجمته وتتكرر هذه الخطوة حتى يتم تصحيح جميع الأخطاء.

1

عدد أوجه القصور في لغة التجميع؟

2

وضح بالرسم الفرق بين عملية الترجمة والتنفيذ في كل من المترجم والمفسر.

3

أكتب ثلاث من إيجابيات لغات البرمجة عالية المستوى؟

4

صِل كل لغة برمجة مع التصنيف الذي تنتمي إليه.

1 FORTRAN	○ اللغات ذات التوجه العلمي
2 BASIC	○ لغات ذكاء اصطناعي
3 PROLOG	○ لغات متعددة المجالات
4 C	○ لغات برمجة الأنظمة

5

اختر الكلمة المناسبة لإكمال الفقرة:

مكتبات قياسية	المفسر	أخطاء لغوية	الذاكرة	الرابط
الكود المصدري	المترجم	أخطاء وقت التشغيل	الكود الكائن	

1. يقبل _____ البرنامج المصدري كمدخل، وينتج برنامجًا مكافئًا بلغة الآلة يسمى _____ .
2. _____ التي يستخدمها المفسر أقل من التي يستخدمها المترجم.
3. يتم ترجمة _____ فقط مرة واحدة في مرحلة الترجمة عند استخدام المترجم.
4. استخدام _____ يعتبر ميزة من حيث التصحيح الآني، ولكن تنفيذ البرنامج يكون أبطأ.
5. _____ هو الذي يقوم باخراج الكود النهائي القابل للتطبيق عند استخدام المترجم.
6. لا يستطيع المترجم تحويل الجمل البرمجية التي تشير إلى _____، لذلك يحتاج إلى ربط وتحويل تلك الجمل.
7. يمكن إنشاء البرنامج التنفيذي إذا لم يكن هناك _____ في البرنامج المصدري.
8. تدعى الأخطاء التي تتم أثناء تنفيذ البرنامج بـ _____.

الدرس الثاني

أدوات تطوير البرمجيات

يستخدم المطورون مجموعة كبيرة من برامج الحاسوب للتطبيقات والبرامج، والتي يوجد لكل منها العديد من المزايا والعيوب. تتطلب عملية البرمجة أن يتحلى المطورون بالمرونة والقدرات الإبداعية للاستفادة الكاملة من الإمكانات المتوفرة في أدوات تطوير البرمجيات وذلك لتقديم أعمال عالية الجودة للشركات.

تستخدم أدوات وبرامج تطوير البرمجيات لمساعدة فريق تطوير البرمجيات في المهمات المختلفة بما فيها إنشاء وتعديل وصيانة البرامج وكذلك تصحيح الأخطاء و تنفيذ المهام البرمجية وعمليات التطوير، كما توجد الكثير من البرامج المستقلة أو المتخصصة والتي توفر أو تدعم مهمات محددة في مراحل دورة تطوير البرمجيات.

تصنيف أدوات تطوير البرمجيات

فيما يلي مجموعة من الأدوات المستخدمة في تطوير البرمجيات:

محررات التعليمات البرمجية Code Editors	تستخدم لكتابة وإجراء التغييرات الخاصة بالتعليمات البرمجية.
المترجمات والرابطات Compilers and Linkers	ترجمة البرامج إلى لغة الآلة القابلة للتنفيذ.
المصححات Debuggers	تساعدنا في تصحيح الأخطاء في البرامج.
منشئات المشروع Project Builders	تتأكد من أن الملفات المحددة سيتم ترجمتها وربطها ببرنامج نهائي واحد.
أدوات إدارة التعليمات البرمجية Code Management Tools	تضمن الحفاظ على الملفات البرمجية من الاستبدال عن طريق الخطأ عندما يعمل العديد من المبرمجين على نفس البرنامج بشكل متزامن.
بيئة التطوير المتكاملة (IDE)	تتيح للمبرمجين بيئة برمجية متكاملة تتضمن محرر النصوص والمترجم والرابط والمصحح.
المحللات Profilers	تمنحنا في العادة فكرة جيدة عن حاجات وتعامل البرنامج مع وقت المعالج وموارد الذاكرة أثناء التشغيل.
محللات الشبكة Network Analyzer	تكون ضرورية عند كتابة برمجيات خاصة بتطبيقات تتعلق بالشبكات بشكل خاص.
مستكشفات ومحلل قواعد البيانات Database Explorer and Analyzers	تتيح التعامل مع قواعد البيانات وتحليل الأداء الخاص باستعلامات قواعد البيانات المحددة.

سبق وتناولنا موضوع المترجمات والرابطات، وسنتحدث في هذا الدرس تفصيلاً عن محررات التعليمات البرمجية وبيئات التطوير المتكاملة.

محرر التعليمات البرمجية Code Editor

يسمح لنا هذا المحرر بإنشاء وتحرير مجموعة من ملفات لغة البرمجة، وعادة ما يمكنه التعامل مع العديد من اللغات المختلفة مثل **HTML** و**CSS** و**JavaScript** و**PHP** و**Ruby** و**Python** و**C** وغيرها.

لا يمكن استخدام معالجات النصوص الاعتيادية مثل ميكروسوفت وورد للتعامل مع الأكواد البرمجية وذلك لتنسيقها نصوص المستندات بشكل لا يتناسب مع المقاطع البرمجية.

ومن خصائص محررات التعليمات البرمجية:

- سهولة الاستخدام والتنقل بين القوائم وأوامر البرنامج.
- إمكانية استخدام أوامر البحث والاستبدال.
- توفير وظائف أوامر النسخ والقص واللصق كما في معالجات النصوص.
- إمكانية تمييز الجمل البرمجية بألوان مختلفة حسب طبيعة الجملة.
- قابلية تخصيص المظهر والقوائم في المحرر.
- قابلية التوسع لدعم لغات برمجية أخرى.

average.py

```
# calculate the average class grade
total_grades = 0
total_students = int(input("Enter the number of students: "))
for  n in range (1, total_students + 1):
    print("Student #", n)
    student_name = input("Enter the name of the student: ")
    student_grade = input("Enter the grade of " + student_name + ": ")
    total_grades = total_grades + float(student_grade)
average_grade = total_grades / total_students
print("The average grade of the class is ", average_grade)
```

توجد العديد من محررات التعليمات البرمجية التي يمكن اختيارها من قِبل المبرمج حسب تفضيلاته، ويبقى المعيار الوحيد لاختيار المحرر هو كفاءة ذلك المحرر.

بعض الأمثلة على محررات التعليمات البرمجية:

- Sublime Text
- Atom
- Visual Studio Code
- Espresso
- Python IDLE
- Coda 2
- Notepad++
- Vim
- BBedit
- Ultraedit

ميزات وتحديات استخدام محررات التعليمات البرمجية

الميزات

- يمكنها أن تنافس محرر بيئة التطوير المتكاملة (IDE) للقيام بالمهام البرمجية الاعتيادية عند احتوائها أو دعمها بالامتدادات المناسبة لدعم لغات البرمجة المختلفة.
- أصغر حجمًا وأسرع في التحميل من بيئات IDE.
- تُسهِّل واجهاتها المبسطة التركيز على التعليمات البرمجية الخاصة بنا.

تحديات الاستخدام

- تفتقر إلى الكثير من ميزات التحرير التي لا توفرها سوى IDE مثل التحرير الذكي.
- قد يحتاج المستخدمين إلى تهيئة محرر النص Text Editor بالامتدادات المناسبة قبل الاستخدام.

بيئة التطوير المتكاملة Integrated Development Environment (IDE)

عادةً ما يتم تقديم بيئات البرمجة الحديثة مع تطبيقاتها المدمجة، حيث تتضمن عدد من أدوات تطوير البرمجيات مثل المفسر **Interpreter** لاستخدامه خلال مرحلة إنشاء البرنامج، والمترجم **Compiler** لإصدار البرنامج بصورته النهائية ونشره.

لا تقتصر بيئات البرمجة المتكاملة الحديثة على توفير مترجم للغة البرمجة فقط، بل تحتوي على جميع البرامج والأدوات اللازمة للمساعدة في الكتابة، والتنفيذ، والأهم من ذلك تشخيص البرامج وتصحيحها، ومن أهم الأداوت التي تتضمنها بيئات البرمجة المتكاملة:

- مستكشف الملفات **File Explorer**
- المحرر **Code Editor**
- المفسر **Interpreter**
- المترجم **Compiler**
- الرابط **Linker**
- المصحح **Debugger**
- مستعرض المخرجات **Output Viewer**

يجب أن تتضمن بيئة البرمجة المرئية بالضرورة محررًا مخصصًا لتسهيل إنشاء الكائنات ذات الطبيعة الرسومية كالنماذج والقوائم وصناديق الحوار، وذلك لتزويد المطور بالأدوات المناسبة لإنشاء المقاطع البرمجية المتعلقة بتلك الكائنات.

قد تحتوي بيئة **IDE** أيضًا بعض الخصائص مثل:

- التعبئة الذكية للكود البرمجي في محرر التعليمات البرمجية.
- التكامل مع مستودعات التعليمات البرمجية **(Source Code Repositories)** مع التحكم في الإصدار.
- أدوات الاختبار المتقدمة **(Testing Tools)**
- مكتبات الكود المصدري **(Source Code Libraries)**
- أدوات إنشاء الأتمتة **(Automation)** ونشرها.

يمكن الوصول لجميع هذه الخدمات من خلال واجهة مستخدم موحدة.

أمثلة على بيئات تطوير IDE

في الماضي، كانت غالبية بيئات **IDE** تدعم لغة برمجة واحدة فقط وعادة ما تم إنشاؤها من قبل شركات البرمجيات أو المؤسسات التي أنشأت تلك اللغة المحددة.

أصبحت معظم المشاريع حاليًا تدمج تقنيات ولغات برمجة مختلفة، مما توجب ظهور بيئات تطوير **IDE** يمكنها العمل ودعم مجموعة واسعة من اللغات.

على سبيل المثال، يدعم **Microsoft Visual Studio** كلًا من لغات **C** و**C++** و**C#** و**VB.Net** و**Python** و**Ruby** و**Node.js** و**JavaScript** و**HTML / CSS** وغيرها....

تشمل أدوات **IDE** الشهيرة الأخرى **NetBeans** و**Eclipse** و**Atom** و **Xcode** و**PyCharm**.

سنستخدم **Visual Studio Code** كبيئة تطوير **IDE** مع **Python**، والذي يختلف عن **Visual Studio** ويتميز بأنه مفتوح المصدر ويعمل على أنظمة تشغيل مختلفة ويمكنه دعم أي لغة برمجة من خلال الإضافات (**Extensions**).

بعض اللغات البرمجية التي يدعمها Visual Studio Code:

- **C**
- **C++**
- **C#**
- **Java**
- **PHP**
- **Python**
- **Go**
- **Ruby**
- **T-SQL**
- **JavaScript**
- **HTML/CSS**
- **TypeScript**

يتم استخدام **Xcode** لتطوير برامج التطبيقات للأجهزة المحمولة للأجهزة التي تعمل بنظام **iOS** ، أما أجهزة **Android** فيتم استخدام **Android Studio**. توجد بيئات **IDE** مثل **Xamarin** و **App Inventor** والتي تدعم النظامين معًا من خلال مجموعة متنوعة من التقنيات الأخرى، وذلك في بيئات تطوير برمجية مختلفة.

البيئات البرمجية السحابية

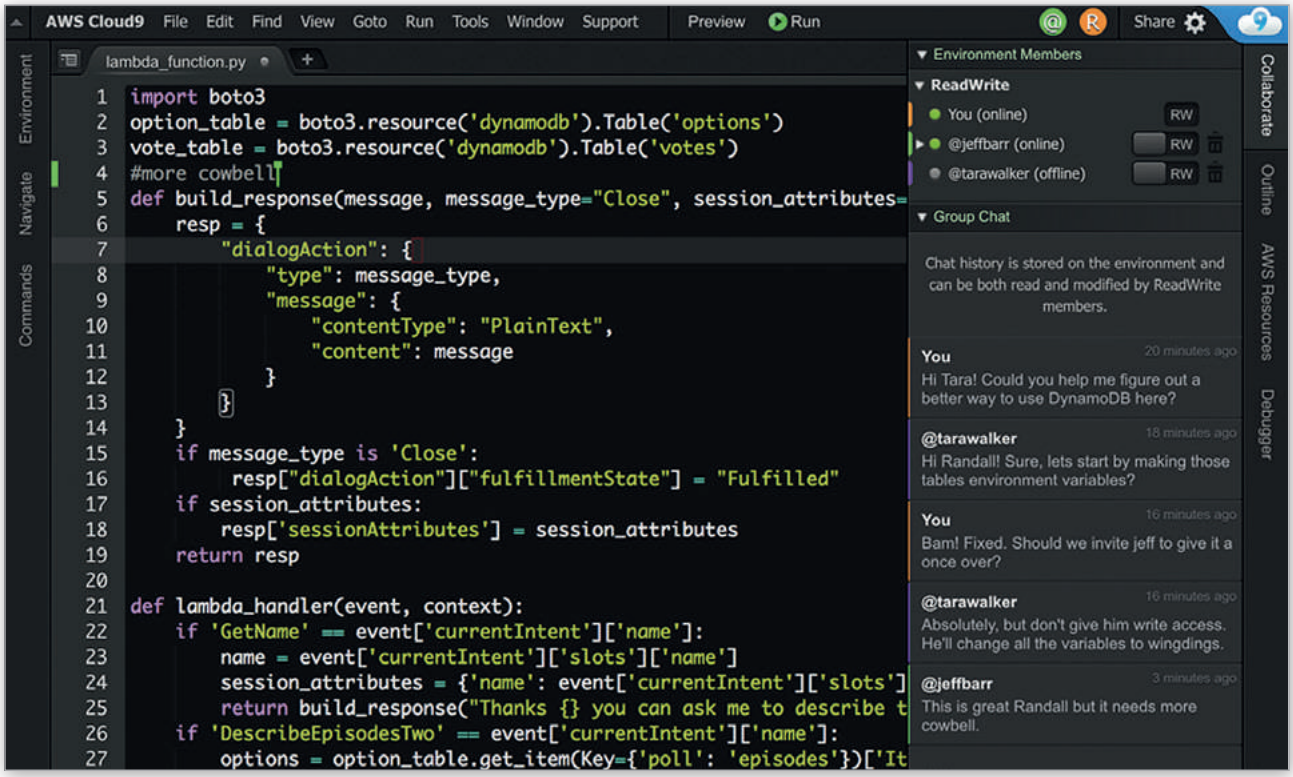
إلى جانب بيئات التطوير البرمجية التقليدية، توجد هناك بيئات تطوير سحابية مستندة على الويب، مثل **Amazon Cloud9**.

توفر البيئات البرمجية السحابية إمكانية العمل على مشروعنا من أي مكان توجد به جميع بيانات مشروعنا الخاص بتطوير البرمجيات سحابيًا.

أحد العيوب الرئيسة لهذه البيئات هو الحاجة إلى الاتصال بالإنترنت للوصول للبيانات والقيام بالعمل.

فوائد استخدام بيئات التطوير البرمجية السحابية:

- إمكانية الوصول إلى أدوات تطوير البرمجيات من أي مكان في العالم.
- إمكانية استخدام أي جهاز مع متصفح ويب.
- لا يوجد متطلبات لتنزيل وتثبيت البيئة البرمجية.
- يمكنها تسهيل عملية التعاون بين المطورين عن بعد.




```
import boto3
option_table = boto3.resource('dynamodb').Table('options')
vote_table = boto3.resource('dynamodb').Table('votes')
#more cowbell
def build_response(message, message_type="Close", session_attributes=
    resp = {
        "dialogAction": {
            "type": message_type,
            "message": {
                "contentType": "PlainText",
                "content": message
            }
        }
    }
    if message_type is 'Close':
        resp["dialogAction"]["fulfillmentState"] = "Fulfilled"
    if session_attributes:
        resp['sessionAttributes'] = session_attributes
    return resp

def lambda_handler(event, context):
    if 'GetName' == event['currentIntent']['name']:
        name = event['currentIntent']['slots']['name']
        session_attributes = {'name': event['currentIntent']['slots']
        return build_response("Thanks {} you can ask me to describe t
    if 'DescribeEpisodesTwo' == event['currentIntent']['name']:
        options = option_table.get_item(Key={'poll': 'episodes'})['It
```

ميزات وتحديات استخدام بيئات التطوير المتكاملة IDE

الميزات

- توفر أدوات تحليل وإكمال للتعليمات البرمجية بصورة ذكية لبرمجة أسرع وأقل خطأ.
- توفر أدوات تصفح واستكشاف قوية للتعليمات البرمجية وتتميز بسهولة الوصول لأي جزء من البرنامج بسهولة مهما كبر حجم المشروع.
- تقدم طرقًا متعددة لتصحيح التعليمات البرمجية وكتابتها دون مغادرة المحرر.
- تدعم العديد من لغات البرمجة بشكل أصلي (**Native**) وتوفر العديد من أدوات التنقل بين التعليمات البرمجية وتحليل التعليمات البرمجية لتسهيل العمل والإنتاجية في المشاريع الكبيرة.

تحديات الاستخدام

- تمتلئ الواجهات بالكثير من الميزات مما قد يجعل من استخدامها أمرًا معقدًا وصعبًا.
- تتطلب كمًا نوعيًا من التدريب لإتقان استخدامها.
- كثيراً ما تؤدي الوظائف الزائدة إلى بطء أدائها.

يقضي المبرمجون معظم وقت البرمجة في عمليات الاختبار والتصحيح، ولذلك يجب أن يتكامل محرر التعليمات البرمجية مع المترجم و مصحح الأخطاء، وهذه هي الميزة الأساسية في بيئة التطوير **IDE**.

الأدوات المتخصصة لبعض مراحل تطوير البرمجيات

يحتاج إنشاء الحلول البرمجية الاحترافية إلى العمل في فريق واستعمال مجموعة متنوعة من الأدوات الضرورية التي لا تقتصر على مرحلة البرمجة فقط بل تمتد لتشمل العملية ككل.

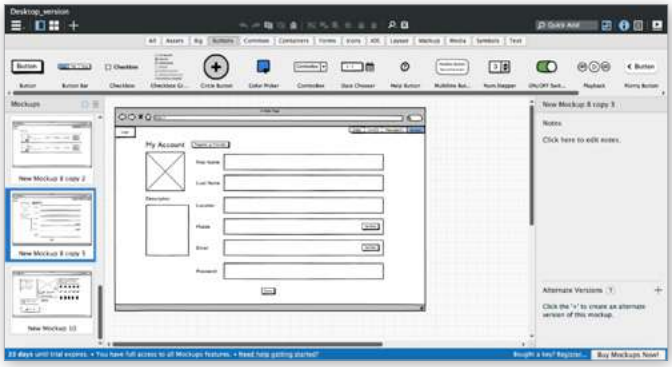
توجد العديد من الأدوات التي يمكن استخدامها أثناء دورة حياة النظام للبرنامج المنتج، وقد يكون من الصعب حصر جميع البرمجيات والأشياء الضرورية الأخرى اللازمة لتطوير برامج الأعمال.

إنشاء النموذج الأولي Prototype

عادة ما يكون النموذج الأولي للبرنامج عبارة عن مخطط هيكلي أو صورة أو مجموعة من الصور التي تعرض العناصر الوظيفية لتطبيق ما، أو قد يكون موقع ويب يستخدم لتخطيط التطبيقات أو هيكلية ووظائف الويب.

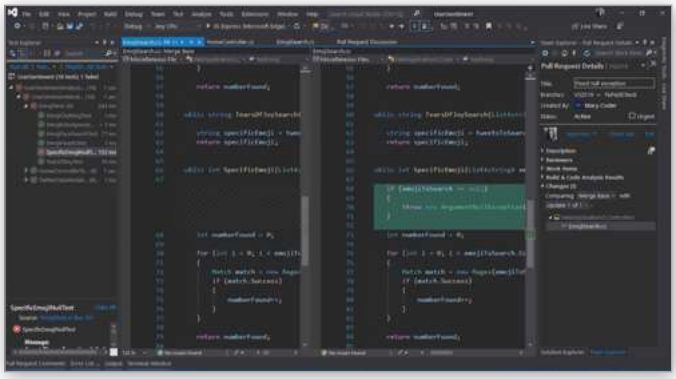
أمثلة للأدوات المستخدمة:

- Pencil
- Balsamiq Mockups
- Adobe Xd



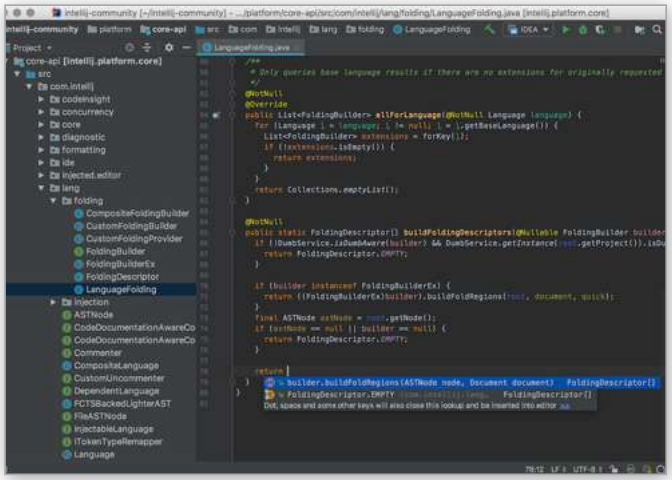
البرمجة

البرمجة هي أهم جزء من عملية تطوير البرمجيات، حيث يستخدم المطور الميزات المختلفة الموجودة في بيئة تطوير البرمجيات IDE لكتابة التعليمات البرمجية وللقيام بالتصحيحات الضرورية على البرنامج.



أمثلة للأدوات المستخدمة:

- Visual Studio Code
- Netbeans
- Visual Studio
- IntelliJ
- Xcode
- Android Studio



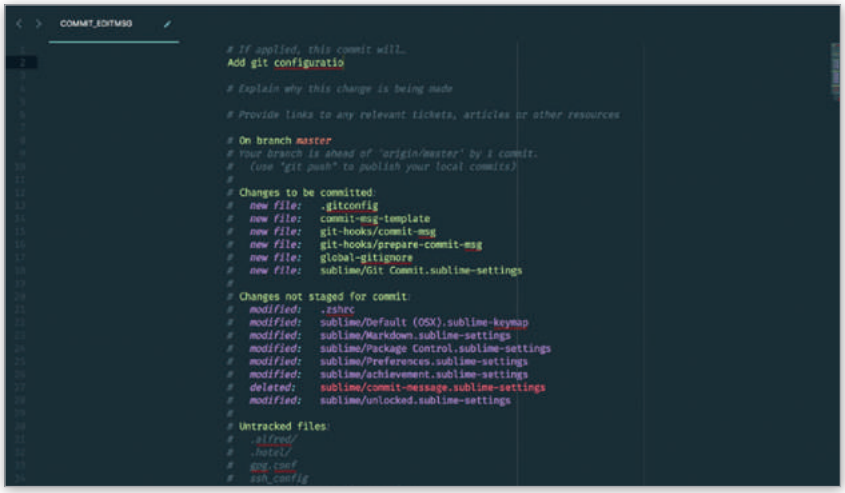
إدارة التحكم في الإصدار - الكود المصدري
(Version Control – Source Code Management)

يخضع الكود البرمجي المصدري إلى الكثير من التعديل والتغيير أثناء عملية التطوير، وقد يتطلب الأمر التراجع عن خطوات معينة في البرنامج أواستخدام كود برمجي قد تم تغييره أو حذفه. عند العمل في فريق من المبرمجين، قد يحتاج اثنان منهم إلى العمل على نفس الملفات في نفس الوقت وإجراء تغييرات على نفس الكود البرمجي.

يطلق على الأداة التي يمكننا استخدامها للتحكم في هذه العملية "إدارة التحكم في الإصدار" أو "إدارة الكود المصدري"، وتمكن هذه الأداة من التالي:

1. تمكين أعضاء الفريق المختلفين من الوصول إلى الكود المصدري في وقت واحد دون إحداث تعارض في العمل.
2. الاحتفاظ بالإصدارات السابقة من ملفات التعليمات البرمجية للرجوع إليها عند حدوث مشكلات.

يستخدم التحكم في الإصدار مستودعًا (**Repository**) لتسجيل جميع التغييرات التي تتم وينشئ نسخة عمل من ملفات الكود الخاصة بالمشروع، ويطلق على نسخة العمل أحيانًا اسم نسخة المراجعة أو الفحص (**Checkout**)، وهي النسخة الشخصية لجميع الملفات الخاصة بالمشروع، ويتم اعتماد جميع التغييرات التي تمت في المستودع عند انتهاء العمل.



أمثلة للأدوات المستخدمة:

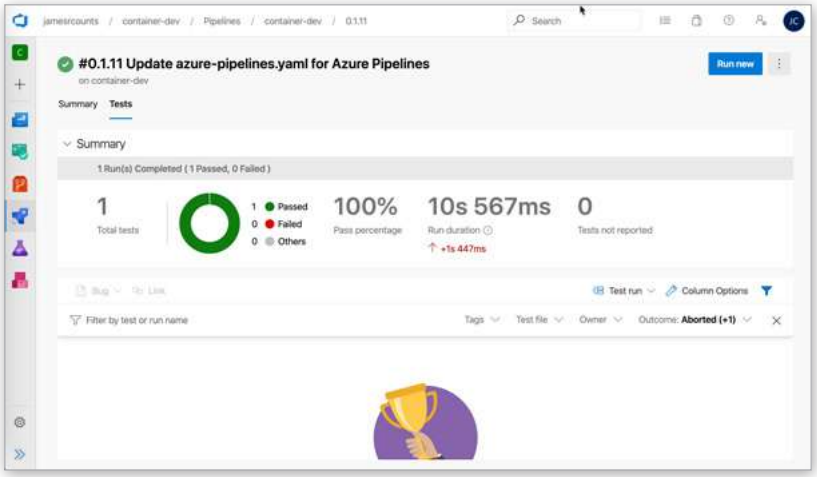
- **Git**
- **Subversion**
- **Mercurial**
- **Azure Devops**
- **Diffmerge**

توجد ميزة مفيدة للغاية في أداة التحكم في الإصدار وهي "التفرع" (**branching**)، وتعرف بأنها إمكانية نسخ جميع التعليمات البرمجية للمشروع كمشروع موازي جديد مما يتيح إمكانية اختبار أو إجراء التغييرات لإنشاء نسخة محدثة أو جديدة من التطبيق، ويمكن لاحقًا نقل أجزاء من الكود الجديد إلى المشروع الأصلي واستخدامها في التطبيق الأصلي أيضًا.

نشر الكود البرمجي Code Deployment

حتى بضع سنوات مضت، كان من السهل نشر أي تطبيق حيث أن مخرجات ترجمة البرنامج توضع على قرص لتكون جاهزة للاستخدام.

مع ظهور الإنترنت أصبح من الضروري "نشر" التطبيقات عبر الإنترنت وذلك كبرنامج قابل للتثبيت عبر متاجر التطبيقات أو بشكل مباشر كتطبيقات الويب، وظهرت بناء على ذلك برامج وأدوات خاصة لنشر الكود البرمجي على الشبكة.



أمثلة للأدوات المستخدمة:

- TeamCity
- Google Cloud Deployment Manager
- GitLab
- Jenkins
- AWS CodeDeploy
- Azure DevOps

الاختبار Testing

لا يعد الاختبار مجرد تصحيح للمقطع البرمجي، وإنما بتجاوز ذلك إلى اختبار تشغيل البرنامج وفعالية استخدامه من قبل عدد كبير من المستخدمين وكذلك اختبارات الأمان من الاختراق ... وغيرها، سنتناول موضوع الاختبار بشكل موسع في درس لاحق من هذه الوحدة.

الأدوات :

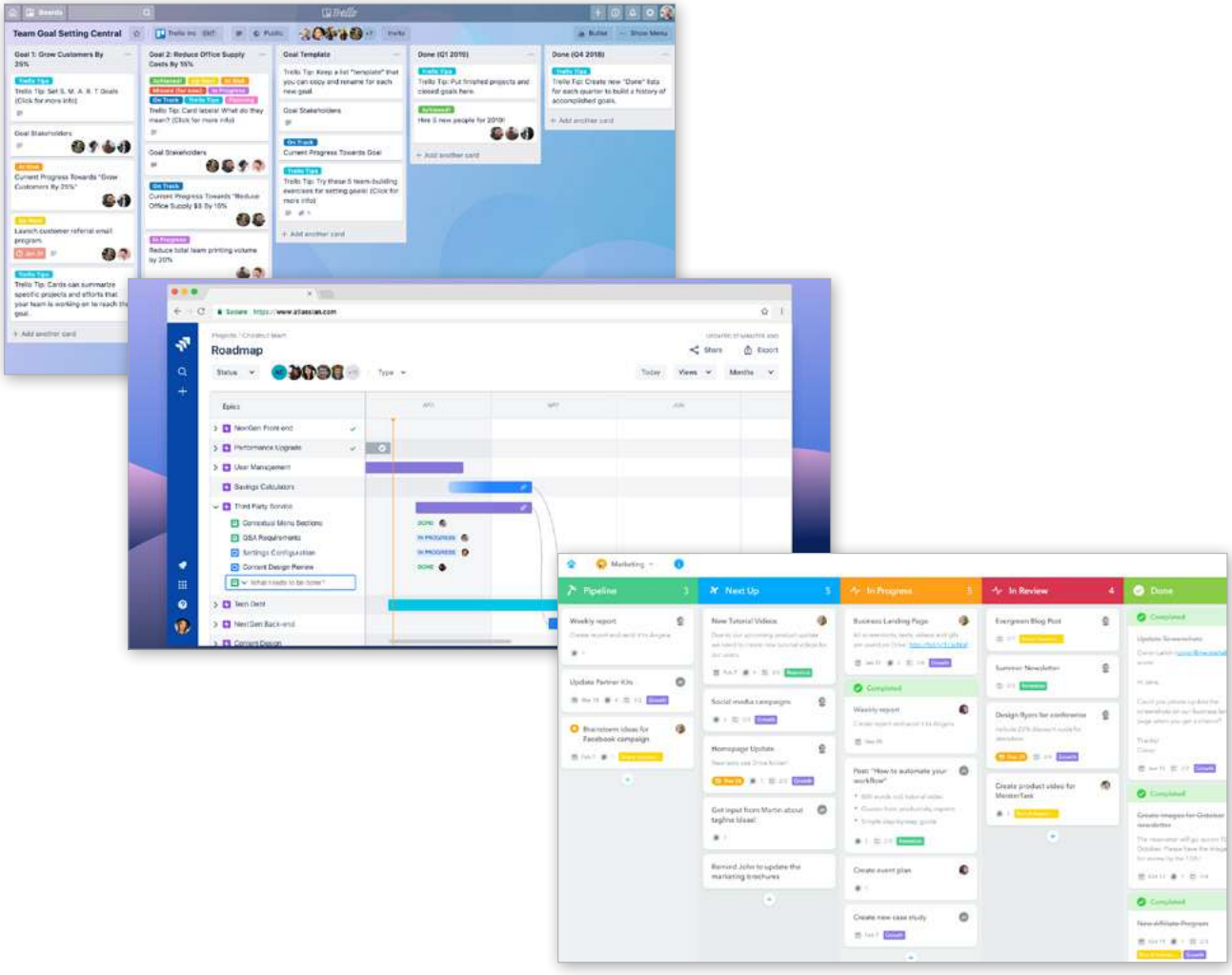
- Apache JMeter
- GhostLab
- Selenium
- Telerik Test Studio
- Azure DevOps
- Iron Wasp
- Zed Attack Proxy
- Wapiti

إدارة المشروع والتعاون وتتبع المشاكل

كما تعلمنا بالفعل فإن الحصول على منتج ناجح يتطلب التحكم الكامل في العملية برمتها ومشاركة المعرفة مع الفريق بأكمله خصوصًا عند اتساع حجم الفريق، وهنا تصبح عملية إدارة المشاريع ذات أهمية كبرى.

أمثلة للأدوات المستخدمة:

- **Microsoft Teams** للتعاون والتواصل.
- **Scrum Trello** للتخطيط بالمنهجية الرشيقة **Agile Planning** والتتبع.
- **Jira** لتتبع مسائل معينة وإدارة المشاريع.
- **MeisterTask** لإدارة المهام.
- **Slack** للتعاون والتواصل.
- **Basecamp** لإدارة المشاريع والتواصل مع العملاء.



استخدام أدوات التطوير لتقديم حلول مختلفة

تعتمد فرق التطوير على الأدوات التي ذكرناها سابقًا لإنتاج مجموعة واسعة من حلول تكنولوجيا المعلومات التي نستخدم العديد منها اليوم لبناء التطبيقات بأنواعها المختلفة، مثل:

- تطبيقات الويب.
- تطبيقات الهواتف الذكية.
- التطبيقات العامة.
- الأنظمة المدمجة.

بناء تطبيق ويب Building a web application

تطبيق الويب هو برنامج تفاعلي يتم إنشاؤه باستخدام تقنيات الويب **HTML** و**CSS** و **JS** ليقوم بتخزين البيانات على خوادم قواعد البيانات، ويتم استخدام هذا التطبيق من قِبل المستخدمين الذين يؤدون المهام عبر الإنترنت.

مراحل بناء تطبيق الويب

1. مرحلة التفكير Ideation Stage

قبل إنشاء تطبيق ويب، يجب أن نضع الأهداف الخاصة بالتطبيق والفكرة الرئيسة له.

2. تحليل السوق Market Research

يجب علينا القيام بما يسمى تحليل السوق لمعرفة:

- ما إذا كان هناك لدى المستهلك المستهدف حاجة لهذا المنتج أو الخدمة.
- وجود منتج أو خدمة مماثلة.

3. تحديد وظائف تطبيق الويب Web Application Functionality

يجب علينا تحديد الوظائف التي تقدم الحلول لمشاكل السوق المستهدف.

4. التخطيط الهيكلي / ونمذجة تطبيق الويب الخاص بنا Wireframing/Prototyping

تختص عملية إنشاء المخطط الهيكلي **Wireframing** بتصميم مخطط لتطبيق الويب الخاص بنا، أما النمذجة الأولية **Prototyping** فتأخذ المخطط الهيكلي خطوة إلى الأمام بإضافة التفاعلية لاختبار وظائف التطبيق.

5. طلب المصادقة Seek Validation

في هذه المرحلة يتم جمع آراء وردود الفعل البناءة من الأطراف ذات العلاقة، والمستخدمين المحتملين فيما يتعلق بالتصميم.

6. التخطيط وإنشاء قاعدة البيانات Architect And Build Database

يتم في هذه المرحلة تحديد البيانات التي نحتاجها كمبرمجين وتلك التي يحتاجها المستخدمون، كما يتم تحديد الأداة المستخدمة في بناء قاعدة البيانات المطلوبة لتطبيق الويب.

هناك العديد من أدوات تصميم قواعد البيانات التي تستخدم لأغراض مختلفة، ولكن طبيعة البرنامج و الكيفية المقترحة لنشر حلنا البرمجي ستحدد خياراتنا، ومن أمثلة الأدوات المستخدمة في تصميم وبناء قواعد البيانات:

- MySQL
- MongoDB
- Amazon Dynamo-DB
- SQL Server
- Firebase
- Azure SQL

7. بناء الواجهة الأمامية Building the Frontend

الواجهة الأمامية هي العنصر المرئي لتطبيق الويب الخاص بنا، وهي تمثل واجهة أو وسيط بين المستخدم والنظام. وتمثل هذه الواجهة ما نراه ونتفاعل معه.

ومن الأمثلة على الأدوات المستخدمة لبناء واجهة مستخدم محسنة للويب:

- jQuery
- Vue.js
- Django
- React.js
- Angular

8. بناء الواجهة الخلفية الموقع Building the Backend

تستخدم الواجهة الخلفية لإدارة البيانات في البرنامج حيث تشير إلى قواعد البيانات والخوادم، وكذلك الجزئيات الغير ظاهرة للمستخدم داخل تطبيق الويب.

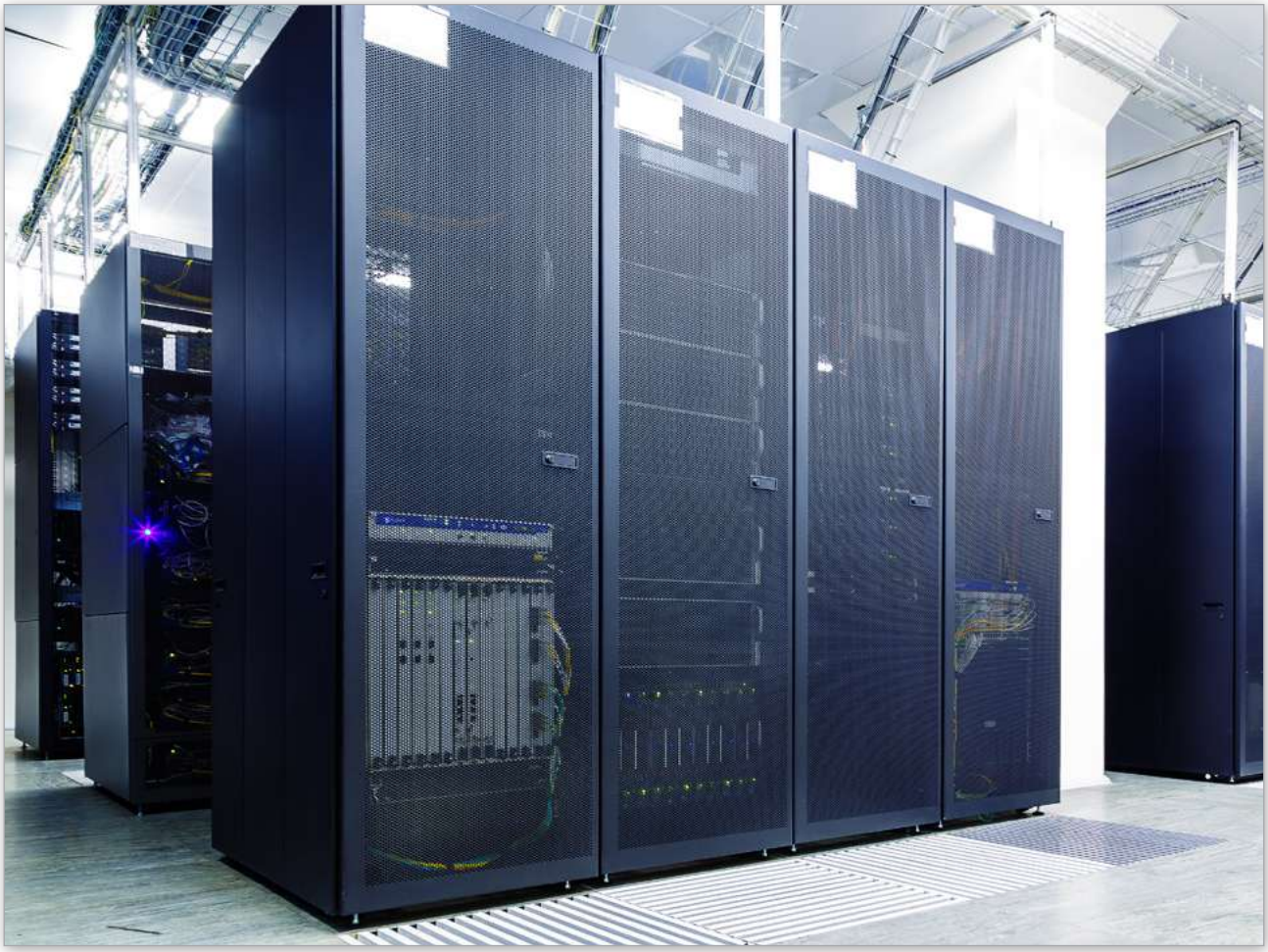
عند ازدياد درجة تعقيد التطبيق ليصبح نظام برمجيات، يكون الكود البرمجي هو أحد الاعتبارات وكذلك إعداد قاعدة البيانات والشبكات والتكامل بين النظم الفرعية المختلفة، كما وتصبح اعتبارات الأمن والأداء ذات أهمية خاصة، ومن أمثلة الأدوات المستخدمة في بناء الواجهة الخلفية:

- Node.js
- PHP
- Ruby
- ASP.NET
- Java
- Python

9. استضافة تطبيق الويب الخاص بنا Host Web Application

لتشغيل تطبيق الويب الخاص بنا على خادم معين، نحتاج إلى مزود استضافة الويب. قد تكون خدمة الاستضافة بسيطة ورخيصة أو قد تكون خدمة حوسبة سحابية كبيرة تسمح لبنيتنا السحابية بالنمو مع ازدياد عدد مستخدمي التطبيق ونمو احتياجاتنا.

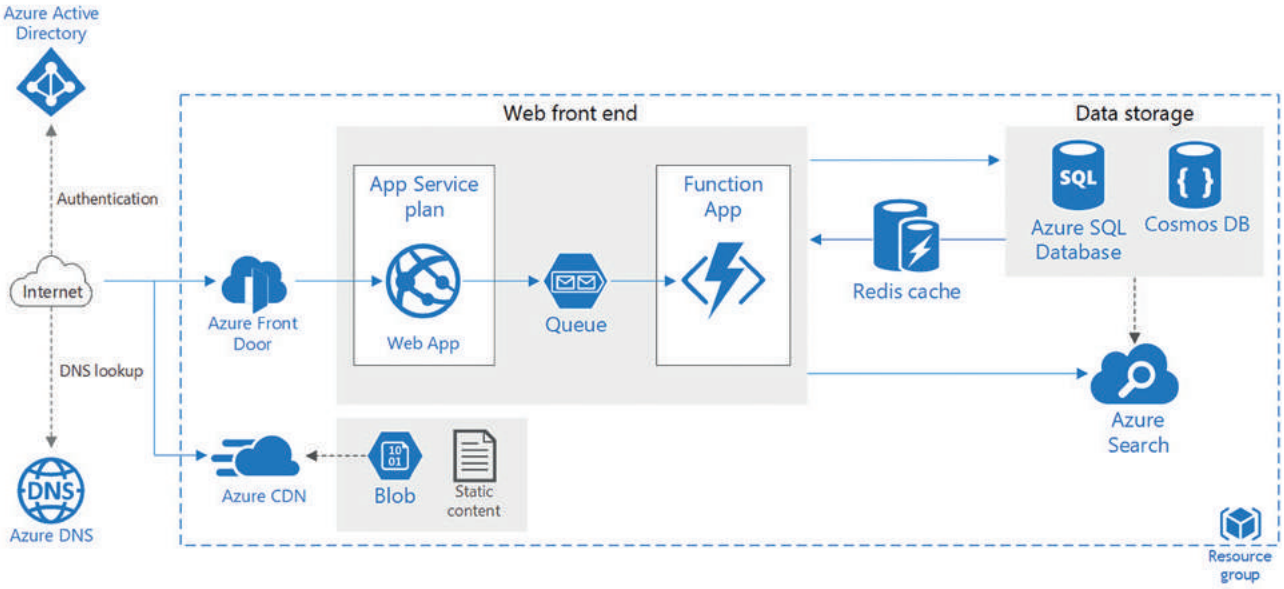
مزودي الاستضافة Hosting Providers	• Bluehost • HostGator • GoDaddy • Rackspace
مزودي الخدمات السحابية Cloud Service Providers	• IBM Cloud • Microsoft Azure • Amazon Web Services • Google Cloud Platform • Alibaba Cloud



هيكلية التطبيقات السحابية The Cloud-ready Application Architecture

يفضل تطوير ونشر التطبيقات السحابية كمجموعة من الخدمات السحابية. تتضمن العملية بناء هيكليات البيانات ثم إنشاء الخدمات، ويتم دمجهما لتشكل نظامًا متكاملاً.

يظهر المخطط التالي كيفية بناء تطبيق ويب يتمتع بقابلية التوسع والأداء، وذلك باستخدام خدمات **Microsoft Azure**، ويجدر الإشارة إلى أن المفهوم نفسه ينطبق على جميع مزودي الحوسبة السحابية.



أهم النقاط التي ينبغي أخذها بعين الاعتبار عند استخدام هيكلية التطبيقات السحابية:

- تصميم التطبيق كمجموعة من الخدمات.
- الفصل بين البيانات ومعايير الأمان والأداء.
- الأخذ بعين الاعتبار متطلبات الاتصال من خلال الشبكات بين مكونات التطبيق.
- النمذجة والتصميم بقابلية التوسع.
- جعل أمن النظام جزءًا أساسيًا داخل التطبيق وليس أمرًا يتم التخطيط له لاحقًا.
- اختيار مراكز استضافة البيانات (**Data Centers**) الأقرب للمستخدمين.

تشبه خطوات إنشاء تطبيق للهاتف المحمول تلك الخاصة بتطبيق الويب ولكن مع وجود بعض الاعتبارات الخاصة، فتطبيق الهاتف المحمول يتم استخدامه على الهاتف والذي يمتلك بالعادة شاشة صغيرة، وكما يوحي الاسم، فإن المستخدم سيستخدم التطبيق "أثناء التنقل"، مما يعني أخذ بعض الاعتبارات كملاءمة واجهة المستخدم لحجم الشاشة وإظهار المعلومات ذات الأهمية بطريقة واضحة وبسيطة، وأيضا ملاحظة أن اختلاف الأجهزة يؤدي إلى ضرورة وجود متطلبات التطبيقات المتجاوبة (**Responsive Applications**).

تدعم كل واحدة من منصتي الجوال الرئيستين **iOS** و**Android** مجموعة مختلفة ولكن متشابهة من التقنيات. على سبيل المثال، يقترح **iOS** كلًا من **Xcode** و**Swift** لتطوير البرمجيات، بينما يقترح **Android** كلًا من **Android Studio** و**Java**.

تتيح بيئات العمل هذه إنشاء تطبيق نهائي جاهزًا للنشر في متجر التطبيقات المحدد في بيئة العمل تلك فقط، ولذلك أصبحت هناك بيئات العمل التي تحاول حل هذه المشكلة من خلال دعم نشر التطبيق في متاجر متعددة.

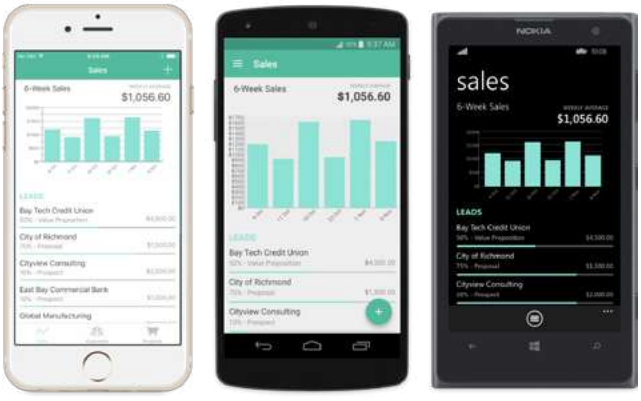
باستخدام الأدوات التالية، يمكن تطوير تطبيق واحد يعمل على بيئات برمجية مختلفة:

- **Xamarin**
- **Adobe PhoneGap / Apache Cordova**
- **Reach Native**
- **Flutter**
- **Ionic**

يُعدُّ اختبار تطبيقات الهاتف المحمول تحديًا كبيرًا، فمن الصعب أن يمتلك المبرمج أو حتى شركة تطوير البرمجيات جميع الأجهزة المحمولة المتوفرة في السوق وذلك للقيام بالاختبار، لهذا توجد بعض الخدمات عبر الإنترنت التي تقدم محاكاة لمجموعة واسعة من الأجهزة المحمولة حيث يمكننا محاكاة اختبار تطبيقنا للتوافق على الأجهزة المختلفة.

ومن الأمثلة على الأدوات التي تمكن من اختبار التطبيق:

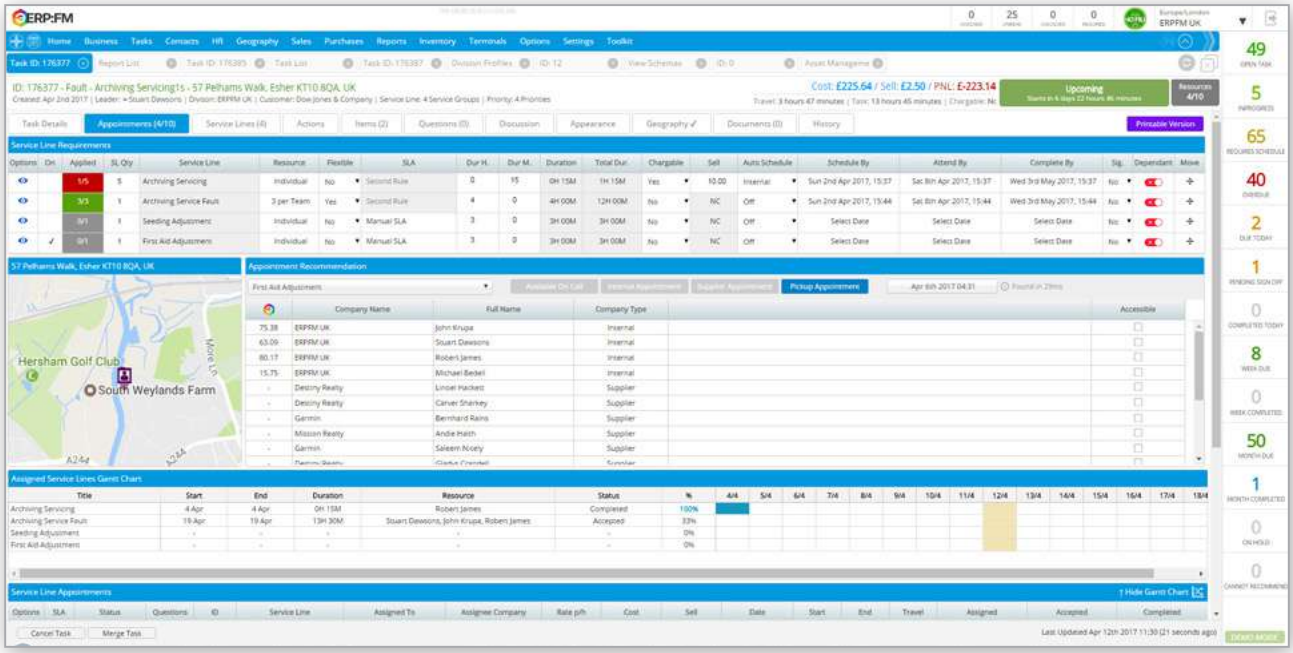
- **Xamarin Test Cloud**
- **BrowserStack**
- **Firebase Test Lab**



بناء تطبيق للأغراض العامة Building a general-purpose application

تطلق تسمية برنامج الأغراض العامة على نوع من التطبيقات الذي يمكن استخدامه لأداء مهام عديدة كما هو الحال في البرامج المكتبية التقليدية مثل معالجات النصوص وبرامج التصميم الرسومي وتطبيقات الأعمال **ERP (Enterprise Resource Planning)** أو **CRM (Customer Relationship Management)**.

على الرغم من تركيز تقنيات تطوير البرامج الجديدة على الويب وتطبيقات الهاتف المحمول، إلا أن هذه التطبيقات التقليدية ما زالت تحتفظ بأهميتها. يعتمد تطوير مثل هذه التطبيقات على مكتبات الأكواد البرمجية الجاهزة والقابلة لإعادة الاستخدام، خاصة مكونات واجهة المستخدم وأدوات التقارير.



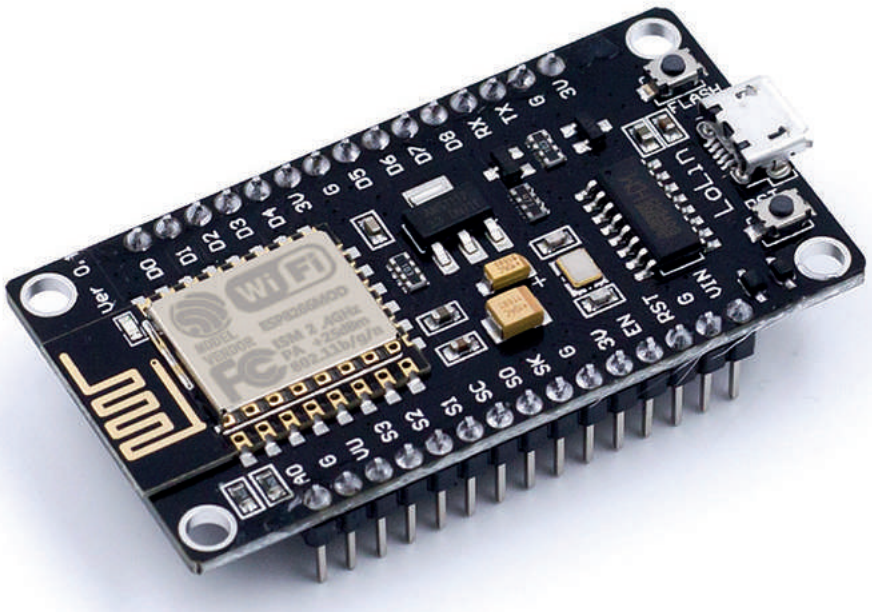
بناء تطبيق نظام مدمج Building an Embedded System Application

إن النظام المدمج هو عبارة عن حاسوب خاص مزود بنظام تشغيل آني (**Real-time**) وغالبًا ما يكون دون واجهة مستخدم. تتعامل البرامج الموجودة على النظام المدمج مع أجهزة استشعار ومشغلات وآليات تبادل البيانات سلكيًا أو لاسلكيًا. يجب أن تكون هذه البرامج موثوقة وآمنة وسريعة. تتطلب هذه التطبيقات أنظمة التشغيل ذات الطبيعة الآنية مثل **RTLinux** و**Windows 10 IoT** و **QNX** ولغات برمجة تم تحسينها لمعالجة البيانات، كما تحتاج إلى الاتصال بالشبكات.

أمثلة على الأنظمة المدمجة إشارات المرور، إنذارات الحريق وأنظمة أمن المنزل.

يمكن برمجة الأنظمة المدمجة من خلال استخدام لغات البرمجة التالية:

- لغة التجميع **Assembly** لغة صعبة ولا تصلح للاستخدام العملي.
- لغة **C**.
- لغات كائنية التوجه (**Object Oriented**) مثل **C#** و**C++** و**Java**.



1

اكتب أربعة من تصنيفات أدوات تطوير البرمجيات؟

2

أكتب ثلاثة من الأمثلة على محررات التعليمات البرمجية.

3 اختر الإجابة الصحيحة:

1. منشئات المشروع Project Builders	تتأكد من أن جميع الملفات التي حددتها سيتم ترجمتها وربطها في برنامج نهائي واحد. ○
	تترجم برنامجك إلى كود قابل للتنفيذ على الجهاز. ○
	ضرورية في حالة إنشاء البرامج المتخصصة بما يتعلق با لشبكا ت . ○

2. أدوات إدارة التعليمات البرمجية:	تساعدك في تصحيح أخطاء البرنامج. ○
	تعمل مع قواعد البيانات وتقوم بتحليل أداء استعلامات بعض قواعد البيانات. ○
	تضمن الحفاظ على الملفات البرمجية من الاستبدال عن طريق الخطأ عندما يعمل العديد من المبرمجين على البرنامج بشكل متزامن ○

3. المحللات Profilers	تزود أو تدعم مهمة معينة في أي حالة من دورة التطوير أو البرمجة. ○
	تمنحنا في العادة فكرة جيدة عن تعامل وحاجات البرنامج من وقت المعالج وموارد الذاكرة أثناء التشغيل. ○
	هي حواسيب خاصة مع أنظمة تشغيل آنية (Real Time) وعادة ما تكون دون واجهة مستخدم. ○

4

أكمل الجمل التالية بالمفردات المناسبة:

تطبيقات الويب	التحكم في الإصدار	النموذج الأولي للبرنامج	معالجات النصوص
الأغراض العامة	محررات التعليمات البرمجية		IDE

1. _____ تساعدك في الكتابة والقيام بالتغييرات على الكود البرمجي.
2. لا يمكن استخدام _____ في البرمجة لوجود تنسيقات خاصة بالنصوص أو المستندات المعقدة.
3. يوجد في _____ محرر نصوص، ومترجم، ورابط ومصحح للأخطاء.
4. _____ عبارة عن مخطط هيكلي ويستخدم في تخطيط التطبيقات وبنية مواقع الويب والدوال.
5. تضمن أدوات _____ تكامل العمل بشكل متزامن من مختلف أعضاء الفريق.
6. _____ برنامج تفاعلي يقوم بتخزين البيانات على خوادم قواعد البيانات ويستخدمه عدد من المستخدمين الذين يؤدون المهام عبر الإنترنت.
7. برامج _____ تستخدم لأداء نطاق واسع من المهام المتعددة.

5

عدِّد الخطوات الأساسية لبناء تطبيق ويب.

6

أكتب ثلاثة من الميزات الأساسية الشائعة لـ IDE؟

7

طابق ما يلي:

1. محرر النصوص:
2. برنامج التحكم في الإصدار:
3. IDE:
4. الواجهة الأمامية frontend:
5. الواجهة الخلفية backend:

هي العناصر المرئية لتطبيق ويب، وهي العنصر الوسيط بين المستخدم والنظام.

يمكن من الاحتفاظ بالإصدارات السابقة من ملفات التعليمات البرمجية للرجوع إليها عند حدوث مشكلات.

هو نوع من البرامج المستخدمة لتعديل ملفات النصوص.

يدير بياناتك وقواعد البيانات والخوادم وكل المكونات التي لا يمكن للمستخدم رؤيتها داخل تطبيق الويب.

يحتوي على جميع البرامج والأدوات اللازمة للمساعدة في كتابة البرامج وتنفيذها، والأهم من ذلك تشخيصها وإصلاحها.

8

ما هي أهم النقاط التي ينبغي أخذها بعين الاعتبار عند استخدام هيكلية التطبيقات السحابية؟

9

ما المقصود ببرمجيات الأغراض العامة؟ أعطِ بعض الأمثلة؟

الدرس الثالث

البرمجة التركيبية
Modular Programming

قام الطلبة بدراسة أساسيات حل المشكلات في الصفوف السابقة. سنتعرف في هذا الدرس على البرمجة التركيبية واستراتيجية حل المشاكل المعقدة وكيفية تطبيقها على مشروع.

تقنيات تصميم البرامج Software Design Techniques

استمرت الجهود منذ بداية ظهور أجهزة الحاسوب لتطوير الأساليب والخطط لضمان إنشاء برامج بشكل مبسط وبمظهر جيد، ولكي يتم كتابتها وكذلك صيانتها بسهولة.وتختلف التقنيات حسب طبيعة المشكلة ومدى تعقيدها، ومن التقنيات المستخدمة لحل المشكلات المعقدة هي تقنية التصميم الهرمي.

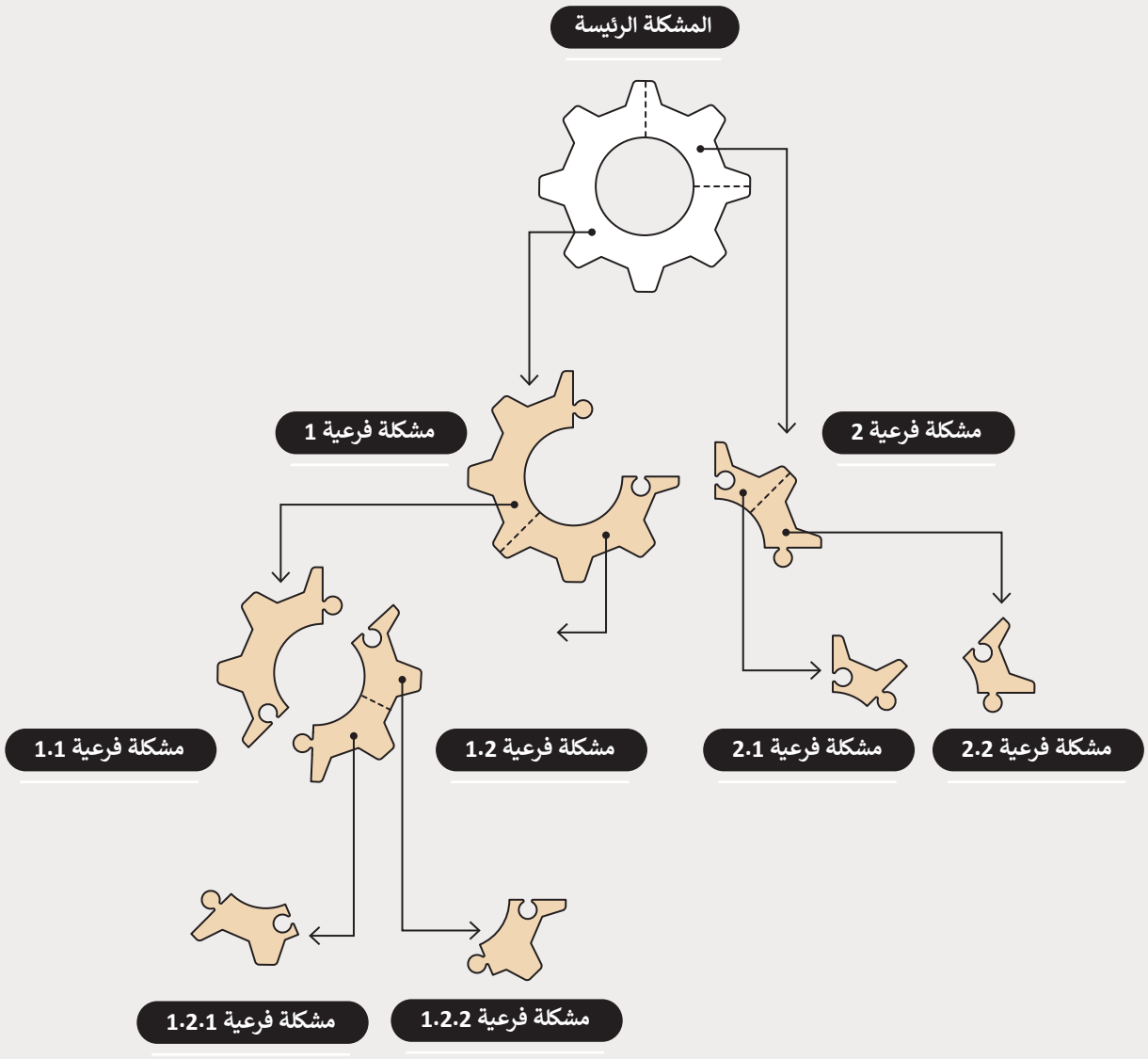
تقنية التصميم الهرمي Hierarchical design technique

تعتبر تقنية التصميم الهرمي أفضل طريقة للتعامل مع المشكلات المعقدة وكتابة البرامج ذات العلاقة ويطلق عليها غالبا التصميم من أعلى إلى أسفل **Top-down**.

تستخدم هذه التقنية استراتيجية تعتمد على تقسيم المشكلة إلى مشاكل فرعية بشكل مستمر وكلما تطلبت الحاجة.

الغرض من تقنية التصميم الهرمي هو تقسيم المشكلة الرئيسة إلى سلسلة من المشاكل الفرعية الأبسط والتي يسهل حلها مما يؤدي في النهاية إلى حل المشكلة الأصلية.

تعتبر المشاكل الفرعية الفردية بسيطة للغاية بحيث يمكن تصميم وكتابة الخوارزميات وأجزاء البرنامج المقابلة لها واللازمة لحلها بسهولة. يتم تبسيط الخوارزمية النهائية للمشكلة إلى العديد من الخوارزميات الفردية المبسطة، وكذلك يتم تبسيط البرنامج النهائي إلى العديد من البرامج الفرعية البسيطة.



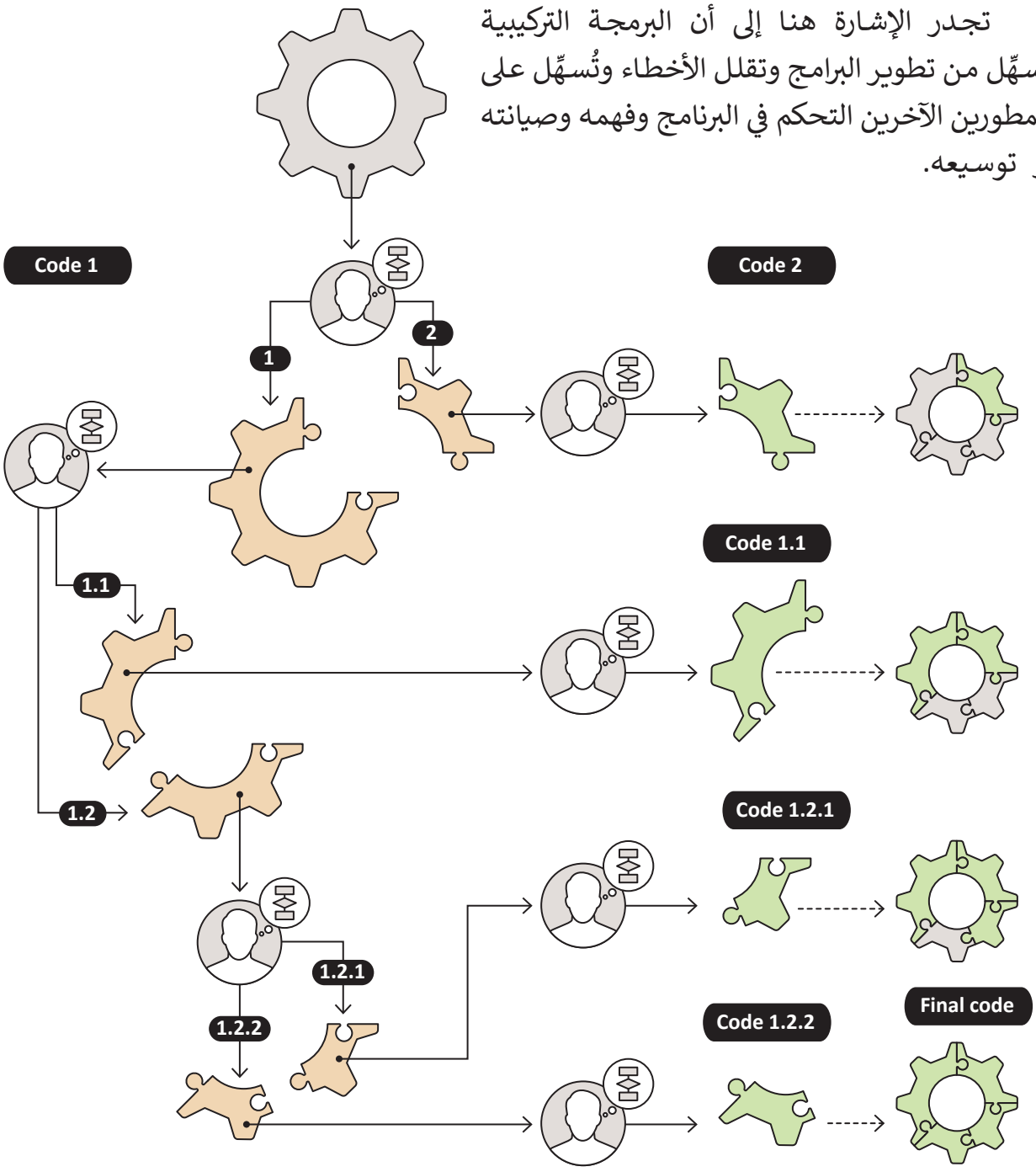
تستمر هذه العملية في جميع المراحل (المستويات) المطلوبة لتقسيم كل مشكلة فرعية إلى مشاكل فرعية أصغر يمكن حلها ببساطة.

استدعت طبيعة التصميم الهرمي، التي تعتمد على تقسيم المشكلة إلى مشكلات فرعية، ظهور نوع جديد من البرمجة يساعد في ترجمة أقسام المشكلة الفرعية إلى مقاطع برمجية، وأطلق على هذا النوع اسم "البرمجة التركيبية".

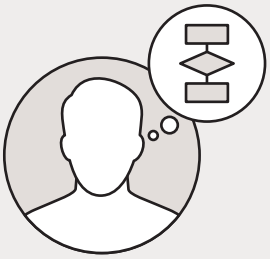
البرمجة التركيبية Modular Programming

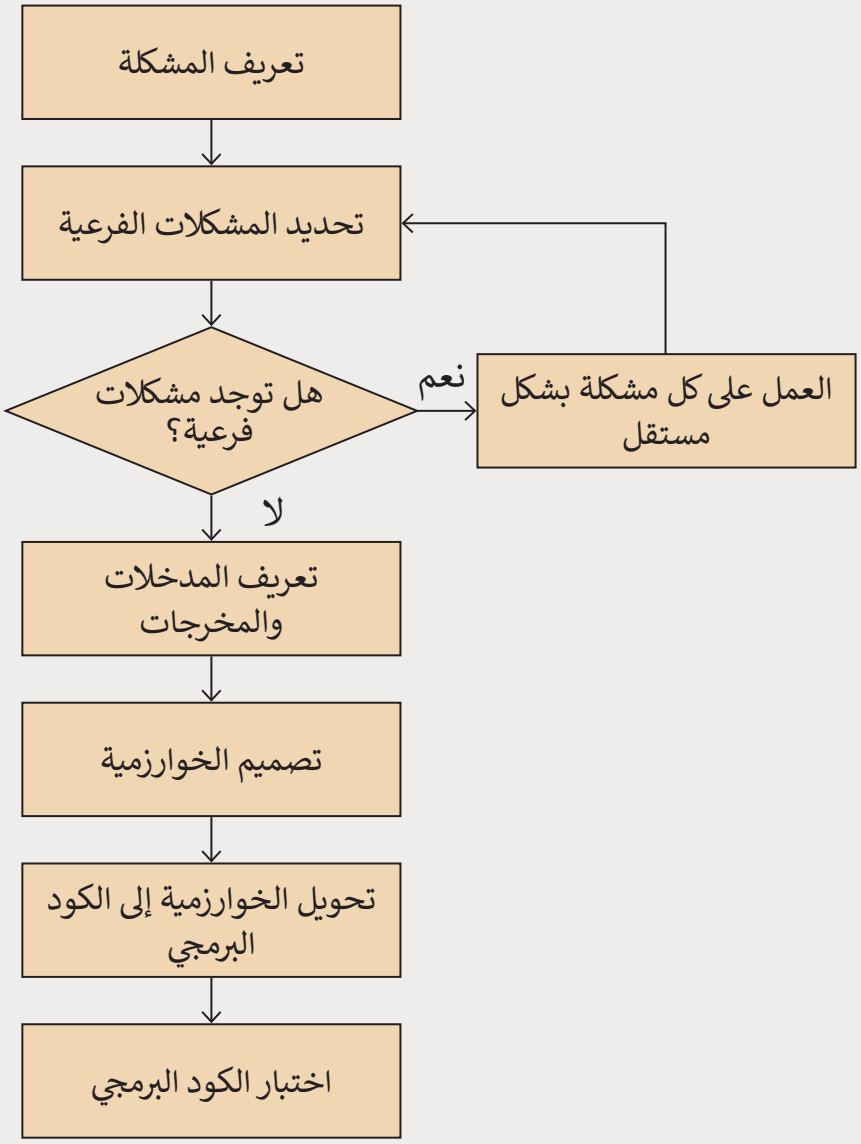
تفصل البرمجة التركيبية وظائف البرنامج إلى وحدات بسيطة ومستقلة، وبعد تحليل المشكلة إلى مشاكل فرعية، يصبح لكل مشكلة فرعية جزء مستقل يتم كتابته بشكل منفصل عن بقية أقسام البرنامج.

تجدر الإشارة هنا إلى أن البرمجة التركيبية تُسهِّل من تطوير البرامج وتقلل الأخطاء وتُسهِّل على المطورين الآخرين التحكم في البرنامج وفهمه وصيانته أو توسيعه.



يتم استخدام عملية التحليل المنطقي التالية عند تطبيق التصميم الهرمي ونهج البرمجة التركيبية لإنشاء برنامج.



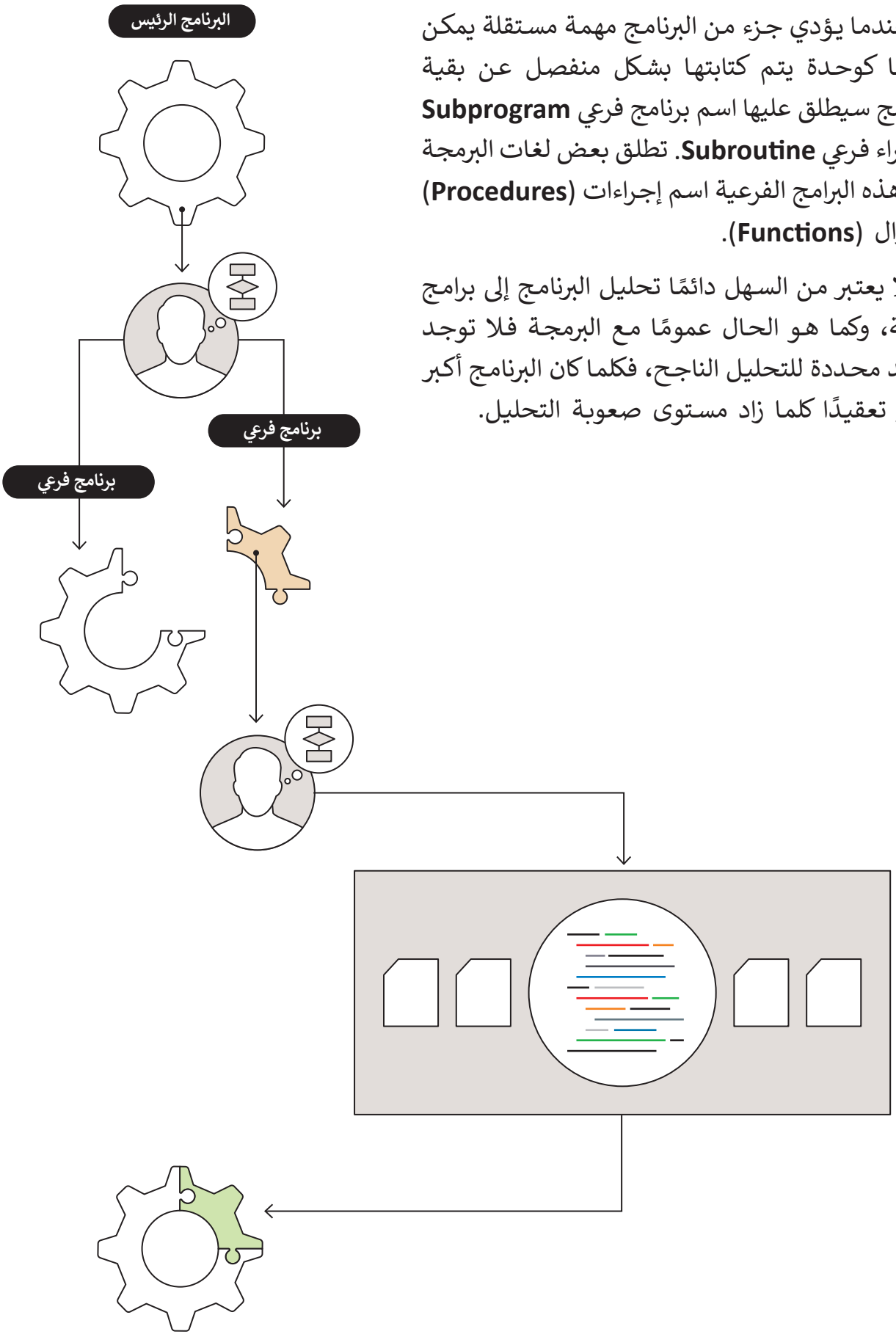


إن تحليل المشكلة هو أحد المفاهيم الرئيسة للتفكير المنطقي، لذلك فإن إنشاء الكود البرمجي النهائي يتم من خلال عملية تقسيم كل مشكلة إلى مشكلاتٍ أصغر. إذا كان من الممكن وصف مشكلة فرعية بواسطة خوارزمية إذن سيكون من السهل كتابة الكود البرمجي الخاص بها، وستمثل جزءًا مستقلاً من الحل (برنامجًا فرعيًا) ويمكننا أيضًا اختباره لتحديد الأخطاء. تجتمع جميع الأجزاء المستقلة معًا لتكون الحل النهائي للمشكلة.

ما هو البرنامج الفرعي Subprogram؟

عندما يؤدي جزء من البرنامج مهمة مستقلة يمكن حزمها كوحدة يتم كتابتها بشكل منفصل عن بقية البرنامج سيطلق عليها اسم برنامج فرعي **Subprogram** أو إجراء فرعي **Subroutine**. تطلق بعض لغات البرمجة على هذه البرامج الفرعية اسم إجراءات **(Procedures)** أو دوال **(Functions)**.

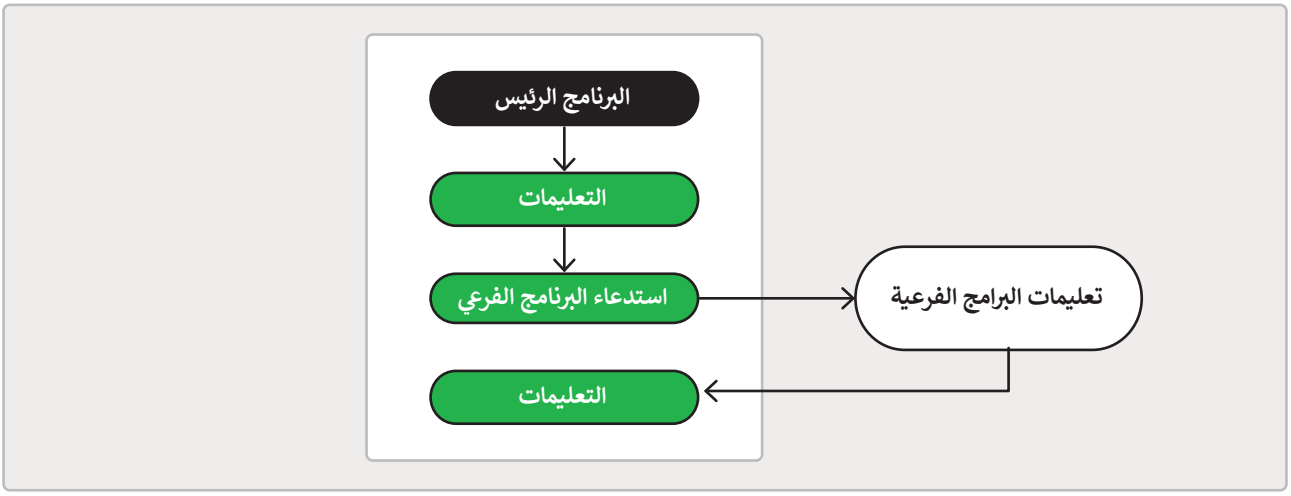
لا يعتبر من السهل دائمًا تحليل البرنامج إلى برامج فرعية، وكما هو الحال عمومًا مع البرمجة فلا توجد قواعد محددة للتحليل الناجح، فكلما كان البرنامج أكبر وأكثر تعقيدًا كلما زاد مستوى صعوبة التحليل.



خصائص البرامج الفرعية:

1. كل برنامج فرعي يمكن أن يتعامل مع البيانات كمدخلات أو مخرجات.

في الواقع، كل برنامج فرعي يبدأ تنفيذه عندما "يتم استدعاؤه" من قِبل البرنامج الرئيس، وينفذ تعليمات معينة وينتهي بالخروج في النهاية.



2. كل برنامج فرعي يجب أن يكون مستقلًا عن البرامج الأخرى.

يقصد بهذا أن كل برنامج فرعي يمكن تصميمه وتطويره وصيانته بشكل مستقل دون التأثير على برنامج فرعي آخر، وهكذا يمكن أيضًا اختباره بشكل صحيح من الناحية العملية، وبالطبع من الصعب تحقيق الاستقلال المطلق.

3. يجب ألا يكون البرنامج الفرعي كبيرًا جدًا.

يجب أن يكون من السهل فهم كل برنامج فرعي يتم إنشاؤه من أجل التحكم فيه، وبشكل عام فإن كل برنامج فرعي يجب أن يؤدي وظيفة خاصة واحدة فقط. إن أي برنامج فرعي يؤدي المزيد من الوظائف يفضل أن يقسم إلى برامج أصغر.

ميزات البرمجة التركيبية

يوفر استخدام البرمجة التركيبية المميزات التالية:

1. تسهيل إنشاء خوارزمية البرنامج وبرمجته

تسمح البرمجة التركيبية بفحص المشكلات البسيطة وحلها دون الحاجة للتعامل مع المشكلة الكلية، بل يتم التدرج في حل المشكلات الفرعية وإنشاء البرامج الفرعية المقابلة وصولًا إلى حل المشكلة العامة ككل.

2. تسهيل عملية فهم وتصحيح البرنامج

يسمح تقسيم البرنامج إلى أقسام مستقلة أصغر للمبرمج بإصلاح جزء معين أو إحداث تغييرات ضرورية بسرعة دون أن تؤثر هذه التغييرات على بقية البرنامج، كما أنه يجعل من السهل على المبرمجين والمطورين الآخرين قراءة وفهم كيفية عمل البرنامج.

3. تقلل من الوقت والجهد في كتابة البرنامج

يمكن استخدام نفس الدالة أو البرنامج الفرعي في أجزاء مختلفة من البرنامج الرئيس، وذلك يقلل من حجم البرنامج ومن الوقت المطلوب لكتابته، ويحُدُّ من احتمالية الخطأ.

4. توسيع قدرات لغات البرمجة

تؤدي كتابة العديد من البرامج الفرعية وإنشاء مكتبات معها إلى توسيع لغة البرمجة نفسها، ولذلك فإن المبرمج المحترف يمتلك العديد من البرامج الفرعية التي يستخدمها بانتظام والتي تم تجربتها واختبارها مسبقًا.

يمكن استخدام برنامج فرعي تم كتابته لبرنامج معين في برامج أخرى بسهولة. إن هذه البرامج الفرعية تشبه الدوال الضمنية التي توفرها لغة البرمجة، ولذلك إذا كنت بحاجة إلى استخدام دوال برمجية لا تدعمها لغة البرمجة بشكل مباشر فيمكنك كتابة البرنامج الفرعي المقابل وإعادة استخدامه لاحقًا.

المعاملات Parameters

يتم استدعاء كل برنامج فرعي ليتم تنشيطه بواسطة برنامج فرعي آخر، أو بواسطة الكود البرمجي المصدري المسمى بالبرنامج الرئيس.

البرنامج الفرعي هو جزء مستقل بذاته ولكنه يتواصل في أغلب الأحيان مع باقي البرنامج.

في العادة يتلقى البرنامج الفرعي القيم (المدخلات) من البرنامج الذي يستدعيه، وبعد التنفيذ يتم إنتاج قيم جديدة تسمى (المخرجات)، يطلق على هذه القيم التي تتنقل بين البرامج الفرعية اسم المعاملات **Parameters**.

تتشابه المعاملات مع المتغيرات الخاصة بالبرنامج ولكنها تختلف في استخداماتها بتمرير القيم للبرامج الفرعية.

الدوال في البايثون Function in Python

لنتذكر ما تعلمناه في المراحل السابقة عن الدوال في **Python**، فالدالة هي مجموعة من الأوامر المنظمة والقابلة لإعادة الاستخدام التي يتم استخدامها لأداء وظيفة معينة. تتكون الدالة من ثلاثة أجزاء: الاسم، والمعاملات، ونص الدالة.

ولتعريف دالة نستخدم الكلمة المفتاحية **def** متبوعة باسم الدالة ثم قوسين يحتويان على المعاملات **Parameters** بشكل اختياري ثم نقطتين (:).

ولكي نستخدم أحد الدوال يجب أن نستدعيها بكتابة اسمها متبوعًا بأقواس، وبشكل اختياري يمكننا إضافة وسيط **Argument** واحد أو أكثر بقيم محددة بحسب عدد المعاملات التي تم استخدامها في الدالة وبنفس الترتيب بحيث سيتم تخزين القيمة الأولى في المعامل الأول والقيمة الثانية في المعامل الثاني وهكذا.

تعريف الدالة

```
def my_function(fname):
```

استدعاء الدالة

```
my_function("Saad")
```

عندما يستدعي برنامج دالة ما، ينتقل التحكم من البرنامج إلى الدالة المطلوبة، والتي تقوم بدورها بتنفيذ مهمة محددة، وعند الانتهاء من تنفيذ الدالة يعود التحكم إلى البرنامج الرئيس مرة أخرى.

الوسيطات **Arguments** والمعاملات **Parameters**

يمكن استخدام مصطلح المعامل **Parameter** أو الوسيط **Argument** للدلالة على ذات الشيء، وهو المعلومات التي يتم تمريرها إلى دالة.

المعامل هو المتغير المذكور داخل الأقواس في تعريف الدالة، بينما الوسيط هو القيمة التي يتم إرسالها إلى الدالة عندما يتم استدعاؤها.

تستخدم الدوال للعديد من الأغراض البرمجية، فلنتعرف على كيفية تَغيُّر صيغة الدالة بناءً على استخداماتها داخل البرنامج.

1. دوال بدون معاملات وبدون إرجاع أي قيمة

لا يتم تبادل أي بيانات بين الدالة المستدعاة والبرنامج.

```
#creating a function
def my_function():
    print("Hello from a function")

#calling a function
my_function()
```

تطبع الدالة رسالة محددة

لاستدعاء الدالة نستخدم اسم الدالة متبوعا بقوسين فارغين.

هذه الدالة:	
ترجع قيمة	X
معاملات	X

```
Hello from a function
```

2. الدوال مع المعاملات التي لا ترجع قيمة

يتم تحديد الوسيطات بعد اسم الدالة داخل الأقواس. يمكنك إضافة العديد من الوسيطات للدالة مع الفصل بينها بعلامة الفاصلة فقط.

يجب استدعاء الدالة بعدد الوسيطات الصحيح، فإذا حاولت استدعاء دالة بعدد مختلف من الوسيطات، ستحصل على خطأ.

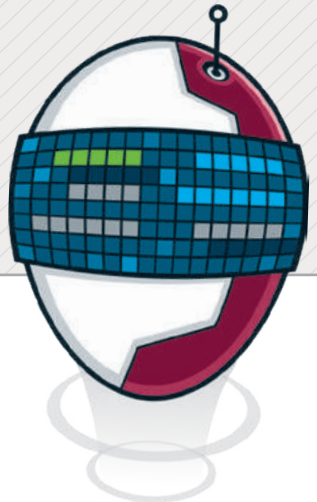
إن تعريف الدالة بكلمة **def** يوضح عدد وأنواع البيانات الخاصة بها.

```
#creating a function
def my_function(fname, lname):
    print("Hello " + fname + " " + lname," from a function")

#calling a function
my_function("Khaled", "Abdullah")
```

عند استدعاء الدالة، تقوم بطباعة الرسالة وفق هذه الوسيطات التي تم تمريرها إلى معاملات الدالة، والتي توافق عدد المعاملات ونوعها وترتيبها.

```
Hello Khaled Abdullah from a function
```

هذه الدالة:	
ترجع قيمة	✗
معاملات	✓



يمكننا إرسال أي نوع من الوسيطات إلى الدالة (نص، رقم، قائمة، قاموس ..إلخ)، وسيتم معاملتها كنفس نوع البيانات داخل الدالة.

لنستعرض مثالًا باستخدام القائمة **List**:

```
#creating a function
def my_function(animals):
    for x in animals:
        print (x)

#define the list
mammals = ["cat", "horse", "tiger", "camel"]

#calling a function
my_function(mammals)
```

```
cat
horse
tiger
camel
```

3. الدوال مع المعاملات التي ترجع قيمة

نستخدم جملة **return** لجعل الدالة ترجع قيمة.

لا تطبع الدالة أي رسالة ولكنها ترجع قيمة.

```
#creating a function
def my_function(x):
    return 5*x

#calling a function with different values of x
print (my_function(3))
print (my_function(10))
print (my_function(182))
```

يمكننا استدعاء الدالة كلما احتجنا لذلك.

```
15
50
910
```

هذه الدالة:

- ☑ ترجع قيمة
- ☑ معاملات

4. الدوال بدون معاملات وترجع قيمة

نرى هنا أن ما بين الأقواس في الدالة **data.today()** فارغ، ورغم ذلك فإن الدالة **data.today()** ترجع كائن من نوع تاريخ **date**، تم تعيينه لمتغير اليوم **today** في البرنامج السابق.

```
#import a module
from datetime import date

#calling a function from the module
today = date.today()
print (today.strftime("%B %d, %Y"))
```

يمكننا استدعاء التاريخ الحالي بصيغ مختلفة.

هذه الدالة:

- ☑ ترجع قيمة
- ☒ معاملات

```
Sep 06, 2020
```

في هذه المرحلة، سنستخدم مثالًا من مشروع قمنا بإنشائه في مرحلة دراسية سابقة وذلك للاطلاع على كيفية تكامل البرمجة التركيبية مع **Python**:

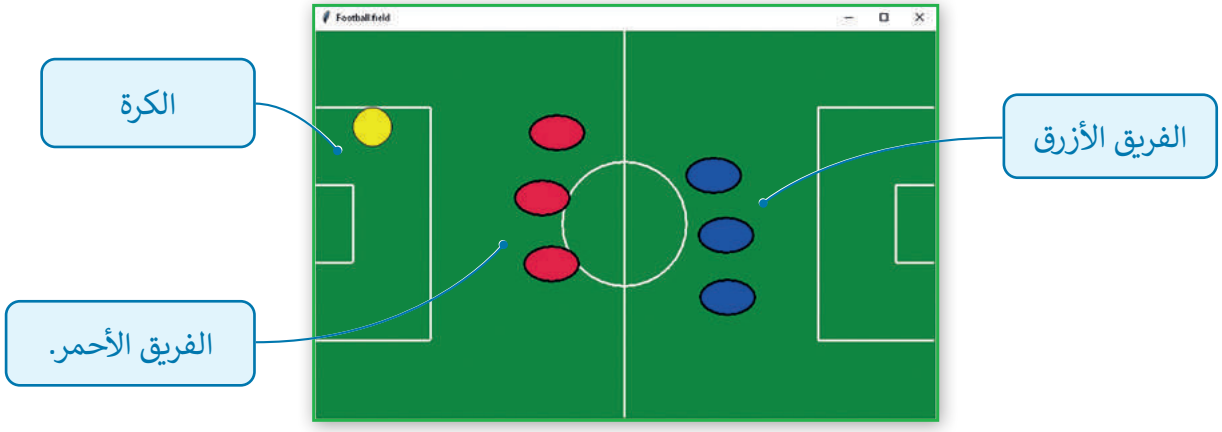
مشروع ملعب كرة القدم:

استخدمنا **Python** في هذا المشروع لتصميم لعبة بسيطة داخل ملعب كرة قدم، حيث قمنا بتطبيق مهارات الرسم التي تعلمناها في **tkinter** لرسم وتخطيط الملعب وتحديد كل من: منطقة الجزاء، منطقة المرمى، خط الوسط، والدائرة المركزية وفي النهاية رسمنا كرة صفراء يتحكم بها اللاعب.

لقد أضفنا ستة لاعبين في الملعب، ثلاثة يشكلون الفريق الأحمر أما البقية فهم الفريق الأزرق.

يلعب المستخدم دور الفريق الأحمر في اللعبة ويحاول التحكم في الكرة وإيصالها إلى منطقة مرمى الفريق الأزرق دون لمس أي لاعب من الفريق المنافس.

قمنا أيضًا ببرمجة الكرة واللاعبين بحيث يتحكم المستخدم في الكرة بواسطة أزرار الأسهم في لوحة المفاتيح، بينما يتحرك اللاعبون تلقائيًا حول الملعب. تنتهي اللعبة وتظهر الرسالة "**Game Over**" (انتهت اللعبة) إذا لمست الكرة أي لاعب في الفريق الأزرق، ولكن إذا نجح المستخدم في إيصال الكرة إلى المنطقة المستهدفة فتظهر الرسالة "**Goal**"، ثم تنتهي اللعبة.

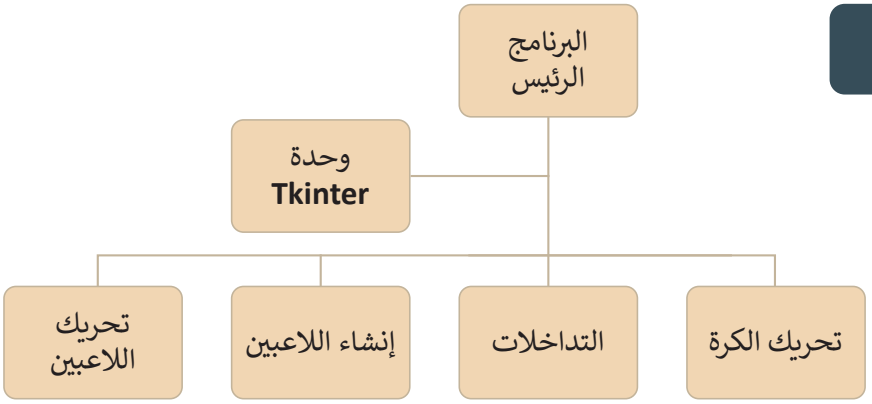


تعريف البرامج الفرعية:

قبل البدء بالبرمجة يجب أن نقوم بتقسيم المشكلة الرئيسة إلى مشاكل فرعية باستخدام تقنية البرمجة التركيبية.

1. بداية الخوارزمية.
2. قم بإنشاء اللوحة الرسومية (النافذة).
3. ارسم منطقة الجزاء اليسرى.
4. ارسم منطقة الهدف اليسرى.
5. ارسم خط منتصف الملعب.
6. ارسم منطقة الجزاء اليمنى.
7. ارسم منطقة الهدف اليمنى.
8. ارسم الدائرة المركزية.
9. ارسم الكرة.
10. حرك الكرة استجابة لضغط مفاتيح الأسهم.
11. أضف اللاعبين عند ضغط زر الفأرة داخل الملعب.
12. حرك اللاعبين داخل الملعب.
13. إذا لمست الكرة اللاعب الأزرق، اذهب إلى الخطوة 14، إذا لم تلمس اللاعب الأزرق اذهب إلى الخطوة 15.
14. أظهر على الشاشة الرسالة "Game Over".
15. إذا لمست الكرة منطقة الهدف للفريق الأزرق اذهب إلى الخطوة 16.
16. اظهر على الشاشة الرسالة "Goal".
17. قم بإيقاف الكل.
18. نهاية الخوارزمية.

تحديد أجزاء المشروع:



إنشاء البرنامج:

سيتم إنشاء البرنامج باستخدام الدوال، وفقًا لخطوات خوارزمية المشكلة، وهكذا سيكون من السهل قراءة البرنامج عند استخدام التعليقات. يحتوي كل جزء من البرنامج على تعليق مع وصف موجز لوظيفته.

الكود البرمجي النهائي للمشروع الحالي

```
from tkinter import *
window=Tk()
window.title("Football field")
canvas=Canvas(bg="green", width=800, height=500)
canvas.pack()

#left penalty area
canvas.create_line(0, 100, 150, 100, width=3, fill="white")
canvas.create_line(0, 400, 150, 400, width=3, fill="white")
canvas.create_line(150,100,150,400, width=3, fill="white")

#left goal area
canvas.create_line(0, 200, 50, 200, width=3, fill="white")
canvas.create_line(0, 300, 50, 300, width=3, fill="white")
canvas.create_line(50,200,50,300, width=3, fill="white")

#half-way line
canvas.create_line(400, 0, 400, 500, width=3, fill="white")

#right penalty area
canvas.create_line(800, 100, 650, 100, width=3, fill="white")
canvas.create_line(800, 400, 650, 400, width=3, fill="white")
canvas.create_line(650,100,650,400, width=3, fill="white")

#right goal area
canvas.create_line(800,200,750,200, width=3, fill="white",
tags="R_goal_area")
canvas.create_line(800,300,750,300, width=3, fill="white",
tags="R_goal_area")
canvas.create_line(750,200,750,300, width=3, fill="white",
tags="R_goal_area")

#center circle
canvas.create_oval(320,170,480,330,width=3,outline="white")

#create the ball
ball_ID=canvas.create_oval(50,100,100,150,fill="yellow")

#move the ball with the arrow keys
def moveBall(event):
    if event.keycode==38:
       canvas.move(ball_ID, 0, -10)
    elif event.keycode==37:
       canvas.move(ball_ID, -10, 0)
    elif event.keycode==39:
       canvas.move(ball_ID, 10, 0)
    elif event.keycode==40:
       canvas.move(ball_ID, 0, 10)
canvas.bind_all("<Key>", moveBall)
```

تتحقق هذه الدالة من الضغط على أي من مفاتيح الأسهم وتحريك الكرة إلى اتجاه معين.

```
#players
def overlaps(pl):
     items = canvas.find_overlapping(*canvas.coords(ball_ID))
     if(pl in items):
         return True
     else:
         return False

#create players
cnt=0
def player(event):
    global cnt
    if cnt<=2:
       canvas.create_oval(event.x+45,event.y+20,event.x-25,
       event.y-25,width=3,fill="crimson", tags=("players", "team_R"))
       cnt=cnt+1
    elif cnt<=5:
        canvas.create_oval(event.x+45,event.y+20,event.x-25,
       event.y-25,width=3,fill="blue", tags=("players", "team_B"))
        cnt=cnt+1
    else:
         move_players()

steps=1
max_steps=200
direction=1

def move_players():
    global steps
    global max_steps
    global direction
    players = canvas.find_withtag("players")
    teamR = canvas.find_withtag("team_R")
    teamB = canvas.find_withtag("team_B")
    right_goal_area = canvas.find_withtag("R_goal_area")
    for pl in players:
        if(overlaps(pl)):
            if(pl in teamB):
                print("Game Over")
                canvas.unbind_all("<Key>")
                return
        canvas.move(pl, 0, direction)
        steps= steps+1
        if steps>=max_steps:
            direction=direction *(-1)
            steps=1
    for line in right_goal_area:
        if(overlaps(line)):
            print("Goal!")
            canvas.unbind_all("<Key>")
            return
    canvas.after(100, move_players)
canvas.bind("<Button-1>", player)
canvas.pack()
window.mainloop()
```

تتحقق هذه الدالة من تداخل اللاعب مع الكرة، فترجع قيمة **True** إذا لمس الكرة و **False** إذا لم يفعل.

تتحقق هذه الدالة من قيمة العداد وتنشئ ثلاثة لاعبين باللون الأزرق وثلاثة لاعبين باللون الأحمر.

هذه الدالة مسؤولة عن الرسائل التي تعرضها اللعبة على الشاشة.

استخدام التخطيط الهيكلي والبرمجة التركيبية في المشروع

يعتبر البرنامج الذي أنشأناه مسبقًا والخاص بلعبة كرة القدم واضحًا بشكل عام، ولكنه يعاني من بعض نقاط الضعف، ومنها على سبيل المثال:

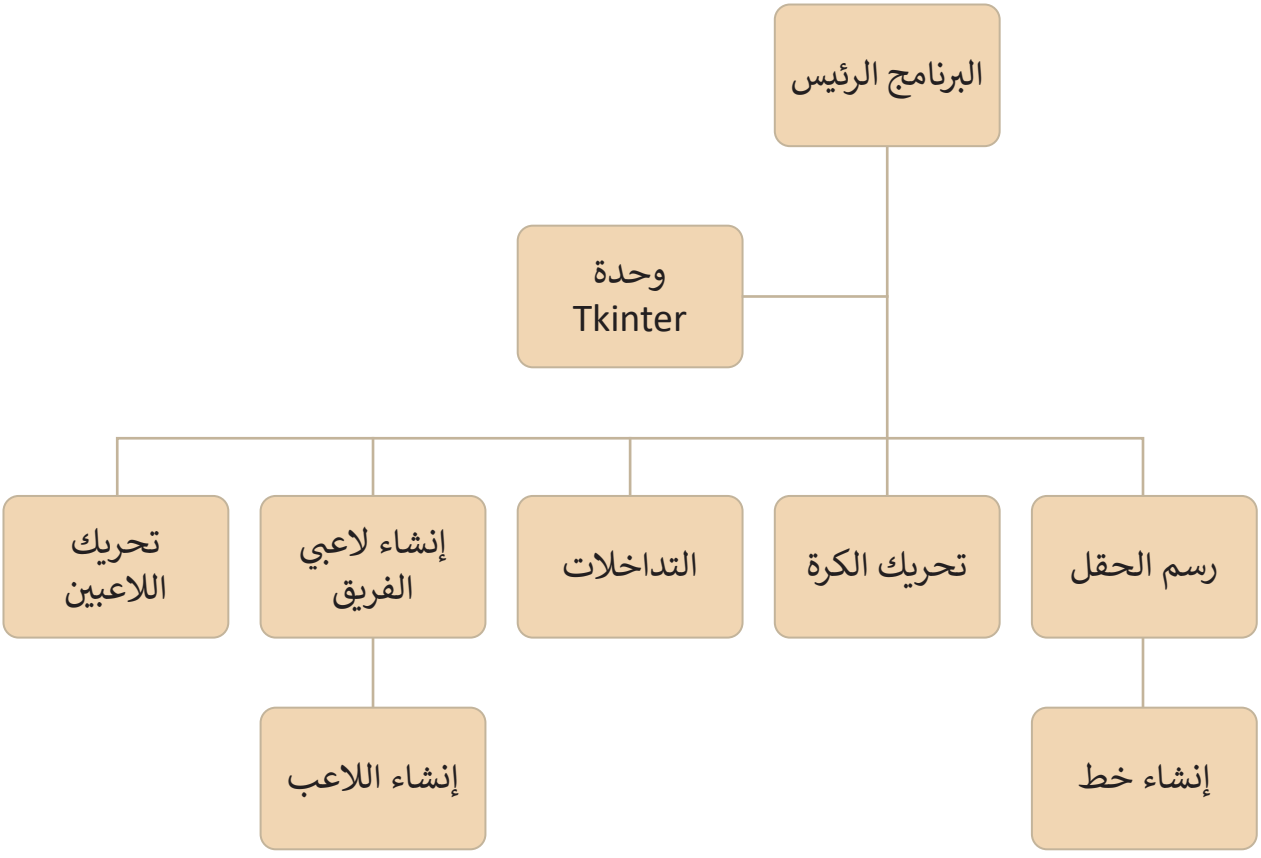
- البرنامج الرئيس كبير نوعا ما.
- من الصعب التمييز بين الدوال وأوامر البرنامج الرئيسة في بعض النقاط.

يمكننا إنشاء برامج فرعية لتحسين وظائف الدوال المستخدمة.

إن الهدف هو بناء برنامج رئيس بسيط يستخدم الدوال لتشغيل المهام. وهذا يعني أن علينا تقسيم المشكلة الرئيسة إلى المزيد من المشكلات الفرعية ومن ثم إنشاء المزيد من البرامج الفرعية المكونة للبرنامج الرئيس.

- سيتم استخدام التخطيط الهيكلي مع تقنية البرمجة التركيبية في مشروعنا الحالي لتحسين البرنامج.

المخطط الجديد للبرنامج



1. الدوال التي ترسم خطوط اللعبة

لتحسين البرنامج سنستخدم الدالة **draw_field_lines** لإنشاء ملعب لكرة القدم. تستدعي هذه الدالة دالة ثانية باسم **createLine** والتي ستنشئ خطوط الملعب. إن استخدام الدوال بدلاً من الأوامر الطويلة مع المعاملات المشتركة يجعل المقطع البرمجي أكثر قابلية للقراءة من ذي قبل، والآن لنغير الجزء الأول من المقطع البرمجي.

أولًا سننشئ الدالة لاستدعائها لاحقًا داخل دالة أخرى مع قيم معاملات مختلفة.

```
from tkinter import *

#create a line
def createLine(x1,y1,x2,y2):
    canvas.create_line(x1,y1,x2,y2, width=3, fill="white")

#draw the football field lines
def draw_field_lines():
    #left penalty area
    createLine(0, 100, 150, 100)
    createLine(0, 400, 150, 400)
    createLine(150,100,150,400)
    #left goal area
    createLine(0, 200, 50, 200)
    createLine(0, 300, 50, 300)
    createLine(50,200,50,300)
    #half-way line
    createLine(400, 0, 400, 500)
    #right penalty area
    createLine(800, 100, 650, 100)
    createLine(800, 400, 650, 400)
    createLine(650,100,650,400)
    #right goal area
    canvas.create_line(800,200,750,200, width=3,
    fill="white",tags="R_goal_area")
    canvas.create_line(800,300,750,300, width=3,
    fill="white",tags="R_goal_area")
    canvas.create_line(750,200,750,300, width=3,
    fill="white",tags="R_goal_area")
    #center circle
    canvas.create_oval(320,170,480,330,width=3,outline="white")
```

يتم استخدام معاملات العرض **width** والتعبئة **fill** مع جميع خطوط ملعب كرة القدم، ولهذا السبب فإننا لا نحتاج إلى تكرار كتابتها، وهكذا يمكن أيضًا تغيير لون وعرض خطوط ملعب كرة القدم في سطر برمجي واحد فقط بدلاً من عشرة سطور.

يستدعي كل سطر من الكود دالة **createLine** بالمعاملات المناسبة.

2. الدوال التي ستنشئ لاعب الفريق

في الجزء التالي من المقطع البرمجي، نقوم بإنشاء دالة إضافية من الدالة **player** الموجودة في برنامجنا، باستخدام الدالة **create_player** يمكننا إنشاء أي فريق بأي لون متاح في **Python**.

```
#create a player
def create_player(event, fp, t):
       canvas.create_oval(event.x+45,event.y+20,event.x-25,
        event.y-25,width=3,fill=fp,tags("players",t))

#create the players of both teams
def player(event):
    global cnt
    if cnt<=2:
       canvas.create_player(event,"crimson", "team_R")
       cnt=cnt+1
    elif cnt<=5:
       canvas.create_player(event,"blue", "team_B")
       cnt=cnt+1
    else:
         move_players()
```

تسمح هذه الدوال بإنشاء عدد كبير ممكن من الفرق بألوان وأسماء مختلفة.

الكود البرمجي النهائي للمشروع:

```
from tkinter import*

#create a line
def createLine(x1,y1,x2,y2):
    canvas.create_line(x1,y1,x2,y2, width=3, fill="white")

#draw the football field lines
def draw_field_lines():
    #left penalty area
    createLine(0, 100, 150, 100)
    createLine(0, 400, 150, 400)
    createLine(150,100,150,400)
    #left goal area
    createLine(0, 200, 50, 200)
    createLine(0, 300, 50, 300)
    createLine(50,200,50,300)
    #half-way line
    createLine(400, 0, 400, 500)
    #right penalty area
    createLine(800, 100, 650, 100)
    createLine(800, 400, 650, 400)
    createLine(650,100,650,400)
    #right goal area
    canvas.create_line(800,200,750,200, width=3,
    fill="white",tags="R_goal_area")
    canvas.create_line(800,300,750,300, width=3,
    fill="white",tags="R_goal_area")
    canvas.create_line(750,200,750,300, width=3,
    fill="white",tags="R_goal_area")
    #center circle
    canvas.create_oval(320,170,480,330,width=3,outline="white")

#move the ball with the arrow keys
def moveBall(event):
    if event.keycode==38:
        canvas.move(ball_ID, 0, -10)
    elif event.keycode==37:
        canvas.move(ball_ID, -10, 0)
    elif event.keycode==39:
        canvas.move(ball_ID, 10, 0)
    elif event.keycode==40:
        canvas.move(ball_ID, 0, 10)

#check if the ball touches a player
def overlaps(pl):
    items = canvas.find_overlapping(*canvas.coords(ball_ID))
    if(pl in items):
        return True
    else:
        return False
```

دالة **draw_field_line** تستدعي الدالة **createLine**.

```
#create a player
def create_player(event, fp, t):
       canvas.create_oval(event.x+45,event.y+20,event.x-25,
        event.y-25,width=3,fill=fp,tags("players",t))

#create the players of both teams
def player(event):
    global cnt
    if cnt<=2:
       canvas.create_player(event,"crimson", "team_R")
       cnt=cnt+1
    elif cnt<=5:
       canvas.create_player(event,"blue", "team_B")
       cnt=cnt+1
    else:
         move_players()

#move players
def move_players():
    global steps
    global max_steps
    global direction
    players = canvas.find_withtag("players")
    teamR = canvas.find_withtag("team_R")
    teamB = canvas.find_withtag("team_B")
    right_goal_area = canvas.find_withtag("R_goal_area")

    for pl in players:
        if(overlaps(pl)):
            if(pl in teamB):
                print("Game Over")
                canvas.unbind_all("<Key>")
                return
        canvas.move(pl, 0, direction)
        steps= steps+1
        if steps>=max_steps:
            direction=direction *(-1)
            steps=1

    for line in right_goal_area:
        if(overlaps(line)):
            print("Goal!")
            canvas.unbind_all("<Key>")
            return

    canvas.after(100, move_players)
```

دالة **player** تستدعي الدالة **create_player**.

في نهاية الكود سنضع البرنامج الرئيس الذي يستدعي الدوال ويحدد متغيرات البرنامج.

```
#main program

#create program's window
window=Tk()
window.title("Football field")

#draw the football field
canvas=Canvas(bg="green", width=800, height=500)
canvas.pack()
draw_field_lines()

#create the ball
ball_ID=canvas.create_oval(50,100,100,150,fill="yellow")

#initialise program's variables
cnt=0
steps=1
max_steps=200
direction=1

#bind the keyboard events to the ball
canvas.bind_all("<Key>", moveBall)
#bind the mouse left button click to the creation of a player
canvas.bind("<Button-1>", player)
canvas.pack()

#go into an infinite loop
window.mainloop()
```

قد لا تقدم البرمجة التركيبية حلولًا لجميع المشكلات ولكنها تجبرك في الواقع على التفكير بشكل منطقي ومنظم فيما يتعلق ببرنامجك. وفيما يلي مجموعة من الفوائد البرمجية لاستخدام البرمجة التركيبية:

- عملية تنظيم التعليمات البرمجية على شكل أجزاء منطقية تساعدك في تنظيم برنامجك بحيث تعرف أين ينتمي كل جزء.
- التفكير في وظيفة كل برنامج فرعي يشجعك على اختيار أسماء واضحة ومناسبة للأجزاء المختلفة من برنامجك.
- استخدم التعليقات والوصف الموجز لوظائف كل برنامج فرعي للمساعدة في سهولة قراءة البرنامج.
- أخيرًا يشجع استخدام الهيكلية التركيبية كل جزء من برنامجك على أداء مهمة واحدة معينة، مما يقلل من احتمالية تأثر كل جزء من البرنامج بالمشكلات الجانبية للأجزاء الأخرى منه.

دوال البرنامج:

دوال مخصصة **Custom functions**:

- `draw_field_lines`
- `createLine`
- `moveBall`
- `overlaps`
- `player`
- `create_player`
- `move_players`

دوال المكتبة **Library functions**:

- `canvas.create`
- `canvas.move`
- `canvas.find`
- `canvas.unbind`
- `canvas.after`
- `canvas.bind`
- `canvas.pack`
- `event.keycode`
- `window.title`
- `window.mainloop`

1

وضح المقصود بالبرنامج الفرعي **subprogram**، وما هي خصائصه؟

2

حدد ما إذا كانت الدوال الآتية ترجع قيمة وتأخذ معاملات أم لا، وذلك بالإشارة إلى المربعات المناسبة بـ ✗ و ✓:

```
#creating a function
def kilos_to_grams(k):
    return 1000*k

#calling a function
print(kilos_to_grams(60))
```

ترجع قيمة	☐
معاملات	☐

```
#creating a function
def kilos_to_grams():
    kilograms = float(input("Please enter kilograms:"))
    return kilograms * 1000

#calling a function
print(kilos_to_grams())
```

ترجع قيمة	☐
معاملات	☐

3

اذكر الميزات الأربعة الرئيسة للبرمجة التركيبية.

.1

.2

.3

.4

4

إنشاء برنامج باستخدام الدوال.

أنشئ برنامج يقارن بين درجات الطلبة في خمس مواد دراسية ويضعها في ترتيب من الأعلى إلى الأدنى، ثم يحسب متوسط درجات جميع الطلبة.

وأخيرًا، يقوم البرنامج بطباعة رسالة لإعلام المستخدم عن أعلى وأدنى درجة، وعرض رسالة حول متوسط درجات الطلبة.

قم بتجزئة المشكلة الرئيسة إلى مشكلات فرعية أولاً، ثم قم بإنشاء البرنامج.

5 اختر الإجابة الصحيحة:

1. البرنامج الفرعي:	يمكن استخدامه في برامج محددة فقط.	○
	يمكن استخدامه في برامج أخرى.	○
	يمكن استخدامه في برامج فرعية أخرى فقط.	○
2. معاملات الدوال هي:	القيم التي يتم طباعتها على الشاشة.	○
	القيم التي يتم ارسالها للدالة عند استدعاءها.	○
	المتغيرات الموجودة بين قوسين في تعريف الدالة.	○
3. البرمجة التركيبية:	تفيد المطورين فقط على فهم وإدارة البرنامج الذي لم يكتبوه.	○
	تقلل من الأخطاء وتجعل من السهل على المطورين التحكم والفهم والتعديل أو توسيع البرنامج.	○
	تقسم المشكلة إلى مجموعة من المشكلات الفرعية التي يسهل حلها وبالتالي حل المشكلة الرئيسة بشكل منطقي فقط.	○

الدرس الرابع

المكتبات البرمجية

لقد قمنا في الدروس السابقة باستخدام دوال Python لكتابة التعليمات البرمجية من مكتبات ووحدات Pyhton القياسية.

في هذا الدرس، سنتعمق في هذا المفهوم ونكتشف المزيد عن المكتبات وكيفية استخدامها.

المكتبات في علم الحاسوب

المكتبة **Library** هي عبارة عن مجموعة من التعليمات البرمجية المدمجة مسبقًا في لغات البرمجة والتي يتم استخدامها لتقليل الوقت المستغرق في البرمجة الفعلية، تمامًا كما هو الحال في المكتبات المادية، وتعتبر هذه المكتبات من الموارد القابلة لإعادة الاستخدام في أي برنامج حيث أنها مستقلة عن البرامج التي يتم كتابتها .

خصائص المكتبة
1. يمكن كتابة المكتبات بأي لغة برمجة، وتستخدم غالبًا في بيئات تطوير البرامج.
2. تُعد المكتبات مفيدة للغاية للوصول إلى التعليمات البرمجية المكتوبة مسبقًا والمستخدمة بشكل متكرر بدلاً من كتابتها من الصفر في كل مرة.
3. قد تحتوي المكتبات على بيانات الإعداد أو التوثيق **Documentation** لكيفية تنظيم المكتبة ومحتوياتها من الدوال والبرامج الفرعية والقيم وغيرها ...
4. يتم تنظيم المكتبات بحيث يمكن استخدامها من قبل برامج متعددة ذات طبيعة مختلفة وليس لها اتصال أو علاقة ببعضها البعض.
5. يستدعي الوظيفة أو المهمة التي تقدمها المكتبة عبر آلية تتوفر في لغة البرمجة.
6. يحتاج المستخدم فقط إلى معرفة وظيفة المكتبة وليس التفاصيل الداخلية لها.

المكتبات في Python

المكتبة في **Python** هي عبارة عن مجموعة من الدوال التي تسمح لك بتنفيذ العديد من الإجراءات دون كتابة كود برمجي كبير. تتوفر في Python مكتبة قياسية، كما ويمكن الوصول إلى آلاف المكتبات التي تم بناؤها من قبل مجتمعات المطورين حول العالم.

قبل أن ننتقل إلى مكتبات بايثون، علينا أن نتعرف على مصطلح الوحدة البرمجية **Module**.

الوحدات البرمجية: هي عبارة عن حزمة من الملفات تحتوي مقاطع برمجية، يتم استيرادها إلى البرنامج لتنفيذ وظائف مختلفة، ويكون امتدادها عادةً py.

من أمثلة الوحدات البرمجية القياسية في Python

- وحدات واجهة المستخدم الرسومية **tkinter module**.
- وحدات برمجية لمعرفة خصائص الحاسوب ونظام التشغيل **Platform module**.
- وحدة الرسم **turtle module**.

أهمية الوحدات البرمجية:

1. إعادة استخدام الكود البرمجي.
2. تساعد في تنظيم المشاريع وتقسيمها.

فكر في الوحدة البرمجية كمجموعة من الدوال المرتبطة معًا.

نصيحة ذكية

تتوفر بعض الوحدات في جميع إصدارات ونسخ Python، بينما يكون البعض الآخر متاحًا فقط عندما يدعمها النظام الأساسي أو يطلبها، كما تتطلب بعض الوحدات الأخرى أن تقوم بتثبيتها وإعدادها عند تثبيت بيئة Python.

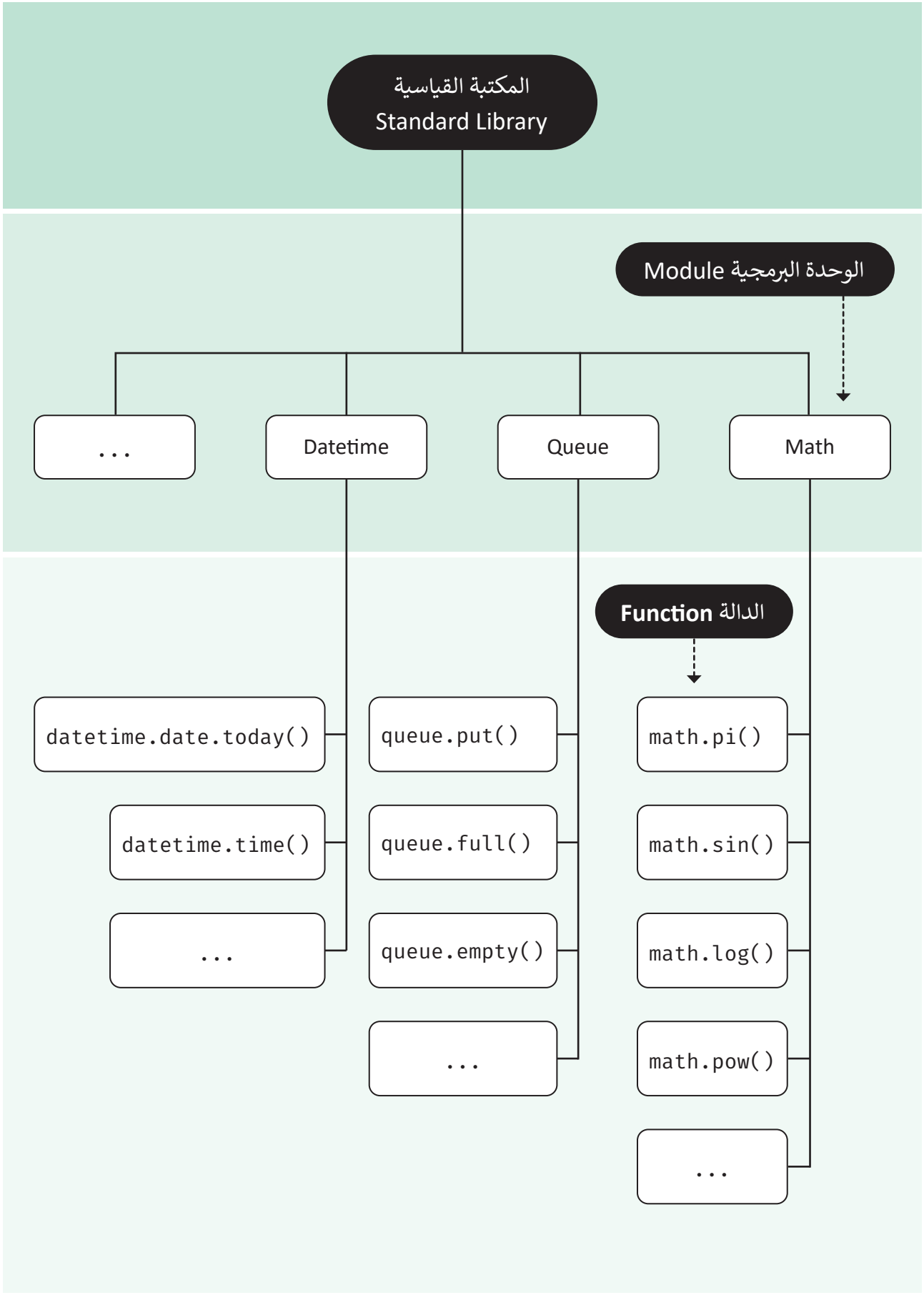
مكتبة Python القياسية

يُستخدم مصطلح المكتبة في **Python** بشكل شائع للإشارة إلى المكتبة القياسية. والتي يتم تثبيتها تلقائيًا عند تثبيت **Python**، مما يجعل وحداتها متاحة بشكل موثوق لأي مقطع برمجي يتم كتابته في **Python**، وبهذا تكون هذه المكتبة جزءًا أساسيًا من لغة **Python** ذاتها. تحتوي هذه المكتبة القياسية على أكثر من 200 وحدة برمجية.

مكتبة **Python** القياسية واسعة للغاية وتقدم مجموعة واسعة من الوحدات البرمجية التي أشرنا إليها، فهي تحتوي على وحدات برمجية مدمجة مكتوبة بلغة برمجة C والتي توفر الوصول إلى وظائف النظام مثل الملفات، وكذلك على وحدات تم كتابتها بلغة **Python**، وتلك توفر حلولًا قياسية للعديد من المشكلات البرمجية.

توجد الدوال داخل وحدات برمجية داخل المكتبات القياسية.

يمثل الرسم البياني أدناه بعض وحدات المكتبة القياسية **Standard Library** وبعض دوالها.



مكتبات Python الأخرى

لا يقتصر الأمر على المكتبة القياسية المثبتة في **Python**، بل يمكن بسهولة تنزيل مكتبات إضافية وتثبيتها لإضافة دوال أخرى قد نحتاجها في برامج أخرى. تأتي معظم المكتبات الإضافية بأدوات التثبيت الخاصة بها أو ببرنامج التثبيت النصي الخاص بها. بمجرد تثبيت المكتبات الإضافية، فإنها تتصرف مثل مكتبة **Python** القياسية، ولا توجد أوامر خاصة تحتاج إلى معرفتها. في هذا الدرس سنركز على المكتبات القياسية.

كيفية استخدام المكتبة القياسية

نظرًا لأن المكتبة القياسية مثبتة بالفعل، فنحن بحاجة فقط إلى استيراد وحداتها البرمجية إلى البرنامج عن طريق إضافة سطر أوامر في أعلى المقطع البرمجي.

هناك عدة طرق للقيام باستيراد وحدات المكتبة القياسية، وأكثرها شيوعًا ما يلي:

1. استيراد الكل

يمكنك تضمين محتويات من المكتبة في المقطع البرمجي باستخدام هذا السطر:

```
#import everything from the module
from module_name import *

#call a function from the imported module
function_name()
```

سيؤدي هذا إلى قراءة كامل محتوى الوحدة البرمجية وإسقاطه مباشرة في المقطع البرمجي.

ثم يمكننا استدعاء أي دالة من الوحدة المستوردة فقط بواسطة اسمها.

نصيحة ذكية

الفرق الوحيد في استخدام مكتبات Python الخارجية هو الحاجة إلى تثبيتها مسبقًا لكي تكون قادرًا على استيرادها إلى البرنامج.

الميزات	العيوب
تتميز هذه الطريقة بتوفير بعض وقت الكتابة، خاصةً عندما نحتاج إلى استخدام الكثير من الدوال من الوحدة القياسية.	إذا قمنا باستيراد جميع الدوال، فسيتم زيادة المقطع البرمجي في البرنامج النهائي دون أي سبب.
يفيد استيراد الكل إذا كنت ترغب في استخدام دالة لا تتذكر إلى أي وحدة برمجية تنتمي.	من الأفضل أن نعرف بالضبط أين يتم استخدام كل دالة لأسباب تتعلق بالصيانة والأمن.

2. استيراد دوال من وحدة برمجية

طريقة أخرى هي استيراد الوحدة البرمجية ودوالها التي ستستخدمها في برنامجك.

```
#import Function from the module
from module_name import function_a, function_b, function_c

#call a function from the imported module
function_a()
function_b()
function_c()
```

يمكنك الآن استخدام الدوال a و b و c في برنامجك.

3. استيراد الوحدة البرمجية

إن أفضل طريقة للتعامل مع الوحدة البرمجية هي استيراد كل محتوياتها وجعلها متاحة فقط من خلال كتابة اسم الوحدة البرمجية ثم اسم الدالة.

```
#import everything from the module
import module_name*

#call a function of the module
module_name.function_name()
```

تحتاج إلى ذكر اسم الوحدة البرمجية ثم اسم الدالة التي تريد استدعاءها.

من المهم أن تدرك أنه ليس عليك بالضرورة فهم المكتبة بأكملها، طالما كنت قادرًا على اختيار الأجزاء التي تحتاجها فقط.

الآن وبعد أن تعرفنا على أساسيات مكتبات Python لنرى كيف يمكنك استخدامها.

استخدام المكتبات القياسية الأكثر شيوعاً

تساعدك المكتبة القياسية على التعامل مع العديد من المهام، لذلك سنستعرض بعضًا من أكثر الوحدات البرمجية استخدامًا من المكتبة القياسية.

1. وحدة sys البرمجية

الهدف من وحدة sys هو مساعدة المطور على معرفة المزيد عن النظام الخاص بجهاز المستخدم ومشغل **Python** الذي تم تثبيته على هذا الجهاز، وكما هو الحال في جميع الوحدات الأخرى، يجب استيراد وحدة **sys** باستخدام الأمر **import**.

```
#import the sys module
import sys

#show the Python version and the locations of the packages
print(sys.version)
print(sys.path)
```

يعرض على الشاشة مسار تخزين جميع وحدات Python القياسية.

يعرض على الشاشة نسخة Python المستخدمة.

يتم عرض الكثير من المعلومات بما فيها نسخة Python 3.7.0

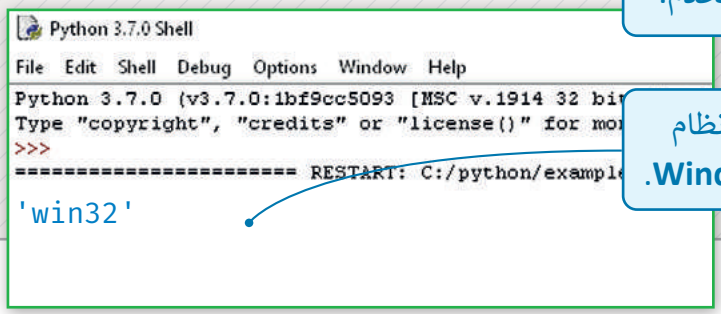
```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
3.7.0 (v3.7.0:c2f86d86e6, Oct 19 2019, 10:49:36) [MSC v.1500 32
bit (Intel)]

['c:\\Users\\BL\\Desktop\\Python Documents CS12\\pyhton codes',
'C:\\WINDOWS\\SYSTEM32\\python37.zip', 'C:\\Python37\\DLLs', 'C:\\
Python37\\lib', 'C:\\Python37\\lib\\plat-win', 'C:\\Python37\\
lib\\lib-tk', 'C:\\Python37', 'C:\\Users\\BL\\AppData\\Roaming\\
Python\\Python37\\site-packages', 'C:\\Python37\\lib\\site-packag-
es']
```

هذه المسارات التي تشير إلى أماكن حفظ جميع الوحدات البرمجية.

في هذا المثال سنقوم بتحديد هوية نظام التشغيل الخاص بنا.

```
#import the sys module
import sys
#show the computer's operating system
print(sys.platform)
```

تعرض هذه الدالة نظام التشغيل المستخدم.



يعمل الجهاز على نظام تشغيل **Windows 32bit**.

2. وحدة os البرمجية

تعد وحدة os مثالًا جيدًا على الوحدة القابلة لإعادة الاستخدام التي توفر بعض الوظائف الأساسية للكود البرمجي للتفاعل مع جهاز المستخدم دون الحاجة إلى أخذ نظام التشغيل للمستخدم بعين الاعتبار (**System Independent**).

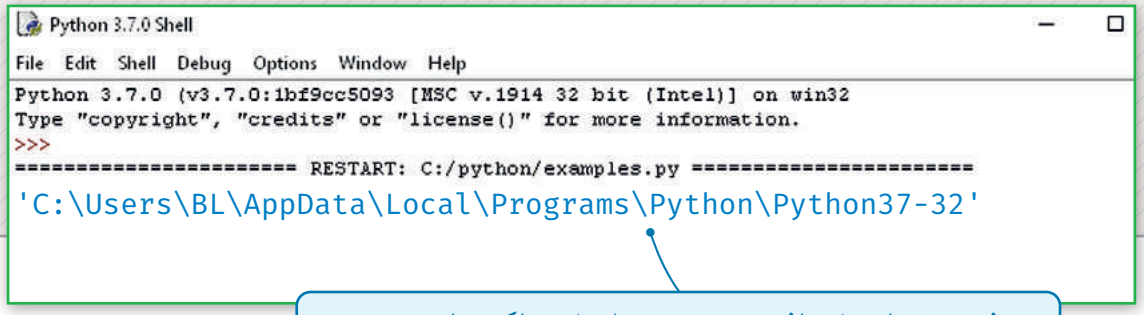
من الممكن إجراء العديد من مهام نظام التشغيل تلقائيًا. توفر وحدة os في **Python** دوال لإنشاء مجلد وإزالته، وجلب محتوياته، وتغيير المسار الحالي أو تحديده.

على سبيل المثال، هناك دالة **getcwd()** في وحدة نظام التشغيل والتي باستخدامها يمكننا معرفة اسم المجلد الذي يتعامل مع الكود البرمجي الخاص بنا.

```
#import the os module
import os
#show the current working directory
print(os.getcwd())
```

استيراد الوحدة القياسية.

استدعاء الدالة **getcwd()** من الوحدة القياسية os



هذا مسار المجلد الذي يتم به حفظ ملف الكود البرمجي به.

استخدام الدوال الخاصة بوحدة **os** بأكثر من ملف لمواقع مختلفة:

سنستخدم دالة **(chdir)**، لتغيير المسار الحالي إلى مسار تم إنشاؤه حديثًا قبل القيام بأي عمليات فيه.

ثم سنقوم بإعادة المسار الحالي إلى المسار الأصلي باستخدام ".." كوسيط في الدالة **(chdir)**.

```
#import the os module
import os
#print the parent directory
print(os.getcwd())
#change the directory to a new one
print(os.chdir("C:\Users\BL\Desktop"))
#print the new directory
print(os.getcwd())
#set the current directory to the parent
print(os.chdir(".."))
#show the current working directory
print(os.getcwd())
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
'C:\Users\BL\AppData\Local\Programs\Python\Python37-32'
'C:\Users\BL\Desktop'
'C:\Users\BL\AppData\Local\Programs\Python\Python37-32'
```

بعد الوصول إلى المسار المطلوب، قد ترغب في الوصول إلى محتواه، ترجع الدالة **listdir()** قائمة تحتوي على أسماء ملفات هذا المجلد.

```
#import the os module
import os
#show the files of the directory
print(os.listdir())
```

تتم طباعة محتويات المجلد الخاص ببرنامج Python.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LI-
CENSE.txt', 'NEWS.txt', 'python.exe', 'python3.
dll', 'python37.dll', 'pythonw.exe', 'Scripts',
'tcl', 'Tools', 'vcruntime140.dll']
```

دالة dir():

إحدى الدوال المفيدة لمعرفة محتويات المكتبة هي **(dir)**، يمكنك استدعاءها على أي كائن لمعرفة الإجراءات التي تدعمها، ولكنها مفيدة بشكل خاص مع الوحدات البرمجية.

على سبيل المثال، يمكنك عرض جميع وظائف وحدة نظام التشغيل os على الشاشة كقائمة.

دعنا نرى ما تحتوي عليه وحدة **os** البرمجية:

```
#import the os module
import os

#show all functions of the os module
print(dir(os))
```

الكائن الذي تريد رؤية كل خصائصه وطرقه.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
['F_OK', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOIN-
HERIT', 'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL', 'O_
SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY',
'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'R_
OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'UserDict',
'W_OK', 'X_OK', '_Environ', '__all__', '__builtins__', '__doc__',
'__file__', '__name__', '__package__', '_copy_reg', '_execvpe',
'_exists', '_exit', '_get_exports_list', '_make_stat_result', '_
make_statvfs_result', '_pickle_stat_result', '_pickle_statvfs_re-
sult', 'abort', 'access', 'altsep', 'chdir', 'chmod', 'close',
'closerange', 'curdir', 'defpath', 'devnull', 'dup', 'dup2',
'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'exe-
clpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fdopen',
'fstat', 'fsync', 'getcwd', 'getcwdu', 'getenv', 'getpid', 'isat-
ty', 'kill', 'linesep', 'listdir', 'lseek', 'lstat', 'makedirs',
'mkdir', 'name', 'open', 'pardir', 'path', 'pathsep', 'pipe',
'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'remove',
'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'spawnl',
'spawnle', 'spawnv', 'spawnve', 'startfile', 'stat', 'stat_float_
times', 'stat_result', 'statvfs_result', 'strerror', 'sys', 'sys-
tem', 'tempnam', 'times', 'tmpfile', 'tmpnam', 'umask', 'unlink',
'unsetenv', 'urandom', 'utime', 'waitpid', 'walk', 'write']
```

هذه الدالة التي استخدمناها في المثال السابق.

نصيحة ذكية

دالة (dir) ليست مفيدة للمكتبات فقط، بل يمكن استخدامها مع جميع كائنات Python، مثل الفئات (classes) والدوال (functions)، كما أنها تدعم أنواع البيانات من النصوص والأرقام.

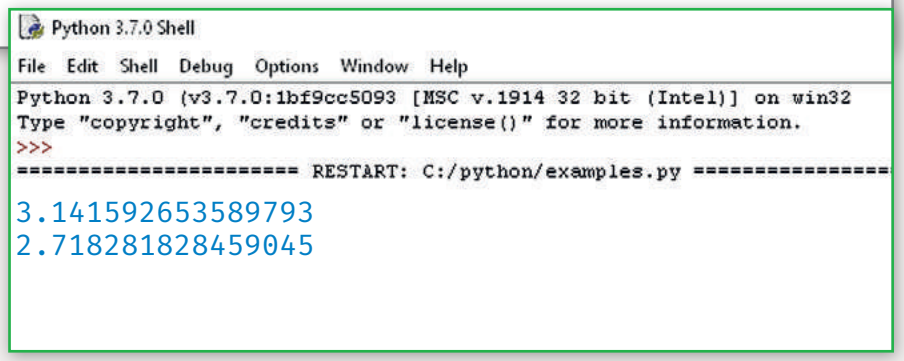
3. وحدة math البرمجية

يتم في وحدة **Math** تعريف بعض الدوال الرياضية الأكثر شيوعًا، مثل الدوال المثلثية، والدوال اللوغاريثمية، ووظائف تحويل الزاوية وغيرها.

يوجد عدد كبير من الدوال في هذه الوحدة الخاصة بالرياضيات، لنستعرض بعضًا منها:

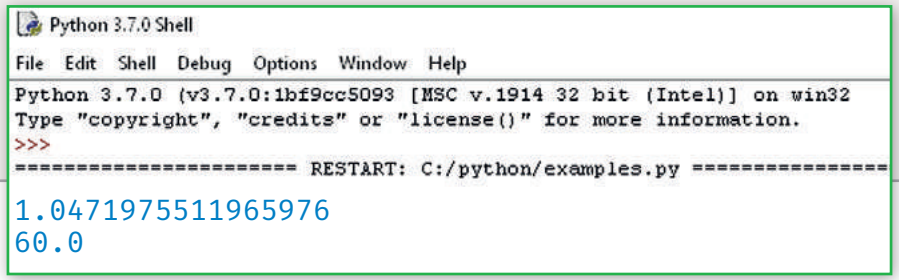
يتم تعريف الثوابت الرياضية في هذه الوحدة أيضًا.

```
#import the math module
import math

#print the pi mathematical constant
print(math.pi)
#print the Euler's number (e)
print(math.e)
```



عندما نريد العمل مع الدوال (sin، cos، tan، ...) نحتاج إلى الزاوية بالتقدير الدائري كوسيط. على سبيل المثال، تحول الجمل البرمجية التالية الزاوية 60 من التقدير الستيني إلى الدائري وبالعكس.

```
#import the math module
import math
#convert an angle from degrees to radians
print(math.radians(60))
#convert an angle from radians to degrees
print(math.degrees(1.0471975511965976))
```



نصيحة ذكية

للتعامل مع الزاويا في بايثون يجب أن تكون الزوايا بالتقدير الدائري.

يمكننا الآن العمل مع دوال (sin, cos, tan, ...) الخاصة بزاوية 60 درجة (1.047197511965976) بالتقدير الدائري.

```
#import the math module
import math

#calculate the sine of 60 degrees
print(math.sin(math.radians(60)))
#calculate the cosine of 60 degrees
print(math.cos(math.radians(60)))
#calculate the tangent of 60 degrees
print(math.tan(math.radians(60)))
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
0.8660254037844386
0.5000000000000001
1.7320508075688767
```

يمكن أن تساعدنا دوال الوحدة البرمجية **math** على إجراء حسابات معقدة.

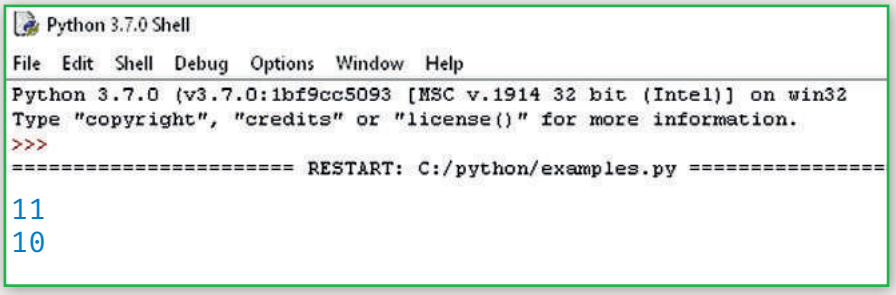
```
#import the math module
import math

#calculate the natural logarithm (base-e) of a given number
print(math.log(10))
#calculate the base-10 logarithm of the given number
print(math.log10(10))
#calculate 2 raised to the power of 8 (2^8)
print(math.pow(2,8))
#calculate the square root of 20
print(math.sqrt(20))
```

تتلقى الدالة **math.pow (a,b)** وسيطين عشرين، وترفع الثانية إلى الأولى وتعيد النتيجة **(a^b)**.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
2.302585092994046
1.0
256.0
4.47213595499958
```

الدالتين التاليتين تساعدان في تقريب الأعداد العشرية.

```
#import the math module
import math

#calculate the ceiling of the number
print(math.ceil(10.1657))
#calculate the floor of the number
print(math.floor(10.1657))
```



انتبه عند التعامل مع هذه الدوال من الأرقام السالبة حاول أن تجرب الدالتين السابقتين على الرقم **-3.4**

4. وحدة tkinter البرمجية

tkinter هي وحدة استخدمناها سابقًا واستخدمنا أيضًا العديد من دوالها. وحدة **tkinter** هي إحدى الحالات التي نستورد فيها كامل الوحدة حيث نستخدم مجموعة كبيرة من دوالها.

```
#import everything of the tkinter module
from tkinter import *

#create a window
window=Tk()
window.title("Hello Python")
window.geometry("300x300")
```

على سبيل المثال، نحتاج إلى استدعاء ثلاث دوال فقط لإنشاء النافذة.

5. وحدة time البرمجية

هناك وحدة برمجية معروفة متاحة في **Python** توفر دوال للعمل مع الأوقات.

```
#import time module
import time

#what time is it?
print(time.strftime("%H:%M"))
#what day of the week is it?
print(time.strftime("%A %p"))
```

نود معرفة الوقت بالساعة والدقيقة وبتنسيق 24 ساعة.

الآن نود معرفة ما هو اليوم وما إذا كان صباحًا أم مساءً.

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
'10:49'
'Wednesday AM'
```

6. وحدة datetime البرمجية

نحتاج إلى العمل مع التواريخ والوقت بكثرة في البرامج المختلفة، ولهذا السبب، توفر المكتبة القياسية وحدة **datetime** لمساعدتنا في العمل مع هذا النوع من البيانات.

```
#import datetime module
import datetime

#today's day
print(datetime.date.today())
```

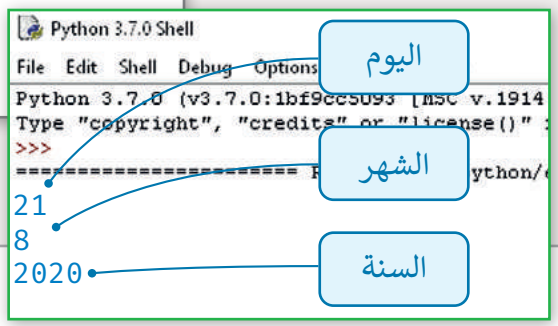
```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
2020-08-21
```

السنة — الشهر — اليوم

دعونا نسأل عن اليوم بطريقة مختلفة باستخدام **Attribute** الدالة. في هذا المثال، نحتاج **Attribute** اليوم **day** والشهر **month** والسنة **year** من دالة **date.today()** يمكننا استخدام نموذج الصيغة التالية:

module_name.function_name( ).attribute_name

```
#import datetime module
import datetime
#today's date in detail
print(datetime.date.today().day)
print(datetime.date.today().month)
print(datetime.date.today().year)
```



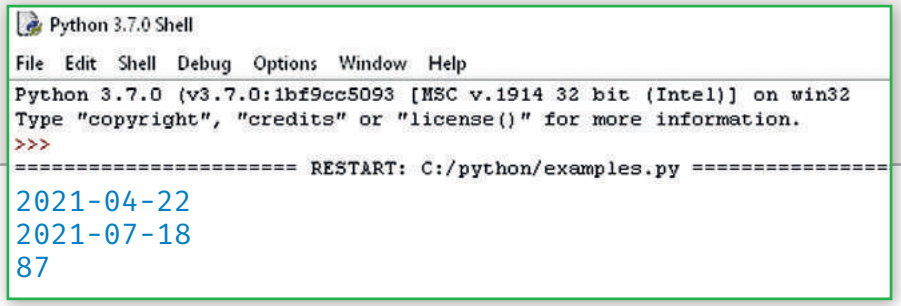
سنستخدم في المثال التالي وحدة **datetime** لحساب عدد الأيام المتبقية ليوم ميلادنا.

ترجع الدالة

date.replace(year=self.year, month=self.month,day=self.day)

تاريخًا بنفس القيمة، باستثناء تلك المعاملات التي يتم منحها قيمًا جديدة من خلال الوسيطات المحددة.

```
#import datetime module
from datetime import date

#print today's date
today = date.today()
print(today)
#my birthday date for this year
my_birthday = date(today.year, 7, 18)
print(my_birthday)

#check if my birthday date has passed
if my_birthday < today:
    my_birthday = my_birthday.replace(year=today.year + 1)
my_birthday

#calculate the days left until my next birthday
time_to_birthday = my_birthday - today
print(time_to_birthday.days)
```



إنشاء المقطع البرمجي الخاص بك

يعتبر إعادة استخدام التعليمات البرمجية من الممارسات الجيدة دائمًا، حيث يوفر السرعة والموثوقية في عملية البرمجة.

قد يتميز المقطع البرمجي الخاص بك بمميزات معينة مقارنةً بذلك الموجود في الوحدات البرمجية القياسية أو تلك المكتوبة من مبرمجين آخرين، وفيما يلي أهم تلك المميزات:

- تلائم المميزات احتياجاتك الحقيقية.
- لديك تحكم كامل بالكود البرمجي والقدرة على تصحيح الأخطاء والقيام بالتغييرات لحظيًا عند الضرورة.
- قد لا تتوفر معلومات التوثيق الكافية في الوحدات الخارجية، أو قد لا تكون معلومات التوثيق صحيحة، فيبقى الكود المكتوب من قبلك أكثر موثوقية.
- قد يحتاج استخدام الوحدات البرمجية الخارجية إلى تحديثات من قبل المطور الأصلي والتي ستتوقف في حال توقف المطور عن عمله مما يضطرك إلى البحث عن حلول بديلة.
- قد تكون تكلفة استخدام أو ترخيص استعمال الوحدات الخارجية مكلفة للغاية أو مقيدة في الاستخدام.

مدير الحزم Python pip

PIP هو مدير الحزم الخاص بحزم **Python** أو كما نطلق عليها اسم الوحدات البرمجية، ويمكننا من خلال استخدامه تثبيت حزم إضافية غير متوفرة في مكتبة Python القياسية.

تثبيت الحزم مع PIP	تحتوي الحزمة على جميع الملفات المطلوبة للوحدة البرمجية.

إضافة إلى مكتبة Python القياسية، يساهم مجتمع Python بمجموعة واسعة من الحزم المصممة لأطر التطوير والأدوات والمكتبات المختلفة. يتم استضافة معظم هذه الحزم ونشرها رسميًا في **Python Package Index (PyPI).pip** والتي تتيح لنا تنزيل هذه الحزم وتثبيتها.

PyPI هو عبارة عن مستودع برامج لـ **Python**. يستخدم **PIP** ال **PyPI** كموقع افتراضي للبحث عن حزمة، ثم يقوم بتثبيت وإدارة حزم البرامج المكتوبة بلغة Python.

يستخدم الأمر **install** لتثبيت الحزم باستخدام **pip**. لنأخذ مثالاً:

يمكننا تثبيت وحدة **pygame** باعتبارها وحدة برمجية شائعة الاستخدام تستخدم في إنشاء برامج الرسم بطريقة أسهل على شاشة الحاسوب. فهي مكتبة برمجية خاصة بـ **Python** مفتوحة المصدر وتستخدم لإنشاء تطبيقات الوسائط المتعددة مثل الألعاب.

لتثبيت pygame:

> من شريط البحث، ❶ اكتب "**cmd**" واضغط زر Enter. ❷

> سيفتح موجه الأوامر.

> اكتب "**pip install pygame**" واضغط Enter. ❸

> سيتم تثبيت الوحدة البرمجية **pygame** بنجاح. ❹

موجه الأوامر **Command Prompt** هو حقل الإدخال في شاشة واجهة مستخدم نصية خاصة بنظام التشغيل أو البرنامج. وكما يوحي الاسم، يتم استخدام موجه الأوامر لإصدار الأوامر المختلفة للنظام.

بدء استخدام وحدة PyGame البرمجية

لقد استخدمت سابقًا وحدة **tkinter** للرسم. سترى الآن كيف يمكنك استخدام وحدة **pygame** لإنشاء أشكال هندسية على الشاشة.

لإنشاء شاشتك الخاصة يجب عليك استخدام الأوامر التالية:

إنشاء شاشة	
الأمر	الوصف
`import pygame`	عند استيراد pygame يتم استيراد جميع الدوال المنتمية لهذه الوحدة.
`pygame.init()`	تهيئة جميع دوال pygame التي يتم استدعاءها.
`screen = pygame.display.set_mode((x,y))`	فتح نافذة بالحجم (x,y) وحفظها في متغير اسمه screen.
`colorName = (r,g,b)`	تعريف متغير اللون في نظام RGB.
`screen.fill(color)`	تعبئة الشاشة باللون الموجود.
`pygame.display.update()`	عرض جميع الرسومات التي قمت بها منذ آخر استدعاء.

- يجب استدعاء الدالة **pygame.init()** بعد استيراد وحدة **pygame** وقبل استدعاء أي دالة أخرى، وهذا يؤدي إلى تهيئة pygame لتكون جاهزة للاستخدام.
- لإعداد نافذة خاصة بـ pygame وتشغيلها نحتاج إلى استدعاء الدالة **pygame.display.set_mode()** وذلك لتحديد حجم النافذة التي نريد إنشاءها.

تستخدم نافذة Pygame نظام إحداثيات محدد بالبكسل. تعمل جميع وحدات البكسل معًا لعرض الصورة التي نراها، فالنافذة المعروضة لها عرض Width بعدد x pixels وارتفاع Height بعدد y pixels.

بمجرد تثبيت حزمة **pygame** فإنها تكون جاهزة للاستخدام.

في المثال التالي سننشئ شكلًا بناءً على خطوط، ستلاحظ بعض الاختلافات عن الطريقة التي استخدمنا بها الألوان في هذه الوحدة، ففي البداية سنعرف الألوان ثم نستخدمها كمعاملات.

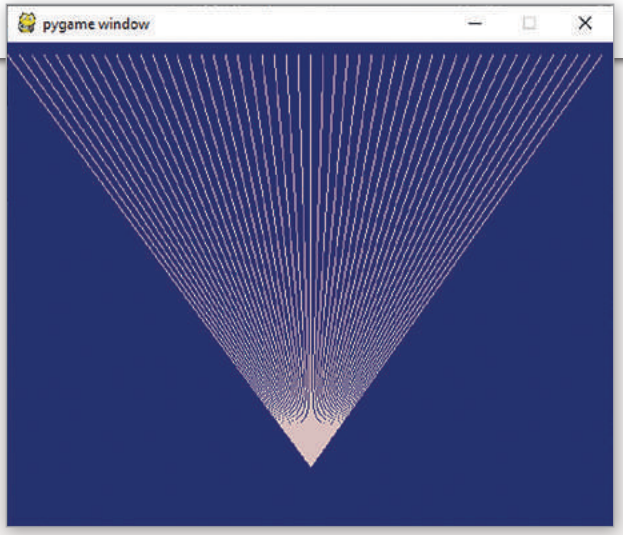
تعريف ألوان البرنامج وفق نظام ألوان **RGB**.

نقطة البداية في رسم الخط.

نقطة النهاية في رسم الخط.

```
import pygame
pygame.init()

darkBlue = (0,0,128)
pink = (255,200,200)

screen = pygame.display.set_mode((500,400))
screen.fill(darkBlue)

for i in range(0,500,10):
    pygame.draw.lines(screen, pink , False, [(i,10), (250,350)], 1)

pygame.display.update()
```

pygame.draw.lines (screen, color, closed, pointlist, thickness).



لا تظهر التغييرات التي تجريها على الشاشة فورًا، فدالة **pygame.display.update()** تطبق ميزة تسمى التخزين المؤقت المزدوج (**double buffering**)، وهي إحدى ميزات **pygame** التي تتيح لك إجراء الكثير من التغييرات على الشاشة ثم إظهارها جميعًا معًا كإطار واحد، أما إذا كان هناك حركة سريعة فإن الشاشة "ستومض" وسيكون ذلك مزعجًا.

عرض صورة بواسطة PyGame

يمكننا باستخدام **Python** تحميل وعرض الصور في تطبيقنا لإنشاء البيئة الخاصة بنا، وتوجد هناك مجموعة متنوعة من الطرق لعرض الصور الرسومية حسب الغرض من المشروع. سنستخدم وحدة **pygame** في هذا المشروع.

يجب أن نستخدم الأوامر التالية لإنشاء نافذة جديدة:

```
#import the pygame module
import pygame

pygame.init()

#create the window
window=pygame.display.set_mode((1200,800), 0,32)
```

أبعاد النافذة.

لتحميل الخلفية في النافذة يجب أن تقوم بإضافة الأوامر التالية إلى الكود البرمجي:

الأمر	الوصف
`background=pygame.image.load("file name").convert()`	تحميل صورة جديدة من ملف.
`window.blit(background,(x,y))`	وضع صورة داخل صورة أخرى
`pygame.display.update()`	تحديث أجزاء من الشاشة لعرض البرامج.

من المهم أن تحفظ الملفين في نفس المجلد لتحميل صورة في ملف برمجي بلغة Python.

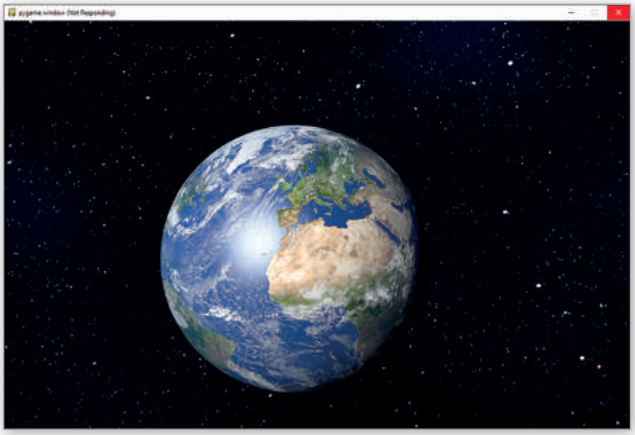
إضافة صورة الخلفية وكائن الخلفية:

- ضع في اعتبارك أنه من أجل تعيين صورة كخلفية، عليك تحديد موضعها عند النقطة (0،0) لملء النافذة بالكامل.
- أيضًا، عند تحميل صورة "**Earth**"، سيظهر العالم بخلفية بيضاء، ولجعل هذه الخلفية شفافة استخدم الإجراء **convert_alpha()** والذي سيغير تنسيق البكسل الخاص بالصورة بما فيها قيم **alpha** الخاصة بالبكسل.

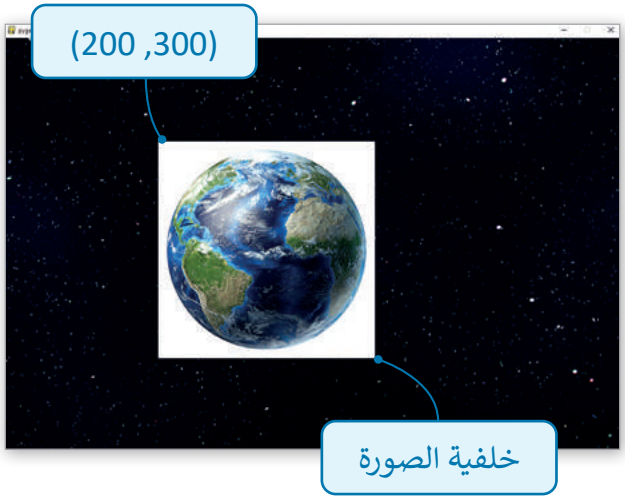
```
#set the "star" image as background object
background=pygame.image.load("stars.png").convert()

#set the "Earth" image as image object
image=pygame.image.load("Earth.png").convert_alpha()

#define the location of the "star" image
window.blit(background,(0,0))

#define the location of the "Earth" image
window.blit(image,(300,200))

pygame.display.update()
```

إزالة خلفية الصورة.

ناتج تنفيذ الكود البرمجي بدون استخدام إجراء **convert_alpha()**



ناتج تنفيذ الكود البرمجي عند استخدام الإجراء **convert_alpha()**



1

أنشئ المقطع البرمجي لكي تجيب عن الأسئلة التالية.

```
from datetime import datetime

odds=[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29,
31, 33,35,37,39,41,43,45,47,49,51,53,55,57,59 ]

right_this_minute = datetime.today().minute

if right_this_minute in odds:
    print("This minute is odd.")
else:
    print("Not odd.")
```

1. المكتبة القياسية Library التي تم استيرادها في البرنامج هي _____ .

2. اسم الوحدة البرمجية Module التي تم استيرادها في البرنامج من المكتبة القياسية هو

_____

3. اسم الدالة Function التي استدعيناها من الوحدة البرمجية هو _____ .

4. وضح وظيفة البرنامج السابق.

_____

_____

_____

5. قم بتشغيل البرنامج وفسر النتائج التي ستحصل عليها.

_____

_____

_____

2

نفذ الكود البرمجي التالي.

```
import sys
print(dir(sys))
```

اشرح الوظيفة التي يقوم بها هذا الكود البرمجي.

3

وضح الفرق بين مكتبات Python القياسية ومكتبات Python الأخرى، واذكر بعض الأمثلة على كل منهما.

4

وضح اثنتين من مميزات إنشاء الكود البرمجي الخاص بك بدلًا من استخدام وحدات برمجية خارجية.

5

أنشئ برنامج يستخدم الدوال التالية:

1. دالة تحسب الجذر التربيعي لرقم يقوم المستخدم بإدخاله.
2. دالة تحسب القيمة التقريبية للجذر التربيعي لذلك الرقم.
3. دالة تطبع رسالة تعرض للمستخدم الرقم الذي قام بإدخاله وجذره التربيعي، والقيمة التقريبية للجذر التربيعي.

على سبيل المثال، إذا كان هناك رقم "30"، فالرسالة التي ستطبع هي "the given number is 30 "، والجذر التربيعي "its root is 5.478" والقيمة التقريبية للجذر التربيعي "round value of the root is 5".

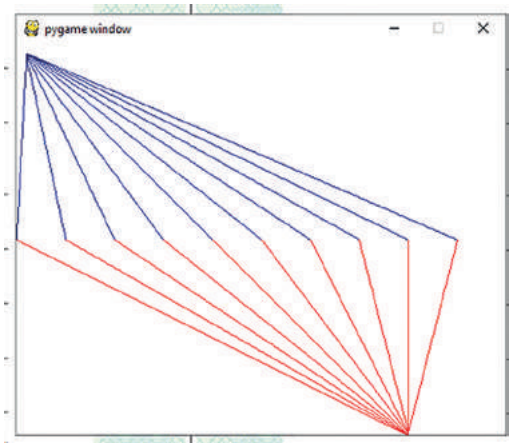
6

اذكر المقصود بالمكتبة البرمجية وعدد خصائصها.

7

املأ الفراغ في البرنامج من أجل إنشاء الأشكال التالية.

```
_________________
pygame.init()

darkBlue = (0, 0, 128)
red = (255, 0, 0)
white = (255, 255, 255)

screen = ___________________.set_mode((500,400))

_________.fill(_______)

for i in range(0,500,50):
    pygame.draw.lines(_______ , _______ , _______ ,
    [(10,10), (i,200)], 1)

for i in range(____,____,____):
    pygame.draw.lines(_______ , _______ , _______ ,
    [(400,400), (i,200)], 1)

_____________________________
```



الدرس الخامس

اختبار البرمجيات

لماذا نقوم بالاختبار؟

قد تشعر عند إكمال كتابتك لبرنامج معين بالثقة من أن كل شيء قد تم تنفيذه كما خططت له، ولكن يجب أن تدرك أن ارتكاب الأخطاء هو طبيعة بشرية وأن البرنامج الأولي قد لا يحقق المستوى المطلوب من الدقة أو التوقعات، ولذلك يجب التحقق من أن البرنامج يعمل كما هو متوقع دون أخطاء أو عيوب.

لا يقتصر الغرض من الاختبار على إظهار أن البرنامج يعمل بشكل صحيح عند إدخال المستخدم لبيانات صحيحة، ولكن أيضًا للعثور على أخطاء لم يتم اكتشافها مسبقًا أو أخطاء تتعلق ببيانات مدخلة غير صحيحة أو غير ذلك ...

التصحيح والاختبار Debugging vs Testing

التصحيح هو عملية إزالة الخلل والأعطال من البرنامج، أما الاختبار فهو عملية التحقق من صحة البرنامج. لذلك، وبدون الاختبار المناسب لا يمكننا المضي قُدمًا في مرحلة التصحيح.

أخطاء برمجية شهيرة

دمرت مجموعة نايت كابيتال **KCG** سمعتها في حوالي 30 دقيقة، ففي 1 أغسطس 2012، قامت خوارزميات التداول في الشركة عن طريق الخطأ بتداول أسهم لحوالي 150 شركة وقامت بشراء تلك الأسهم بأسعار مرتفعة ومن ثم بيعها بسعر منخفض. بحلول الوقت الذي تم فيه إيقاف النظام، كانت مجموعة **KCG** قد خسرت 440 مليون دولار، وكانت الخسارة أربعة أضعاف صافي دخلها السنوي، وفي يوم واحد، فقد سعر سهم الشركة 62% من قيمته السوقية.

في أكتوبر 2005، تم إطلاق سراح 23 سجينًا مبكرًا بسبب خلل في برمجة الحاسوب الخاص بإدارة السجون في ميشيغان.

بدأ مشروع **Mars Climate Orbiter** في 11 ديسمبر 1998. وبسبب خطأ في أنظمة البرمجيات في المحطة الأرضية دخلت المركبة المدارية في الغلاف الجوي للمريخ عند نقطة دخول خاطئة وتفككت بعد 286 يوم من انطلاقها، مما أدى إلى فقدان ناسا (**NASA**) مشروعًا بقيمة 327.6 مليون دولار.

في عام 2003، أثَّر انقطاع التيار الكهربائي في ثماني ولايات أمريكية وكندا على 50 مليون شخص، حيث أدى خطأ برمجي في إحدى عمليات المزامنة إلى تعطل النظام، فكانت النتيجة توقف 256 محطة كهرباء عن العمل.

من الذي يقوم بالاختبار؟

يقوم المبرمج أو المطور باختبار الكود البرمجي الذي يقوم بكتابته في أغلب الأحيان. ولكن لا يعتبر المبرمج الذي كتب الكود البرمجي أفضل شخص يجري الاختبارات، ففي بعض الأحيان قد لا يستطيع رؤية الأخطاء التي ارتكبها. لهذا السبب فإننا نحتاج إلى شخص آخر يسمى المُختبِر (**Tester**) ليقوم بالاختبار مع التركيز على وظائف البرنامج واختبار النتائج من خلال القيام بإدخال مجموعات مختلفة من بيانات الإدخال مثلاً.

يقوم المبرمجون بإجراء الاختبارات الأولية، ولكن المُختبِر هو الذي يحكم بجودة البرنامج وأنه يعمل كما هو متوقع. تضم بعض شركات تطوير البرمجيات قسمًا متخصصًا بالاختبار يطلق عليه عادة اسم قسم ضمان الجودة (**QA – Quality Assurance**)، ويختص بإجراء الاختبارات المختلفة.

لا يضمن الاختبار أن البرنامج صحيح بنسبة 100%، ولكنه يظهر وجود أخطاء بالبرنامج.

1. الأخطاء اللغوية Syntax Errors

تحدث أخطاء بناء الجمل البرمجية عندما يحتوي المقطع البرمجي على خطأ لغوي، فقد يكون الخطأ في بناء الجملة خطأ إملائيًا أو حرفًا مفقودًا أو خطأ في قواعد بناء جمل التحكم. يعتبر الكشف عن هذا النوع من الأخطاء سهلًا حيث أن بيئات تطوير البرمجيات **IDE** الحديثة تظهر على الفور مثل هذه الأخطاء مما يمكن المبرمج من إصلاحها، وعادةً ما تقترح أداة تطوير البرامج نوع الخطأ وكيفية تصحيحه.

تحتوي الجملة التالية على خطأ لغوي، قد لا يكون ملحوظًا ولكن عندما تحاول تشغيل البرنامج سيتم عرض رسالة خطأ تعرض المشكلة.

```
Print("This is a test")
```

```
Traceback (most recent call last):
File "d:/OneDrive/Desktop/G12TCS2/errors.py", line 1, in <module>
    Print("This is a test")
NameError: name 'Print' is not defined
```

للمتابعة، تحتاج إلى تغيير أمر الطباعة **Print** إلى **print** ثم تشغيل البرنامج مرةً أخرى، هذا النوع من الأخطاء شائع في Python عند استخدام الأحرف الكبيرة أو المسافات البادئة خطأً أو نسيان الأقواس الختامية وكذلك عدم وضع النقطتين في نهاية الأوامر الشرطية واستخدام الأوامر من الوحدات أو المكتبات غير المدرجة في البرنامج.

2. أخطاء وقت التشغيل Runtime Errors

تحدث أخطاء وقت التشغيل أو أخطاء التنفيذ أثناء تنفيذ البرنامج. من أكثر أخطاء التنفيذ شيوعًا تجاوز السعة (**Overflow**) والتقريب والاقتطاع (**Rounding and Truncation**) وكذلك الأخطاء الناتجة عن القسمة على صفر. تحدث هذه الأخطاء بسبب أنواع البيانات غير الصحيحة أو نقص الذاكرة أو بسبب خلل في بيانات الإدخال. تظهر أخطاء التنفيذ عادة عند عرض البيانات على الشاشة أو في عملية الطباعة.

التكرار التالي يحتوي على خطأ في التنفيذ حيث أن العدد 1 تم تقسيمه على 0 وبالتالي ظهور خطأ "**division by zero**".

```
for x in range(5,-1,-1):
    print(1/x)
```

```
0.2
0.25
0.3333333333333333
0.5
1.0
Traceback (most recent call last):
File "d:/OneDrive/Desktop/G12TCS2/errors.py", line 2, in <module>
    print(1/x)
ZeroDivisionError: division by zero
```

يحدث الخطأ التنفيذي المعتاد عند إدخال المستخدم لبيانات ليست هي المتوقعة من البرنامج.

```
number = int(input("Enter a whole number: "))
print(number)
```

إذا أدخل المستخدم 5، فكل شيء على ما يرام، ولكن إذا أدخل 5.1 فسيحصل على رسالة خطأ في القيمة "**Invalid Value**".

```
Traceback (most recent call last):
  File "d:\OneDrive\Desktop\G12TCS2\errors.py", line 6, in <mod-
ule>
    number = int(input("Enter a whole number: "))
ValueError: invalid literal for int() with base 10: '5.1'
```

على المطور أن يتحقق من كافة البيانات المدخلة لتجنب مثل هذه الأخطاء.

3. الأخطاء المنطقية Logical Errors

تظهر الأخطاء المنطقية عند تشغيل البرنامج فقط حيث أنها أخطاء في التصميم، ويمكنك رؤية نتائج تلك الأخطاء من خلال الحصول على نتائج غير متوقعة أو غير صحيحة، وفي بعض الأحيان يكون من الصعب العثور على تلك الأخطاء وتصحيحها.

عند تشغيل البرنامج التالي، ستحصل على نتيجة ولكنك لن تحصل على متوسط التقدير الصحيح. لا توجد هنا أخطاء في بناء الجملة البرمجية أو في التنفيذ ولكن هناك خطأ في الحسابات وبالتحديد في أولوية القسمة في العملية الحسابية.

```
Grade1 = 18
Grade2 = 19
Grade3 = 19
AverageGrade = Grade1 + Grade2 + Grade3 / 3
print(AverageGrade)
```

```
43.333333333333336
```

عندما يكون البرنامج طويلًا ومعقدًا، قد يكون من الصعب العثور على أخطاء بسيطة ولكنها مهمة مثل هذا الخطأ، ولهذا السبب، يجب اختبار كل دالة أو جزء من البرنامج بدقة خاصة عند وجود خوارزمية تتعلق بحسابات رياضية أو تحويلات نصية أو عمليات تتعلق باتخاذ القرارات.

قم بتجربة هذا البرنامج:

```
price = 0.0
total = 0.0
price = float(input())
while price > 0.0 :
    total += price
print(total)
```

ما هي نتيجة البرنامج؟ على ما يبدو أنه لن يحدث أي شيء بسبب وجود تكرار لا نهائي، ويرجع هذا لوجود خطأين منطقيين هنا:

الأول: إن جملة الإدخال **(Input Statement)** لا توفر أي رسالة للمستخدم لإدخال القيمة المعينة، فالمستخدم لا يدرك أن عليه إدخال السعر، وأن القيمة قد تكون رقمًا عشريًا وأنه عند إدخاله للرقم 0 سينتهي البرنامج بعرض قيمة الإجمالي.

الثاني: لاحظ أن وضع أمر الإدخال في موضع خاطئ في الكود البرمجي يؤدي إلى إنشاء **while** لتكرار لا نهائي.

استراتيجيات الاختبار

ينقسم الاختبار إلى عدة فئات اعتمادًا على مدى تعقيد الكود البرمجي أو التطبيق الذي يتم اختباره. يتم استهلاك معظم وقت الاختبار على أدنى مستوى، وعلى اختبار الدوال البرمجية والبرنامج الرئيس. توجد العديد من استراتيجيات الاختبار المختلفة المستخدمة من مطوري المشاريع الصغيرة والكبيرة.

اختبارات التشغيل الجاف **Dry Run Testing**	اختبارات التشغيل الجاف هو تقنية اختبار يمكن تطبيقها على خوارزميات صغيرة تتضمن متغيرات وبيانات يمكنك اتباعها باستخدام جدول التتبع. في هذه الطريقة تقوم بتتبع منطق البرنامج حيث يقوم الحاسوب بتنفيذ كل عبارة في الكود وتسجل في جدول التتبع قيمة كل متغير. تتطلب هذه الطريقة التأني والدقة وتستخدم عادةً عندما تحصل المتغيرات على قيم غير صحيحة.
اختبارات قابلية الاستخدام **Usability Testing**	تجرى اختبارات قابلية الاستخدام أو تجربة المستخدم **(UX)** للتأكد من أن البرنامج سهل ومفهوم للمستخدم النهائي.
اختبارات الصندوق الأسود **Black-Box Testing**	اختبارات الصندوق الأسود هي تلك التي تعتبر أجزاء البرنامج التي تختبرها كالصندوق المغلق، فيتم تجاهل الكود البرمجي ويتم التعامل مع بيانات المدخلات والمخرجات فقط بحيث يحصل المختبر على النتائج المتوقعة عند إدخال البيانات، دون النظر في جودة التقنيات المستخدمة في كتابة المقطع البرمجي.

اختبارات الصندوق الأبيض **White-Box Testing**	اختبارات الصندوق الأبيض هي تلك التي تتحقق من كيفية عمل الكود البرمجي بالتفصيل. يشبه اختبار التشغيل الجاف هذا النوع من الاختبارات من حيث كيفية التحقق من دقة المقاطع البرمجية.
اختبارات الوحدة **Unit Testing**	اختبار الوحدة هو عملية اختبار كل دالة من دوال البرنامج بشكل منفصل، للتأكد من صحة عمل الدالة، قبل التحقق الكلي من صحة عمل البرنامج.
اختبارات التكامل **Integration Testing**	تتحقق اختبارات التكامل من كيفية تصرف الأجزاء المختلفة من البرنامج عندما تعمل معًا كنظام متكامل.
اختبارات الأداء **Performance Testing**	تتحقق اختبارات الأداء من أداء البرنامج أو النظام في ظل عمليات تحميل البيانات أو زيادة عدد المستخدمين. سوف تظهر المشكلات التي تحتاج إلى تصحيح لضمان قابلية التوسع.
اختبارات القبول **Acceptance Testing**	ستظهر اختبارات القبول ما إذا كان المنتج يفي بجميع متطلبات المستخدمين على اختلاف احتياجاتهم، ويناسب هذا الاختبار البرمجيات الكبيرة متعددة المستخدمين.
اختبارات الأمان أو الاختراق **Security, Penetration Testing**	تركز اختبارات الاختراق على أمن البرنامج أو النظام. فيتم التحقق من كيفية حماية البرنامج من الهجمات والاختراقات.

لنجرب إحدى استراتيجيات الاختبار، وليكن اختبار التشغيل الجاف، سنقوم بتنفيذ البرنامج التالي ونلاحظ كل متغير أثناء تغير قيمته على جدول التتبع.

```
# Khaled's grades
student1 = { "name":"Khaled",
        "assignment" : [90, 95, 92],
        "test" : [95, 85],
        }

# Sana's grades
student2 = { "name":"Sana",
        "assignment" : [82, 76, 74],
        "test" : [80, 90],
        }

# Function that calculates the average grade
def get_average(marks):
    total_sum = sum(marks)
    total_sum = float(total_sum)
    return total_sum / len(marks)

# Function that calculates the total average grade
def calculate_total_average(student):
    assignment = get_average(student["assignment"])
    test = get_average(student["test"])

# Return the result based on weightage supplied
# 60% from assignments 40% from test
    return (0.6 * assignment + 0.4 * test)
```

```
# Calculate the letter grade of each student
def assign_letter_grade(score):
    if score >= 90: return "A"
    elif score >= 80: return "B"
    elif score >= 70: return "C"
    elif score >= 60: return "D"
    else : return "E"

# Calculate the average marks and letter grade for each student
print("=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=")
averagegrade = calculate_total_average(student1)
print("Average Grade of",student1["name"],"is", averagegrade)
lettergrade = assign_letter_grade(averagegrade)
print("Letter Grade is", lettergrade)
print("=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=")
averagegrade = calculate_total_average(student2)
print("Average Grade of",student2["name"],"is",averagegrade)
lettergrade = assign_letter_grade(averagegrade)
print("Letter Grade is", lettergrade)
print("=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=")
```

أثناء استعراض كل جملة من الكود البرمجي، سيتم ملء جدول التتبع الخاص بالطالب الأول كالتالي.

التقدير بالحروف letter grade	الدرجة score	متوسط الدرجات average grade	الاختبار test	الاختبار Assignment	المجموع الإجمالي Total_ sum	الدرجات marks	اسم الطالب name
					277	[90, 95, 92]	"name":"Khaled", "assignment" : [90, 85, 85], "test" : [95, 85]
					277.0		
				92.333333			
					180	[95, 85]	
			90.0				
	91.4	91.4					
A							

نشاط

لنقم بتصحيح البرنامج التالي والذي تحتاج فيه إلى حساب الوزن الإجمالي لقائمة من المنتجات بالكيلوغرام. يتم إعطاء الوزن لكل منتج بالجرام، يحتوي البرنامج على خطأ تنفيذي واحد وخطأ منطقي واحد على الأقل. استخدم اختبار التشغيل الجاف وجدول التتبع للعثور على الأخطاء وتصحيح البرنامج.

```
#Calculate the total weight of an e-shop order

#Define the constants
#Weight in grams
SMARTPHONE = 400
HEADPHONES = 150
BATTERY = 50

#Read the input from user
print("Enter your order: ")
num_smartphones = int(input("Number of smartphones: "))
num_headphones  = int(input("Number of headphones: "))
num_batteries = int(input("Number of batteries: "))

#Calculate the order weight in grams and kilos
total_weight = num_smartphones*SMARTPHONES + num_
smartphones*HEADPHONES + num_batteries*BATTERY
weight_kilos = float(total_weight/100)

#Display the result
print("The total weight of your order is:",total_weight,"kilos")
```

أكمل جدول التتبع بجميع المتغيرات التي تحتاجها وابدأ عملية التصحيح.

			num_ batteries	num_ headphones	num_ smartphones

التعامل مع الأخطاء Error Handling

تستخدم تقنية معالجة الأخطاء للكشف عن أي مسبب لخطأ في البرنامج. قد يكون من الصعب العثور على جميع الأخطاء في البرنامج رغم تطبيقنا لاستراتيجية اختبار جيدة، حيث أن بعض الأخطاء تتعلق بالأجهزة أو المكونات المادية للحاسوب والتي من المستحيل الحصول عليها خلال عملية الاختبار والتصحيح. قد تؤدي بعض هذه الأخطاء إلى تعطل البرنامج وفقدان البيانات. إن القيام بعملية التعامل مع الأخطاء بالصورة المناسبة يعطي البرنامج القدرة على تجاوز الخطأ وإظهار رسالة إعلام للمستخدم بحدوث الخطأ دون تعطيل البرنامج.

قم بتشغيل البرنامج وأدخل قيمة نصية بدلًا من رقم.

```
age = input("Enter your age: ")
months = int(age) * 12
print("Your age is", months, "months.")
```

سيتعطل البرنامج بسبب وجود خطأ في القيمة، ولكن هناك حل للتعامل مع هذا النوع من الأخطاء. يُطلق المبرمجون على الأخطاء التي تظهر أثناء التنفيذ تسمية **Exception** في البرمجة. يمكن للمبرمج باستخدام تقنية معالجة الأخطاء التحكم في سير البرنامج وعرض الرسائل المناسبة للمستخدم.

يحتوي **Python** على جملة تحكم تسمى **try/except**. عندما نستخدم الكلمة المفتاحية except متبوعة بالخطأ الذي قد ينتج من جملة try، ثم يتبعه تعليمات برمجية معينة.

لتصحيح البرنامج السابق لنقم بإجراء هذه التغييرات على البرنامج:

```
try:
    age = input("Enter your age: ")
    months = int(age) * 12
except ValueError:
    print("I'm sorry. I need a number!")
else:
    print("Your age is", months, "months.")
finally:
    print("Thank you!")
```

إذا أدخلت قيمة خطأ للعمر، يتوقف البرنامج ولن تظهر رسالة خطأ وإنما سيتم التعامل مع الخطأ بشكل صحيح، وسيتم تنفيذ المقطع البرمجي التالي:

```
print("I'm sorry. I need a number!")
```

إذا لم يكن هناك خطأ في البيانات المدخلة، سيتم تنفيذ المقطع:

```
print("Your age is", months, "months.")
```

الجملة البرمجية تحت جملة **finally** ستعمل في الحالتين مع أو بدون خطأ:

```
print("Thank you!")
```

لنقم بتشغيل البرنامج بالقيمتين التاليتين:17 و seventeen.

```
Enter your age: 17
Your age is 204 months.
Thank you!
```

```
Enter your age: seventeen
I'm sorry. I need a number!
Thank you!
```

الاختبارات المؤتمتة Automated Testing

في كثير من الأحيان يكون البرنامج كبيرًا ومعقدًا بوجود العديد من التحديثات والتعديلات التي قد تغير من وظيفته بعد إصداره الأولي، في هذه الحالة، يجب تنفيذ نفس الاختبارات وبعض الاختبارات الجديدة للتحقق من أن البرنامج يعمل بشكل صحيح.

ولكن إذا كانت الاختبارات كثيرة، فسيستغرق إنجازها يدويًا وقتًا وجهدًا كبيرًا. للتعامل مع هذه الحالة يمكن للمُختبر أن ينشئ مجموعة من الاختبارات التلقائية التي يتم تحديثها في كل مرة يتم فيها تغيير البرنامج. في الواقع يقوم المختبِر بكتابة الكود الخاص لاختبار البرنامج الفعلي من خلال عدة أدوات يمكن استخدامها لأتمتة هذه العملية.

اختيار بيانات الاختبار

إن أفضل طريقة لاختبار البرنامج هي "حساب" مخرجات البرنامج قبل تشغيله فعليًا ومعرفة ما إذا كانت النتائج الظاهرة تتطابق مع ما قمت بحسابه يدويًا، وبمعنى آخر، يتم كتابة النتيجة المتوقعة من البرنامج قبل تشغيله ومقارنة نتيجة التشغيل مع المخرجات الحقيقية للبرنامج.

يجب أن يعمل البرنامج بشكل صحيح مهما كانت البيانات التي يُدخلها المستخدم، فإذا تم إدخال بيانات غير صالحة يجب أن يوضح البرنامج أن البيانات غير مقبولة وأن يطلب إدخالًا جديدًا.

في بعض الحالات، تكون البيانات المدخلة صالحة ولكن البرنامج لا يعمل وفق المتوقع، يحدث هذا الأمر عندما لا يأخذ المطور بالحسبان جميع القيم المحتملة للبيانات المدخلة.

لاختبار البرنامج بشكل صحيح، نحتاج إلى اختيار بيانات اختبار تمثل جميع احتمالات الإدخال من المستخدم، تنقسم بيانات الاختبار إلى 3 فئات وهي: بيانات عادية، وبيانات حدية، وبيانات خاطئة.

1. البيانات العادية Normal Data

هي البيانات المستخدمة عادةً عند التعامل مع البرنامج من قِبل المستخدم، وتضم مجموعة القيم من نفس نوع البيانات المتوقعة، فعلى سبيل المثال إذا كان عليك إدخال شهر كعدد صحيح من 1 إلى 12، فإن البيانات العادية هي عدد صحيح من 1 إلى 12.

2. البيانات الحدية أو المتطرفة Boundary or Extreme Data

هي عبارة عن بيانات تقع على أطراف (حدود) نطاق القيم المتوقع، فعلى سبيل المثال إذا كنت تتوقع إدخال سنة ما بين 1900 إلى 2020، فإن القيم المتطرفة هي 1900 و2020، وهكذا فأنت تقوم باختبار البرنامج عند إدخال 1900 أو 2020 كأعداد إلى البرنامج دون حدوث أي أخطاء.

3. بيانات خطأ أو استثنائية Erroneous or Exceptional Data

هي البيانات التي تقع خارج نطاق القيم المتوقعة أو ذات نوع بيانات خطأ. ففي المثال السابق، إذا أدخل المستخدم العدد 0 أو 13 كشهر، أو أدخل كلمة January بدلًا من العدد الصحيح 1، فسيكون هناك خطأ.

يتحقق البرنامج الجيد من صحة جميع البيانات المدخلة، ويجب تحذير المستخدم في حال إدخال بيانات غير صالحة وأن يتيح المحاولة مرة أخرى.

خطة الاختبار Test Plan

خطة الاختبار أو جدول الاختبار هي عبارة عن قائمة بالاختبارات المخطط تطبيقها للتحقق من دقة البرنامج ومن ثَم تسجيل نتائج كل اختبار.

> يتضمن الجدول بيانات الاختبار والغرض من كل اختبار والنتائج المتوقعة والنتائج الفعلية عند تشغيل البرنامج.

> يُطلق على كل صف في جدول الاختبار هذا حالة اختبار (**Test Case**).

> يتحقق سيناريو الاختبار (**Test Scenario**) من صحة جزء محدد من وظيفة البرنامج وقد يحتوي على مجموعة من حالات الاختبار.

> يجب أن تفكر في معايير القبول المتوقعة في كل سيناريو اختبار.

على سبيل المثال، إذا كان سيناريو الاختبار "التحقق من نطاق البيانات الخاصة بحجز أحد الفنادق".

حالات الاختبار ستكون كالتالي:

- "التحقق من تاريخ تسجيل الوصول يقع في المستقبل (بعد تاريخ اليوم)".
- "التحقق من تاريخ تسجيل المغادرة يقع في المستقبل (بعد تاريخ اليوم)".
- "التحقق فيما إذا كان تاريخ تسجيل الوصول يقع بعد تاريخ تسجيل المغادرة"، وأي تغييرات أخرى.

لذلك في حالات الاختبار، ستحاول تطبيق أنواع مختلفة من الحالات المتطرفة.

فإذا كنت تعمل في فريق ولم يكن من يقومون بالاختبار من المشاركين في إنشاء الكود البرمجي، فإن خطة الاختبار هي الأداة التي سيستخدمها المختبرون لعرض النتائج التي بحاجة إلى تصحيح من قِبل المبرمجين. وهنا يكون عمل المبرمجين هو تصحيح الكود وإصلاح كل خطأ.

تبدو وثيقة خطة الاختبار على هذا النحو:

حالة الاختبار	نوع البيانات	الغرض	بيانات الاختبار	النتائج المتوقعة	النتائج الحقيقية	الإجراء المطلوب

يجب عدم الخلط بين حالة الاختبار (**Test Case**) وحالة الاستخدام (**Use Case**)، فكما رأينا سابقًا فإن حالة الاستخدام تحدد كيفية استخدام البرنامج أو النظام لأداء مهمة معينة، وعادة ما تكون عبارة عن مخطط يوضح تسلسل الإجراءات التي سيتبعها المستخدم عند التفاعل مع البرنامج.

لننشئ خطة اختبار خاصة بالكود البرمجي التالي:

```
# calculate the average class grade
total_grades = 0
total_students = int(input("Enter the number of students: "))

for  n in range (1, total_students + 1):
    print("Student #", n)
    student_name = input("Enter the name of the student: ")
    student_grade = input("Enter the grade of " + student_name + ": ")
    total_grades = total_grades + float(student_grade)

average_grade = total_grades / total_students
print("The average grade of the class is ", average_grade)
```

سننشئ 3 سيناريوهات اختبار مختلفة:

- أحدها لاختبار إدخال عدد الطلبة (عدد صحيح).
- أحدها لاختبار إدخال أسماء الطلبة (نص).
- أحدها لاختبار إدخال درجة كل طالب (عدد عشري).

لنفترض أن لدينا فصل دراسي يضم ما يصل إلى 30 طالبًا وأن نظام الدرجات يتراوح بين 0 إلى 20 درجة. بالنسبة لسيناريوهات الاختبار الخاصة بهذه الحالة، فإن حالات الاختبار التي سيتم تطبيقها هي كالتالي:

حالة الاختبار	نوع البيانات	الغرض	بيانات الاختبار	النتائج المتوقعة	النتائج الحقيقية	الإجراء المطلوب
1-1	عادية	اختبار عدد مدخلات الطلبة	3	الاستمرار	الاستمرار	
1-2	طرفية	اختبار أعلى حد	30	الاستمرار	الاستمرار	
1-3	طرفية	اختبار قيم أعلى الحد الأعلى	31	الرفض	الاستمرار	تحقق من البيانات المدخلة
1-4	طرفية	اختبار قيم أعلى الحد الأعلى	99999	الرفض	الاستمرار	تحقق من البيانات المدخلة
1-5	استثنائية	اختبار الصفر	0	الرفض	خطأ قسمة على صفر	التعامل مع الاستثناء
1-6	استثنائية	اختبار أقل من 0	-1	الرفض	مخرج خطأ	تحقق من البيانات المدخلة
2-1	عادية	التحقق من إدخال اسم الطالب	Khaled	الاستمرار	الاستمرار	
2-2	عادية	التحقق من إدخال اسم الطالب	123	الاستمرار	الاستمرار	
2-3	استثنائية	اختبار فارغ	(اضغط فقط **ENTER**)	الرفض	الاستمرار	تحقق من الإدخال
3-1	عادية	التحقق من إدخال عدد الطلاب	19	الاستمرار	الاستمرار	
3-2	طرفية	اختبار الحد الأعلى	20	الاستمرار	الاستمرار	
3-3	طرفية	اختبار الحد الأدنى	0	الاستمرار	الاستمرار	
3-4	استثنائية	اختبار قيم أعلى من الحد الأعلى	21	الرفض	الاستمرار	تحقق من الإدخال
3-5	استثنائية	اختبار قيم أقل من الحد الأدنى	-1	الرفض	الاستمرار	تحقق من الإدخال

الهدف من الاختبار هو إنشاء حالات خطأ عن قصد باستخدام بيانات صالحة وأخرى غير صالحة، وغالبًا ما يتم التخطيط لسيناريوهات الاختبار وحالات الاختبار مقدمًا قبل القيام بالبرمجة الفعلية.

إذا أردنا إحداث خطأ بشكل متعمد في البرنامج فيمكن القيام بإحدى العمليات التالية:

1. إدخال بيانات نصية (نص) عند توقع البرنامج بيانات عددية (رقم).
2. القسمة على صفر.
3. إدخال قيمة خارج النطاق المقبول للقيم.
4. عدم إدخال أي قيمة حين يتوقع البرنامج إدخال قيمة.

تحدث معظم الأخطاء بسبب إدخال بيانات غير صحيحة من خلال لوحة المفاتيح أو قراءة بيانات غير صالحة من ملف. في الحالتين يجب على المبرمج تطبيق تقنيات التحقق لإظهار رسائل الخطأ المناسبة للمستخدم وتجنب معالجة البيانات الخطأ أو تعطل البرنامج وفقدان البيانات.

توثيق عملية الاختبار Test Documentation

تحتاج عملية الاختبار إلى توثيق دقيق للاستفادة منها في اختبارات الإصدارات التالية، وتتضمن وثائق الاختبار ما يلي:

- سياسة الاختبار - وصف المبادئ والأساليب وأهداف الاختبار.
- خطة الاختبار - وصف البرنامج ووظائفه وأجزائه التي يجب اختبارها ونطاق الاختبارات.
- مواصفات الاختبار - تفاصيل كل سيناريو اختبار ومعايير التقييم الخاصة به.
- وصف الاختبار - بيانات الاختبار وإجراءات كل حالة اختبار.
- تقرير تحليل الاختبار - نتائج كل سيناريو اختبار.
- تقرير الخلل - تقرير عن أي خلل أو خطأ أو مشكلة تم العثور عليها في البرنامج.
- تقرير ملخص الاختبار - التقرير النهائي الذي يلخص عملية الاختبار المكتملة.

الإصدار ألفا Alpha	**Alpha** هو إصدار البرنامج الذي لم يتم اختباره بدقة، وقد يحتوي على أخطاء خطيرة يمكن أن تسبب أعطالاً أو فقدان البيانات. عادة ما يتم اختبار هذا الإصدار من قِبَل المطورين والمختبرين.
الإصدار بيتا Beta	**Beta** هو إصدار البرنامج الذي تم إكماله بالميزات ولكنه قد يحتوي على عدد صغير من الأخطاء وكذلك على بعض المشكلات في الأداء. إن محور هذا الاختبار هو تقييم قابلية الاستخدام من المستخدمين الفعليين حيث أن هذا الإصدار يتم إتاحته لعدد صغير من المستخدمين المحتملين خارج شركة التطوير. يمكن أن يكون الإصدار التجريبي مفتوحًا و متاحًا للجمهور أو مغلقًا باستثناء عدد محدود من المستخدمين. تستخدم إصدارات بيتا بشكل عام لأغراض التسويق وللترويج للمنتج النهائي.
الإصدار المرشح Release Candidate	الإصدار المرشح هو الإصدار التجريبي الأخير الذي من المفترض أن يكون خاليًا من الأخطاء وجاهزًا للسوق. خلال هذه المرحلة، يقوم المطورون بتحسين الكود دون إضافة أي ميزات جديدة والتحقق من نجاح جميع الاختبارات النهائية.
الإصدار المستقر Stable Release	يعتبر هذا الإصدار هو الإصدار النهائي للبرنامج والجاهز لنشره على الويب أو بيعه كمنتج، ويجب ألا يحتوي على أي أخطاء ويلبي جميع المتطلبات المخطط لها.

1

صِف باختصار ثلاثة من استراتيجيات الاختبار.

2

ما هي تصنيفات بيانات الاختبار؟ متى يستخدم كل صنف من تلك البيانات؟ اضرب بعض الأمثلة لتبرير إجابتك.

3

المقاطع البرمجية التالية غير صحيحة. ابحث عن الأخطاء وحدد نوعها.

1. نوع الخطأ:

```
# I like coconut smoothies
smoothies = ['coconut', 'strawberry', 'tropical', 'banana']
smoothie_with_coconut = [True, False, True False]

i = 0
while i < len(smoothie_with_coconut)
    if smoothie_with_coconut[i]:
        print ('Your', smoothies[i], 'smoothie contains coconut.')
        i = i + 1
```

2. نوع الخطأ:

```
# I like museums
day=input("Enter D for weekday, W for weekend or H for holiday: ")
visitor = input("Enter A for adult or C for child: ")

if (visitor == "A"):
    if (day == "D"):
        ticket = 3
    elif (day == "W"):
        ticket = 5
    else:
        ticket = 5
elif (visitor == "C"):
    if (day == "D"):
        ticket = 0
    else:
        ticket = 2

print('Your ticket price is',ticket, 'QAR.')
```

أعد كتابة المقطع البرمجي بشكل صحيح مع جميع عمليات التحقق من الصحة لتغطية جميع حالات إدخال البيانات.

4

اتبع هذه الخطوات للتحكم في عمل البرنامج وعرض الرسائل المناسبة للمستخدم.

1. حاول تشغيل البرنامج باستخدام المدخلات التالية: 60 ، 92 ، 78. على الرغم من أن هذه البرامج تبدو صحيحة ولكنها ستتعطل مع الإدخال الخاطئ.

```
def assign_letter_grade(score):
    if score >= 90: return "A"
    elif score >= 80: return "B"
    elif score >= 70: return "C"
    elif score >= 60: return "D"
    else : return "E"

score = int(input("Enter your score: "))

lettergrade = assign_letter_grade(score)
print("Letter Grade is", lettergrade)
```

2. قم بتشغيل هذا البرنامج وأدخل نصًا بديلاً عن أحد الأرقام.

3. ما هي التقنية التي يجب أن نستخدمها لإعلام المستخدم بنوع الإدخال المطلوب مع استمرار البرنامج بالعمل؟

4. قم بإجراء التغييرات المناسبة على البرنامج باستخدام التقنية التي ذكرتها في السؤال السابق.

5. اكتب مخرجات البرنامج الجديد لثلاث حالات إدخال مختلفة.

5

اعتمادًا على جدول التتبع أدناه، استمر باختبار التشغيل الجاف للطالب الثاني في النشاط السابق، واستكمل كامل الجدول.

اسم الطالب name	الدرجات marks	المجموع الإجمالي Total_ sum	الواجب Assignment	الاختبار test	متوسط الدرجات Average grade	الدرجة score	التقدير بالحروف letter grade
"name":"Sana", "assignment" : [82, 76, 74], "test" : [80, 90]							

مشروع الوحدة 1

العنوان: اليوم الرياضي.

الوصف:

في مرحلة دراسية سابقة أنشأت برنامج باستخدام دوال جاهزة في لغة بايثون، كانت وظيفته أن يقرأ اسم ورقم اللاعب ونقاطه في المراحل الثلاثة للسباق، ثم القيام بحساب إجمالي نقاط اللاعب. ويترشح المتسابق إلى الدور نصف النهائي للسباق الثلاثي إذا حصل على 15 نقطة على الأقل وفق المعادلة التالية:

TriathlonPoints = 0.6* (SwimmingScore + CyclingScore RunningScore) / 3

الأدوات: لغة برمجة بايثون Python.

خطوات التنفيذ:

- إدخال بيانات اللاعبين الحاضرين فعليًا (لا نعلم مسبقا عددهم).
- عرض اسم ونتيجة جميع المتسابقين المشاركين فعليًا (في أي وقت أثناء المسابقة).
- عرض اسم ونتيجة جميع المتسابقين المتأهلين فقط إلى الدور نصف النهائي للسباق الثلاثي (في أي وقت أثناء المسابقة).
- عرض اسم ونتيجة جميع اللاعبين غير المتأهلين فقط إلى الدور نصف النهائي للسباق الثلاثي (في أي وقت أثناء المسابقة).
- حفظ بيانات اللاعبين الحاضرين فعليًا بالمسابقة والمتأهلين وغير المتأهلين في ملفات (txt) خاصة بهم، لاستخدامها لاحقًا عند الحاجة لمعالجة البيانات.

مشروع الوحدة 2

العنوان: تصميم متجر إلكتروني.

الوصف: قم بتصميم هيكل برنامج متجر إلكتروني باستخدام منهاج البرمجة التركيبية. اكتب البرنامج الرئيس ووظائفه وقم باختبار البرنامج النهائي.

الأدوات: لغة برمجة بايثون Python.

خطوات التنفيذ:

سيقوم البرنامج بمحاكاة الوظائف الرئيسية لمتجر إلكتروني قائم على النصوص: عرض قائمة بالمنتجات المتاحة، وعرض مزيد من التفاصيل وسعر منتج معين، أو إضافة المنتج إلى سلة التسوق، أو عرض محتويات العربة أو مسحها، حساب المبلغ الإجمالي والخروج.

يقدم المتجر الإلكتروني كل يوم خميس خصم 10% على جميع المنتجات.

استخدم وحدة datetime و date.weekday() لمعرفة ما إذا كان العميل سيحصل على الخصم.

استخدم تقنيات الاختبار التي تعرفها للتحقق من دقة البرنامج.

ماذا تعلمت

تعلمت في هذه الوحدة:

- لغات البرمجة، تاريخها وتصنيفاتها ومجالات استخدامها.
- كيف يفهم الحاسوب لغات البرمجة ويتعامل مع أخطائها بواسطة المترجم والمفسر.
- الأدوات المختلفة لتطوير البرمجيات وسياقات استخدامها في المراحل المختلفة لتطوير البرمجيات وإنتاج الحلول البرمجية المختلفة.
- استخدام التصميم الهرمي والبرمجة التركيبية في تحسين الكود أو إنتاج مقاطع برمجية ذات جودة عالية.
- الاستفادة من المكتبة القياسية والمكتبات الخارجية في Python لاختصار الوقت المستغرق في عملية البرمجة.
- اختبار البرمجيات والكشف عن أخطاء البرمجة باستخدام الاستراتيجيات المختلفة للاختبار.

المصطلحات

الدرس 1	لغة التجميع Assembly Language	المترجم Compiler	لغات برمجة عالية المستوى Higher-level Programming Language
	المفسر Interpreter	لغة الآلة Machine language	لغات الجيل الرابع Fourth-generation Language
	الرابط Linker	برنامج تنفيذي Executable Program	

الدرس 2	محرر التعليمات البرمجية Code Editor	بيئة التطوير المتكاملة Integrated Development Environment (IDE)	نظام مدمج Embedded System
	تطبيق للأغراض العامة General-purpose Application	تطبيق للهاتف المحمول Mobile Application	أداة تطوير البرمجيات Software Development Tool
	تطبيق الويب Web Application	إدارة التحكم في الإصدار- الكود المصدري Version Control-Source Code Management	

الدرس 3	الوسيطات Arguments	التصميم الهرمي Hierarchical Design	البرمجة التركيبية Modular Programming
	المعاملات Parameters	مشكلة فرعية Subproblem	إجراء – برنامج فرعي Subroutine

الدرس 4	إعادة استخدام الكود Code Reuse	التوثيق Documentation	دالة Function
	استيراد Import	وحدة برمجية Module	إجراء Procedure
	مكتبة بايثون Python Library	المكتبة القياسية Standard Library	

الدرس 5	الإصدار ألفا Alpha Release	الإصدار بيتا Beta Release	البيانات الحدية أو المتطرفة Boundary or Extreme Data
	بيانات خطأ أو استثنائية Erroneous or Exceptional Data	التعامل مع الأخطاء Error Handling	خطأ برمجي Exception
	الأخطاء المنطقية Logical Errors	الإصدار المرشح Release Candidate	أخطاء وقت التشغيل Runtime Error
	اختبار البرمجيات Software Testing	الأخطاء اللغوية Syntax Error	حالة الاختبار Test Case
	بيانات الاختبار Test Data	سيناريو الاختبار Test Scenario	استراتيجيات الاختبار Testing Strategies

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

علوم الحاسب المستوى الثاني عشر المسار التكنولوجي
الفصل الدراسي الأول 2022 - 2021
كتاب الطالب

تم النشر بواسطة: **دار النشر MM Publications**
www.mmpublications.com
info@mmpublications.com
المكاتب
المملكة المتحدة، الصين، قبرص، اليونان، كوريا، بولندا، تركيا، الولايات المتحدة الأمريكية، الشركات المنتسبة والممثلين في جميع أنحاء العالم.

تم الإنتاج في الاتحاد الأوروبي

الرقم الدولي المعياري للكتاب: **978-618-05-5850-0** N2106002172-18805

ISBN: 978-618-05-5850-0

PUBLISHED BY **MM PUBLICATIONS**