علوم الحاسب

COMPUTER SCIENCE

كتاب الطالب

11

المسار التكنولوجي

الفصل الدراسي الثاني
2020–2021

الطبعة الأولى

binarylogic

binarylogic

علوم الحاسب المستوى الحادي عشر

المسار التكنولوجي

كتاب الطالب / الفصل الدراسي الثاني 2020 - 2021

ISBN: 978-618-05-5243-0

PUBLISHED BY **MM PUBLICATIONS**

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 حقيقية في 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

معالجة البيانات وتخزينها هي أحد الإستخدامات الرئيسة للحاسوب، وعادةً عند كتابة برنامج معين نحتاج لحفظ المدخلات والمخرجات ونتائج بعض التعليمات مؤقتاً لاستعمالها خلال البرنامج. وقد سبق وتعرفنا على مفهوم المتغير الذي يحفظ قيمة واحدة إلا أننا في حالات كثيرة نحتاج إلى التعامل مع عدد كبير من البيانات والتي من الصعب استخدام متغيرات منفردة لحفظها وإنما نحتاج إلى طريقة أخرى للتعامل مع هذه البيانات من خلال استخدام هياكل البيانات.

ماذا سنتعلمْ؟

في هذه الوحِدة سنتعلم:

> تعريف واستخدام القوائم Lists.

> تعريف واستخدام الصفوف Tuples.

> تعريف واستخدام المصفوفات Arrays.

> تعريف واستخدام القاموس Dictionary.

مواضيع الوحدة

> القوائم Lists

> الصفوف Tuples

> المصفوفات Arrays

> القاموس Dictionary

الأدوات

> Python

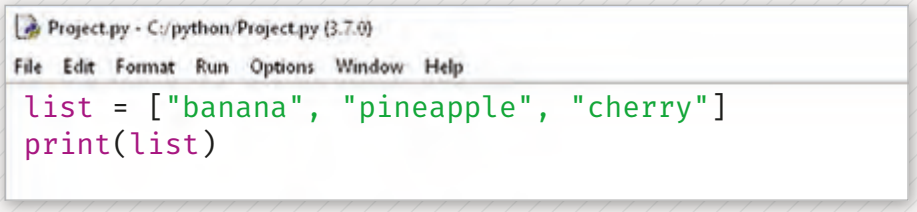
هل تذكر؟

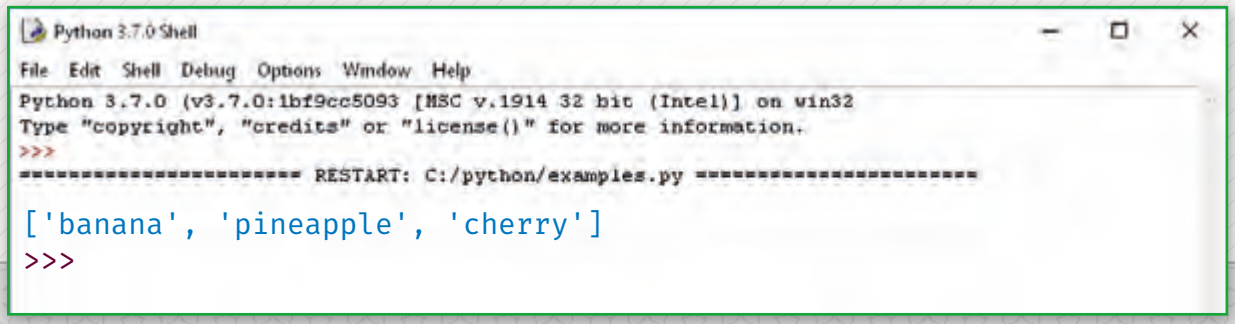
أنواع البيانات

تحتوي كل لغة برمجة تصنيفات لأنواع البيانات التي يمكن استخدامها في تلك اللغة، يحدد نوع البيانات طبيعة القيم بالإضافة للعمليات التي من الممكن القيام بها، يوجد في بايثون Python خمسة أنواع أساسية من البيانات وسيتم التركيز على ثلاثة منها:

- بيانات عددية **Numbers**
- بيانات نصية **String**
- بيانات على شكل قائمة **List**
- بيانات على شكل صفوف **Tuple**
- قاموس بيانات **Dictionary**

بيانات على شكل قائمة List





المتغيرات

المتغير هو اسم رمزي يشير لمكان في ذاكرة الحاسوب لتخزين البيانات أثناء تنفيذ البرنامج. حيث يمكن استخدامه للدلالة على معلومات، وتتغير قيم المتغيرات أثناء تنفيذ البرنامج.

يمكن للمتغيرات أن تمثل أنواعاً عديدة من البيانات.

الأرقام
(متغيرات عددية)

```
Level=3
Score=1200
TotalAmount=120.50
```

النصوص
(متغيرات نصية)

```
Message="Do you want to play again Y/N?"
MyName="Khaled"
EmailAddress="Khaled@education.qa"
```

عند استخدام المتغيرات النصية عليك دوماً كتابة النص بين علامتي التنصيص " ".

الإدخال input

عند استخدام أمر الإدخال سيكون مطلوبًا من المستخدم أن يكتب نصًا معينًا ثم يضغط على مفتاح Enter ↵.

Project.py - C:/python/Project.py (3.7.0)

File Edit Format Run Options Window Help

```
print("Please Enter x value: ")
x=input()
print("The x value is: ",x)
```

File Edit Shell Debug Options Wi

```
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 ==========
Please Enter x value:
10
The x value is: 10
>>>
```

القوائم Lists

هياكل البيانات

حين يتعامل البرنامج مع بيانات كثيرة فان طريقة تخزينها تؤثر على كفاءة البرنامج ولذلك من الضروري تخزينها بشكل يجعل الوصول إليها سريعاً. ولهذا يتعين على المبرمج اختيار هيكل البيانات المناسب للمشكلة التي يكتب برنامج لحلها.

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

القائمة List

هي أكثر هياكل البيانات شيوعاً في بايثون وقد سبق لك التعرف عليها واستخدامها بشكل بسيط. في هذا الدرس سندرسها بشكل أكثر تعمقاً ونتعلم كيفية استخدامها في البرامج لحل المشكلات.

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

الصيغة العامة لتعريف القائمة

يتم تعريف القائمة بالصيغة التالية :

```
List_Name=[item1,item2,...,item n]
```

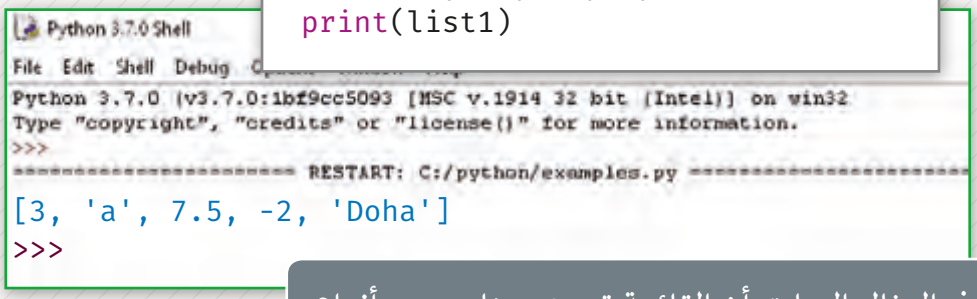
متغير يمثل اسم القائمة

عناصر القائمة

حيث يتم وضع عناصر القائمة بين القوسين المربعين [ ] , ويفصل بين كل عنصر وآخر بالفاصلة.

مثال 1

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

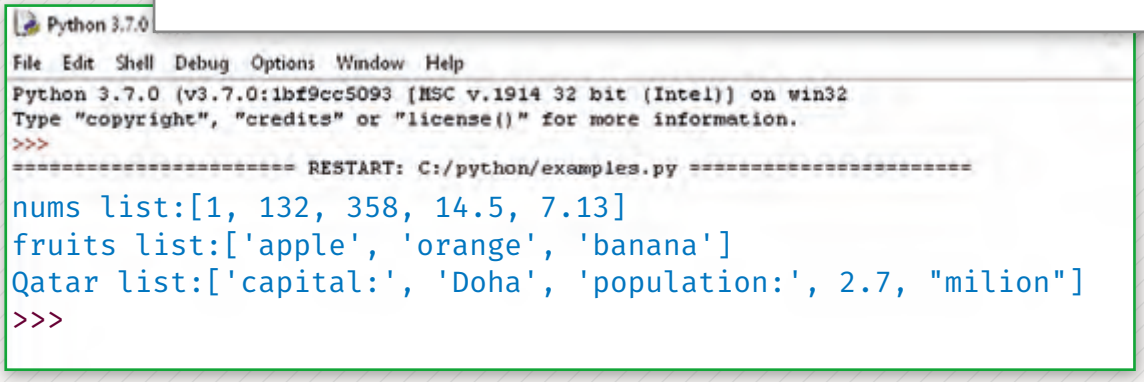


لاحظ في المثال السابق أن القائمة تحوي عناصر من أنواع مختلفة: أعداد صحيحة وحقيقية وحرف ونص.

مثال 2

إنشاء القائمة

```
nums=[1,132,358,14.5,7.13]
print("numbers list:",nums)
fruits=["apple","orange","banana"]
print("fruits list:",fruits)
Qatar=["capital:","Doha","population:",2.7,"million"]
print("Qatar list:",Qatar)
```

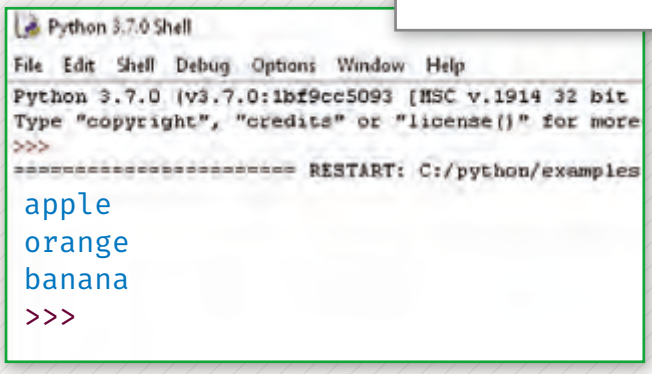


لاحظ في المثال السابق أنه يتم طباعة القائمة بجميع عناصرها باستخدام جملة print واحدة.

ولطباعة عناصر القائمة عنصراً بعد الآخر يمكن استخدام جملة For كما في المثال التالي:

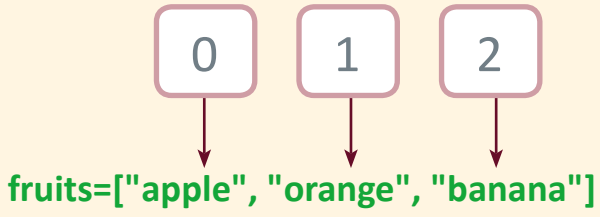
مثال 3

```
fruits=["apple","orange","banana"]
for f in fruits:
    print(f)
```



الوصول إلى عناصر القائمة

يمكن التعامل مع كل عنصر من عناصر القائمة كعنصر قائم بذاته وذلك بطباعته أو تغيير قيمته أو استخدامه في عمليات أخرى. كل عنصر في القائمة له رقم تسلسي **(index)** يمثل موقعه في القائمة ويبدأ ترقيم المواقع في القائمة بالصفر.



نظام ترقيم القائمة يبدأ من 0 وليس من 1.

وللتعامل مع عنصر معين في القائمة نستخدم اسم القائمة متبوعاً بقوسين مربعين وبينهما رقم موقعه في القائمة، فمثلاً لطباعة العنصر الثالث في القائمة fruits (رقم موقعه 2 في القائمة):

```
print(fruits[2])
```

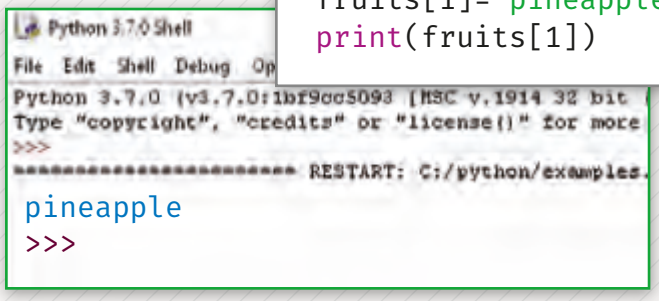
في المثال التالي نستخدم رقم موقع العنصر لاستبدال عنصر في القائمة بعنصر آخر مباشرة أو عن طريق إدخاله من المستخدم.

مثال 4

fruits[0]	apple	0
fruits[1]	orange	1
fruits[2]	banana	2

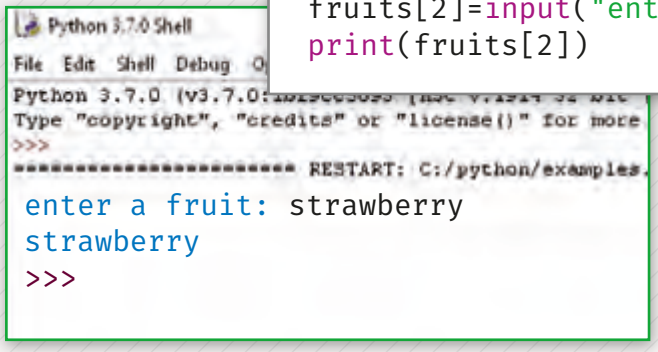
لتعيين قيمة إلى موقع محدد داخل القائمة نستخدم الأوامر التالية:

```
fruits=["apple","orange","banana"]
fruits[1]="pineapple"
print(fruits[1])
```



يُمكننا استخدام أمر الإدخال لإضافة وتعيين قيمة.

```
fruits=["apple","orange","banana"]
fruits[2]=input("enter a fruit:")
print(fruits[2])
```

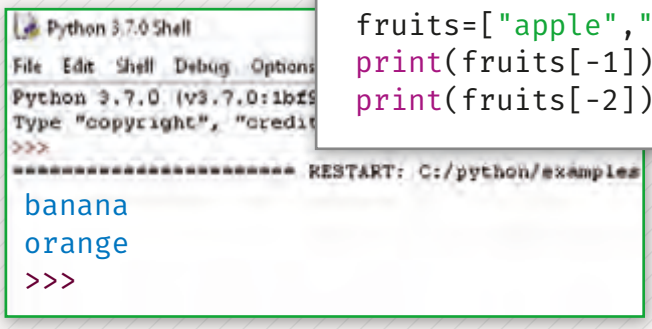


تسمح لنا لغة Python باستخدام الأرقام السالبة لترقيم مواقع العناصر داخل القوائم، فالموقع رقم 1- يشير إلى آخر عنصر في القائمة، بينما 2- يشير إلى العنصر قبل الأخير في القائمة وهكذا.

مثال 5

fruits[-3]	apple	-3
fruits[-2]	orange	-2
fruits[-1]	banana	-1

```
fruits=["apple","orange","banana"]
print(fruits[-1])
print(fruits[-2])
```

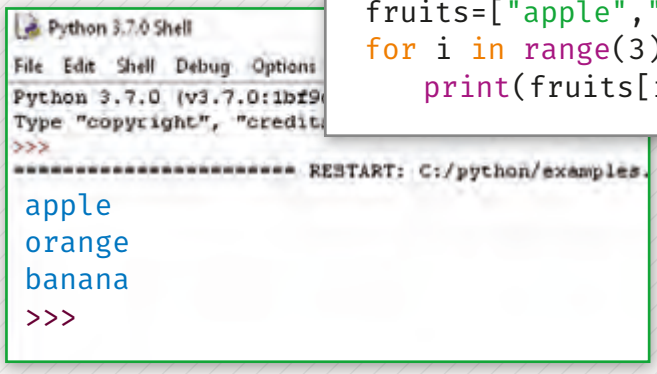


مثال 6

يُمكننا أيضًا استخدام تكرار for ورقم الفهرس للتعامل مع عناصر القائمة.

fruits[0]	apple	0
fruits[1]	orange	1
fruits[2]	banana	2

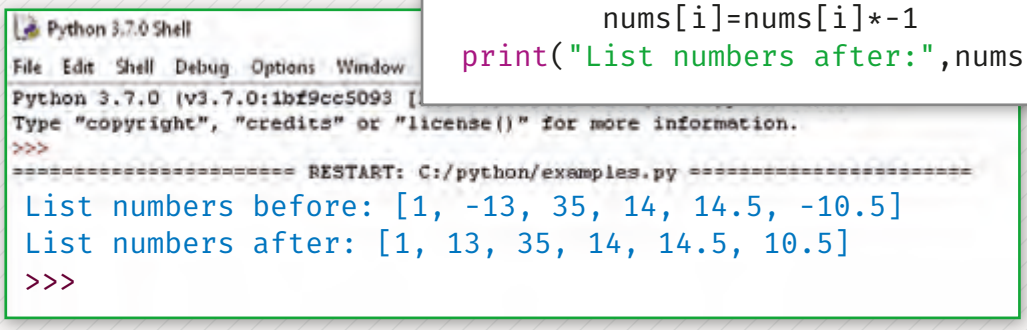
```
fruits=["apple","orange","banana"]
for i in range(3):
    print(fruits[i])
```



مثال 7

قم بكتابة برنامج بايثون يقوم بـ :
> انشاء قائمة Nums التي تحتوي على العناصر التالية: 1,-13,35,14,14.5,-10.5
> تغيير قيم العناصر السالبة إلى موجبة.
> إظهار محتوى القائمة على الشاشة.

```
nums=[1,-13,35,14,14.5,-10.5]
print("List numbers before:",nums)
for i in range(6):
    if nums[i]<0:
        nums[i]=nums[i]*-1
print("List numbers after:",nums)
```



استخدام الدوال مع القوائم:

سبق وتعرفنا على بعض الدوال الجاهزة التي يمكن استخدامها مع القوائم وسنتعلم هنا كيفية استخدامها وكذلك بناء دوال تقوم بنفس الوظيفة لغرض التدريب على استخدام الدوال مع القوائم.

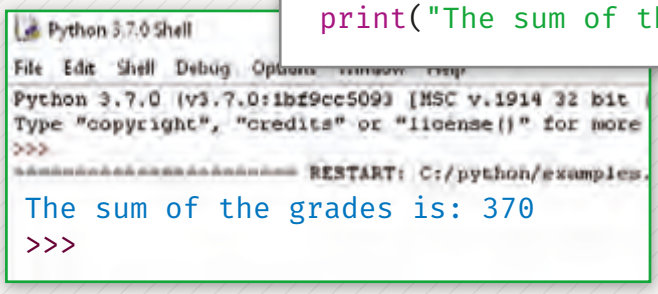
من أجل القيام بحساباتك يمكنك استخدام الدوال الجاهزة التالية:	
len()	عدد العناصر الموجودة في قائمة أو عدد الحروف لمتغير نصي أو عدد الأرقام لمتغير عددي.
sum()	مجموع عدة عناصر.
max()	أكبر عنصر موجود في قائمة.
min()	أصغر عنصر موجود في قائمة.

محاكاة وظيفة الدالة sum()

لكي نتدرب على إنشاء الدوال واستخدامها مع القوائم دعنا الآن نقوم ببناء دالة تقوم بنفس الوظيفة التي تقوم بها دالة **sum()** وهي جمع عناصر القائمة المرسلة للدالة وإعادة النتيجة. سنقوم بتسمية هذه الدالة **mySumGrade**.

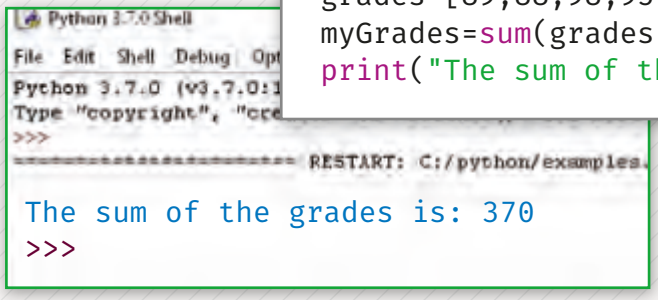
لكي نقوم بعملية الجمع نحتاج إلى متغير جديد يتم فيه عملية تجميع الأرقام ويجب أن تكون القيمة الأولية لهذا المتغير هي صفر. المتغير الذي يستخدم لمثل هذا النوع من العمليات يعرف بـ "المجمع" "**Accumulator**". سنقارن النتيجة التي ستعيدها الدالة التي قمنا بكتابتها مع النتيجة التي تعيدها الدالة **sum()**.

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

#program section
grades=[89,88,98,95]
myGrades=mySumGrade(grades)
print("The sum of the grades is: ",myGrades)
```



فلنقم بمقارنة الدالة التي أنشأناها مسبقًا مع الدالة الجاهزة Sum().

```
grades=[89,88,98,95]
myGrades=sum(grades)
print("The sum of the grades is: ",myGrades)
```



دالة sum() تتعامل فقط مع قيم عددية.

محاكاة وظيفة الدالة ()max

سنقوم بانشاء دالة **myMaxGrade** والتي تستقبل قائمة من الأرقام (الدرجات) وترجع القيمة الأكبر في القائمة وهي نفس الوظيفة التي تقوم بها دالة **()max**. لكي نقوم بهذه الوظيفة فإننا في البداية سنفترض أن القيمة الأولى في القائمة هي القيمة الأكبر ثم نقارنها ببقية القيم، وفي كل مرة نجد قيمة أكبر نحتفظ بها وهكذا حتى الانتهاء من مقارنة جميع القيم.

```
def myMaxGrade (gradesList):
    maxGrade=grades[0]
    l=len(gradesList)
    for i in range(l):
        if grades[i]>maxGrade:
            maxGrade=grades[i]
    return maxGrade

#program section
grades=[89,88,98,95]
maxGrade=myMaxGrade(grades)
print("The maximum grade is:",maxGrade)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Optio
Python 3.7.0 (v3.7.0:1b
Type "copyright", "credits" or "license()" for more
>>>
========== RESTART: C:/python/examples.
The maximum grade is: 98
>>>
```

فلنقم بمقارنة الدالة التي أنشأناها مسبقًا مع الدالة الجاهزة ()max.

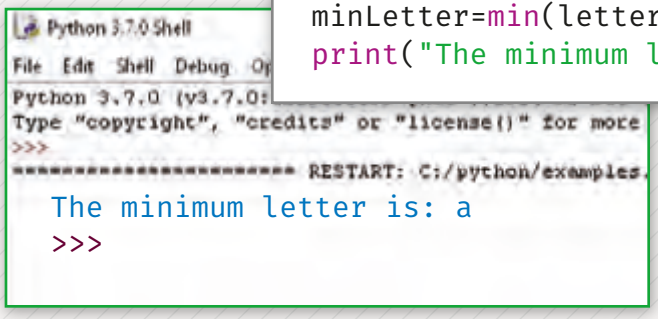
```
grades=[89,88,98,95]
maxGrade=max(grades)
print("The maximum grade is:",maxGrade)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debu
Python 3.7.0 (v3.
Type "copyright",
>>>
========== RESTART: C:/python/examples.
The maximum grade is: 98
>>>
```

مسألة للنقاش

ما الذي سيحدث إذا كان لدينا قائمة من الأحرف؟ ما هو العنصر الأكبر والعنصر الأصغر؟ جرب بنفسك!

```
letters=["b", "a", "x"]
minLetter=min(letters)
print("The minimum letter is:",minLetter)
```



دوال min و max لا تتعامل مع القوائم التي تدمج ما بين الأحرف والأرقام.

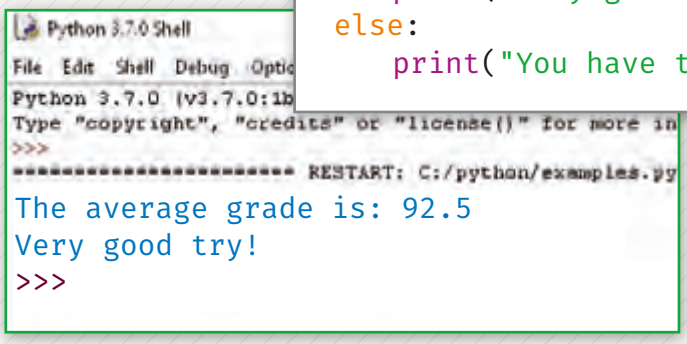
الآن حان الوقت لحل مشكلة.

سنقوم بإنشاء برنامج في Python، فإذا كان لدينا خمس درجات تمثل درجات أحد الطلبة في خمس مواد مختلفة، ونريد أن نحسب متوسط درجاته، فإذا كان هذا المتوسط أكبر من أو يساوي 70 فسيتم عرض الرسالة "Very Good Try!"، وإذا لم يكن كذلك فسيتم عرض الرسالة "You have to try harder".

ما العمليات الحسابية اللازمة لإيجاد قيمة متوسط الدرجات للطالب؟

لطباعة الرسالة المناسبة.

```
def avgFunc (gradesList):
    sumGrade=sum(gradesList)
    l=len(gradesList)
    avg=sumGrade/l
    return avg

#program section
grades=[89,88,98,95]
averageGrade=avgFunc(grades)
print ("The average grade is:",averageGrade)
if averageGrade>70:
    print("Very good try!")
else:
    print("You have to try harder.")
```



مزيد من الدوال الجاهزة المستخدمة مع القوائم:

الدوال المستخدمة مع القوائم	
العملية	الوصف
listName.append(x)	إضافة العنصر x في آخر القائمة.
listName.remove(x)	يزيل العنصر x من القائمة.
listName.count(x)	تحسب عدد مرات ظهور العنصر x داخل القائمة.
listName.sort()	فرز العناصر في القائمة.
listName.reverse()	عكس عناصر القائمة.
listName.clear()	حذف جميع العناصر من القائمة.

في Python تتصل جميع العمليات بكائن محدد، يُمكن للعملية تغيير البيانات التي تحتوي عليها.

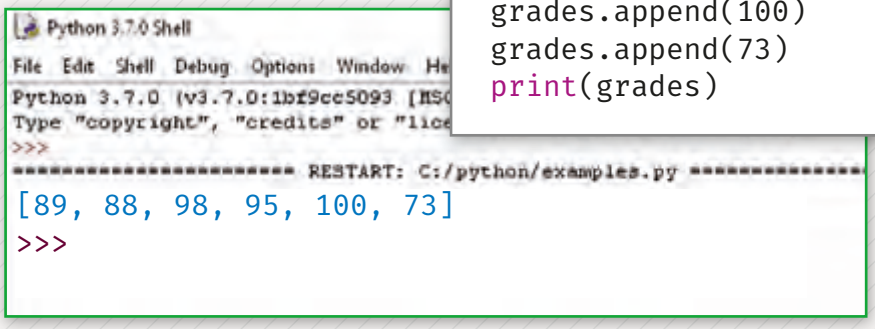
عملية **append()** تضيف عنصرًا جديدًا إلى نهاية القائمة.

مثال 8

listName.append(x)

تُنشئ قائمة بالدرجات.

```
grades=[89,88,98,95]
grades.append(100)
grades.append(73)
print(grades)
```



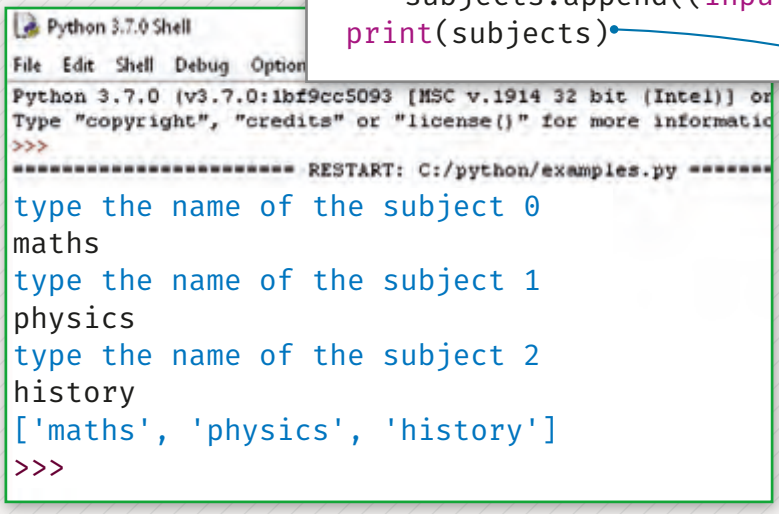
يمكننا استخدام عملية **append()** لإنشاء قائمة كمدخلات من المستخدم، وللقيام بذلك علينا إنشاء قائمة فارغة أولًا.

مثال 9

إنشاء قائمة فارغة.

```
subjects=[]
for i in range(3):
    print("type the name of the subject",i)
    subjects.append((input()))
print(subjects)
```

طباعة القائمة التي قمنا بإنشائها.

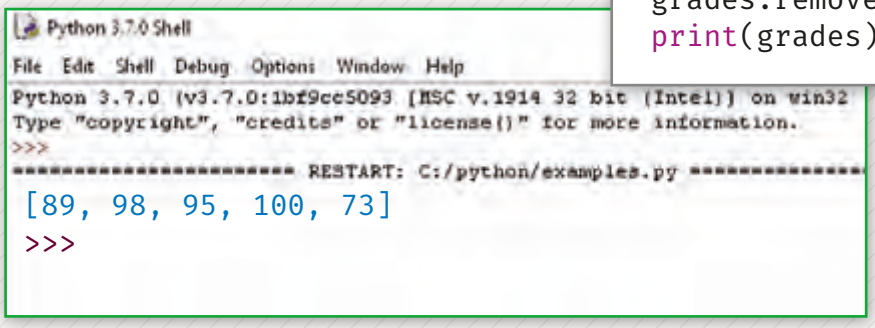


عملية **remove()** تحذف عنصر محدد من القائمة.

مثال 10

listName.remove(x)

```
grades.remove(88)
print(grades)
```



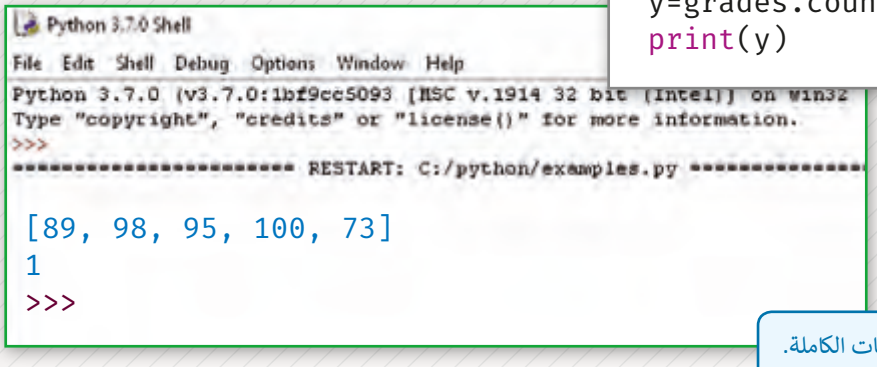
ما الذي يتعين عليك تغييره في المقطع البرمجي لحذف أدنى درجة؟

عملية **count()** تقوم بحساب عدد مرات وجود عنصر محدد داخل القائمة.

مثال 11

listName.count(x)

```
print(grades)
y=grades.count(100)
print(y)
```



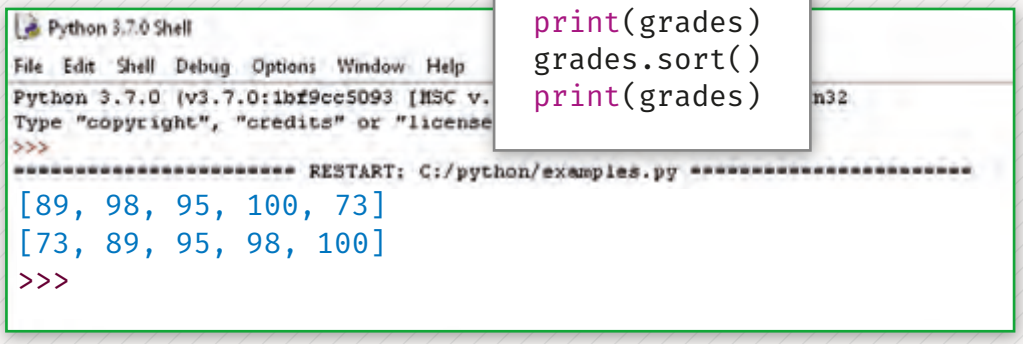
تقوم بحساب جميع العلامات الكاملة.

عملية **sort()** تقوم بترتيب عناصر القائمة تصاعديًا.

مثال 12

listName.sort()

```
print(grades)
grades.sort()
print(grades)
```

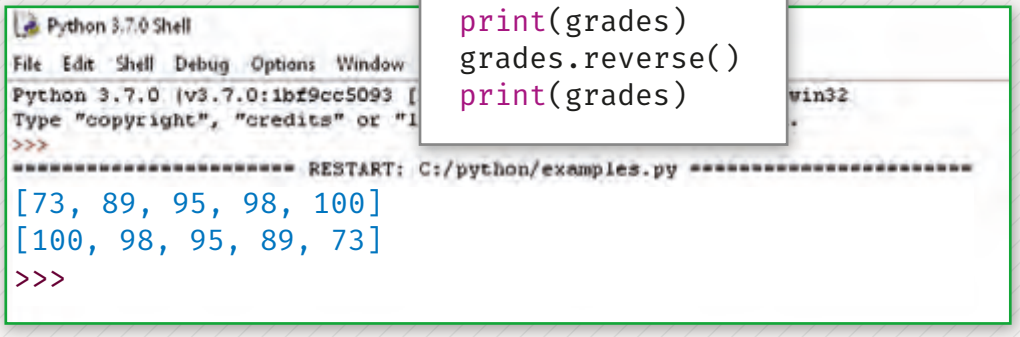


عملية **reverse()** تقوم ترتيب عناصر القائمة عكسيًا.

مثال 13

listName.reverse()

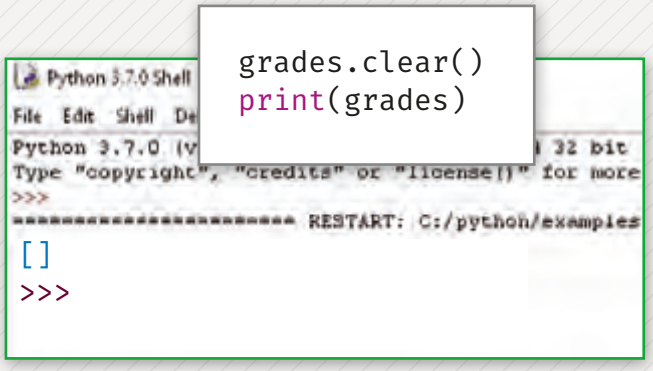
```
print(grades)
grades.reverse()
print(grades)
```



عملية **clear()** تقوم بحذف جميع عناصر القائمة.

مثال 14

```
listName.clear()
```



مثال 15

يعتبر الطالب ناجحًا في اختبارات مادة تكنولوجيا المعلومات لمنتصف الفصل الأول إذا حصل على 20 درجة أو أكثر.

إذا اعتبرنا أن صفك يتكون من عدد N من الطلاب:

قم بكتابة برنامج بلغة بايثون يقوم بـ:

> إنشاء قائمة بأسماء الطلاب.

> إنشاء قائمة بدرجات الطلاب في مادة تكنولوجيا المعلومات.

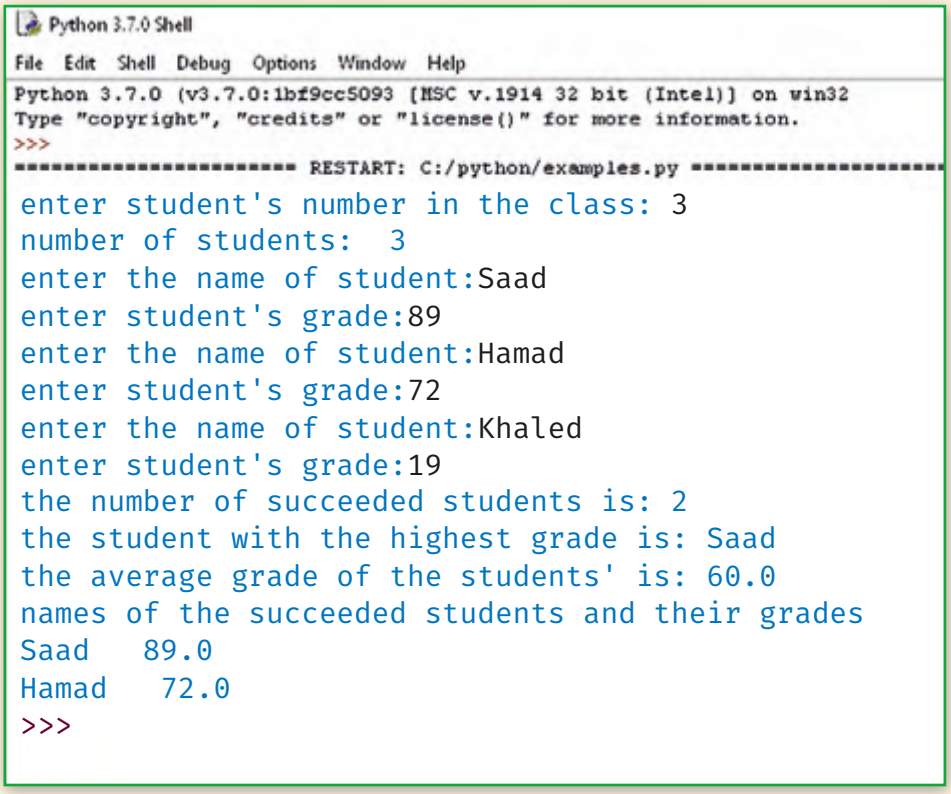
> إيجاد عدد الطلاب الناجحين.

> طباعة اسم أفضل طالب بالصف.

> إيجاد معدل الصف.

> طباعة أسماء الطلاب الناجحون ودرجاتهم.

```
#define the lists students names and students grades
Names=[]
grades=[]

#enter the number of students in the class
N=int(input("enter student's number in the class: "))
print("number of students: ",N)

#enter students' names and their grades
for i in range (N):
    St_Name=str(input("enter the name of student:"))
    St_Deg=float(input("enter student's grade:"))
    Names.append(St_Name)
    grades.append(St_Deg)

#calculation of succeeded students
Nb_s=0
for i in range(N):
    if grades[i]>=20:
        Nb_s=Nb_s+1
print("the number of succeeded students is:",Nb_s)

#search the name of the student with the highest grade
max_d=0
for i in range(N):
    if grades[i]> grades[max_d]:
        max_d=i
print("the student with the highest grade is:",Names[max_d])

#calculation of the average grade of the students
total=0
for i in range(N):
    total=total+grades[i]
Average=total/N
print("the average grade of the students is:", Average,)

#showing the names of succeeded students and their grades
print("names of succeeded students and their grades")

for i in range(N):
    if grades[i]>=20:
      print(Names[i], " ", grades[i])
```


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 ==========
enter student's number in the class: 3
number of students: 3
enter the name of student:Saad
enter student's grade:89
enter the name of student:Hamad
enter student's grade:72
enter the name of student:Khaled
enter student's grade:19
the number of succeeded students is: 2
the student with the highest grade is: Saad
the average grade of the students' is: 60.0
names of the succeeded students and their grades
Saad 89.0
Hamad 72.0
>>>

1

ما المقصود بهياكل البيانات؟

_____

_____

_____

_____

_____

_____

2

أنشيء القائمة التالية: foods =["salad", "fruit", "vegetables", "dairy"]

> اطبع أول وآخر عنصر في القائمة.

> اطبع عدد العناصر داخل القائمة.

3

اختر الإجابة الصحيحة مما يلي وتحقق من إجابتك باستخدام حاسوبك.

1. ترقيم القائمة يبدأ من:	0	○
	1	○
	-1	○

2. الدالة التي تُرجع أدنى عنصر في القائمة هي:	sum()	○
	min()	○
	max()	○

3. الدالة التي تُرجع أكبر عنصر في القائمة هي:	sum()	○
	min()	○
	max()	○

4. لإضافة عنصر إلى القائمة نستخدم الدالة:	sum()	○
	append()	○
	count()	○

4

أنشيء القائمة التالية: colors=["red", "green", "blue"]

استخدم قائمة عمليات Python لتعديل القائمة.

> أضف "white" إلى نهاية القائمة.

> احذف "red" من القائمة.

> أضف "blue" إلى نهاية القائمة.

> قم بحساب عدد مرات وجود كلمة "blue" داخل القائمة.

> قم بترتيب الألوان الموجودة في القائمة.

> احذف جميع عناصر القائمة.

5

اكتب دالة myMinGrade تقوم بمحاكاة الدالة ()min حيث تستقبل قائمة من الأرقام (الدرجات) وتعيد الرقم الأصغر. استخدمها في برنامج وقارن نتيجة الدالة myMinGrade مع الدالة () min.

6

أنشيء قائمة إدخال من قِبل المستخدم بحيث يقوم بكتابة اسمه الأول، واسمه الأخير، وعمره، ثم سيتم عرض هذه القائمة على الشاشة.

7 قم بكتابة برنامج في Python بحيث:

> يقرأ سبع درجات لطالب.

> يطبع عدد الدرجات التي هي أعلى من المتوسط.

> يحذف جميع الدرجات التي أقل من المتوسط.

> يقوم بحساب عدد جميع عناصر القائمة المتبقية بعد إحداث التغييرات السابقة.

8 اكتب برنامجًا يدخل المستخدمه له نتيجة ستة سباحين ثم يُخرج أعلى ثلاث نتائج منها. اكتب الخوارزمية اللازمة للحل وكود Python البرمجي لإنشاء هذا البرنامج.

9 أنشيء قائمة باستخدام لغة Python من خلال الآتي:

> قم بفتح برنامج IDLE(Python 3.7 32-bit).

> افتح ملفًا جديدًا New File.

> احفظ الملف باسم List1 في مكان من اختيارك.

> اكتب برنامج Python يقوم بـ:

- إنشاء قائمة Qatar التي تحتوي على العناصر التالية:

capital, Doha, population, 2.7, million ثم نفذ البرنامج

- اطبع عناصر قائمة Qatar عنصرًا بعد الآخر ثم نفذ البرنامج.

10

أنشيء قائمة باستخدام لغة Python من خلال الآتي:

> افتح ملفًا جديدًا New File.

> احفظ الملف باسم List2 في مكان من اختيارك.

> اكتب برنامج Python يقوم بـ:

- إنشاء قائمة Subject التي تحتوي على العناصر التالية:

Arabic, Math, Social, Biology

- طباعة العنصر الثالث بالقائمة (رقم موقعه 2 في القائمة).

- تعيين English كقيمة للعنصر الثالث بالقائمة.

- استخدام أمر الادخال لإضافة وتعيين قيمة للعنصر الثالث بالقائمة.

11

أنشيء قائمة باستخدام لغة Python من خلال الآتي:

> افتح ملفًا جديدًا New File.

> احفظ الملف باسم List3 في مكان من اختيارك.

> اكتب برنامج Python يقوم بـ:

- إنشاء قائمة fruits التي تحتوي على العناصر التالية: apple ,orange ,kiwi

- اطبع عناصر قائمة fruits عنصرًا بعد الآخر.

- قم بطباعة العنصر الثاني فقط من القائمة.

- قم بتعيين banana كقيمة للعنصر الثاني بالقائمة.

- استخدام أمر الادخال لإضافة وتعيين قيمة للعنصر الثاني بالقائمة.

12

أكمل الجدول التالي بكتابة نتيجة الأمر البرمجي الموجود في العمود الأول، وذلك عند تطبيقه على القائمة C والموضحة أدناه.

C=["red", "orange", "green", "blue", "white"]

كود برمجي	النتيجة
print(C)	
print(C[1])	
print(C[-1])	
print(C[3])	
print(C[0], C[4])	
print(C[0], C[-5])	
C[3]= "red" C[-3]= "blue" C[0]= "green" print(C)	
C[0]= C[4] C[1]= C[-2] print(C)	

13

أكمل الجدول التالي بكتابة نتيجة الأمر البرمجي الموجود في العمود الأول، وذلك عند تطبيقه على القائمتين A و B الموضحة أدناه.

A=[2,4,3,5,0,7,1]

B=[90, -5, 55, -50, 3.14, -90]

كود برمجي	النتيجة
print(sum(A))	
print(sum(B))	
print(max(A))	
print(max(B))	
print(min(A))	
print(min(B))	
print(len(A))	
print(len(B))	
print(sum(A)+ sum(B))	
print(len(A)+sum(B))	

الدرس الثاني

الصفوف Tuples

الصفوف **tuples** هي أحد هياكل البيانات في لغة بايثون وتحوي عددًا من العناصر المتتالية من نفس النوع أو من أنواع مختلفة. وتختلف عن القوائم أنها غير قابلة للتغيير ولذلك نقوم باستخدامها في الحالات التي تكون فيها البيانات ثابتة ونحتاج فقط للوصول لهذه البيانات وليس لتغييرها.

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

الصيغة العامة لتعريف الصف

يتم تعريف القائمة بالصيغة التالية :

```
tuple_Name=(item1,item2,...,item n)
```

متغير يمثل اسم الصف

عناصر الصف

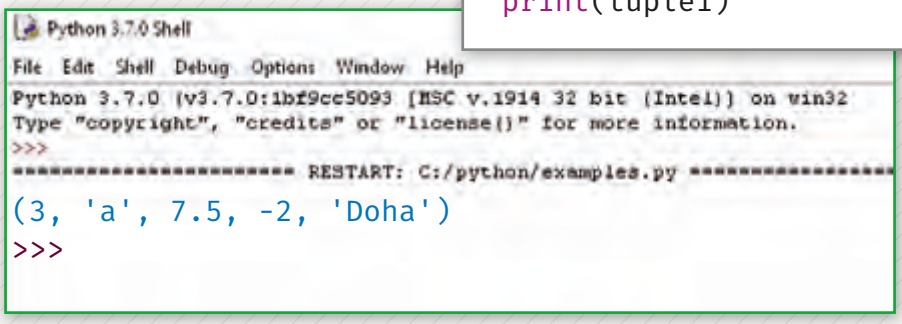
حيث يتم وضع عناصر الصف بين القوسين ( ) , ويفصل بين كل عنصر وآخر بالفاصلة.

كن حذرًا

تتطلب البرمجة قدرًا كبيرًا من الوقت أمام الشاشة، أثناء عملك احرص أن تكون شاشة الحاسوب في خط البصر دون أي تعديل في الوضع المعتاد لعنقك.

مثال 1

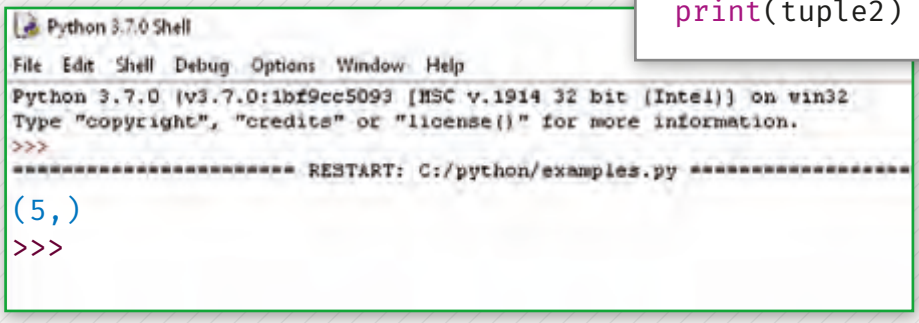
```
tuple1=(3,"a",7.5,-2,"Doha")
print(tuple1)
```



مثال 2

عند تعريف صف يحوي عنصراً واحداً فقط يتم كتابة العنصر بين القوسين ولكن متبوعاً بفاصلة.

```
tuple2=(5,)
print(tuple2)
```



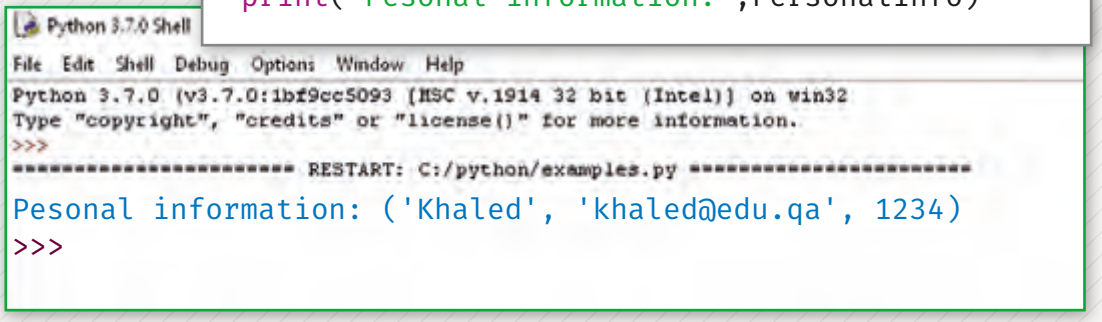
استخدام الفاصلة في هذه الحالة يعتبر أمرًا إجباريًا حتى يفهم مترجم بايثون أنك تنوي تعريف tuple وليس متغير عادي قيمته 5. ولا نحتاج لوضع فاصلة إضافية كما فعلنا هنا في حال كان الصف يحتوي على أكثر من قيمة.

يستخدم الصف غالبًا عندما نريد التعامل مع بيانات لا يتم تغييرها، على سبيل المثال، إذا أردنا تخزين بعض المعلومات الشخصية مثل رقم الهوية أو البريد الإلكتروني للشخص.

مثال 3

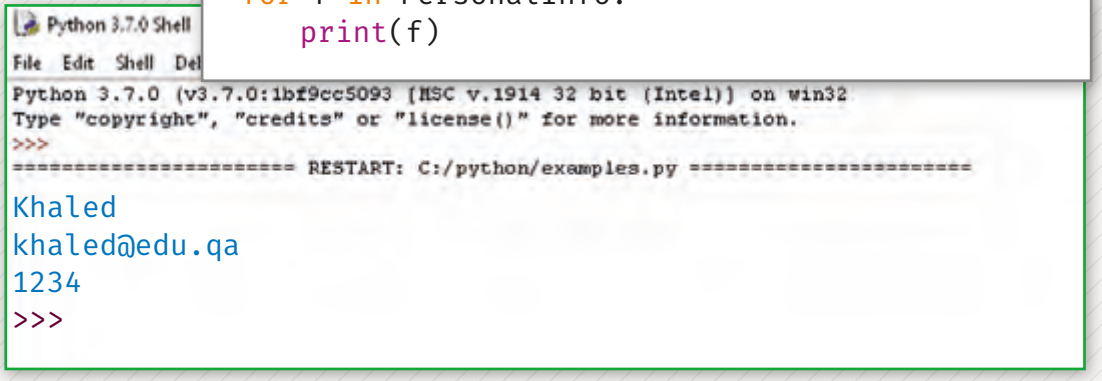
PersonalInfo[0]	Khaled	0
PersonalInfo[1]	khaled@edu.qa	1
PersonalInfo[2]	1234	2

```
PersonalInfo =("Khaled","khaled@edu.qa", 1234)
print("Pesonal information:",PersonalInfo)
```



يمكن طباعة عناصر الصف باستخدام for

```
PersonalInfo =("Khaled","khaled@edu.qa", 1234)
for f in PersonalInfo:
    print(f)
```



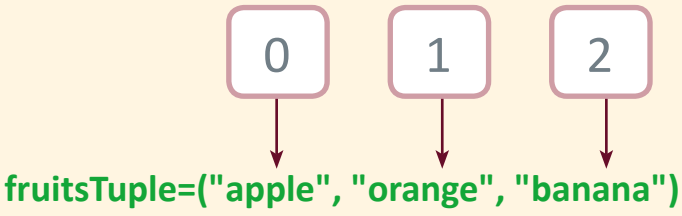
وجه الاختلاف بين القوائم Lists والصفوف Tuples

- يتم تعريف الصفوف بإحاطة العناصر داخل القوسين ( ) بدلاً من [ ].
- تعتبر الصفوف Tuples غير قابلة للتغيير.

الوصول إلى عناصر الصف

كما هو الحال في القوائم, يمكن التعامل مع كل عنصر من عناصر الصف كعنصر قائم بذاته وذلك بطباعته أو استخدامه في عمليات أخرى. كل عنصر في الصف له رقم تسلسي **(index)** يمثل موقعه في الصف ويبدأ ترقيم المواقع في الصفوف كما سبق في القوائم بالصفر.

تذكر أنه لا يمكن تغيير العنصر في الصف كما سبق بالنسبة لعناصر القائمة



وللتعامل مع عنصر معين في الصف **Tuple** نستخدم اسم الصف متبوعاً بقوسين مربعين وبينهما رقم موقعه في الصف، فمثلاً لطباعة العنصر الثالث في الصف **fruitsTuple** (رقم موقعه 2 في الصف):

```
print(fruitsTuple[2])
```

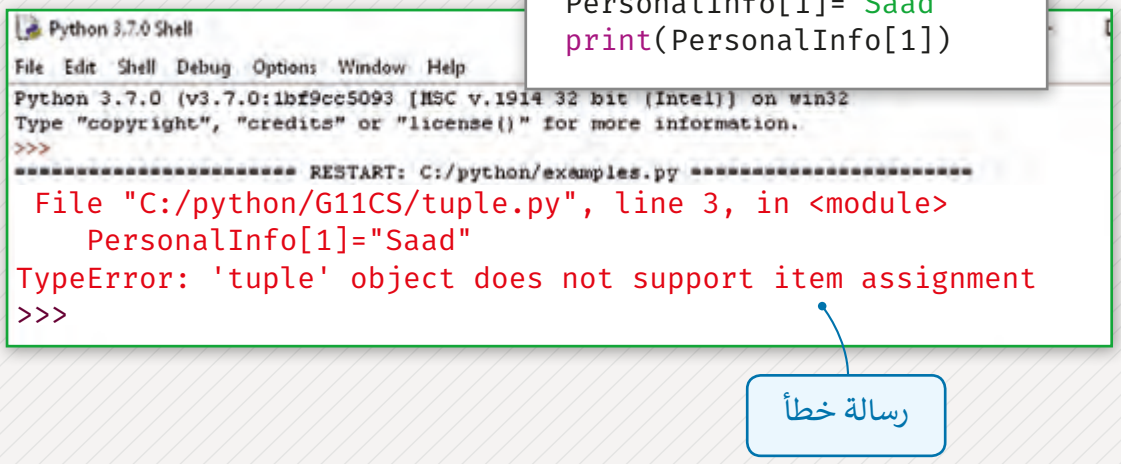
كذلك أيضاً يمكن استخدام الأرقام السالبة لتحديد موقع عنصر معين في الصف كما هو الحال في القوائم فرقم 1- يشير إلى العنصر الأخير ورقم 2- يشير إلى العنصر قبل الأخير وهكذا.

بصفته تركيب بيانات غير قابل للتغيير لا يمكننا إضافة أو إزالة العناصر بعد إنشاء tuple.

مثال 4

PersonalInfo[0]	Khaled	0
PersonalInfo[1]	khaled@edu.qa	1
PersonalInfo[2]	1234	2

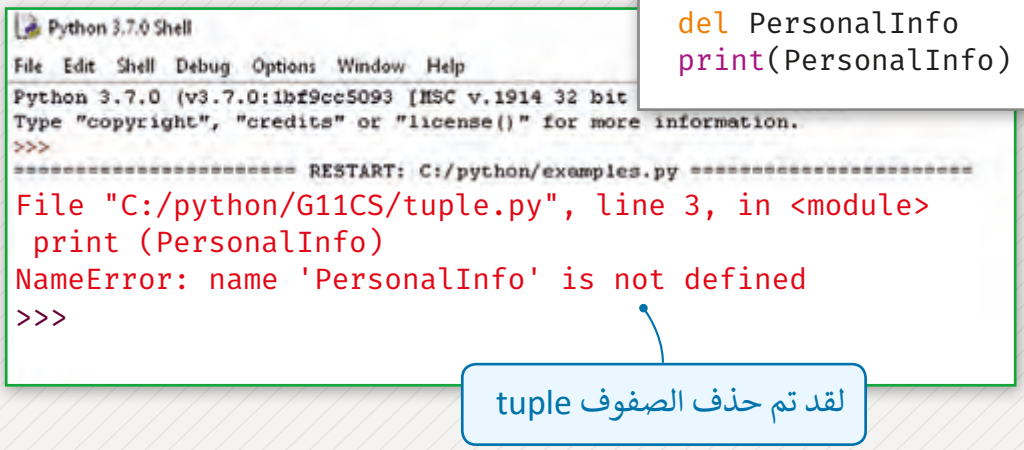
```
PersonalInfo[1]="Saad"
print(PersonalInfo[1])
```



كما أنه من غير الممكن حذف أي عنصر من عناصر **tuple**، ولكن يمكننا حذف المجموعة بالكامل باستخدام الكلمة المفتاحية del.

مثال 5

```
del PersonalInfo
print(PersonalInfo)
```

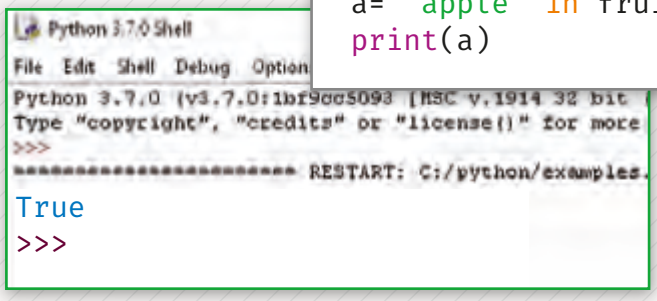


معاملات In و not in

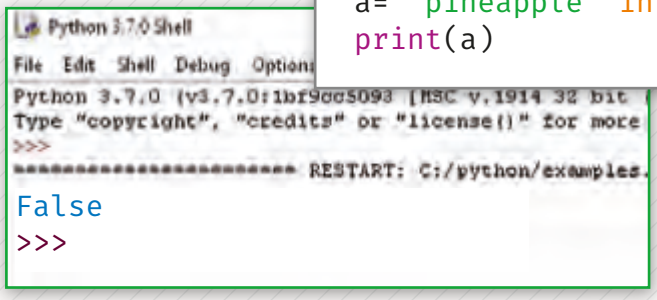
تقدم **Python** معاملات **In** و **not in** التي تتحقق من وجود عنصر في المجموعة، وتكون نتيجة تطبيق هذه المعاملات إما صحيحة أو خطأ.

مثال 6

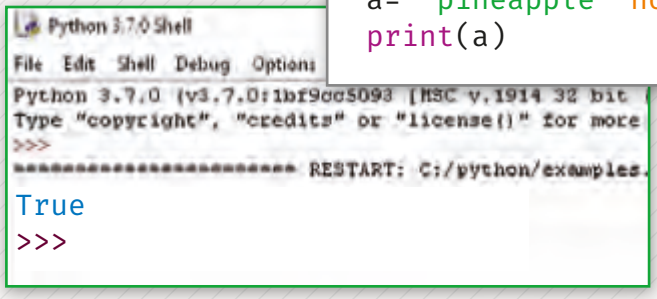
```
fruitsTuple =("apple","orange","banana")
a= "apple" in fruitsTuple
print(a)
```




```
fruitsTuple =("apple","orange","banana")
a= "pineapple" in fruitsTuple
print(a)
```




```
fruitsTuple =("apple","orange","banana")
a= "pineapple" not in fruitsTuple
print(a)
```



نصيحة ذكية

يمكن استخدام معاملات in و not in مع القوائم Lists وليس فقط مع الصفوف، جرب ذلك.

تُقدم لنا **Python** دالتين يُمكننا استخدامهما مع الصفوف **tuples**.

الدوال المستخدمة مع الصفوف	
العملية	الوصف
tupleName.count(x)	تقوم بحساب عدد مرات ظهور العنصر x داخل الصفوف.
tupleName.index(x)	ترجع الموقع الحالي للعنصر x داخل الصفوف.

مثال 7

tupleName.count(x)

fruitsTuple[0]	apple	0
fruitsTuple[1]	orange	1
fruitsTuple[2]	banana	2

```
numOfBanana=fruitsTuple.count("banana")
print("Banana was found in tuple:",numOfBanana, "time(s)")
```

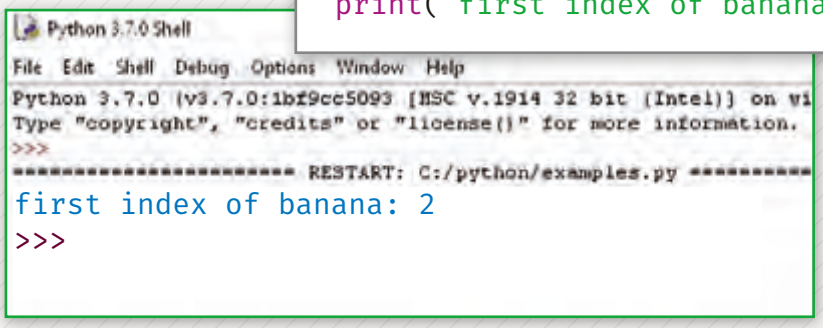
```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.1914 32 bit (Intel)] on vi
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
Banana was found in tuple: 1 time(s)
>>>
```

مثال 8

tupleName.index(x)

fruitsTuple[0]	apple	0
fruitsTuple[1]	orange	1
fruitsTuple[2]	banana	2
fruitsTuple[3]	banana	3

```
pos=fruitsTuple.index("banana")
print("first index of banana:", pos)
```



إذا تكرر نفس العنصر أكثر من مرة، يتم إرجاع الموقع الأول له.

حاول تجربة المقطع البرمجي التالي، ما الذي سيحدث؟

```
pos=fruitsTuple.index("pineapple")
print("index of pineapple:", pos)
```

1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

.1	يجب أن تكون جميع العناصر في الصف Tuple من نفس نوع البيانات.	
.2	الصفوف Tuple هو من نوع هياكل البيانات غير القابلة للتغيير.	
.3	يتم ترقيم مواقع العناصر في الصف بدءًا بالرقم (1).	
.4	يتم إحاطة العناصر داخل الصف Tuple بأقواس مربعة.	
.5	لا يمكننا إضافة عناصر جديدة في Tuple بعد إنشائه.	

2

ما أوجه الاختلاف بين القائمة List والصفوف Tuple؟

_____

_____

_____

_____

_____

3

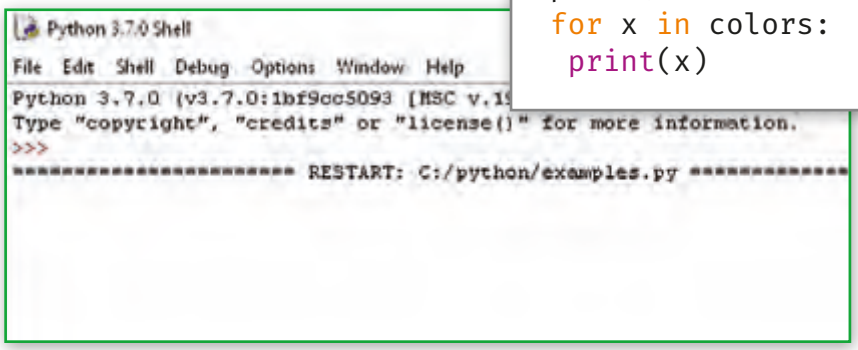
قم بإنشاء tuple باسم Colors يحتوي الألوان red, green, blue, yellow, blue, purple, blue ثم قم باستخدام عمليات الصفوف في Python للقيام بما يلي:

> حساب عدد مرات وجود لون "blue" داخل القائمة.

> طباعة قيمة الفهرس الخاصة باللون "green".

> ما ناتج تنفيذ المقطع البرمجي التالي؟

```
print("My color is:", colors[0])
print("Available colors:")
for x in colors:
 print(x)
```



4

قم بإنشاء لعبة تخمين الحرف بحيث يتوجب على اللاعب تخمين الحرف، يحق للاعب المحاولة ثلاث مرات كحد أقصى لاكتشاف ما إذا كان الحرف موجودًا داخل Tuple أم لا، وستظهر استجابة الحاسوب فقط بكلمة "yes" أو"no"، قم بكتابة كود Python المناسب لإنشاء هذا البرنامج.

5 أنشيء صفًا Tuple باستخدام لغة Python وذلك بتنفيذ الآتي:

> قم بفتح برنامج IDLE(Python 3.7 32-bit)
> أنشيء ملفًا جديدًا New File.
> احفظ الملف باسم Tuple1 في مكان من اختيارك.
> اكتب برنامج Python يقوم بـ:
- إنشاء الصف Info الذي يحتوي العناصر التالية:
اسمك، صفك، سنك، درجة مادة علوم الحاسب.
- طباعة عناصر الصف Info (باستخدام الأمر print فقط).
- طباعة عناصر الصف Info عنصرًا بعد الآخر (باستخدام جملة التكرار For) ثم نفذ البرنامج .

6 أنشيء صفًا Tuple باستخدام لغة Python وذلك بتنفيذ الآتي:

> أنشيء ملفًا جديدًا New File.
> احفظ الملف باسم Tuple2 في مكان من اختيارك.
> اكتب برنامج Python يقوم بـ:
- إنشاء الصف Subject الذي يحتوي العناصر التالية:
Arabic, Math, Social, Biology
- طباعة الأول الثالث بالصف
- تعيين English كقيمة للعنصر الثاني بالصف.
ماذا تلاحظ:.....
> اكتب الأمر البرمجي الذي يمكن من حذف الصف.

7

أنشيء صفًا Tuple باستخدام لغة Python وذلك بتنفيذ الآتي:

> أنشيء ملفًا جديدًا New File.

> احفظ الملف باسم Tuple3 في مكان من اختيارك.

> اكتب برنامج Python يقوم بـ:

- إنشاء الصف degrees الذي يحتوي العناصر التالية: 11 ، 6 ، 12.5 ، 15 ، 2

- طباعة عناصر الصف degrees عنصرًا بعد الآخر.

- طباعة مجموع درجات العنصر الأول والأخير من القائمة.

- احتساب وعرض مجموع عناصر القائمة.

- حساب عدد العناصر التي قيمتها دون الـ 7.5.

- حذف الصف.

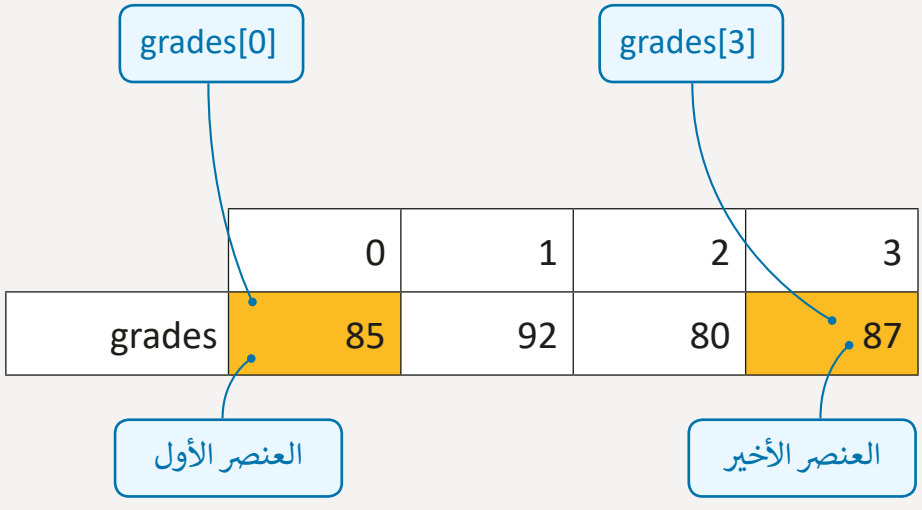
المصفوفات Arrays

توفر جميع لغات البرمجة تقريبًا هيكل البيانات المعروف بالمصفوفة Array، يُستخدم هذا الهيكل لتخزين بيانات من نفس النوع، لذا يمكننا اعتبار المصفوفة كمجموعة متغيرات من نفس النوع.

يستخدم Python القائمة List لتمثيل كل مصفوفة، لذلك فعندما نتحدث عن المصفوفات في لغة البرمجة فإننا نشير فعليًا إلى القوائم، ولكن الفرق الرئيس ما بين المصفوفة والقائمة أن المصفوفات يمكن أن تحتوي فقط على قيم من نفس نوع البيانات، بينما يمكن أن تحتوي القوائم على قيم لأنواع مختلفة من البيانات، أيضًا يمكن أن تكون المصفوفات ذات بعد واحد (1D) أو ثنائية الأبعاد (2D) أو حتى عدد N من الأبعاد.

المصفوفة أحادية الأبعاد (1D)

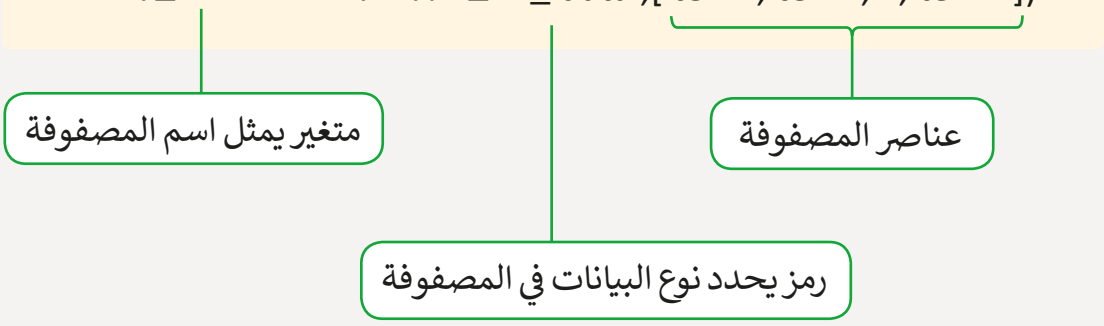
تشبه هذه المصفوفة القوائم، فكل عنصر في المصفوفة لديه فهرس يقوم بتحديد موقعه داخل المصفوفة، ويُمكننا الوصول إلى عناصر كل مصفوفة عن طريق كتابة اسم المصفوفة والرقم التسلسلي للعنصر بين قوسين مربعين.



الصيغة العامة لتعريف المصفوفة

يتم تعريف المصفوفة بالصيغة التالية :

```
Array_Name=array('type_ of_ data',[item1,item2,...,item n])
```



حيث يتم وضع عناصر المصفوفة بين القوسين المربعين [ ] , ويفصل بين كل عنصر وآخر بالفاصلة.

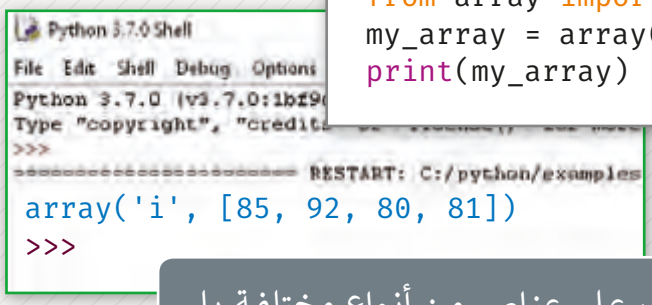
الرموز المحددة لنوع عناصر المصفوفة	
رمز النوع	نوع البيانات
i	(عدد صحيح)
d	(عدد عشري)
u	(حرف أو رمز)

إنشاء المصفوفة

لاستخدام المصفوفة **Array** في **Python**، علينا استيراد الوحدة القياسية Array من المكتبة الخاصة بالمصفوفة، وبما أن المصفوفة تحتوي على عناصر من نفس نوع البيانات، فيجب تحديد نوع البيانات عند إنشائها.

مثال 1

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



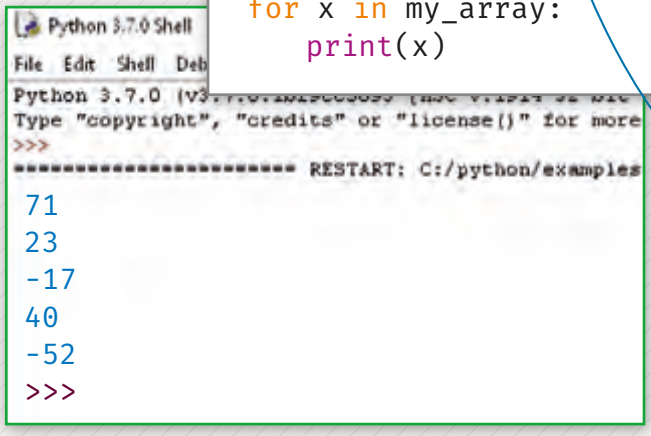
المصفوفة لا يمكن أن تحتوي على عناصر من أنواع مختلفة بل يجب أن تحتوي فقط على قيم من نفس نوع البيانات.

مثال 2

فلننشئ مصفوفة تحتوي على أعداد صحيحة في المثال التالي ونقوم بطباعة عناصرها.

array[0]	71	0
array[1]	23	1
array[2]	-17	2
array[3]	40	3
array[4]	-52	4

```
from array import *
my_array = array('i', [71,23,-17,40,-52])
for x in my_array:
    print(x)
```

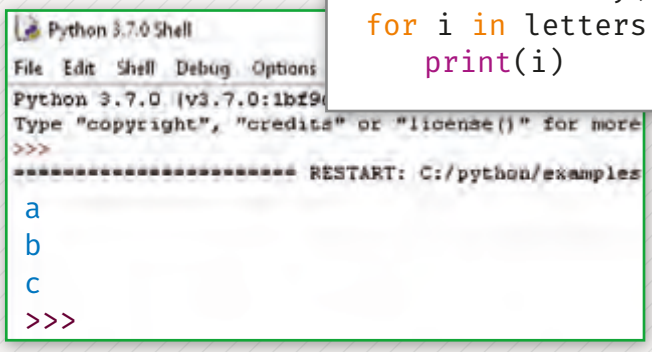


هذه مصفوفة تحتوي على قيم أعداد صحيحة.

مثال 3

فلننشئ مصفوفة تحتوي على أحرف أو رموز في المثال التالي.

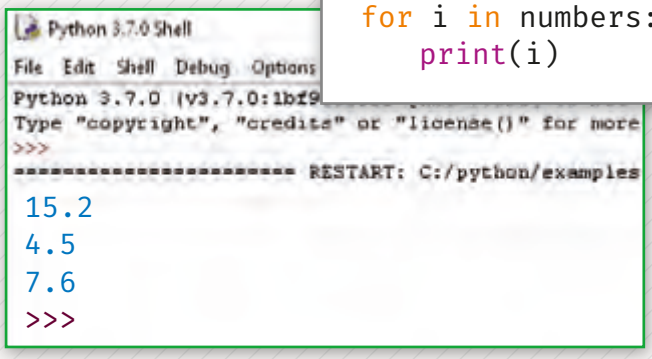
```
from array import *
letters = array('u',['a','b','c'])
for i in letters:
    print(i)
```



مثال 4

فلننشئ مصفوفة تحتوي على أعداد عشرية في المثال التالي.

```
from array import *
numbers=array('d',[15.2, 4.5, 7.6])
for i in numbers:
    print(i)
```



حاول تجربة المقطع البرمجي التالي، هل ظهرت رسالة خطأ؟

```
from array import *
my_array = array('i', [1,2,3.7,4.3,5])
for i in my_array:
    print(i)
```

مثال 5

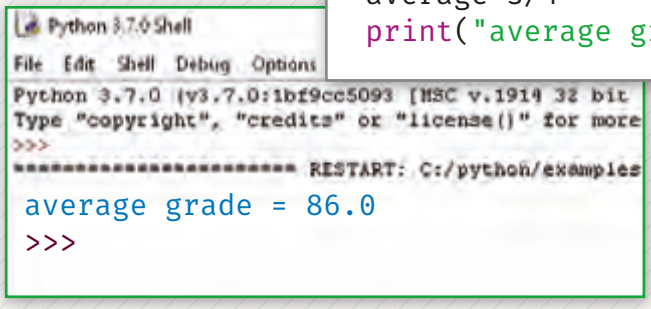
فلننشئ مصفوفة تحتوي على درجات طالب في أربع مواد دراسية.

```
from array import *
grades = array('i', [85,92,80,87])
```

	0	1	2	3
grades	85	92	80	87

سنقوم بحساب متوسط الدرجات، وبما أننا نعرف عدد عناصر المصفوفة فسنستخدم تكرارًا لمعالجة بياناته.

```
from array import *
grades = array('i', [85,92,80,87])
s=0
for i in grades:
    s=s+i
average=s/4
print("average grade =",average)
```



عندما نستخدم في أحد البرامج متغيرًا يضيف قيمًا بشكل متكرر، يُطلق عليه اسم متغير تراكمي accumulator.

عادةً ما يتم تعيين القيمة الأولية للمتغير التراكمي إلى صفر.

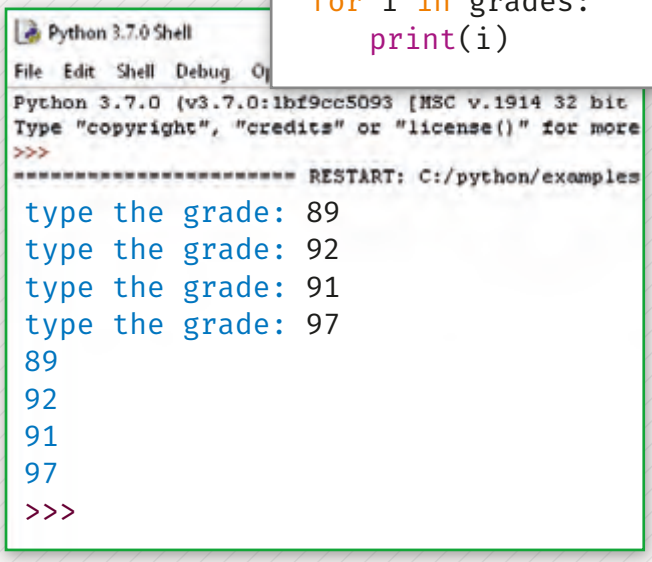
هل يُمكنك التفكير بطرق مختلفة للحصول على نفس النتيجة؟

في الأمثلة السابقة تم تحديد عناصر المصفوفة داخل البرنامج،، سنقوم الآن بإنشاء مصفوفة وإدخال عناصرها من قِبل المستخدم.

مثال 6

```
from array import *
grades = array('i', [0,0,0,0])
for i in range(4):
    grades[i]=int(input('type the grade: '))
for i in grades:
    print(i)
```

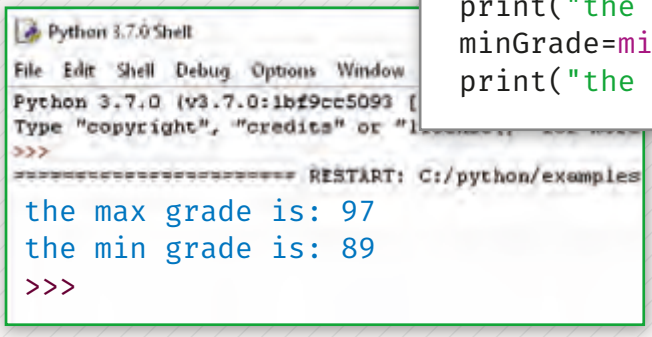
علينا إعادة ضبط قيم المصفوفة لتكون أصفار.



مثال 7

دوال القوائم يُمكن استخدامها أيضًا مع المصفوفات.

```
from array import *
grades = array('i', [89,92,91,97])
maxGrade=max(grades)
print("the max grade is:",maxGrade)
minGrade=min(grades)
print("the min grade is:",minGrade
```



إذا كان لدينا خمس طلبة فسنحتاج إلى خمسة جداول مختلفة.

```
Student_1 = [85, 92, 80, 87]
Student_2 = [93, 85, 89, 76]
Student_3 = [87, 83, 85, 80]
Student_4 = [90, 86, 78, 83]
Student_5 = [91, 96, 84, 92]
```

ما الذي سيحدث في حالة وجود 800 طالب؟
في هذه الحالة سنستخدم مصفوفة ثنائية الأبعاد (2D Array).

المصفوفة ثنائية الأبعاد (2D Array)

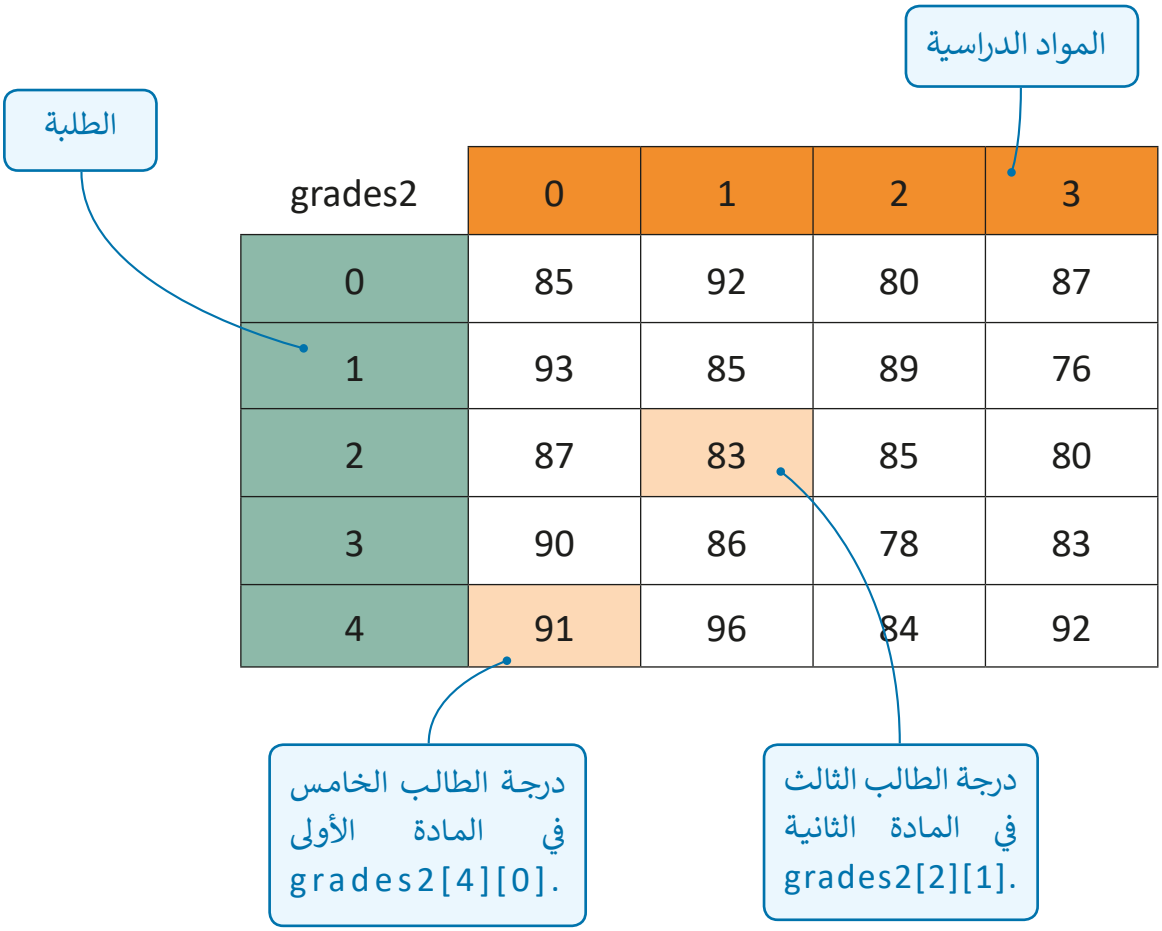
في المصفوفة ثنائية الأبعاد **(2D Array)**، يتم تمثيل البيانات داخل صفوف وأعمدة، وتتم فهرسة كل عنصر في المصفوفة ثنائية الأبعاد بواسطة فهرسين، أحدهما يشير إلى الصف والآخر للعمود.

	العمود 0	العمود 1	العمود 2
الصف 0	x[0][0]	x[0][1]	x[0][2]
الصف 1	x[1][0]	x[1][1]	x[1][2]
الصف 2	x[2][0]	x[2][1]	x[2][2]

مثل المصفوفة أحادية الأبعاد (1D) يجب أن يكون كل عنصر في المصفوفة ثنائية الأبعاد (2D) من نفس النوع.

يمثل كل صف درجة كل طالب في أربع مواد دراسية.

يمثل كل عمود درجات الطلبة في مادة دراسية واحدة.



grades2	0	1	2	3
0	85	92	80	87
1	93	85	89	76
2	87	83	85	80
3	90	86	78	83
4	91	96	84	92

مثال 8

للإشارة إلى عنصر معين في المصفوفة، نستخدم الصف والعمود. اسم المصفوفة [الصف] [العمود]

Array Name [row][column]

```
grades2[3][0]=90
grades2[0][3]=87
```

أكمل الفراغات بدرجة الطالب وفق المصفوفة.

```
grades2[0][2]=________
grades2[1][1]=________
grades2[2][3]=________
grades2[4][1]=________
```

1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

.1	يمكن أن تحتوي المصفوفة على بيانات مختلفة النوع.
.2	يستخدم Python القوائم لتمثيل المصفوفات.
.3	عند إنشاء مصفوفة يجب علينا تحديد نوع البيانات.
.4	تتم فهرسة كل عنصر في المصفوفة ثنائية الأبعاد برقم فهرس واحد.

2

أكمل المصفوفة الخاصة بدرجات الحرارة بالقيم المعطاة.

```
temperature[0]=32
temperature[3]=41
temperature[1]=43
temperature[2]=28
temperature[4]=30
```

	0	1	2	3	4
temperature					

3

أنشئ المصفوفة السابقة لدرجات الحرارة بلغة Python.

> قم بطباعة قيم المصفوفة.

> اجمع درجات الحرارة.

> قم بإيجاد متوسط درجات الحرارة وطباعته.

> استخدم دوال min و max لطباعة الحد الأدنى والحد الأقصى لدرجة الحرارة.

4

أنشئ المصفوفة التالية باسم letters الخاصة بالحروف.

	0	1	2	3	4
letters	Q	A	T	A	R

5

أنشئ مصفوفة خاصة بإدخال حروف اسمك من قِبل المستخدم.

6

اكتب دالة تستقبل مصفوفة أحادية الأبعاد وتُرجع عدد العناصر الموجبة فيها.

7

اكتب برنامجًا في Python يقوم بتخزين أعمار 15 شخصًا في مصفوفة، وعلى البرنامج أن يحسب ويعرض التالي:

> عدد الأشخاص الذين أعمارهم أكبر من 40.

> متوسط الأعمار.

> عدد الأشخاص الذين أعمارهم أكبر من متوسط الأعمار.

8

أكمل المصفوفة ثنائية الأبعاد بالقيم المعطاة.

```
temperature[0][2]=32
temperature[1][1]=42
temperature[2][3]=39
temperature[2][1]=40
temperature[1][3]=43
temperature[0][3]=27
```

temperature	0	1	2	3
0				
1				
2				

الدرس الرابع

القاموس Dictionary

القاموس هو هيكل بيانات قابل للتغير يحوي عددًا من العناصر، كل عنصر في القاموس عبارة عن قيمتين الأولى تمثل المفتاح والثانية تمثل العنصر نفسه. يختلف القاموس عن الهياكل السابقة أن الوصول للعنصر يتم عبر المفتاح وليس عبر رقم الموقع كما هو الحال في القوائم والصفوف والمصفوفات. قيم المفاتيح يمكن أن تكون من أي نوع من أنواع البيانات.

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

الصيغة العامة لتعريف القاموس

```
dictionary_name={key1:item1,key2:item2,...,key n:item n}
```

متغير يمثل اسم القاموس

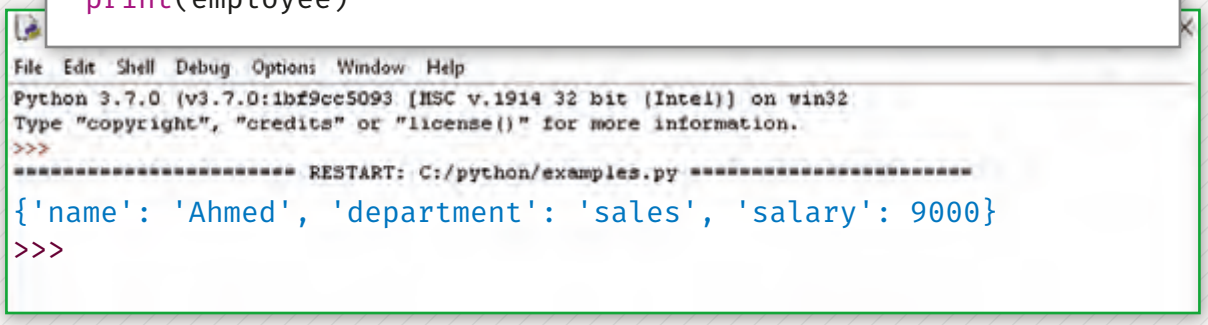
عناصر القاموس

> تستخدم الأقواس المعقوفة { }عند تعريف القاموس، ويتم الفصل بين العنصر والمفتاح باستخدام علامة :

> لا يمكن وجود عنصرين في القاموس لهما نفس المفتاح، كل مفتاح يسمح لك بالوصول لقيمة واحدة من القيم الموجودة في القاموس.

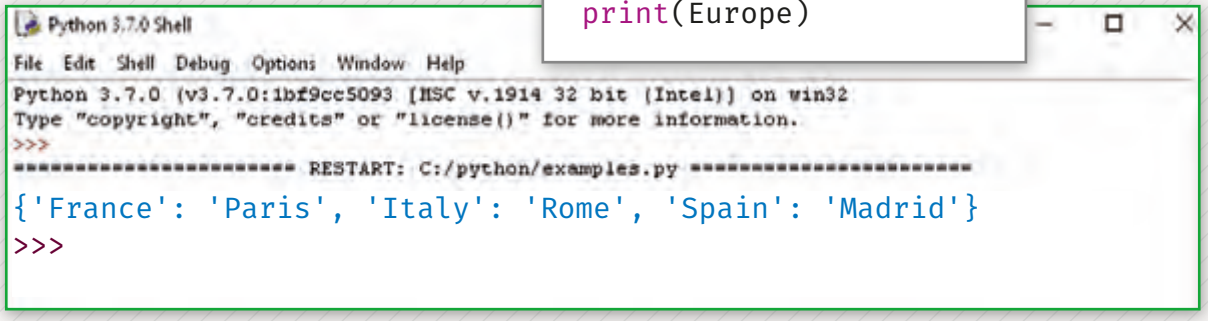
مثال 1

```
employee={"name":"Ahmed","department":"sales","salary":9000}
print(employee)
```



مثال 2

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



الاختلاف بين القائمة List والقاموس Dictionary

- القائمة هي سلسلة من العناصر المتتابعة، في حين أن القاموس يضم أزواجًا من العناصر غير المرتبة.
- الفرق الرئيس يتلخص في كيفية الوصول إلى العناصر، فعناصر القائمة يتم وضعها في قائمة يتم الوصول إليها عن طريق رقم الموقع، بينما يتم الوصول إلى عناصر القاموس من خلال المفاتيح.

إنشاء القاموس

يُمكننا إنشاء القاموس من خلال استخدام أمر الإنشاء **dict()**.

مثال 3

```
Europe=dict(France="Paris", Italy="Rome", Spain="Madrid")
print(Europe)
```

```
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 ==========
{'France': 'Paris', 'Italy': 'Rome', 'Spain': 'Madrid'}
>>>
```

كما يُمكننا إنشاء قاموس يتم تعبئة مدخلاته من قِبل المستخدم.

مثال 4

إنشاء قاموس فارغ

```
myDict = dict()
key = input("Enter the key: ")
value = input("Enter the value: ")
myDict[key] = value
print(myDict)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
Enter the key: a
Enter the value: 1
{'a': '1'}
>>>
```

ما الذي ستضيفه إلى المقطع البرمجي لإنشاء قاموس بثلاثة أزواج؟

الدوال المستخدمة مع القاموس

يُقدم لنا **Python** مجموعة من الدوال الجاهزة يُمكن استخدامها مع القواميس.

الدوال	
الدالة	**الوصف**
dictName.get(x)	ترجع القيمة المرتبطة بالمفتاح x، وإذا لم يتم العثور على المفتاح في القاموس فإنها ترجع None.
dictName.update(x)	تضيف زوج/أزواج جديدة من العناصر إلى القاموس إذا كانت المفاتيح غير موجودة مسبقًا فيه. أو تقوم بتحديث محتوى القيم المرتبطة بالمفاتيح الموجودة.
dictName.values()	تُرجع جميع القيم الموجودة في القاموس.
dictName.keys()	تُرجع جميع المفاتيح الموجودة في القاموس.
dictName.clear()	تحذف جميع العناصر داخل القاموس.

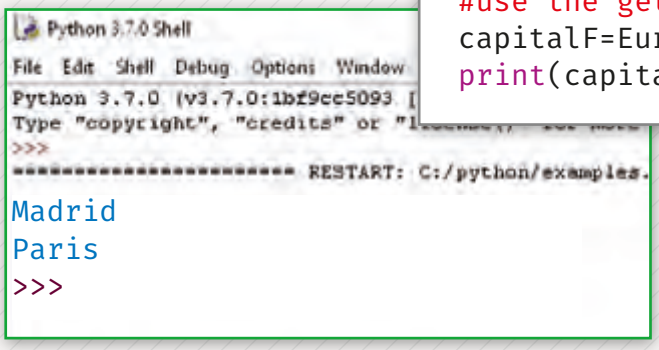
الوصول إلى عناصر القاموس

لا يحتوي عنصر القاموس على رقم فهرس، ولكن توجد طريقتان للوصول إلى العناصر:

- باستخدام المفتاح الخاص بالعنصر الذي يكتب داخل الرمز [ ].
- باستخدام دالة get().

مثال 5

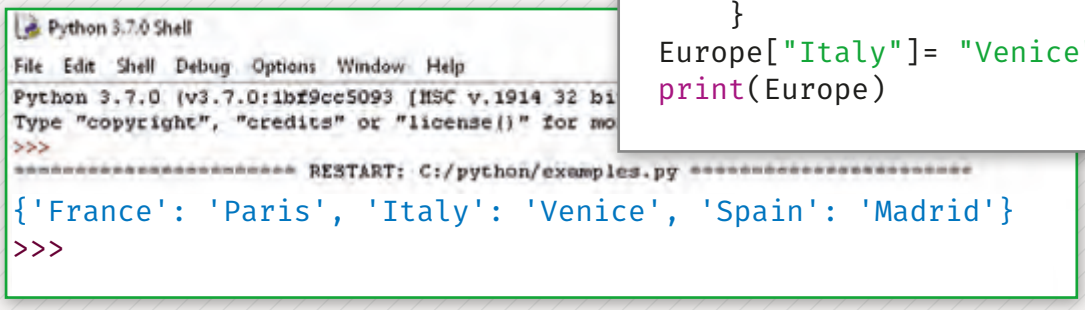
```
capitalS=Europe["Spain"]
print(capitalS)

#use the get operation
capitalF=Europe.get("France")
print(capitalF)
```



مثال 6

لتغيير قيمة عنصر داخل القاموس يُمكننا استخدام الأوامر التالية:

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

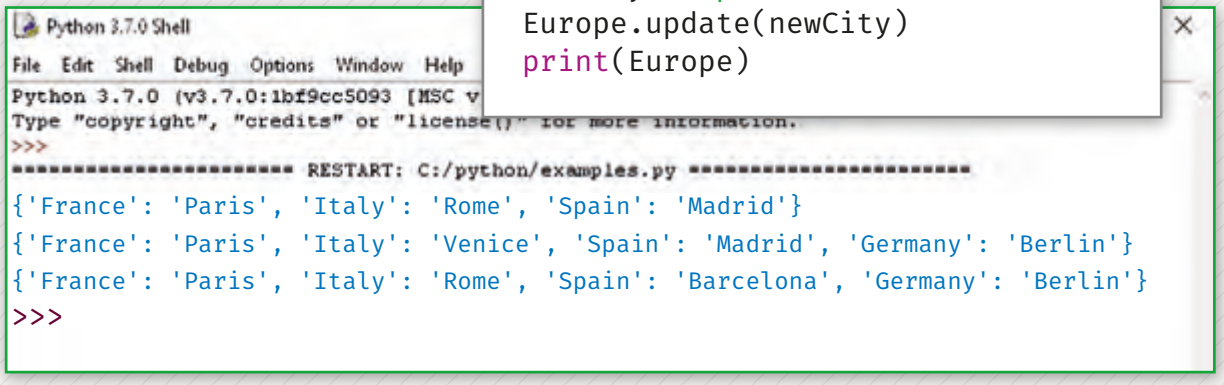


مثال 7

لنجرب إضافة زوج جديد باستخدام الدالة:

dictName.update(key:value)

```
Europe= {
    "France" : "Paris",
    "Italy" : "Rome",
    "Spain" : "Madrid",
    }
print(Europe)
newCountry={"Germany": "Berlin"}
Europe.update(newCountry)
print(Europe)
#update a value of an existing key
newCity={"Spain": "Barcelona"}
Europe.update(newCity)
print(Europe)
```

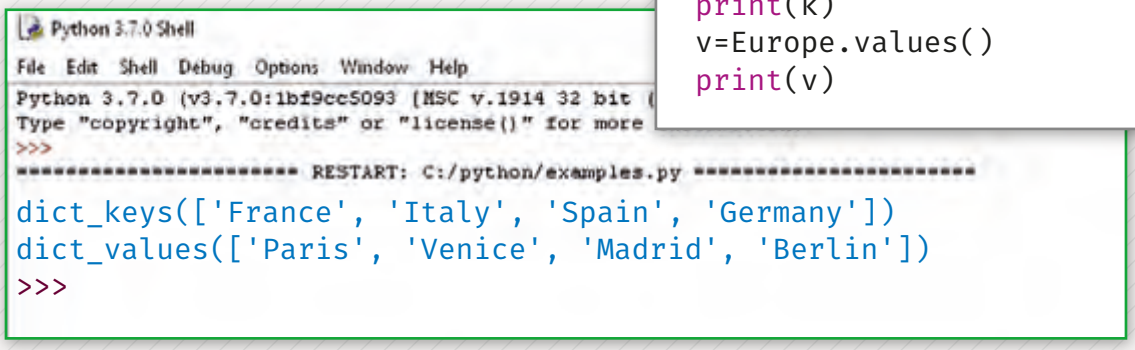


مثال 8

لنجرب طباعة أسماء الدول (مفاتيح القاموس) والمدن الموجودة فيها (القيم)، باستخدام الدوال:

dictName.keys()
dictName.values()

```
Europe= {
    "France" : "Paris",
    "Italy" : "Rome",
    "Spain" : "Madrid",
    }
k=Europe.keys()
print(k)
v=Europe.values()
print(v)
```

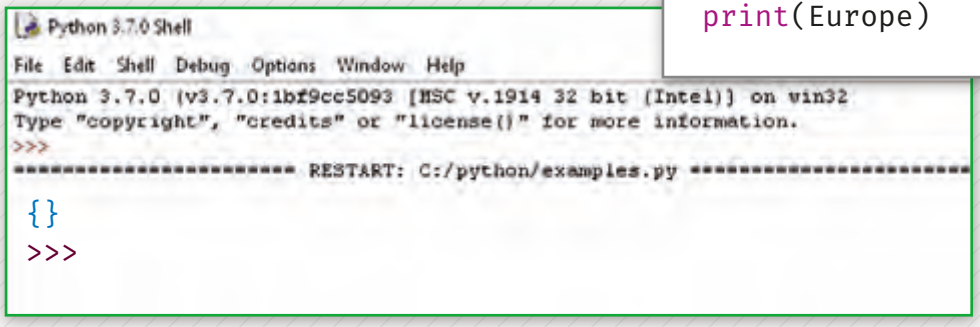


إذا أردنا إزالة جميع عناصر القاموس فإننا نستخدم العملية **clear()**.

مثال 9

dictionaryName.clear()

```
Europe.clear()
print(Europe)
```



حذف عنصر من القاموس

إذا أردنا حذف عنصر من القاموس، فإننا نستخدم الكلمة المفتاحية **del** متبوعة باسم القاموس والمفتاح بين قوسين مربعين، كما يُمكننا أيضًا حذف القاموس بأكمله بواسطة الكلمة المفتاحية del متبوعة باسم القاموس.

مثال 10

```
del Europe["Italy"]
print(Europe)
```

```
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 ==========
{'France': 'Paris', 'Spain': 'Madrid'}
>>>
```

```
del Europe
print(Europe)
```

```
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
Type "copyright", "credits" or "license()" for more infor
>>>
========== RESTART: C:/python/examples.py ==========
    print(Europe)
NameError: name 'Europe' is not defined
>>>
```

الاستخدامات المختلفة لهياكل البيانات

يُستخدم كل هيكل من هياكل البيانات في حالات مختلفة.

استخدام هياكل البيانات	
هيكل البيانات	الاستخدام
القائمة List	عندما نحتاج إلى مجموعة يتم تعديلها بشكل متكرر.
المصفوفة Array	عندما نريد تخزين بيانات من نفس النوع.
الصفوف Tuple	عندما نحتاج إلى تخزين بيانات لا نريد تغييرها.
القاموس Dictionary	عندما نحتاج إلى وجود ارتباط منطقي ما بين (المفتاح:القيمة).
	عندما نحتاج إلى البحث عن البيانات بناءً على مفتاح مخصص.

مشروع البنك

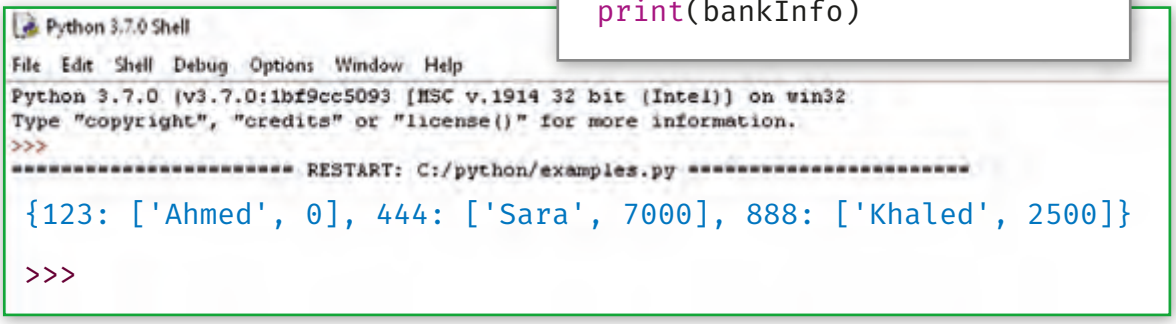
سنقوم الآن بإنشاء مشروع بسيط خطوة بخطوة. سنقوم بمحاكاة برنامج بنك.

لكل عميل سجل في البنك يتضمّن:

- رقم الحساب
- الاسم
- رصيد الحساب

سنستخدم القاموس لحفظ بيانات العميل، بحيث يكون رقم الحساب هو المفتاح (قيمة فريدة) ولكن سنواجه مشكلة في تعيين القيمة المرتبطة، لأن سجل العميل يحتوي على اسمه ورصيده كذلك، وعليه فيمكننا استعمال القوائم Lists كقيمة مرتبطة بالمفتاح.

```
bankInfo={
    123:['Ahmed',5000],
    444:["Sara",7000],
    888:['Khaled',2500]
    }
print(bankInfo)
```

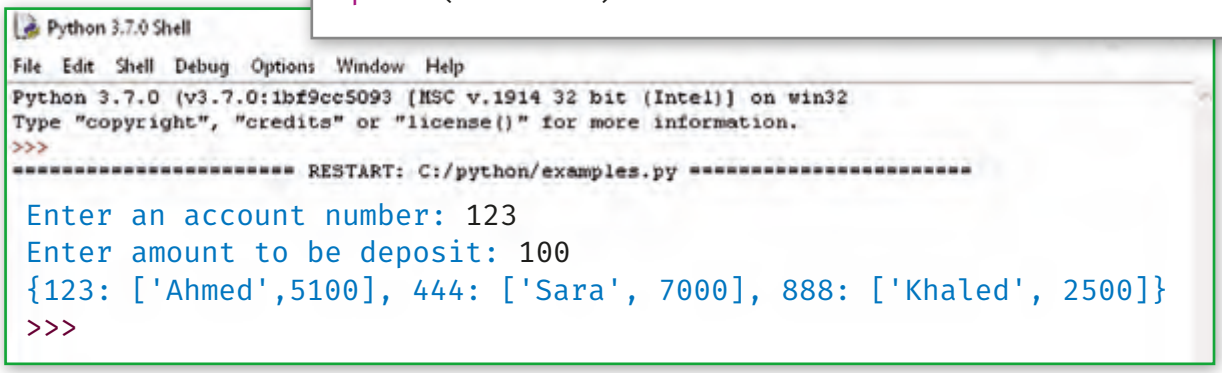


عبر المقطع البرمجي التالي نقوم ببرمجة عملية الإيداع.

عملية الإيداع تعني أننا نقوم بإضافة المبلغ المدخل إلى رصيد الحساب.

```
bankInfo={
    123:['Ahmed',5000],
    444:["Sara",7000],
    888:['Khaled',2500]
    }

#deposit operation
accountNo=int(input("Enter an account number: "))
account=bankInfo.get(accountNo)
balance=account[1]
amount=float(input("Enter amount to be deposit: "))
newBalance=balance+amount
account[1]=newBalance
print(bankInfo)
```

القائمة تمثل بيانات الحساب.



> أضف المقطع البرمجي التالي لبرنامجك.
- قم بوصف طريقة تشغيله.
- ماذا يحسب هذا البرنامج برأيك؟

```
s = 0
for x in bankInfo:
    y = []
    y = bankInfo.get(x)
    s = s + y[1]
print(s)
```

> استعن بالمثال السابق في كتابة برنامج يقوم بعملية السحب من الحساب، قبل تنفيذ السحب، يجب أن تتأكد إذا كان رصيد الحساب كافياً. مالم يتم عرض الرسالة: "No enough balance".

> اكتب مقطعًا برمجيًا يقوم بحذف الحساب الذي يتم إدخال رقمه من قبل المستخدم، على أن يكون رصيد الحساب صفرًا. وإلا سيتم عرض رسالة "The account has a balance, cannot be deleted"

1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

1.	يتم كتابة العناصر داخل القاموس بين قوسين مربعين.
2.	تكتب أزواج القاموس بالشكل التالي: (key:value).
3.	يمكن الوصول إلى عناصر القاموس باستخدام أرقام مواقعها.
4.	يمكن أن تكون مفاتيح القواميس أي نوع من أنواع البيانات.

2

تعريف القاموس.

> عرف القاموس Dictionary.

_____

_____

> اكتب الصيغة العامة للقاموس Dictionary.

_____

_____

_____

3

ما هي أوجه الاختلاف ما بين القاموس والقائمة؟

_____

_____

_____

4

أنشئ قاموسًا خاصًا باسمك، واسم العائلة.

5

قم بإنشاء قاموس يتم تعبته من قبل المستخدم.

6

قم بإنشاء قاموس

> يحتوي على الاسم الأول والاسم الأخير لثلاثة من أصدقائك

> قم بإضافة صديق جديد.

> احذف جميع عناصر القاموس.

7

استخدم عمليات القواميس في Python للقيام بما يلي:

> قم بإنشاء قاموس يحتوي على الاسم الأول والأخير لخمسة من أصدقائك.

> اطبع قائمة بالمفاتيح.

> اطبع قائمة القيم.

> قم بإضافة صديق جديد.

> اطبع القائمة الجديدة للقيم.

> احذف جميع عناصر القاموس.

8 أنشئ دالة تسجيل دخول باستخدام القاموس.

> أنشئ قاموسًا بأسماء المستخدمين وكلمات المرور.

```
Dictionary
"User1": 12345,
"User2": 56789,
"User3": 33333,
```

> أنشئ دالة باسم login تختص بتسجيل الدخول تتضمن (المستخدمين users، اسم المستخدم username، كلمة المرور password) بثلاثة معاملات:

- قاموس بأسماء المستخدمين وكلمات المرور.
- اسم المستخدم username.
- كلمة المرور password.

> يجب أن ترجع الدالة True إذا كان اسم المستخدم موجودًا وكلمة المرور صحيحة، و سترجع False خلاف ذلك.

هيكل بيانات المصفوفة (Array) في Visual Basic

تعتبر المصفوفات واحدة من تراكيب البيانات الأكثر شيوعًا، فيمكننا العثور عليها في جميع لغات برمجة الحاسوب تقريبًا.

فيما يلي مثال على استخدام المصفوفة بلغة برمجة Visual Basic.

```
Module IterateArray
   Public Sub Main()
      Dim numbers = {10, 20, 30}

      For index = 0 To numbers.GetUpperBound(0)
         Console.WriteLine(numbers(index))
      Next
   End Sub
End Module
' The example displays the following output:
'   10
'   20
'   30
```

القوائم (Lists) في C#

إضافة الى Python، تُستخدم القوائم في لغات برمجة أخرى. مثال على استخدام القوائم (Lists) في C#.

```
#include <iostream>
#include <list>

void print(std::list<std::string> const &list)
{
    for (auto const& i: list) {
        std::cout << i << "\n";
    }
}

int main()
{
    std::list<std::string> list = { "blue", "red", "green" };
    print(list);

    return 0;
}
```

مشروع الوحدة

العنوان: قاموس عربي – انجليزي

الوصف: باستخدام البرمجة بلغة Python عليك إنشاء قاموس لغوي عربي – انجليزي.

الأدوات: لغة برمجة Python

خطوات التنفيذ: أنشئ القائمة التالية:

```
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 ==========
Welcome to your own Arabic-English dictionary!
Please choose one of the following options:
1. Add
2. Remove
3. Edit
4. Search
5. List
0. Exit
What do you want to do?
>>>
```

> قم بقراءة اختيار المستخدم وتنفيذ الدالة المقابلة.

> إذا كان اختيار المستخدم هو 1، فستتم إضافة الكلمات إلى القاموس اللغوي، يجب أن يكون تنسيق كل زوج كالتالي، الكلمة بالإنجليزية: الكلمة بالعربية.

> إذا كان اختيار المستخدم هو 2، فقم بإزالة الكلمة المحددة.

> إذا كان اختيار المستخدم هو 3، فقم بتعديل الكلمة المحددة.

> إذا كان اختيار المستخدم هو 4، فقم بالبحث عن ترجمة الكلمة المحددة.

> إذا كان اختيار المستخدم هو 5، فقم بطباعة القاموس بأكمله.

> إذا كان اختيار المستخدم هو 0، فأغلق القاموس.

ماذا تعلمت

تعلمت في هذه الوحدة:

> التمييز بين بعض أنواع هياكل البيانات.

> الاستخدامات المختلفة لهياكل البيانات.

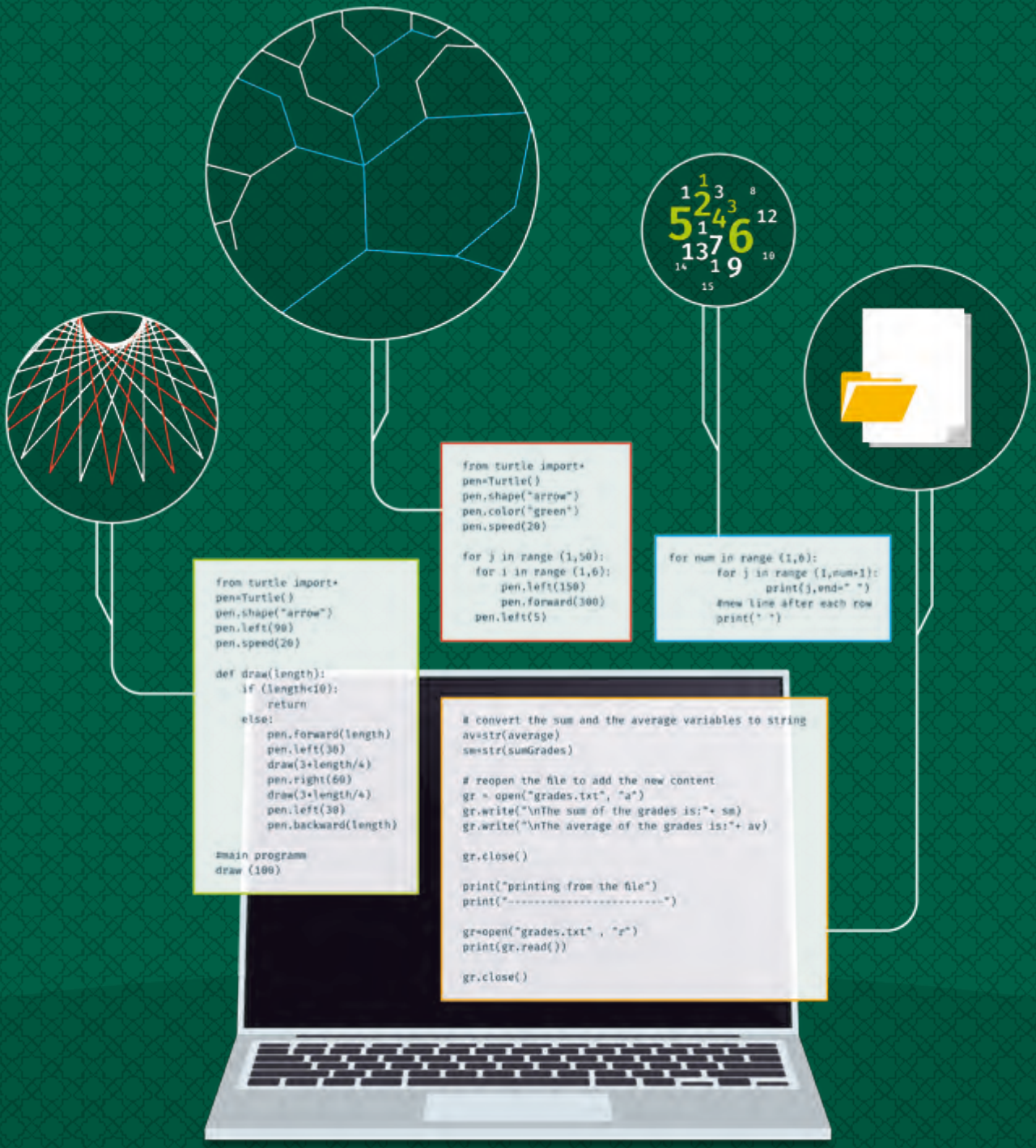
> استخدام هياكل البيانات في تطبيقات برمجية متنوعة.

المصطلحات

الدرس 1	هيكل بيانات Data Structure	الموقع Index	الدوال Functions	
	قابلة للتغيير Mutable	غير قابلة للتغيير Immutable	قائمة List	
الدرس 2	صف Tuple			
الدرس 3	مصفوفة Array	مصفوفة أحادية الأبعاد (1D) One-dimensional (1D) array	مصفوفة ثنائية الأبعاد (2D) Two-dimensional (2D) array	
	عمود Column	صف Row		
الدرس 4	قاموس Dictionary	زوج Pair	مفتاح Key	
	قيمة Value			

2. تقنيات البرمجة

في هذه الوحدة سنتعلم كيفية التعامل مع الملفات النصية باستخدام لغة البرمجة بايثون من حيث الإنشاء، والفتح، والكتابة، والقراءة، والتعديل، وأيضا سنتطرق لمجموعة من تقنيات البرمجة ومنها التكرارات البرمجية المتداخلة وتقنية الإستدعاء Recursion الذاتي للدوال.



ماذا سنتعلمْ؟

في هذه الوحِدة سنتعلم:

> العمليات الأساسية لمعالجة ملفات البيانات.

> كيفية فتح، وقراءة، وإلحاق، والكتابة على ملف البيانات.

> كيفية إنشاء ملف نصي txt. برمجيًا في Python.

> ما المقصود بالتكرار المتداخل.

> كيفية استخدام التكرارات المتداخلة.

> كيفية طباعة الأنماط باستخدام التكرارات المتداخلة.

> ما المقصود بالقوائم المتداخلة.

> كيفية التعامل مع القوائم المتداخلة.

> كيفية تمثيل Python للمصفوفات ثنائية الأبعاد باستخدام القوائم المتداخلة.

> كيفية طباعة صف موجود ضمن مصفوفة ثنائية الأبعاد.

> كيفية طباعة عمود موجود ضمن مصفوفة ثنائية الأبعاد.

> ما المقصود بالاستدعاء الذاتي.

> ما المقصود بدوال الاستدعاء الذاتي.

> كيفية إنشاء دوال الاستدعاء الذاتي باستخدام Python.

> إيجابيات وسلبيات الاستدعاء الذاتي.

> متى يتم استخدام الاستدعاء الذاتي.

مواضيع الوحدة

> ملفات البيانات

> الجمل التكرارية المتداخلة

> القوائم المتداخلة والمصفوفات ثنائية الأبعاد

> الاستدعاء الذاتي

الأدوات

> Python

هل تذكر؟

التكرارات Loops

في بعض الأحيان نحتاج إلى تكرار تنفيذ مجموعة أوامر برمجية عدة مرات داخل البرنامج، مما يستغرق وقتاً وجهداً كبيرين. تقدم لنا لغة بايثون مجموعة جمل للتحكم البرمجي تسمى بأوامر التكرار loop لمساعدتنا على تجنب إعادة كتابة الأوامر البرمجية مرةً أخرى. تسمح جمل التكرار بتنفيذ جملة برمجية واحدة أو مجموعة جمل برمجية عدة مرات. سوف نتعرف الآن على نوعين من هذه التكرارات وهما تكرارfor و تكرار while.

تدعم لغة برمجة بايثون هذه الأنواع من الجمل التكرارية.	
for loop_variable in range: statements	تكرار for
while condition: statements	تكرار while

```
# Prints out the value of i
for i in range(5):
    print("i is now", i)
```

```
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 (In
Type "copyright", "credits" or "license()" for more in
>>>
========== RESTART: C:/python/examples.py
i is now 0
i is now 1
i is now 2
i is now 3
i is now 4
```

```
i=1
while i<6:
    print(i)
    i=i+2
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093 [MSC v.191
Type "copyright", "credits" or "license()"
>>>
========== RESTART: C:/python
1
3
5
```

القائمة List

هي أكثر هياكل البيانات شيوعاً في بايثون وقد سبق لك التعرف عليها واستخدامها بشكل بسيط. في هذا الدرس سندرسها بشكل أكثر تعمقاً ونتعلم كيفية استخدامها في البرامج لحل المشكلات.

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

الصيغة العامة لتعريف القائمة

يتم تعريف القائمة بالصيغة التالية :

```
List_Name=[item1,item2,...,item n]
```

متغير يمثل اسم القائمة

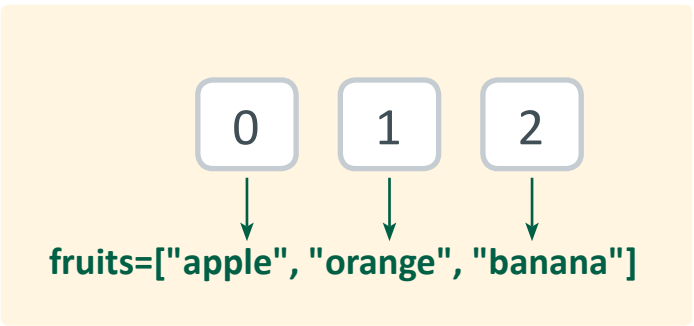
عناصر القائمة

حيث يتم وضع عناصر القائمة بين القوسين المربعين [ ] , ويفصل بين كل عنصر وآخر بالفاصلة.

الوصول إلى عناصر القائمة

يمكن التعامل مع كل عنصر من عناصر القائمة كعنصر قائم بذاته وذلك بطباعته أو تغيير قيمته أو استخدامه في عمليات أخرى. كل عنصر في القائمة له رقم تسلسي (index) يمثل موقعه في القائمة ويبدأ ترقيم المواقع في القائمة بالصفر.

نظام ترقيم القائمة يبدأ من 0 وليس من 1.



الدرس الأول

ملفات البيانات

تعرفنا في السابق على مجموعة من البرامج التي تستخدم البيانات التي يتم إنشاؤها أثناء وقت التشغيل فقط، حيث قمنا بتخزين هذه البيانات على شكل متغيرات وتراكيب بيانات داخل ذاكرة الحاسوب **(RAM)**.

ولكن في كل مرة يتم فيها إيقاف البرنامج، يتم فقد البيانات التي كانت قد حُفظت في الذاكرة. ولتجنب هذه المشكلة، ظهرت فكرة تخزينها في ملفات البيانات التي يتم حفظها على القرص الثابت أو وحدات التخزين الأخرى مما يسمح لنا باستعادتها لاحقًا ليتم معالجتها. إن أبسط نوع من حاويات حفظ البيانات هو الملف النصي **Text File**.

الملفات النصية Text Files

الملف النصي هو عبارة عن سلسلة من النصوص (تشمل الحروف والأرقام والرموز)، يُمكن إجراء عمليات مختلفة على الملفات النصية كالحذف والإضافة والتعديل، ويتم ذلك من خلال أوامر برمجية محددة.

عمليات المعالجة الأساسية على ملف البيانات:

1 فتح الملف (Open).

لفتح ملف، يتعين علينا تحديد موقع الملف في وحدة التخزين، ثم الاختيار ما إذا كنا نريد أن نقرأ من الملف أو نقوم بالكتابة فيه.

2 القراءة من الملف (Read).

عندما نقرأ البيانات الموجودة داخل الملف، فإننا نقوم بقراءة هذه البيانات الموجودة بالملف إلى متغيرات وهياكل بيانات داخل برنامجنا الموجود في الذاكرة لمزيد من عمليات المعالجة.

3 الكتابة إلى الملف (Write).

عندما نكتب البيانات في الملف، فإننا ننقل قيم المتغيرات وهياكل البيانات المستخدمة في البرنامج إلى الملف المحفوظ في وحدة التخزين، حيث يمكننا الكتابة إلى ملف جديد أو الإضافة إلى محتويات ملف موجود مسبقًا.

4 إغلاق الملف (Close).

عندما نغلق ملفًا، فإن نظام التشغيل يتأكد من انتهاء جميع عمليات القراءة والكتابة.

تقدم لنا Python دوال جاهزة لاستخدامها في إنشاء وقراءة وتحديث وحذف الملفات.

دالة فتح الملف Open

لفتح ملف في **Python** فإننا نستخدم دالة **Open**.تأخذ هذه الدالة معاملين: أولهما هو مسار الملف الذي نريد فتحه، والثاني عبارة عن الحرف الذي يمثل العملية التي نريد القيام بها على الملف.

الصيغة العامة لدالة فتح الملف Open:

```
<object>=open(filename, mode)
```

علمًا بأن الـ **object** هوكائن متغير يمثل الملف المراد فتحه داخل البرنامج

file name: مسار/اسم الملف على وسائط التخزين.

mode: حرف العملية (للكتابة ، للقراءة ، ...)

التعامل مع الملفات في Python	
حرف العملية	العملية
r	قراءة البيانات من ملف موجود مسبقًا.
w	الكتابة إلى ملف.
a	إلحاق البيانات في نهاية ملف موجود.

r+: تفتح هذه الدالة ملف للقراءة والكتابة. يمكننا إضافة البيانات إلى الملف وقراءته من البداية ولكن إذا لم يكن الملف موجودًا، فلا تقوم الدالة بإنشاء ملف جديد.

دالة إغلاق الملف Close

عند الانتهاء من عمليات الكتابة والقراءة من الملف يجب علينا إغلاقه وذلك باستخدام Close، وتؤمن هذه الدالة حفظ جميع التغييرات إذا قمنا بإجرائها على الملف.

تستخدم دالة **close** لإغلاق الملف الذي قمنا بفتحه.

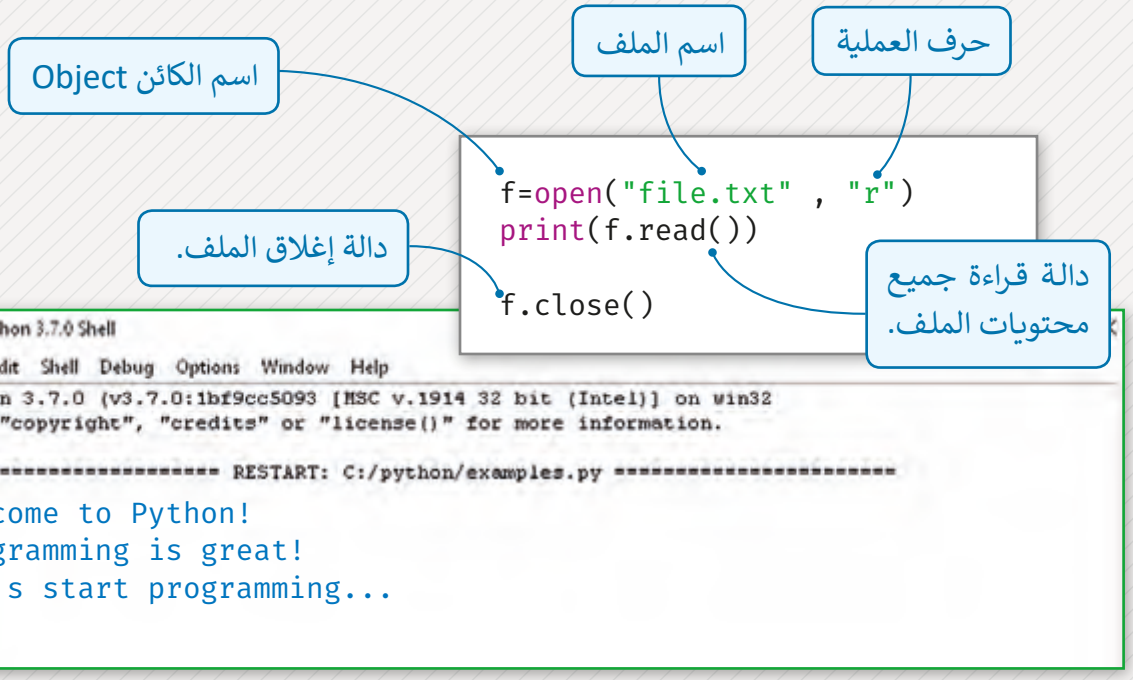
الصيغة العامة لدالة إغلاق الملف Close:

```
object.close()
```

افتح برنامج المفكرة notepad وأنشئ ملفًا نصيًا بداخله جملة "Welcome to Python!".
قم بحفظ الملف باسم "file".

مثال 1

قم بفتح الملف لقراءة محتوياته.



يجب أن يكون البرنامج .py والملف .txt الذي تريد فتحه في نفس المجلد، وفي حالة استخدام مسار خاص بملفك، فيجب عليك التأكد من صحة اسم المسار.

نصيحة ذكية

قبل استخدامك لدالة Open بغرض قراءة الملف، تأكد من أن الملف موجود لتتجنب ظهور رسالة خطأ.

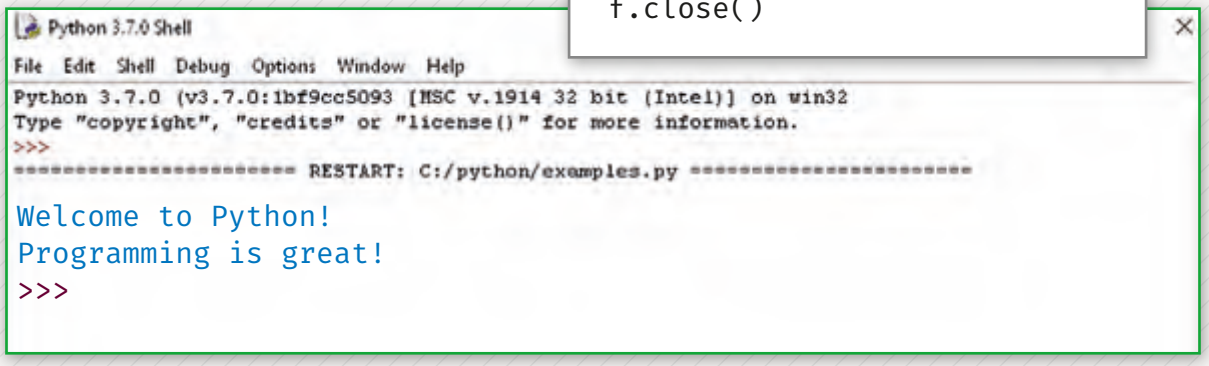
يُمكننا أيضًا قراءة الملف سطرًا بعد سطر.

إن دالة **readline()** تُرجع سطرًا واحدًا من الملف.

مثال 2

أنشئ الملف النصي التالي واحفظه باسم file.txt


```
f=open("file.txt", "r")
print(f.readline())
print(f.readline())

f.close()
```



هل يُمكنك تخمين نتيجة تنفيذ المقطع البرمجي التالي؟

تحقق من إجابتك بواسطة Python shell.

```
f=open("file.txt", "r")

line = f.readline()
while line:
    print(line)
    line=f.readline()

f.close()
```

الإلحاق إلى ملف

يُمكننا إضافة نص جديد إلى ملف، وسيتم إضافة المحتوى الجديد في نهايته وذلك بفتح ملف موجود مسبقا بوضعية الالحاق a.

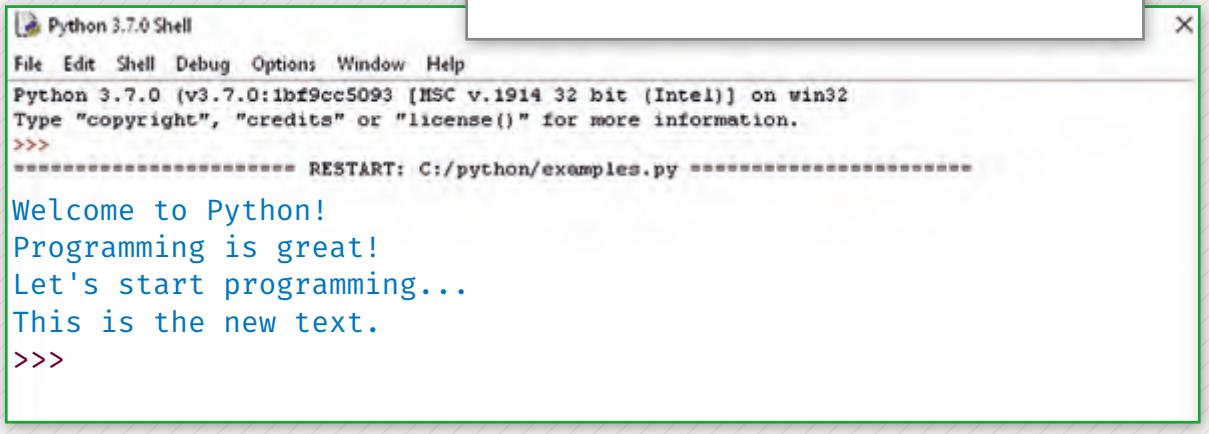
الصيغة العامة للالحاق: <object>=open("<file_path>","a").

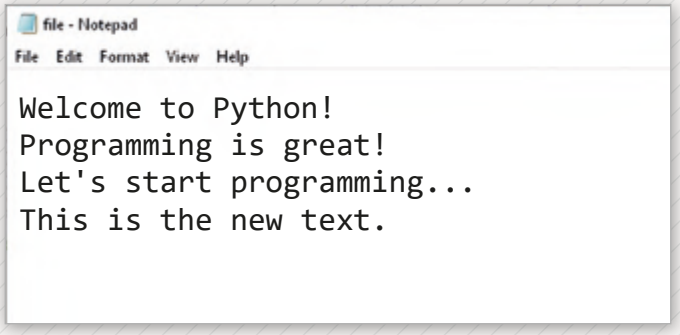
مثال 3

لنقم بإضافة نص إلى الملف file.txt الذي استخدمناه في مثال (1):

```
f=open("file.txt", "a")
f.write("\nThis is the new text.")

#open and read the file after the
appending:
f=open("file.txt", "r")
print(f.read())

f.close()
```

نستخدم \n للانتقال إلى سطر جديد.



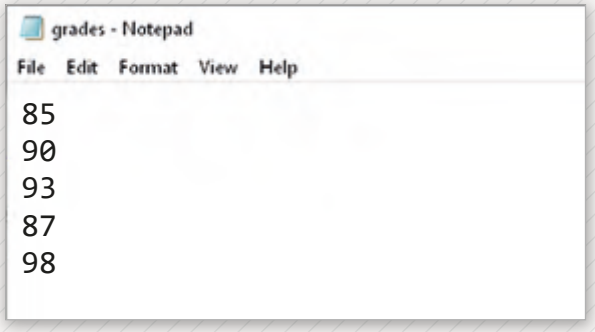


سننشئ في هذا الدرس ملفًا يحتوي على درجات لطالب في 5 مواد دراسية، وبعدها سنستخدم كود **Python** لفتح الملف وقراءة الدرجات ثم سنقوم بحساب المجموع ومتوسط درجات الطالب وكتابتها إلى الملف.

من خلال الأمثلة التالية، سنتعرف على طرق قراءة البيانات من الملف.

مثال 4

أنشئ الملف النصي التالي الخاص بدرجات الطالب واحفظه باسم grades.txt

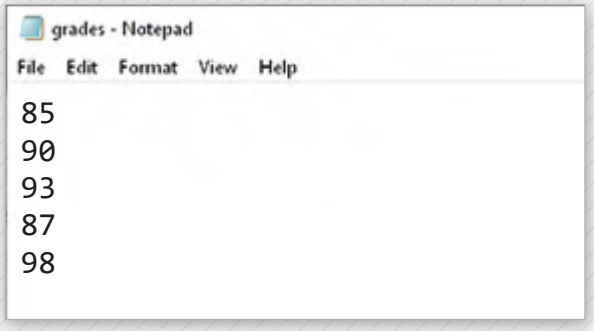


اكتب مقطعًا برمجيًا لفتح الملف في بايثون وقراءة محتوياته.

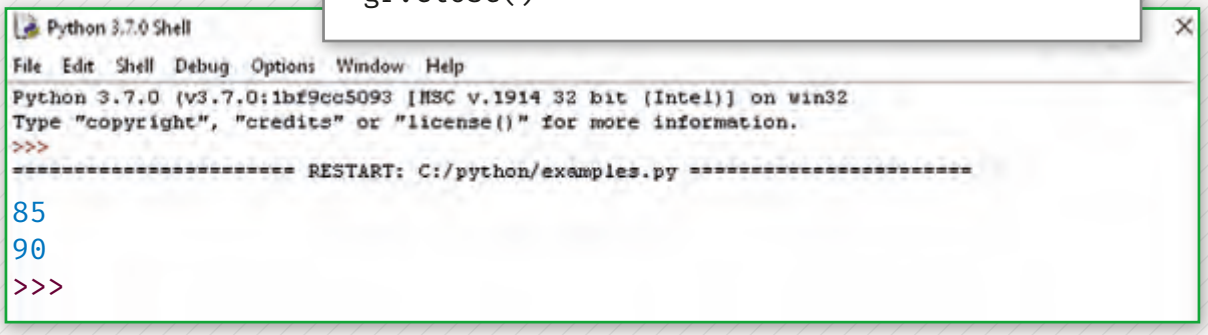
يُمكننا استخدام المسار لفتح الملف.

```
gr = open("C:\python\G11CS\grades.txt", "r")
print(gr.read())

gr.close()
```

مثال 5




```
gr=open("C:\python\G11CS\grades.txt", "r")
print(gr.readline())
print(gr.readline())

gr.close()
```




```
gr=open("C:\python\G11CS\grades.txt", "r")

grade = gr.readline()
while grade:
    print(grade)
    grade=gr.readline()

gr.close()
```

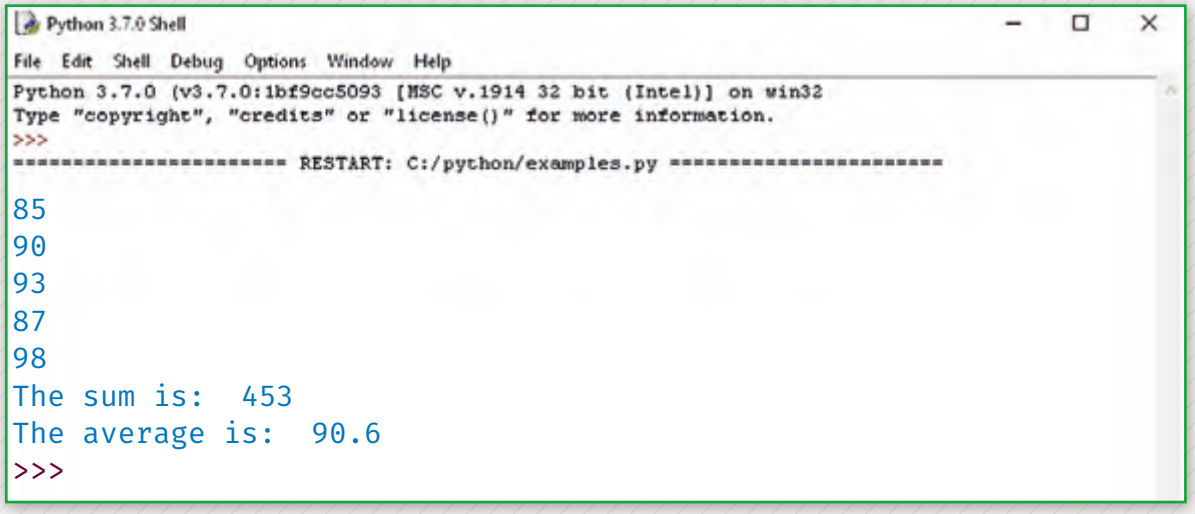
استخدم المقطع البرمجي التالي لطباعة جميع أسطر الملف.

يضمن هذا الشرط قراءة محتويات الملف إلى نهايتها، ويتوقف بانتهاء محتويات الملف.

مثال 6

لكي نقوم بحساب متوسط الدرجات، علينا أولًا أن نقوم بحساب مجموع تلك الدرجات.

```
gr=open("C:\python\G11CS\grades.txt", "r")

sumGrades=0
grade=gr.readline()
cnt=0
while grade:
    # convert string to integer
    gradeInt=int(grade)
    print(gradeInt)
    # count the grades
    cnt=cnt+1
    sumGrades=sumGrades+gradeInt
    grade=(gr.readline())

print("The sum is: ",sumGrades)
average= sumGrades/cnt
print("The average is: ",average)

gr.close()
```



يعمل هذا البرنامج على افتراض أن محتويات الملف هي أرقام فقط (بدون مسافات أو أسطر فارغة)، وفيما عدا ذلك ستظهر رسالة خطأ.
ماذا يحدث لو كان الملف فارغًا؟
هل بإمكانك حل المشكلة؟

قم بإضافة المقطع البرمجي التالي إلى برنامجك لإضافة المجموع والمتوسط إلى ملفك.

```
# convert the sum and the average variables to string
av=str(average)
sm=str(sumGrades)

# reopen the file to add the new content
gr = open("C:\python\G11CS\grades.txt", "a")
gr.write("\nThe sum of the grades is:"+ sm)
gr.write("\nThe average of the grades is:"+ av)

gr.close()

print("printing from the file")
print("-----------------------")

gr=open("grades.txt" , "r")
print(gr.read())

gr.close()
```

```
Python 3.7.0 Sh
File  Edit  Shell
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 ==========
printing from the file
-----------------------
85
90
93
87
98
The sum of the grades is:453
The average of the grades is:90.6
>>>
```

grades - Notepad

File Edit Format View Help

```
85
90
93
87
98
The sum of the grades is:453
The average of the grades is:90.6
```

ملف grades.txt

الكتابة إلى الملف

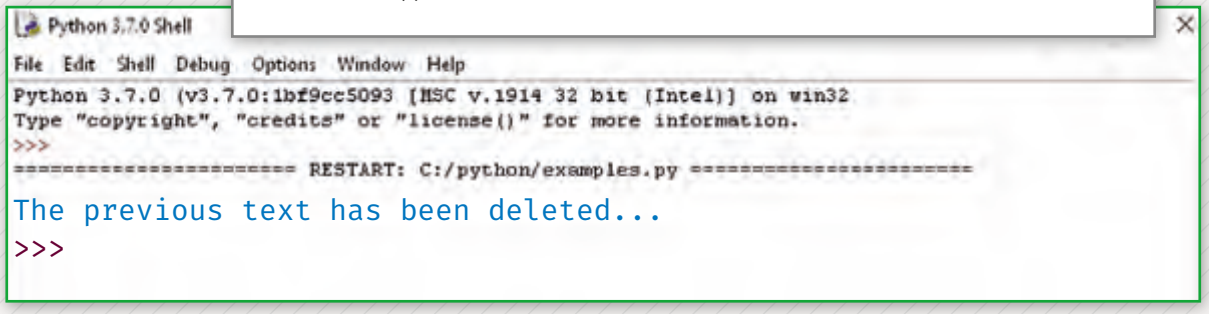
تقوم الدالة **Write** بالكتابة على الملف الحالي بحيث يتم حذف كافة المحتوى السابق.

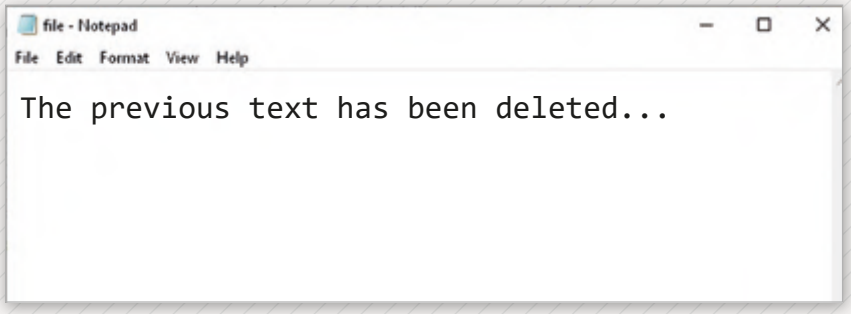
الصيغة العامة لدالة الكتابة <object>. write("message")

حيث أن: **message** هو النص المراد كتابته داخل الملف.

مثال 7

```
f=open("file.txt", "w")
f.write("The previous text has been deleted...")

#open and read the file after writing:
f=open("file.txt", "r")
print(f.read())

f.close()
```



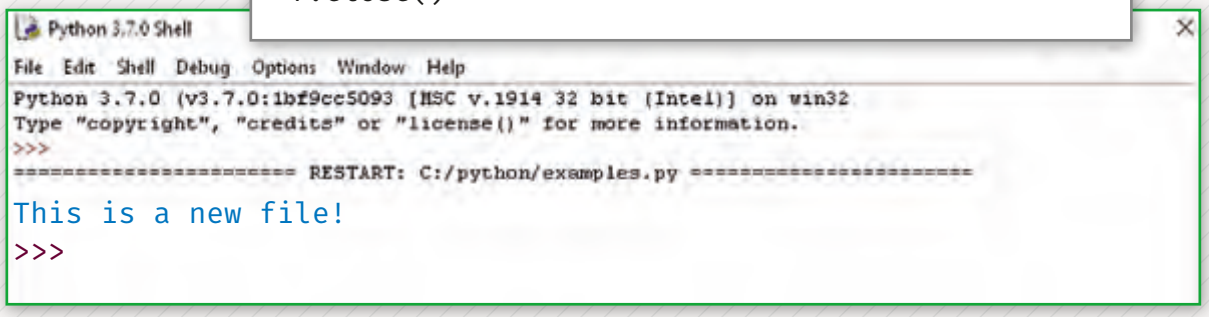


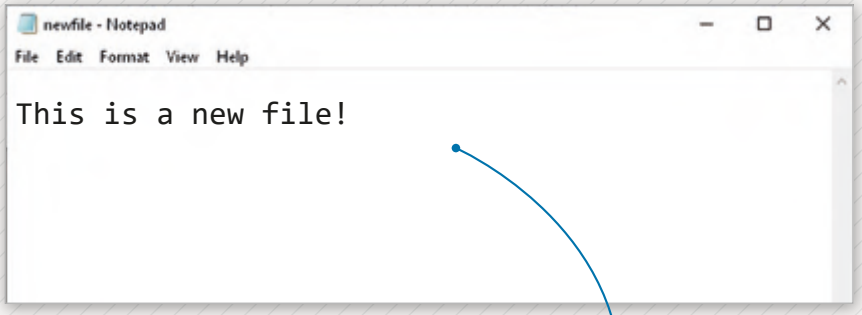
مثال 8

إذا قمنا بفتح ملف للكتابة ولم يكن موجودًا، فسينشئ البرنامج ملفًا جديدًا.

لاحظ إنشاء الملف النصي في نفس مجلد البرنامج.

```
f=open("newfile.txt", "w")
f.write("This is a new file!")

#open and read the file after writing:
f=open("newfile.txt", "r")
print(f.read())

f.close()
```



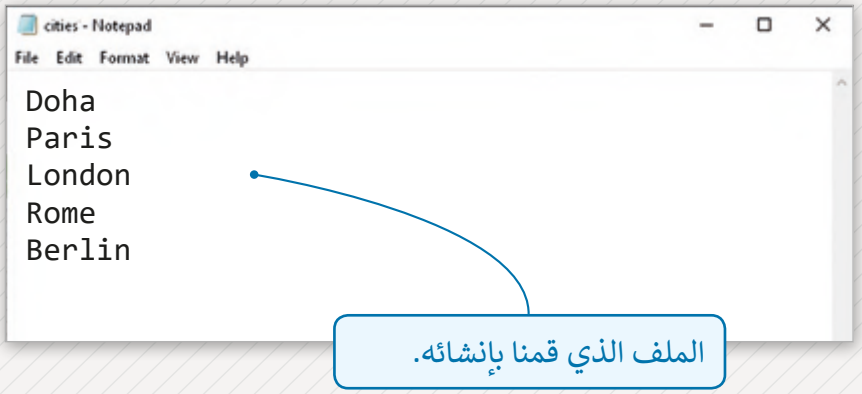


الملف الذي قمنا بإنشائه

مثال 9

فلنستخدم Python لإنشاء ملف نصي .txt يتم فيه كتابة أسماء 5 مدن مختلفة.

```
f=open("cities.txt", "w")
f.write("Doha\n")
f.write("Paris\n")
f.write("London\n")
f.write("Rome\n")
f.write("Berlin")

f=open("cities.txt", "r")
print(f.read())

f.close()
```



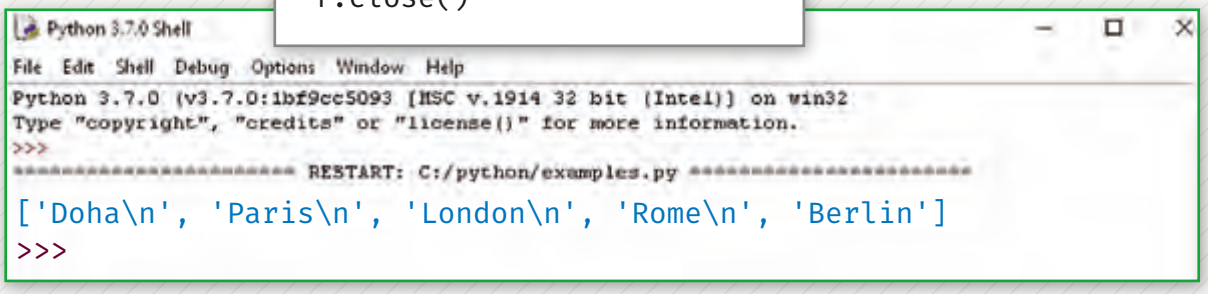
قراءة محتويات الملف إلى قائمة

ذكرنا سابقًا أنه يمكن قراءة محتويات الملف إلى متغيرات أو إلى هياكل بيانات، وسنستعرض في المثال الآتي قراءة محتويات الملف إلى قائمة بحيث نتمكن من إجراء مختلف عمليات القوائم على البيانات.

مثال 10

سنقوم الآن بتخزين كل سطر من الملف الخاص بنا إلى قائمة.

```
citiesList=[ ]
f=open("cities.txt", "r")
for i in range(5):
   city= f.readline()
   citiesList.append(city)

print(citiesList)
f.close()
```



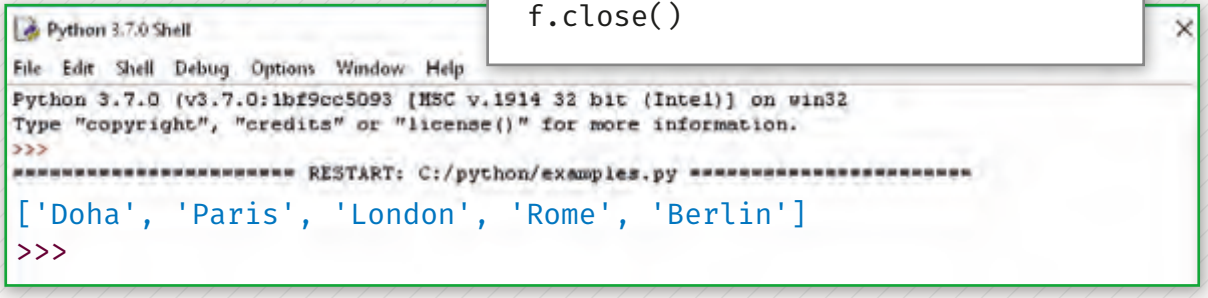
دالة ()replace

لاحظ أنه في نهاية اسم كل من المدن في الملف، يوجد الحرف \n والذي يستخدم للانتقال إلى سطر جديد، من أجل استخدام دوال القوائم فإننا نحتاج إلى إزالة الرمز \n، حيث تقوم دالة () **replace** باستبدال حرف واحد بحرف آخر.

مثال 11

قم بتغيير مقطعك البرمجي بهذه الأسطر البرمجية.

```
citiesList=[ ]
f=open("cities.txt", "r")
for i in range(5):
   city= f.readline()
   #replace \n with empty space
   c=city.replace("\n","")
   citiesList.append(c)

print(citiesList)
f.close()
```

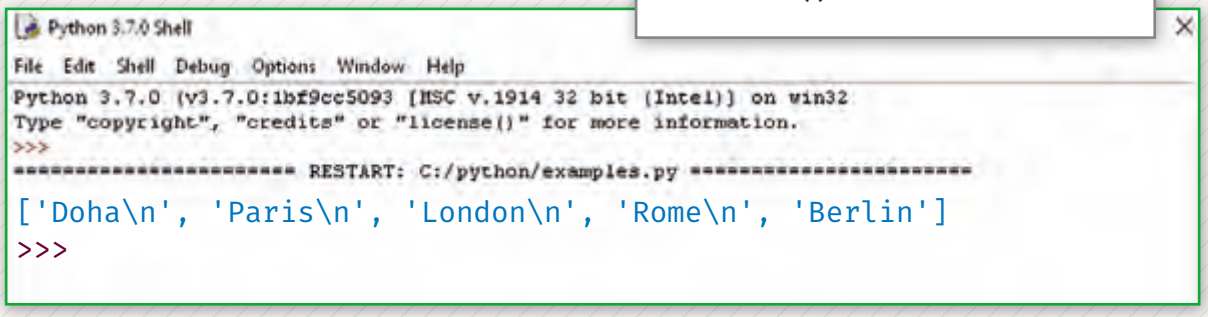


القراءة من الملف (الدوال Read, Readline, Readlines)

تقرأ دالة **readline()** في كل مرة سطرًا واحدًا من الملف، أما دالة **readlines()** فتقوم بقراءة جميع أسطر الملف وترجع قائمة تحتوي على جميع أسطر الملف.

مثال 12

```
f=open("cities.txt", "r")
print(f.readlines())

f.close()
```

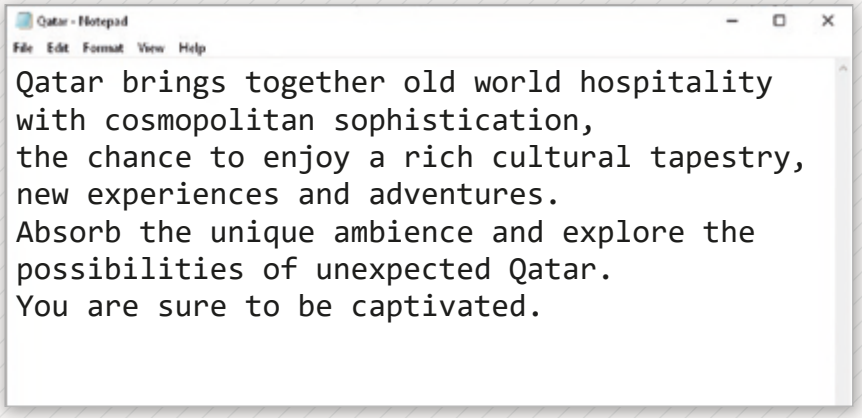


قم بتجربة المقطع البرمجي التالي،
ما الذي نقوم بإنشائه؟

```
f=open("cities.txt", "r")
citiesList=[]
citiesList=f.readlines()
l=len(citiesList)
for i in range(l):
   if "\n" in citiesList[i]:
        citiesList[i]=citiesList[i].replace("\n"," ")

print(citiesList)

f.close()
```

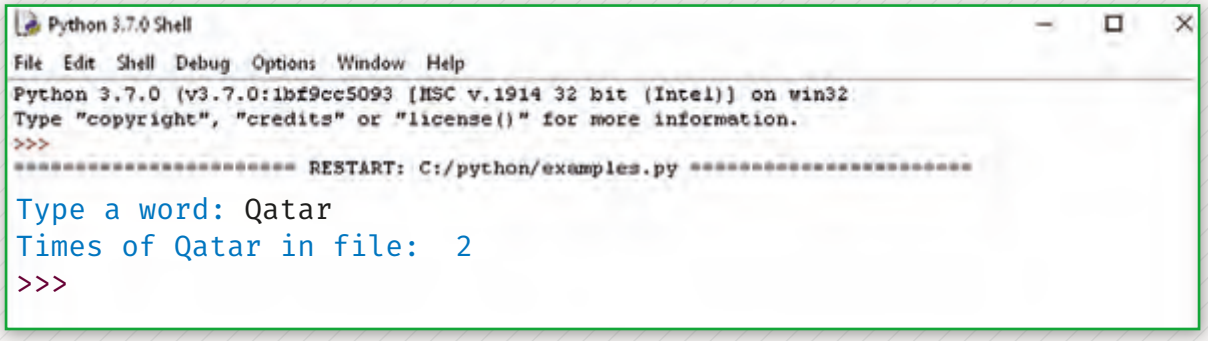
مثال 13

أنشئ المستند النصي التالي:



سنستخدم دالة readlines() للبحث عن كلمة وحساب عدد مرات ورودها في النص.

```
f=open("Qatar.txt", "r")
counter=0
word=input("Type a word: ")
datafile = f.readlines()

for line in datafile:
    nb=line.count(word)
    counter=counter+nb

print("Times of", word, "in file: ",counter)
f.close()
```



1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

.1	عندما نقرأ من ملف txt. فإننا نضيف أسطرًا جديدة.
.2	تقوم عملية الإلحاق إلى ملف بإضافة المحتوى إلى بداية الملف.
.3	عملية الكتابة write تقوم باستبدال النص الموجود داخل الملف.
.4	يمكننا فتح الملف النصي فقط إذا ما كان بنفس المجلد الموجود به ملف Python البرمجي.
.5	لإلحاق نص جديد في الملف فإننا نستخدم معامل "a".

2

ما هي العمليات الأساسية لمعالجة ملف البيانات؟ اشرح وظيفة كل منها.

3 ما هي نتيجة تنفيذ البرنامج التالي؟

```
f=open("file.txt", "w")
f.write("Qatar")
f.write("\n2020")

f=open("file.txt", "r")
print(f.read())

f.close()
```

4 أنشئ ملفًا نصيًا يحتوي اسمك الثلاثي واحفظه باسم name.txt.

> استخدم مقطع Python برمجي لقراءة ملفك.

> استخدم مقطع Python برمجي لإضافة عنوانك.

5 استخدم مقطع Python برمجي لإنشاء ملف نصي جديد يحتوي أسماء مجموعة من زملائك بالفصل.

> استخدم مقطع Python لإنشاء قائمة، وانقل محتويات الملف النصي إليها.

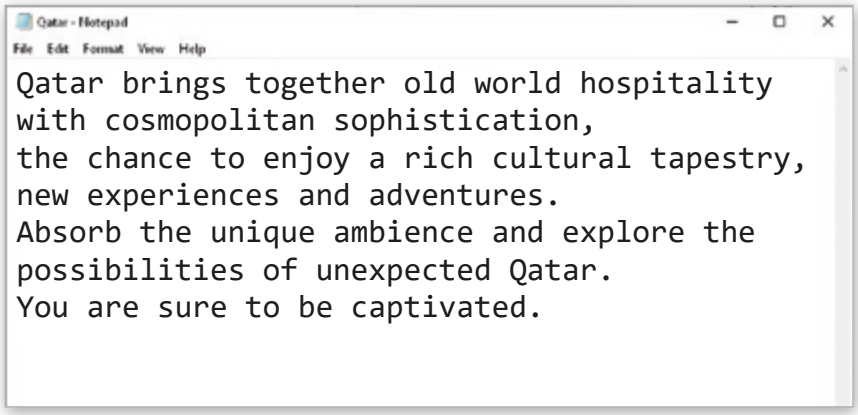
> اختر اسمًا وقم بحساب عدد المرات التي ظهر بها بالقائمة.

6

استخدم ملف Qatar.txt لإنشاء مقطع Python برمجي يقوم:

> عد الكلمات في الملف.

> عد الجمل في الملف.



7

قم بإنشاء مقطع Python برمجي.

> قم بإنشاء الملف النصي "temperature.txt".

> اكتب في الملف متوسط درجة الحرارة الأسبوعية لمدينة الدوحة، خلال شهر يناير.

> احسب درجة الحرارة المتوسطة في شهر يناير وقم بإضافتها في نهاية الملف.

8

ساعد إدارة مدرستك في تحليل درجات الطلاب وقم بكتابة الخوارزمية، والمخطط الانسيابي والمقطع البرمجي المناسب بلغة بايثون لحل المشكلة الآتية:

يعتبر الطالب ناجحًا في اختبارات مادة تكنولوجيا المعلومات لنهاية الفصل الأول إذا كانت درجته تعادل أو تفوق 50، إذا اعتبرنا أن صفك يتكون من عدد N من الطلاب، نفذ الآتي:

> إنشاء ملف Names.txt وتعبئته بأسماء جميع الطلبة.

> إنشاء ملف Degrees.txt وتعبئته بدرجات الطلاب في مادة تكنولوجيا المعلومات.

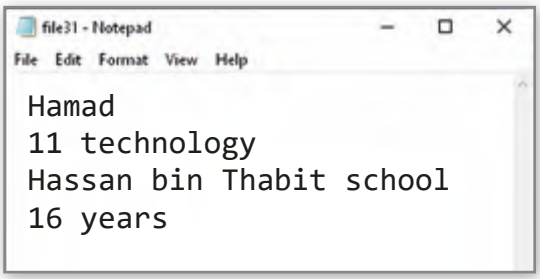
> إنشاء ملف Success.txt يحمل أسماء الطلاب الناجحين ودرجاتهم.

> إنشاء ملف Fails.txt يحمل أسماء الطلاب الراسبين ودرجاتهم.

9

ما الفرق بين دالة readline ودالة readlines؟ نفذ الآتي واستنتج الفرق.

> أنشيء الملف النصي file31.txt وقم بكتابة النص التالي داخله، ثم احفظه بمجلد العمل الخاص بك.



> شغل البيئة البرمجية IDLE (Python 3.7).

> اكتب البرنامج الآتي ونفذه.

```
f=open("file31.txt", "r")
Anonymous=f.readline()

print(Anonymous)

f.close()
```

	قيمته	نوعه
Anonymous		

> استنتج دور الدالة readline؟:.....

> على نفس البرنامج عدل readline بـ readlines ثم أعد تنفيذ البرنامج.

	قيمته	نوعه
Anonymous		

> استنتج دور الدالة readlines؟:.....

10

اكتب مقطعًا برمجيًا لقراءة محتويات ملف نصي باستخدام الدالة readlines، وذلك بتنفيذ الآتي:

> أنشئ برنامجًا جديدًا تحت مسمى file312.py ثم قم بحفظه في مكان من اختيارك.

> اكتب برنامج يمكن من:

- إنشاء الملف النصي file312.txt وفتحه لغرض الكتابة.
- كتابة أسماء المدن العربية الاتية (Doha, Beirut, Kuwait, Erbed, Aleppo) داخل الملف (كل مدينة في سطر).
- إغلاق الملف.
- إعادة فتح الملف لغاية القراءة منه.
- قراءة محتوى الملف وطباعته كقائمة.

11

اكتب برنامجًا يمكن من التعامل (قراءة، بحث، حساب...) مع الملفات النصية، وذلك بتنفيذ الآتي:

> أنشئ برنامجا جديدا تحت مسمى file313.py ثم قم بحفظه في مكان من اختيارك.

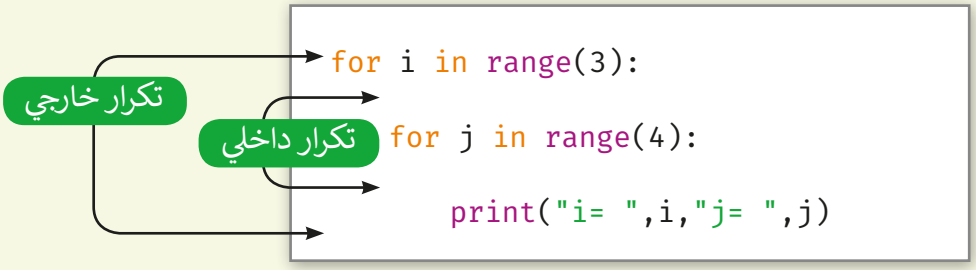
> اكتب برنامج يمكن من :

- فتح الملف file311.txt لغرض الكتابة.
- قراءة الملف وحفظ محتواه في القائمة City.
- احتساب وعرض عدد المساحات الفارغة (" " space) الواردة في الملف.

الجمل التكرارية المتداخلة

لقد قمنا مسبقًا باستخدام العديد من أنواع التكرارات. سنتعرف الآن ما الذي يحدث عند الجمع بين إجراءات التكرار. تُسمى عملية وضع تكرار داخل تكرار آخر بعملية التداخل **(nesting)**.

يمكن دمج أي نوع تكرار داخل نوع آخر، ومن أكثر التكرارات المتداخلة شيوعًا هي تكرارات **for**.



مثال 1

```
for i in range(3):
    for j in range(2):
        print("i= ",i,"j= ",j)
```

```
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
Type "copyright", "credits" or "license()" for more
>>>
========== RESTART: C:/python/examples
i= 0 j= 0
i= 0 j= 1
i= 1 j= 0
i= 1 j= 1
i= 2 j= 0
i= 2 j= 1
>>>
```

خطوات التنفيذ:

1. i تأخذ القيمة 0

   سيتم تنفيذ التكرار الداخلي مرتين لقيم for j=0, j=1

2. الآن ستزيد قيمة i، ولقيمة for i=1

   سيتم تنفيذ التكرار الداخلي مرتين أيضًا لقيم for j=0, j=1

3. الآن ستزيد قيمة i، ولقيمة for i=2

   سيتم تنفيذ التكرار الداخلي مرتين أيضًا لقيم for j=0, j=1

في النهاية يكون التكرار الخارجي قد تم تنفيذه 3 مرات والتكرار الداخلي 6 مرات.

جدول القيم

i	j	
0	0	1
	1	
1	0	2
	1	
2	0	3
	1	

تكرار خارجي

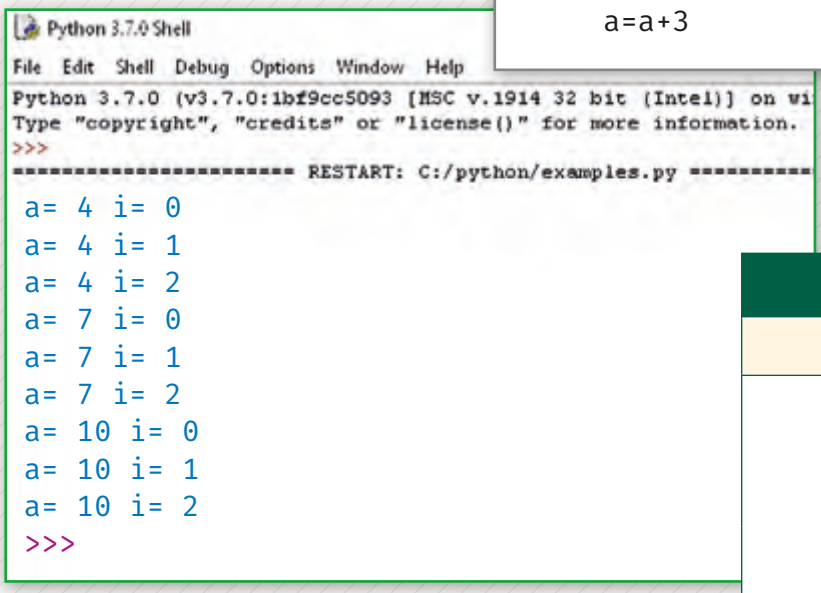
تكرار داخلي

```
a=4
while a<=12:
    for i in range(3):
        print("a=",a,"i=",i)
    a=a+3
```

مثال 2

لنستعرض مثالًا آخر على التكرارات المتداخلة.

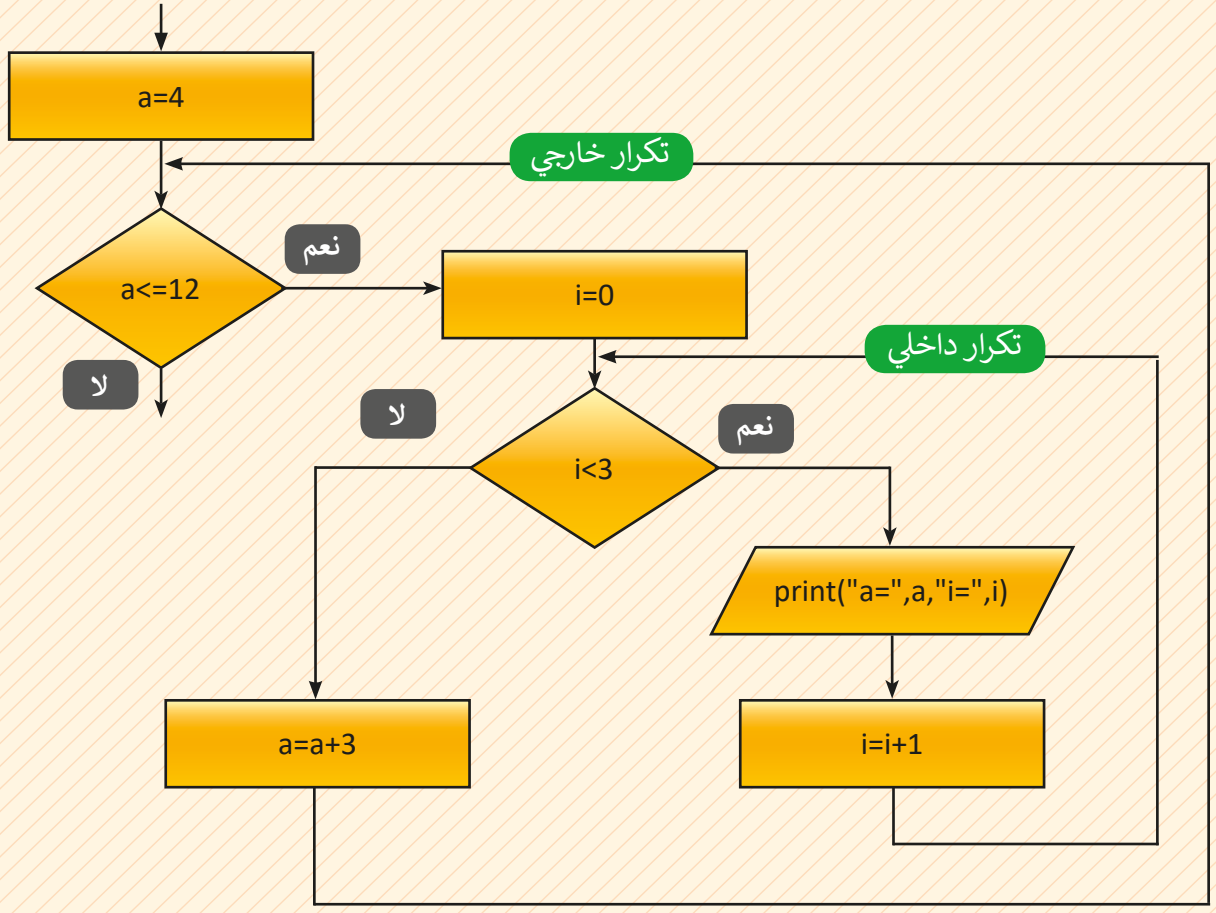
```
a=4
while a<=12:
    for i in range(3):
        print("a=",a,"i=",i)
    a=a+3
```



جدول القيم

a	i
4	0
	1
	2
7	0
	1
	2
10	0
	1
	2
13	

المخطط الانسيابي للمقطع البرمجي السابق.

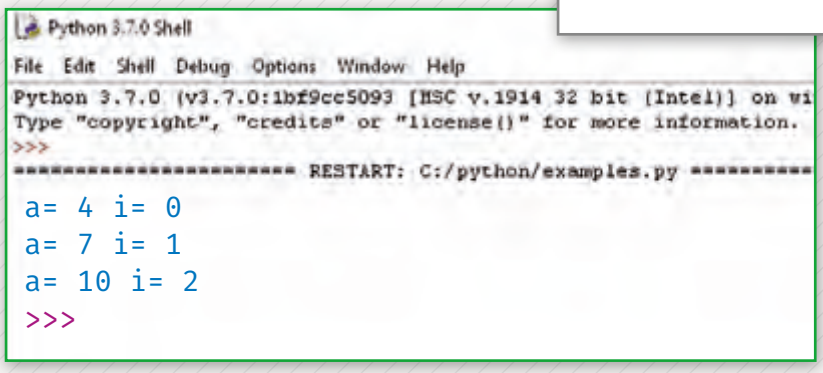


عليك أن تكون حذرا للغاية عند التعامل مع المسافة البادئة في التكرارات المتداخلة، فهي تحدد الأوامر التي يتم تضمينها في كل تكرار.

مثال 3

لنستعرض هنا مثالًا على مسافات بادئة مختلفة.

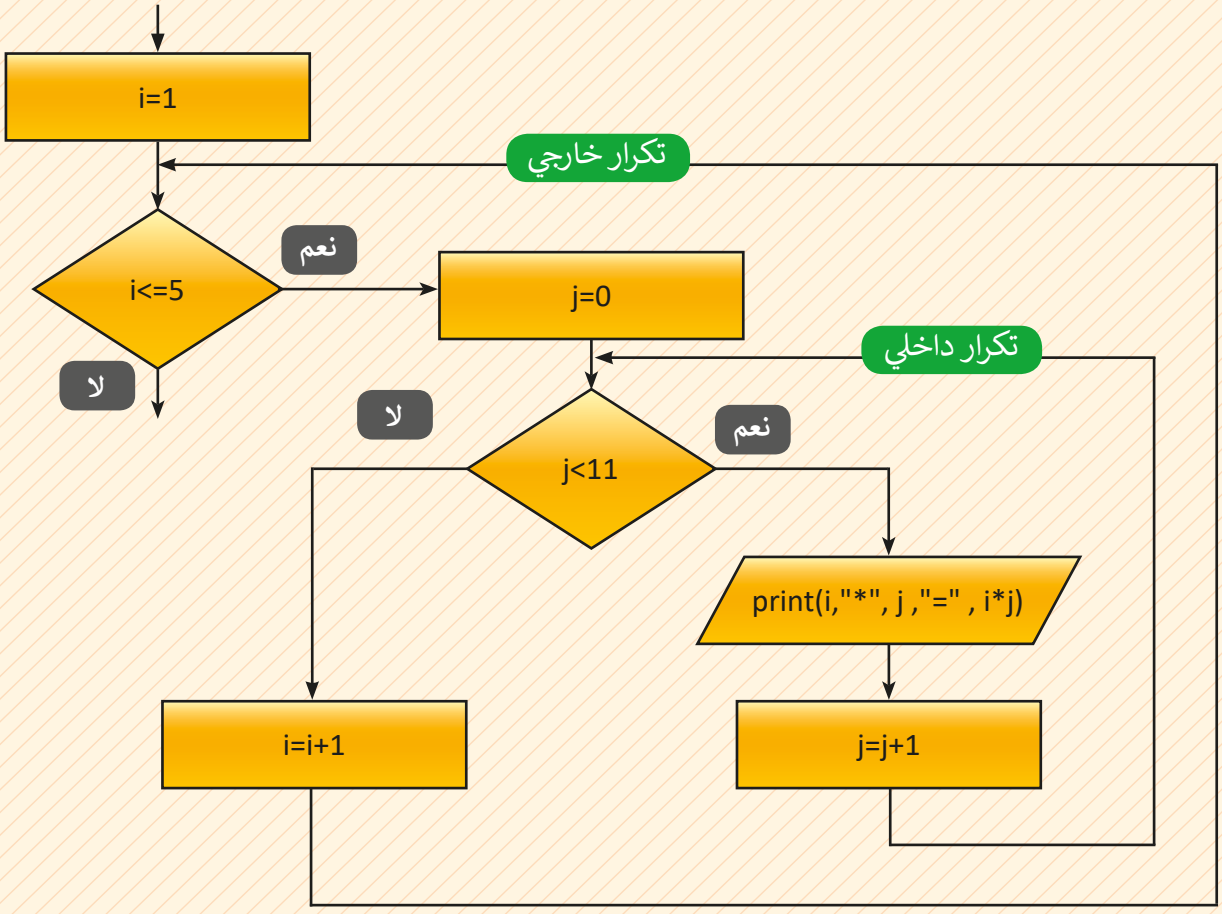
```
a=4
while a<=12:
    for i in range(3):
        print("a=",a,"i=",i)
        a=a+3
```



القواعد المطبقة على التكرارات المتداخلة:

- يجب أن تكون كامل الجملة التكرارية الداخلية داخل الجملة التكرارية الخارجية.
- ينبغي استخدام اسم مختلف للعداد في كل من التكرار الداخلي والخارجي.
- انتبه إلى أن التكرار الداخلي يكتمل أولًا.
- في كل دورة من التكرار الخارجي، يتم تنفيذ جميع دورات التكرار الداخلي.

قم بتجربة المخطط الانسيابي التالي، ما الذي نقوم بإنشائه؟ ثم اكتب الخوارزمية المناسبة وترجمها إلى مقطع برمجي بلغة Python.



التكرارات المتداخلة في حياتنا الواقعية

توجد العديد من الأمثلة على التكرارات المتداخلة في حياتنا اليومية، فمثلًا تعتبر الساعة الرقمية إحدى الأمثلة البسيطة عليها.

في الساعة الرقمية نحتاج 3 تكرارات

- الأول خاص بتتبع الساعة.
- والثاني يختص بتتبع الدقائق.
- والثالث يختص بتتبع الثواني.

مثال 4

```
for hour in range (24):
    for minute in range (60):
        for sec in range (60):
            print(hour,":",minute,":",sec)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  W
Python 3.7.0 (v3.7.0:1bf9cc5
Type "copyright", "credits"
>>>
========== RESTART: C:/python/examples.py ==========
0:0:0
0:0:1
0:0:2
.
.
.
.
23:59:59
>>>
```

سيقوم البرنامج بعرض عدد الثواني خلال فترة 24 ساعة.

- التكرار الداخلي سيتكرر 60 مرة لكل تكرار من التكرار الأوسط.
- التكرار الأوسط سيتكرر 60 مرة لكل تكرار من التكرار الخارجي.
- التكرار الخارجي سوف يتكرر 24 مرة.

عندما يتكرر التكرار الخارجي 24 مرة، يكون التكرار الأوسط قد تكرر 1440 مرة والتكرار الداخلي 86400 مرة.

ارسم المخطط الانسيابي للبرنامج.

طباعة الأنماط

يُمكننا استخدام **Python** لعرض أنماط رياضية معينة على الشاشة.

مثال 5

```
for num in range (6):
    for j in range (num):
        print(num,end=" ")
    #new line after each row
    print(" ")
```

```
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 wi
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
>>>
```

end= تمنع النص التالي من الطباعة في سطر جديد.

```
for num in range (1,6):
    for j in range (1,num+1):
        print(j,end=" ")
    #new line after each row
    print(" ")
```

```
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 wi
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
>>>
```

الرسم باستخدام التكرارت المتداخلة Nested Loops

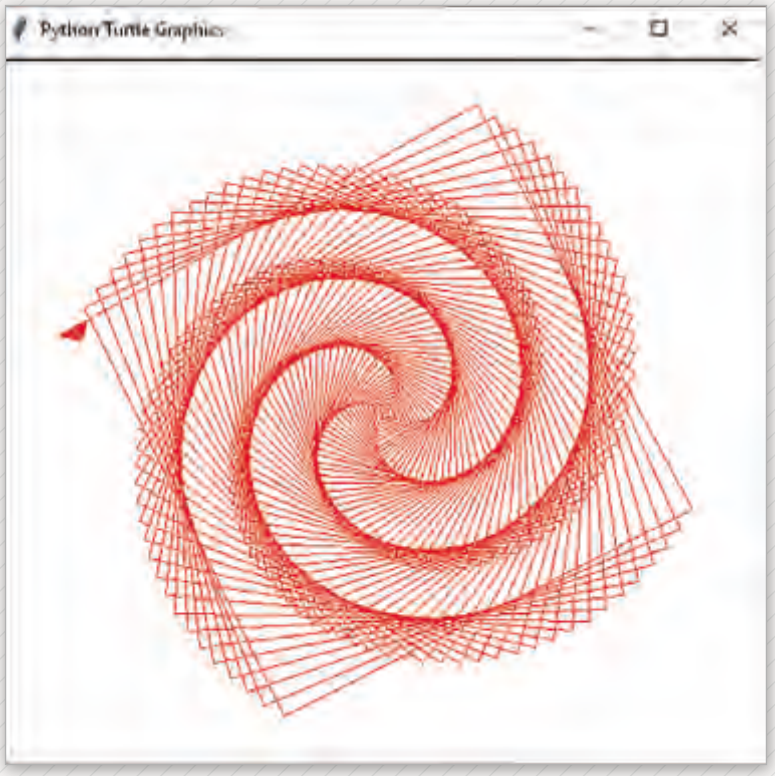
لقد استخدمنا في الماضي السلحفاة لإنشاء الأشكال الأساسية وبعض الرسومات الجميلة. الآن سنستخدم رسومات السلحفاة مع التكرارات لإنشاء رسومات أكثر تعقيدًا.

مثال 6

```
from turtle import*
pen=Turtle()
pen.shape("arrow")
pen.color("green")
pen.speed(20)

for j in range (1,50):
  for i in range (1,6):
      pen.left(150)
      pen.forward(300)
  pen.left(5)
```

مثال 7

```
from turtle import*
pen=Turtle()
pen.shape("arrow")
pen.color("red")
pen.speed(20)

length=500
angle=91

for side in range (length):
    pen.forward(side)
    pen.left(angle)
```



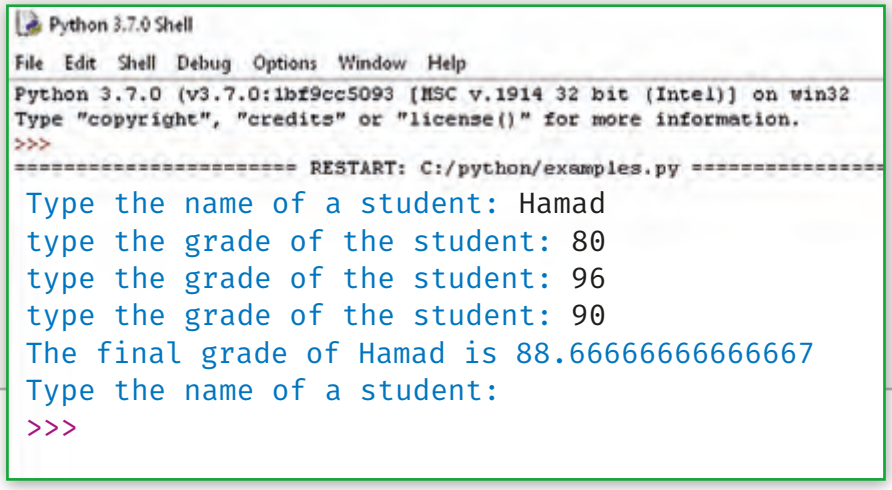
مشروع الطلبة

سنقوم الآن بإنشاء مشروع بسيط خطوة بخطوة.

احسب متوسط درجات الطلبة في صف دراسي يتكون من 30 طالب، وقد تقدم كل طالب لاختبارين فصليين واختبار نهائي.

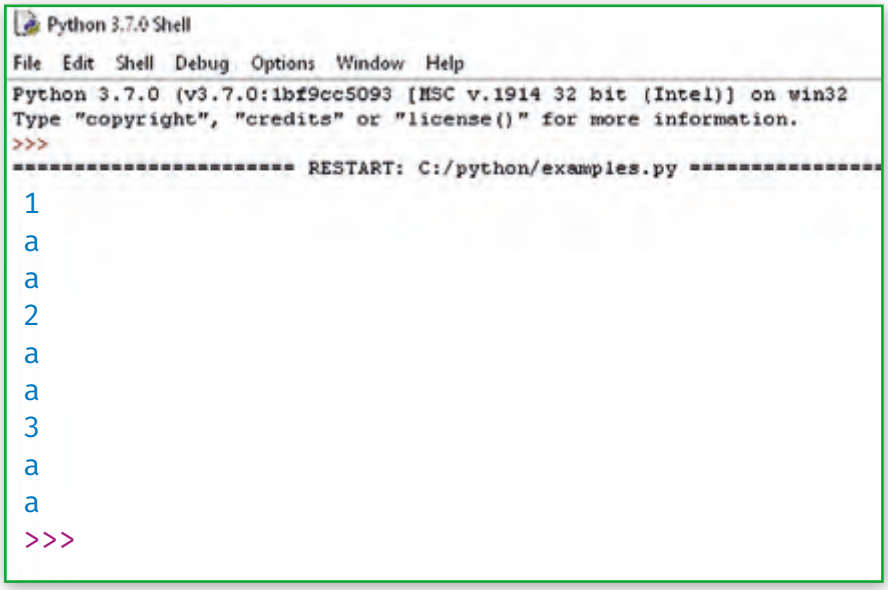
ستكون درجة الطالب النهائية هي متوسط الدرجات الثلاث.

```
#outer loop for the 30 students
for student in range(30):
    name=input("Type the name of a student: ")
    #initialize sumGrades for every student
    sumGrades=0
    #inner loop for the 3 grades
    for gr in range(3):
        grade=int(input("type the grade of the student: "))
        #sum the 3 grades of the student
        sumGrades=sumGrades+grade
    #calculate the final grade
    finalGrade=sumGrades/3
    print("The final grade of",name, "is", finalGrade)
```



1

تأمل النتيجة التي تظهر على الشاشة الآتية، والمقاطع البرمجية أدناها:



> اكتب بجانب كل مقطع برمجي ما هو الخطأ الذي يحتويه. ليعطي نفس النتيجة على الشاشة أعلاه:

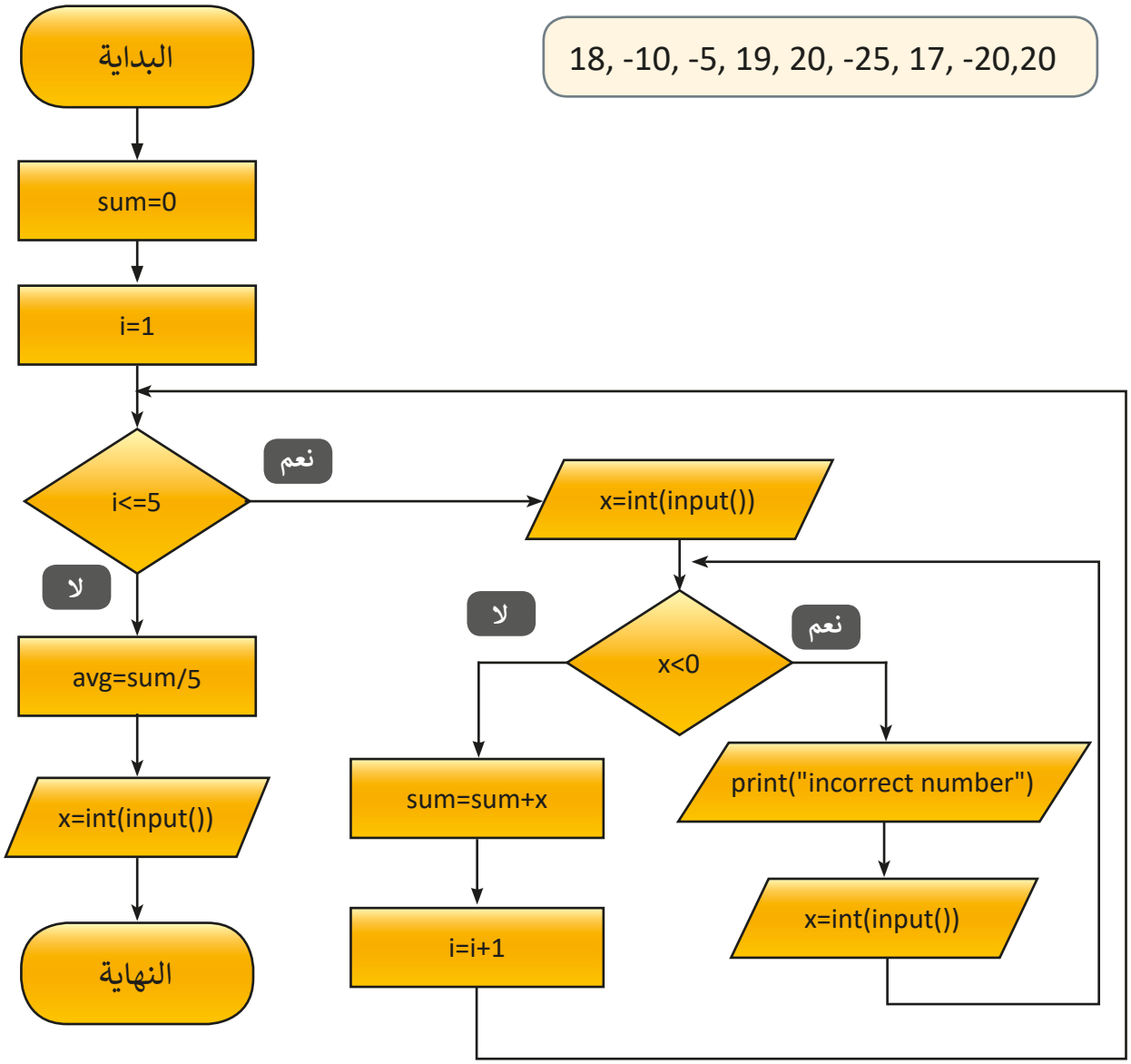
```
for i in range(1,4):
    print(i)
for j in range(2):
    print("a")
```

```
for i in range(1,4):
    print(i)
    print("a")
    for i in range(2):
        print("a")
```

2

صف وظيفة المخطط الانسيابي التالي. ما الذي سيتم عرضه في الشاشة إذا قمنا بإدخال القيم التالية؟ ثم قم بترجمة المخطط الانسيابي إلى خوارزمية ومقطع برمجي بلغة Python.

18, -10, -5, 19, 20, -25, 17, -20,20



3 قم بتشغيل المقطع البرمجي التالي وأكمل الجدول.

```
x=2
c=5
while c>0:
    for i in range (7,12,2):
        x=x+3
        print(i,x)
    c=c-3
    print(x,c)
```

جدول القيم

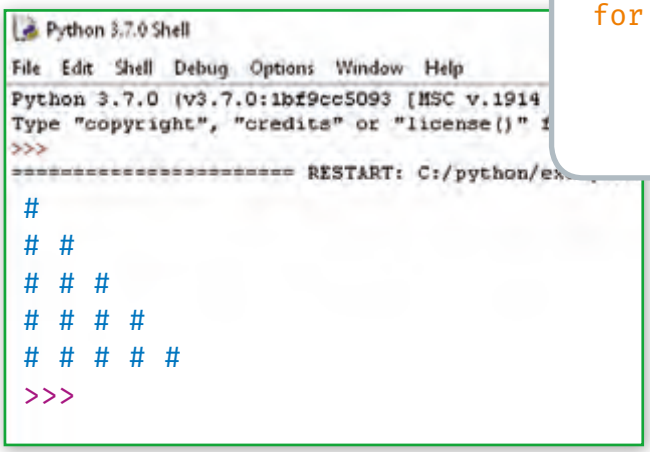
x	c	i	screen
2	5		
5		7	7 5
	2		
		7	7 14
			20 -1

4 ارسم نتيجة المقطع البرمجي التالي.

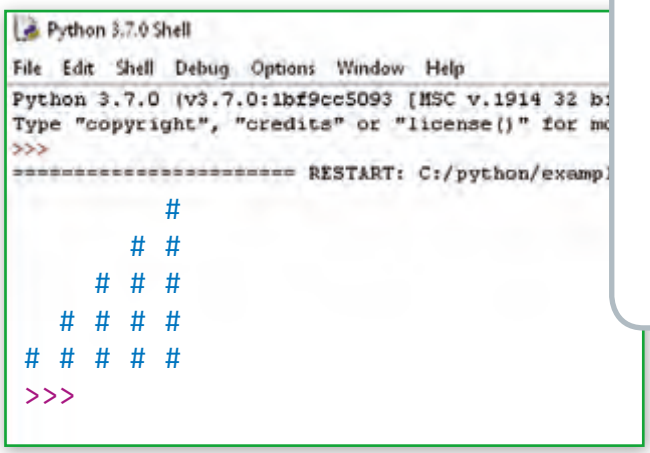
```
n = 5
for i in range(n):
    for j in range(i):
        print('* ', end="")
    print('')

for i in range(n,0,-1):
    for j in range(i):
        print('* ', end="")
    print('')
```

5

قم بإكمال الفراغات الموجودة في مقطع Python البرمجي لعرض النمط التالي على الشاشة.



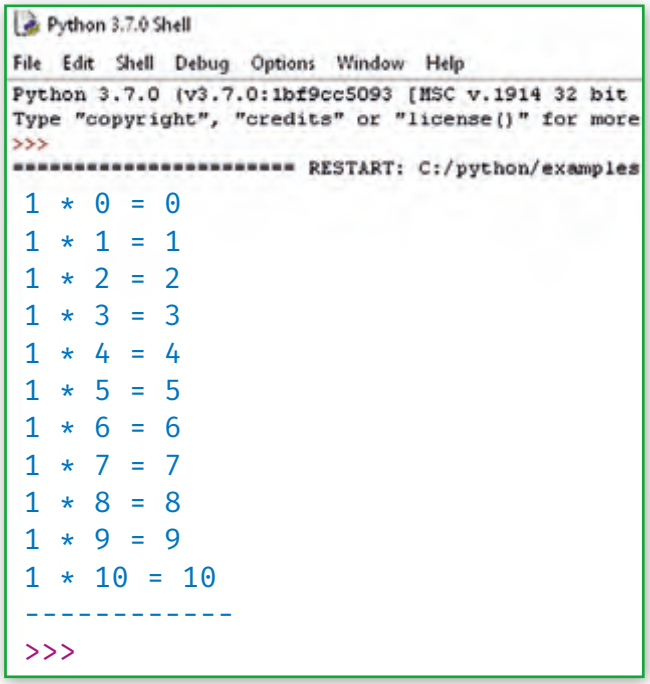

```
for num in range (______):
    for ____ in range (______):
        print("#",end=" ")
    print(" ")
```




```
k = 8
for ____ in range(______):
    for j in range(____, k):
        print(end=" ")
    k = k - 2
    for j in range(0, i+1):
        ______("#",end=" ")
    print()
```

6

أنشئ برنامجًا بلغة Python يعرض جدول الضرب على الشاشة.



7

في بطولة العالم لألعاب القوى، يمكن للمتسابقين في رياضة الوثب الثلاثي القيام بثلاث محاولات، قم بإنشاء برنامج Python يقوم بالتالي:

> قراءة أسماء 20 متسابق.

> قراءة قيم القفزات الثلاث لكل متسابق.

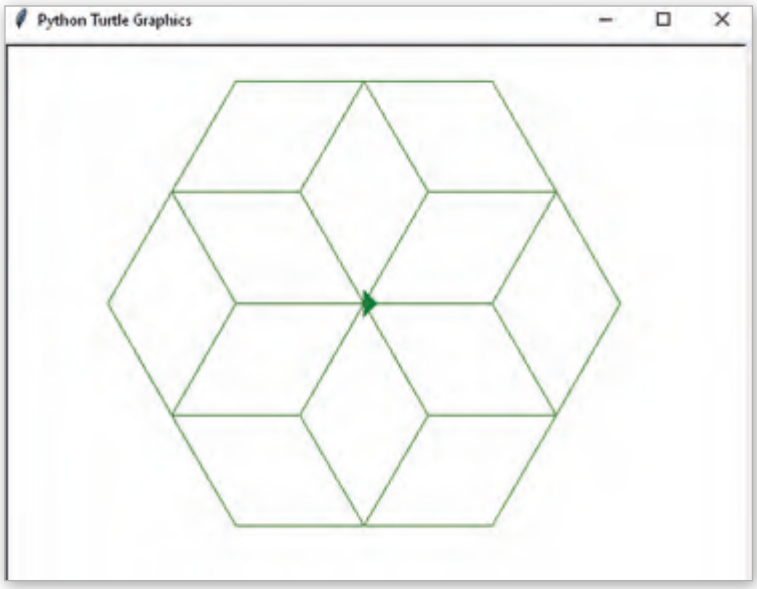
> حدد القفزة الأعلى بين المحاولات الثلاثة للاعب.

> عرض اسم المتسابق الذي حاز على الميدالية الذهبية على الشاشة.

8

قم بإكمال الفراغات الموجودة في مقطع Python البرمجي لعرض الرسم التالي.

```
from turtle import*
pen=Turtle()
pen.shape("arrow")
pen.color("green")
pen.speed(20)

for j in range (______):
  for ______ in range (6):
      pen.forward(60)
      pen.left(60)
  ______.left(60)
```



9

إذا اعتبرنا أن السنة تتكون من 12 شهر وأن الشهر يتكون من 30 يوم ، اكتب برنامجًا بلغة البرمجة Python يمكن من إظهار التواريخ الممكنة لسنة 2020 على الشاشة (أول تاريخ هو 2020/1/1 وآخر تاريخ هو 2020/12/30).

10

قام قسم تكنولوجيا المعلومات بالمدرسة بترقيم أجهزة الحاسوب في المعمل كالتالي:

> الصف الأول من الحواسيب: 14 13 12 11

> الصف الثاني من الحواسيب: 24 23 22 21

> الصف الأخير من الحواسيب: 64 63 62 61

مثال:

> الجهاز رقم 42 هو ثاني جهاز في الصف الرابع

> الجهاز رقم 63 هو ثالث جهاز في الصف السادس.

lab	1	2	3	4	5	6
1	61	51	41	31	21	11
2	62	52	42	32	22	12
3	63	53	43	33	23	13
4	64	54	44	34	24	14

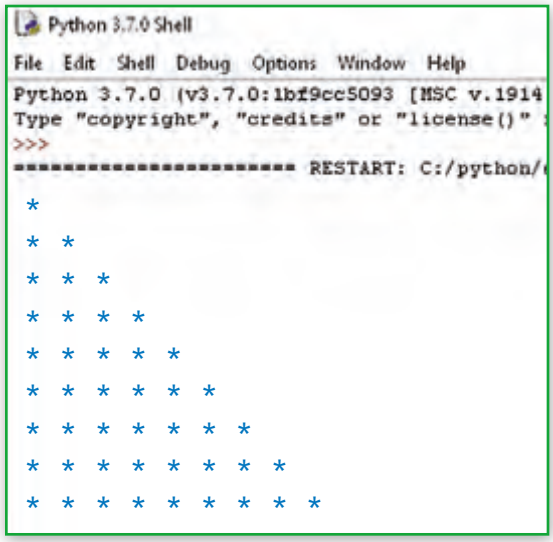
> أكتب برنامجا بلغة Python يمكن من طباعة أرقام جميع الحواسيب الموجودة بالمعمل مع العلم أن المعمل يتكون من ستة صفوف (تمثلها أرقام من 1 الى 6) وكل سطر يتكون من أربعة حواسيب (تمثلها أرقام من 1 الى 4).

11

اكتب مقطعًا برمجيًا لعرض نمط هندسي على الشاشة، وذلك بتنفيذ الآتي:

> اكتب برنامج ex332.py بلغة Python يمكن من رسم وعرض الشكل التالي على الشاشة.

> احفظ البرنامج في مجلد العمل الخاص بك.



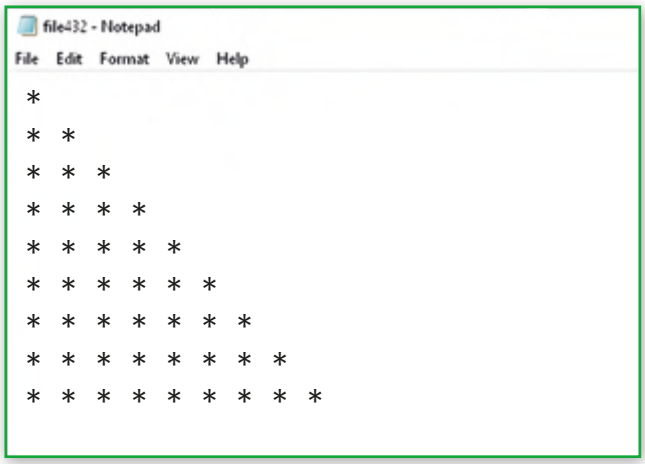
12

اكتب مقطعًا برمجيًا لكتابة نمط هندسي في ملف نصي، وذلك بتنفيذ الآتي:

> إنشاء ملف نصي file432.txt في مجلد العمل الخاص بك.

> فتح الملف لغرض الكتابة.

> رسم الشكل التالي داخل الملف.



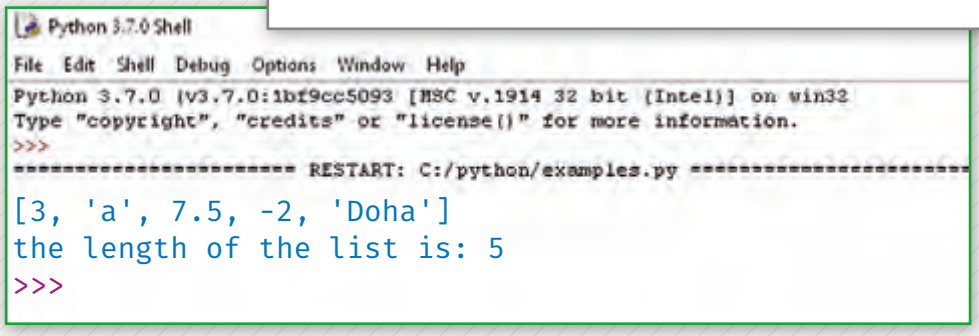
الدرس الثالث

القوائم المتداخلة والمصفوفات ثنائية الأبعاد

القائمة هي إحدى هياكل البيانات التي تستخدم في **Python** لحفظ البيانات. لا تحتاج عناصر القائمة إلى أن تكون من نفس النوع بل ويمكن تغييرها أثناء تنفيذ البرنامج، كما ويمكن أيضًا تغيير حجم القائمة أثناء تنفيذ البرنامج بحيث تتم إضافة عناصر أو إزالتها منها.

مثال 1

```
list1=[3,"a",7.5,-2,"Doha"]
print(list1)
l=len(list1)
print("the length of the list is:",l)
```



لاحظ في المثال السابق أن القائمة تحوي عناصر من أنواع مختلفة: أعداد صحيحة وحقيقية وحرف ونص.

القوائم المتداخلة

يمكن أن تحتوي القائمة على أنواع مختلفة من العناصر وقد تحتوي على قائمة أخرى، ويُطلق عليها حينها مُسمى "القوائم المتداخلة".

مثال 2

list1[0]	3	0
list1[1]	[a,b,c]	1
list1[2]	7.5	2
list1[3]	-2	3
list1[4]	Doha	4

العنصر الثاني من القائمة في المثال التالي عبارة عن قائمة.

```
list1=[3,["a","b","c"],7.5,-2,"Doha"]
print(list1)
```

```
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,['a','b','c'], 7.5, -2, 'Doha']
>>>
```

القائمة الداخلية تشبه العناصر الأخرى في القائمة، فيتم التعامل معها كبقية العناصر دون اعتبار لطول القائمة.

```
list1=[3,["a","b","c"],7.5,-2,"Doha"]
print(list1[1])
l=len(list1)
print("the length of the list is:",l)
```

```
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 ==========
['a', 'b', 'c']
the length of the list is: 5
>>>
```

مثال 3

list2[0]	[1,2]	0
list2[1]	[3,4]	1
list2[2]	[5,6,7]	2

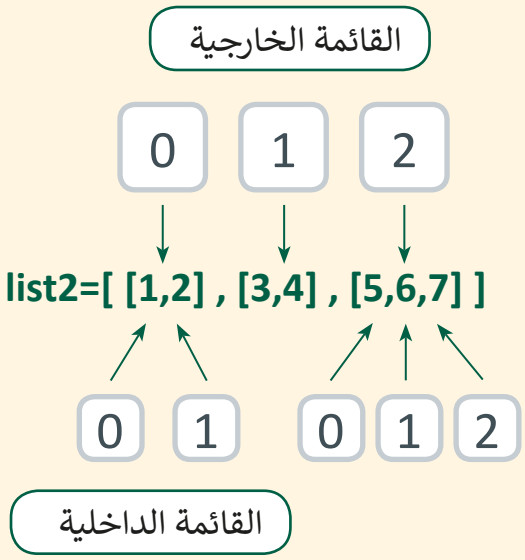
في هذا المثال فإن جميع عناصر القائمة عبارة عن قوائم.

```
list2=[[1,2],[3,4],[5,6,7]]
print(list2)
print(list2[1])
```

```
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 ==========
[[1, 2], [3, 4], [5, 6, 7]]
[3, 4]
>>>
```

لعرض عنصر من عناصر قائمة داخلية، نحتاج إلى رقمين، الأول هو رقم فهرس العنصر في القائمة الخارجية والثاني هو فهرس القائمة الداخلية.

نظام ترقيم القائمة يبدأ من 0 وليس من 1.



أكمل الفراغات بالعنصر المناسب طبقًا لقائمة list2.

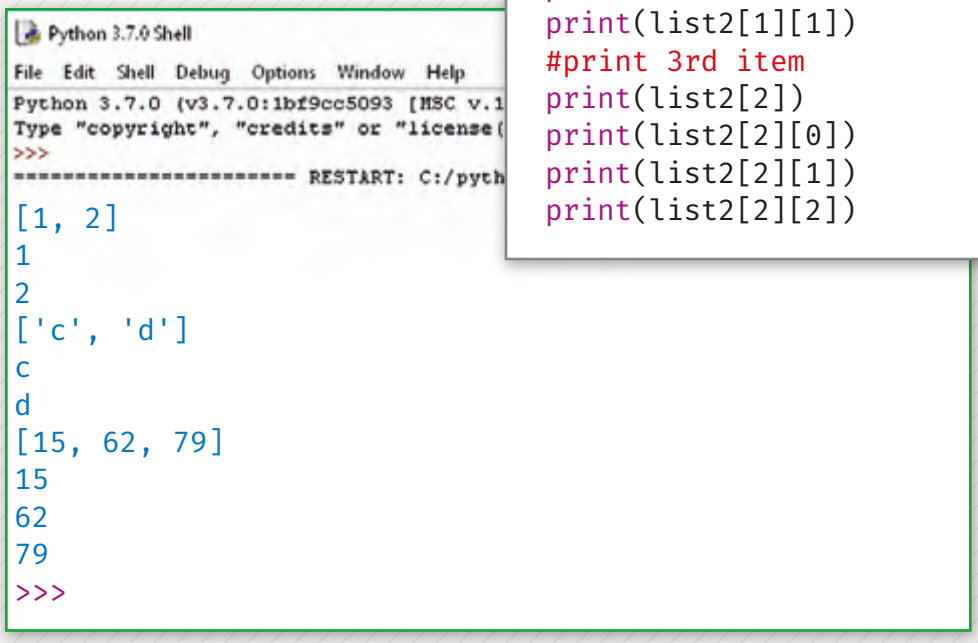
```
list2[0][1]= 2
list2[1][0]= 3
list2[1][1]=______
list2[2][2]=______
list2[0][0]=______
list2[2][0]=______
```

مثال 4

list2[0]	[1,2]	0
list2[1]	["c","d"]	1
list2[2]	[15,62,79]	2

في هذا المثال نقوم بطباعة القوائم المتداخلة وجميع عناصرها.

```
list2=[[1,2],["c","d"],[15,62,79]]
#print the 1st item
print(list2[0])
print(list2[0][0])
print(list2[0][1])
#print the 2nd item
print(list2[1])
print(list2[1][0])
print(list2[1][1])
#print 3rd item
print(list2[2])
print(list2[2][0])
print(list2[2][1])
print(list2[2][2])
```



يمكننا استخدام التكرارات المتداخلة لإخراج نفس النتيجة. قم بالتحقق بهذا المقطع البرمجي.

```
list2=[[1,2],["c","d"],[15,62,79]]
for i in list2:
    #print the items of the outer list
    print(i)
    for j in i:
        #print the items of the inner list
        print(j)
```

علينا أن نكون حذرين بأرقام الفهرس لكي لا يتم الإشارة إلى عنصر غير موجود في القائمة الخاصة بنا.

مثال 5

list2[0]	[1,2]	0
list2[1]	[3,4]	1
list2[2]	[5,6,7]	2

```
list2=[[1,2],[3,4],[5,6,7]]
print(list2[1][3])
```

```
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 ==========
Traceback (most recent call last):
  File "C:/python/G11CS/nested lists/test.py", line 16, in <module>
    print(list2[1][3])
IndexError: list index out of range
>>>
```

اكتب الأوامر المناسبة لعرض الرقم 7 على الشاشة.

```
list2=[[1,2],[3,4],[5,6,7]]
```

المصفوفة ثنائية الأبعاد Array [2D]

لقد تعرفنا سابقًا على المقصود بالمصفوفة ثنائية الأبعاد، وسنرى في هذا الدرس كيفية التعامل مع تلك المصفوفة في **Python**.

يتم تمثيل المصفوفات ثنائية الأبعاد في **Python** بقوائم متداخلة.

كما رأينا بالفعل، يتم تمثيل البيانات في المصفوفة ثنائية الأبعاد بصفوف وأعمدة، وتتم فهرسة كل عنصر في المصفوفة ثنائية الأبعاد بواسطة فهرسين، أحدهما للصف والآخر للعمود.

الصيغة العامة للمصفوفة ثنائية الأبعاد:

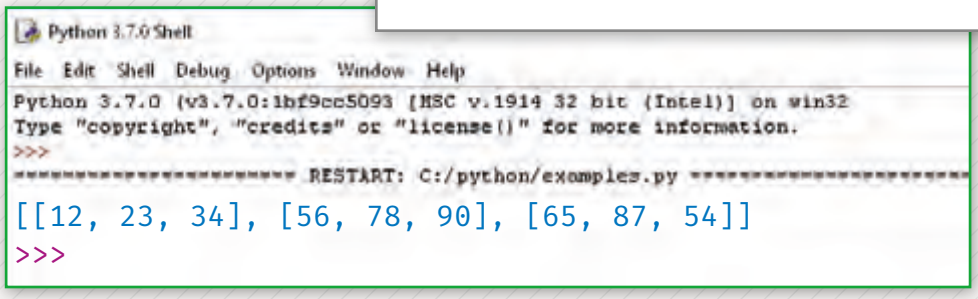
x	عمود 0	عمود 1	عمود 2
الصف 0	x[0][0]	x[0][1]	x[0][2]
الصف 1	x[1][0]	x[1][1]	x[1][2]
الصف 2	x[2][0]	x[2][1]	x[2][2]

array2D	عمود 0	عمود 1	عمود 2
الصف 0	12	23	34
الصف 1	56	78	90
الصف 2	65	87	54

مثال 6

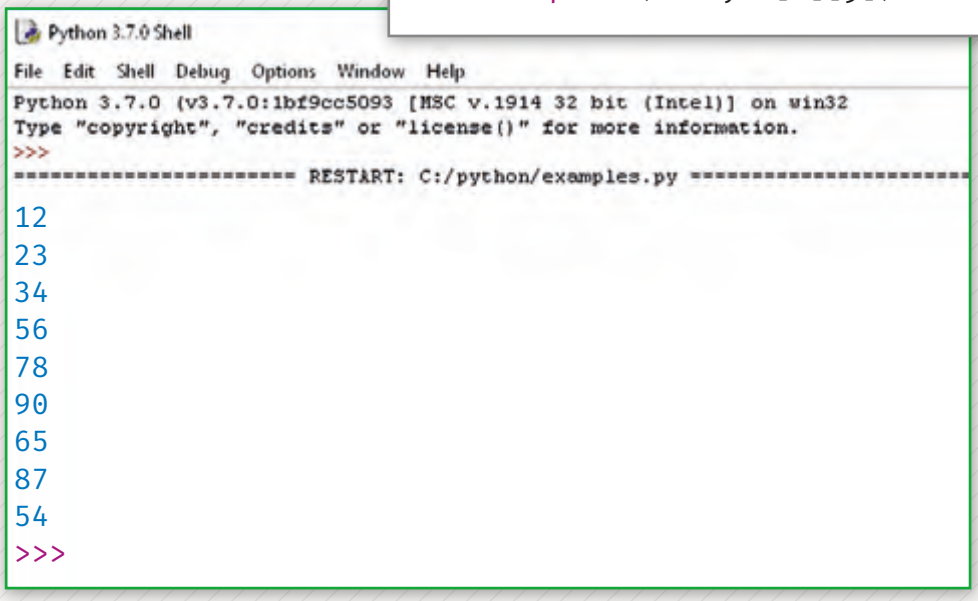
array2D[0]	[12,23,34]	0
array2D[1]	[56,78,90]	1
array2D[2]	[65,87,54]	2

```
array2D=[[12,23,34],[56,78,90],[65,87,54]]
print(array2D)
```



لطباعة جميع عناصر المصفوفة نحتاج إلى استخدام التكرارات المتداخلة، إحداها للصف والآخر للعمود.

```
for i in range (3):
    for j in range (3):
        print(array2D[i][j])
```



مثال 7

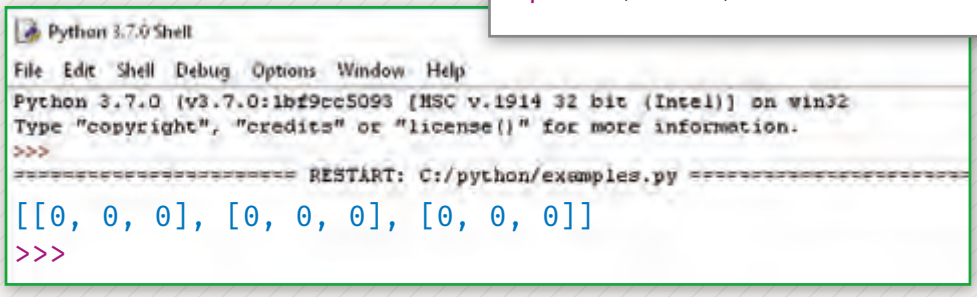
فلننشئ الجدول التالي.

zeros	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0

إنشاء مصفوفة خارجية فارغة.

إنشاء مصفوفة داخلية فارغة.

```
row=3 #number of rows
col=3 #number of columns

zeros = []
for r in range(row):
    zeros.append([])
    for c in range(col):
        zeros[r].append(0)
print(zeros)
```



حاول إنشاء الجدول التالي.

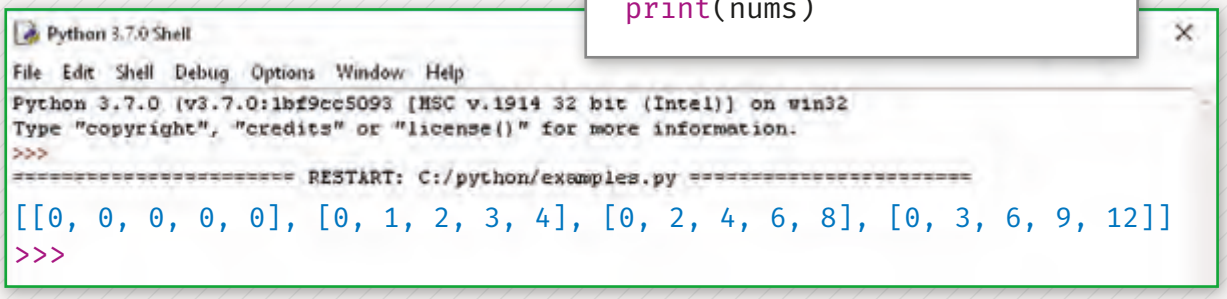
nums	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

مثال 8

لنقم بإنشاء مصفوفة محتواها ناتج ضرب الفهرس الخاص بالصف مع العمود.

nums	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	6	8
3	0	3	6	9	12

```
row=4 #number of rows
col=5 #number of columns

nums = []
for r in range(row):
    nums.append([])
    for c in range(col):
        nums[r].append(r*c)
print(nums)
```

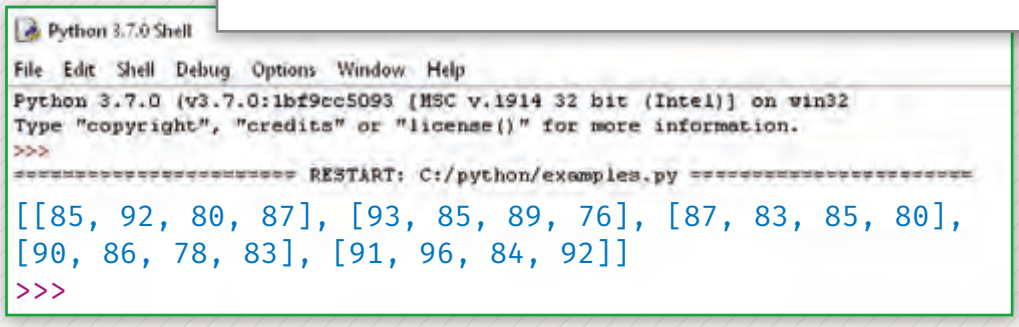


في المثال التالي لدينا درجات 5 طلاب في 4 مواد مختلفة.

- نريد حساب متوسط الدرجات لخمس طلاب في أربع مواد. سنقوم بجمع كافة البيانات في مصفوفة ثنائية الأبعاد.
- يمثل كل صف درجات كل طالب في 4 مواد.
- يمثل كل عمود درجات الطلبة في مادة دراسية واحدة.

مثال 9

```
grades=[[85,92,80,87],[93,85,89,76],[87,83,85,80],
        [90,86,78,83],[91,96,84,92]]
print(grades)
```



المواد الدراسية

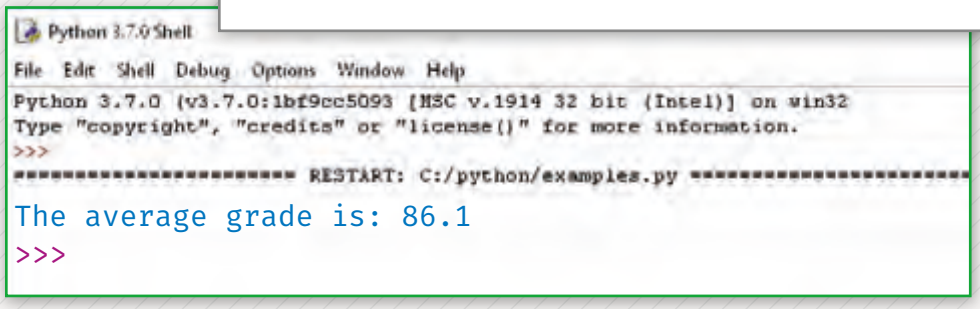
الطلبة

grades	0	1	2	3
0	85	92	80	87
1	93	85	89	76
2	87	83	85	80
3	90	86	78	83
4	91	96	84	92

مثال 10

لحساب متوسط درجات جميع الطلبة، علينا أن نجمع جميع درجات المصفوفة، وكما فعلنا سابقًا في أمر الطباعة، سنحتاج إلى تكرارين متداخلين من خلال الجدول، إحداها للصف والآخر للعمود.

```
grades=[[85,92,80,87],[93,85,89,76],[87,83,85,80],
        [90,86,78,83],[91,96,84,92]]

row=5 #number of rows
col=4 #number of columns
#initialize the accumulator
sumGrades=0
for r in range (row):
    for c in range (col):
        sumGrades=sumGrades+grades[r][c]

averageGrade=sumGrades/(row*col)
print("The average grade is: ", averageGrade)
```



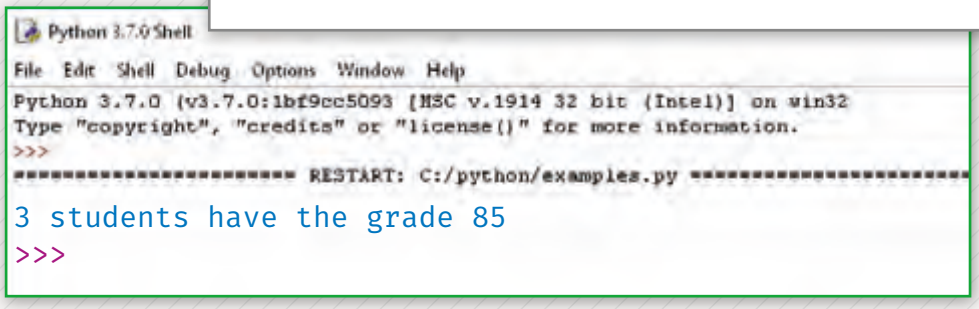
نصيحة ذكية

لا تكتفِ بقراءة الكود البرمجي فقط، بل تدرب عليه.

مثال 11

فلنقم بحساب عدد مرات وجود الدرجة 85 داخل المصفوفة.

```
grades=[[85,92,80,87],[93,85,89,76],[87,83,85,80],
       [90,86,78,83],[91,96,84,92]]

row=5 #number of rows
col=4 #number of columns
gr=85 #the grade we want to count
#initialize the counter
counter=0
for r in range (row):
    for c in range (col):
       if grades[r][c]==gr:
           counter=counter +1

print(counter,"students have the grade 85")
```



لا تنسَ أن فهرسة القوائم تبدأ من 0

نصيحة ذكية

تحقق دومًا من المقطع البرمجي الخاص بك لتجنب الأخطاء.

في الأمثلة السابقة تم إدخال الدرجات من قِبل المبرمج. سنقوم الآن بإنشاء مصفوفة وإدخالها يدويا من قبل المستخدم.

مثال 12

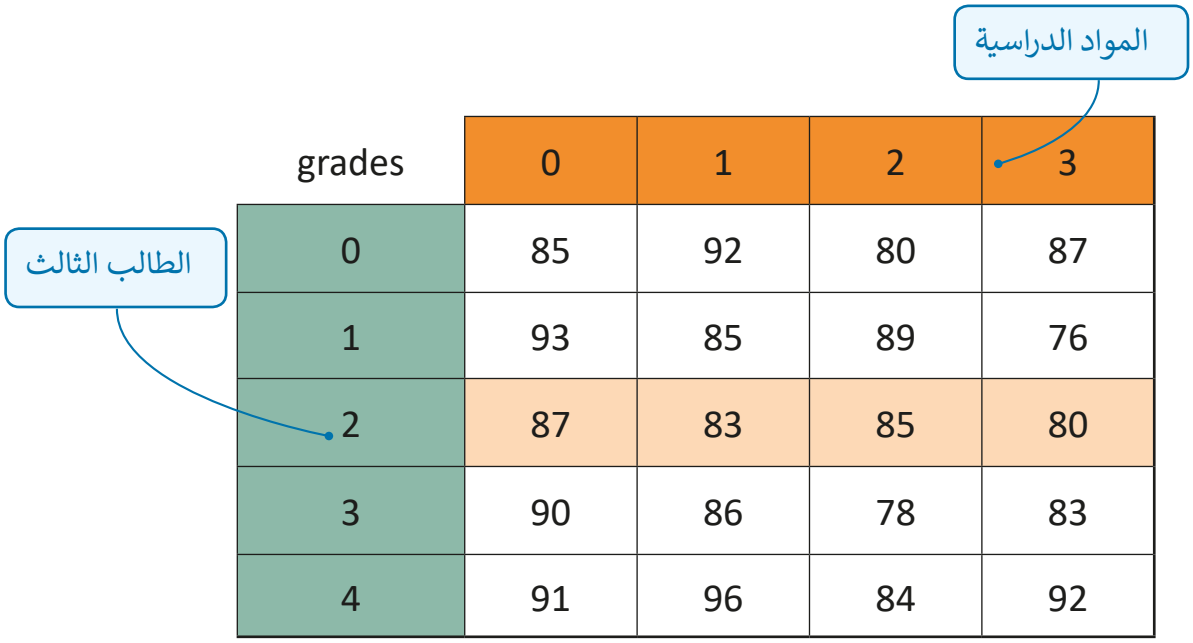
إنشاء قائمة فارغة.

إنشاء قائمة متداخلة.

```
row=5
col=4
grades = []
for r in range(row):
    print("student #",r+1)
    grades.append([])
    print("Enter the grades")
    for c in range(col):
        print("Grade:",c+1,)
        grades[r].append(int(input()))
print(grades)
```

```
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 wir
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
student #1
Enter the grades
Grade: 1
85
Grade: 2
92
Grade: 3
80
student #2
Enter the grades
Grade: 1
93
Grade: 2
85
Grade: 3
89
......
......
[[85,92,80,87],[93,85,89,76],[87,83,85,80],
[90,86,78,83],[91,96,84,92]]
>>>
```

نريد حساب متوسط درجات الطالب الثالث. علينا أن نجمع فقط درجات الصف الثالث من المصفوفة.



grades	0	1	2	3
0	85	92	80	87
1	93	85	89	76
2	87	83	85	80
3	90	86	78	83
4	91	96	84	92

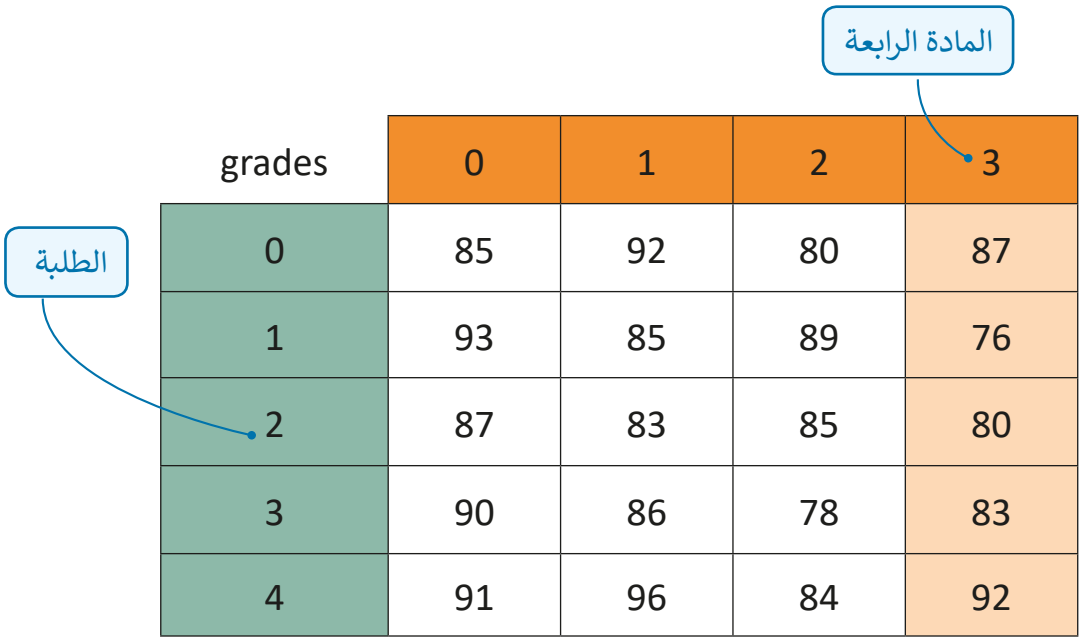
مثال 13

فهرس الصف الثالث هو رقم 2 ويبدأ كفهرس للعد من الصفر.

```
sumGrades=0
print("the grades of the 3rd student are:")
for c in range(col):
    print(grades[2][c])
    sumGrades=sumGrades + grades[2][c]

averageGrade=sumGrades/col
print("the average grade of the 3rd student is:", averageGrade)
```

```
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 ==========
the grades of the 3rd student are:
87
83
85
80
the average grade of the 3rd student is: 83.75
>>>
```

نريد حساب متوسط درجة المادة الرابعة وهي مادة البرمجة، ، تابع الخطوات الآتية:



grades	0	1	2	3
0	85	92	80	87
1	93	85	89	76
2	87	83	85	80
3	90	86	78	83
4	91	96	84	92

مثال 14

فهرس العمود الرابع هو الرقم 3 والذي سيبدأ العد كفهرس بدءًا من الصفر.

```
sumGrades=0
print("the grades in programming are:")
for r in range(row):
    print(grades[r][3])
    sumGrades=sumGrades+ grades[r][3]

averageGrade=sumGrades/row
print("the average grade in programming is:", averageGrade)
```

```
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
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/examples.py ==========
the grades in the 4th lesson are:
87
76
80
83
92
the average grade in programming is: 83.6
>>>
```

1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

.1	يقوم Python بإنشاء مصفوفة ثنائية الأبعاد 2D array باستخدام القوائم.	○
.2	يتم تمثيل البيانات في المصفوفة ثنائية الأبعاد 2D array فقط من خلال الصفوف.	○
.3	تتم فهرسة كل عنصر في المصفوفة ثنائية الأبعاد 2D array بواسطة فهرسين.	○
.4	إن الفهرسين في عنصر المصفوفة ثنائية الأبعاد 2D arrow، أحدهما للعمود والآخر لحجم المصفوفة.	○
.5	لطباعة جميع عناصر المصفوفة، نحتاج إلى استخدام التكرارات المتداخلة.	○

2

استخدم المصفوفة ثنائية الأبعاد لملء الفراغات.

nums	0	1	2	3
0	0	1	2	3
1	4	5	5	7
2	8	9	10	11
3	12	13	14	15

```
nums[0][2]=________
nums[1][1]=________
nums[2][3]=________
nums[3][2]=________
nums[0][3]=________
nums[1][0]=________
nums[1][3]=________
nums[2][1]=________
```

3

اختر الإجابة الصحيحة مما يلي وتحقق من إجابتك باستخدام حاسوبك.

ما الذي سيتم طباعته بواسطة المقطع البرمجي التالي؟

.1

```
list=[3,4,["a","b"]]
print(list[2][1])
```

- 3
- 4
- a
- b

.2

```
list=[1,"a",[0,"b"]]
print(list[1])
```

- 1
- a
- 0
- b

.3

```
list=[3,["c","d"],["a","b"]]
print(list[1][1])
```

- a
- d
- 3
- c

.4

```
list=[[7,"a"],4,["c","b"]]
print(list[0][1])
```

- 7
- a
- c
- b

4

أنشئ برنامج Python يقوم بحساب جدول الضرب التالي.

nums	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

5

تمثل المصفوفة التالية المقاعد المتاحة في الملعب الرياضي.

> 0 يعني أن المقعد متاح.

> 1 يعني أن المقعد غير متاح.

seats	0	1	2	3	4
0	0	0	0	1	1
1	0	1	0	0	0
2	0	1	0	1	0
3	1	1	1	0	1
4	1	1	1	1	0

أنشئ برنامج Python يقوم بالتالي:

> إنشاء مصفوفة.

> سؤال المستخدم عن عدد التذاكر التي يريد شراءها.

> عرض رسالة على الشاشة تخبر المستخدم بتوفر العدد المطلوب من المقاعد.

6

أنشئ الجدول التالي والذي يمثل درجات الحرارة المسجلة في 5 مدن أوروبية خلال فصل الربيع.

temperatures	March	April	May
London	12	15	18
Rome	17	19	24
Paris	13	17	20
Berlin	9	14	19
Madrid	16	18	22

أنشئ برنامج Python يقوم بالتالي:

> حساب متوسط درجة الحرارة في المدن الخمس للثلاث شهور.

> حساب متوسط درجة الحرارة في مدينة باريس.

> حساب متوسط درجة الحرارة في شهر أبريل.

> عرض أقصى درجة حرارة موجودة في الجدول.

> حفظ جميع درجات الحرارة للمدن الخمس في شهر مارس بالملف march.txt

7 عرف المصفوفة ثنائية الأبعاد والموضحة في الجدول أدناه، مستخدمًا في ذلك البرمجة بلغة Python.

> الجدول التالي يعبر عن أماكن جلوس بعض الطلبة بمعمل الحاسوب:

lab	1	2	3	4
1	st3	st6	st9	st12
2	st2	st5	st8	st11
3	st1	st4	st7	st10

> مستخدمًا الصيغة العامة لتعريف القوائم، قم بتعريف قائمة Class تعبر عن أماكن جلوس الطلبة.

8 اكتب مقطعًا برمجيًا بلغة Python يستخدم المصفوفات ثنائية الأبعاد لتمثيل بعض بيانات العائلات التي تسكن في حي ما، وذلك يتنفيذ الآتي:

> شغل البيئة البرمجية IDLE (Python 3.7) الموجودة على جهازك

> أنشئ برنامجًا تحت مسمى List422.py ثم قم بحفظه داخل المجلد الخاص بك.

> اكتب مقطعًا برمجيًا يمكن من:

- إنشاء القائمة Families التي تتكون عناصرها من اسم الأب، عدد الأبناء الذكور و عدد البنات الإناث كالتالي:

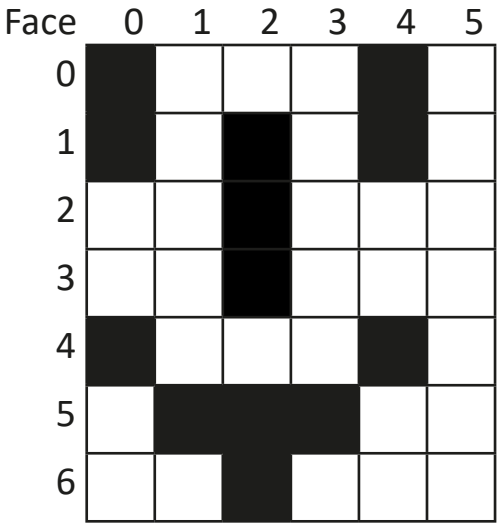
[[Hamad,2,3], [Jassem,2,4], [Ibrahem,4,0]]

- عرض جميع عناصر القائمة عنصرًا بعد الآخر على الشاشة.

9

في مادة تكنولوجيا المعلومات تعلمنا أن ترميز الصور النقطية يكون عبر النقاط بكسل، يعبر الرسم أدناه عن تكبير لصورة نقطية حيث نرمز للون الأسود ب 1 واللون الأبيض ب 0.

قم بتمثيل الشكل أدناه برمجيًا باستخدام القوائم والصيغ البرمجية المناسبة باتباع الخطوات الآتية:



> شغل البيئة البرمجية IDLE (Python 3.7) الموجودة على جهازك.

> أنشئ برنامجًا جديدًا باسم List423.py ثم قم بحفظه داخل المجلد الخاص بك.

> اعتمادًا على ما درست اقترح طريقة لتمثيل الصورة أعلاه:

- اسم القائمة المقترحة: .....
- عدد عناصر القائمة المقترحة: .....
- نوع عناصر القائمة المقترحة: .....
- وصف لعناصر القائمة المقترحة: .....

> برمجيًا، وباستخدام الصيغ التكرارية المناسبة، أنشئ الجدول التالي في شكل مصفوفة ثنائية الأبعاد (قوائم متداخلة):

Face	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

> قم بالتعديلات اللازمة على القائمة Face للتمكن من ترميز الصورة الآتية:

Face	0	1	2	3	4	5
0	■				■	
1	■		■		■	
2			■			
3			■			
4	■				■	
5		■	■	■		
6			■			

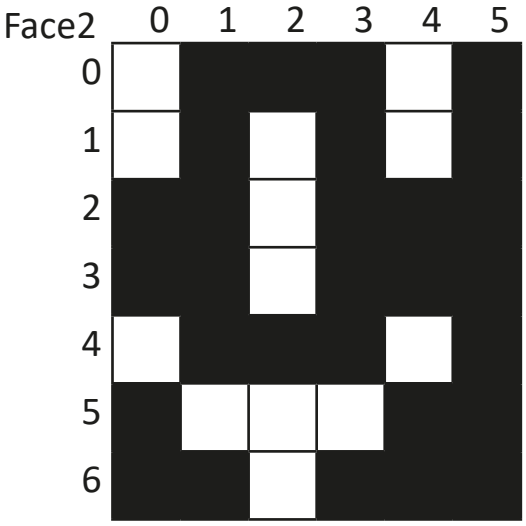
10

اكتب مقطعًا برمجيًا لإنشاء مصفوفة ثنائية الأبعاد محتواها ناتج العمليات الحسابية على عناصرها، وذلك بتنفيذ الآتي:

> على نفس البرنامج السابق List423 قم بإضافة المقطع البرمجي الذي يمكن من:

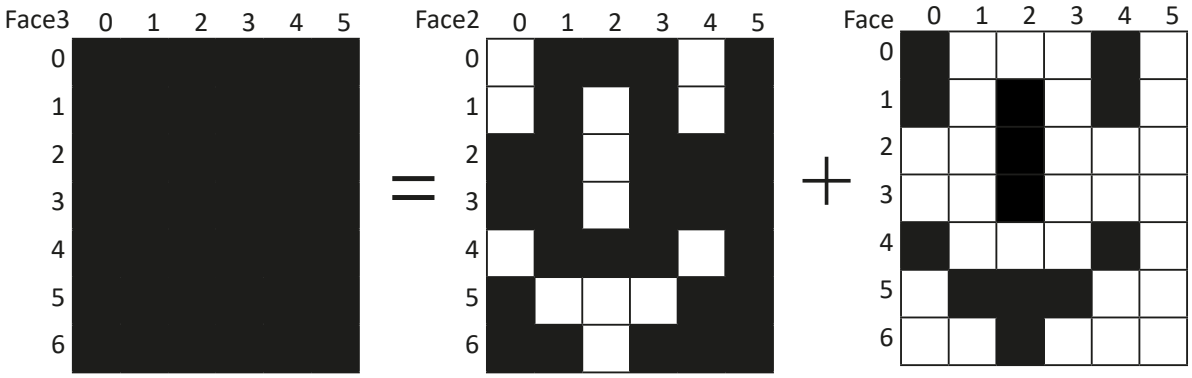
> نسخ القائمة Face في قائمة جديدة Face2.

> عكس ترميز الصورة Face2 لتصبح كالتالي:



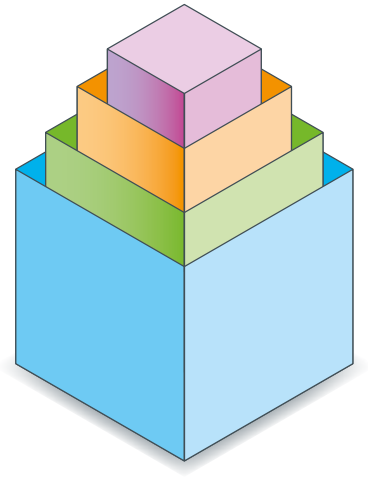
ملاحظة: استخدم التكرارات المتداخلة.

> مطابقة الصورتين Face و Face2: جمع عناصر القوائم Face و Face2 للحصول على القائمة Face3.



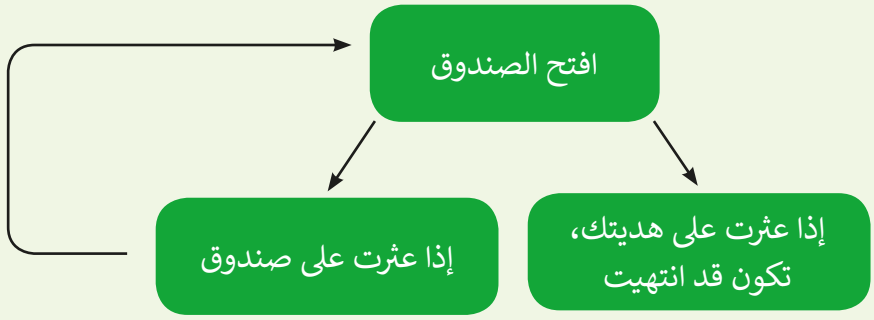
الدرس الرابع

الاستدعاء الذاتي



لو أحضر والداك لك هدية، وأنت تتوق الآن لمعرفة ماهيتها، ولكن حين تقوم بفتح الصندوق فإنك تجد صندوقًا جديدًا داخل ذلك الصندوق، وبالتالي سيكون هناك صناديق داخل صناديق ولن تعرف في أي صندوق من تلك الصناديق توجد الهدية.

خطوات العثور على هديتك هي:



المثال السابق الذي تم وصفه في المخطط السابق مثال على الاستدعاء الذاتي (التكرار).

يحدث الاستدعاء الذاتي عندما نكرر نفس التعليمات لكن بمعطيات مختلفة وأقل تعقيدا.

يعتبر الاستدعاء الذاتي إحدى طرق حل المشكلات في علم الحاسوب وذلك من خلال القيام بتقسيم المشكلة إلى مجموعة مشاكل صغيرة شبيهة بالمشكلة الأصلية بحيث يمكننا استخدام نفس الخوارزمية لحل تلك المشاكل. يتم استخدام الاستدعاء الذاتي من قِبل نظام التشغيل وبواسطة التطبيقات الأخرى، كما وتدعمه معظم لغات البرمجة أيضًا.

الدوال

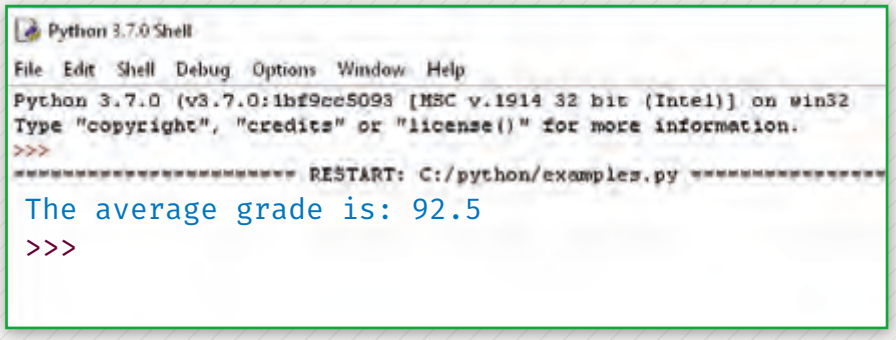
الدالة هي عبارة عن مقطع برمجي قابل لإعادة الاستخدام بحيث يُستخدم لتنفيذ إجراء معين في البرمجة، يتم استخدام الدوال لجعل البرنامج أكثر كفاءة، ويتم استدعاؤها بواسطة البرنامج الرئيس أو بواسطة دوال أخرى في البرنامج.

مثال 1

لنستعرض مثالًا على دالة تستدعي دالة أخرى.

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

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

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

استدعاء للدالة mySumGrade

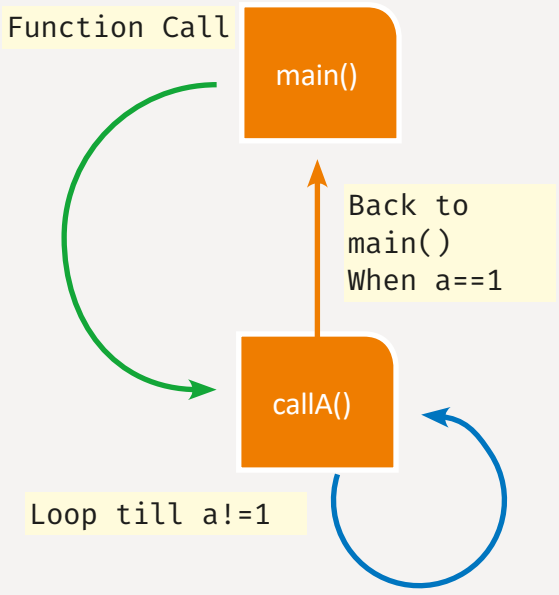


دالة الاستدعاء الذاتي Recursive Function

في بعض الحالات، من الممكن أن تستدعي الدالة نفسها، وهذه الخاصية تُسمى بالاستدعاء الذاتي.

الاستدعاء الذاتي هي العملية التي تستدعي بها الدالة نفسها.

التركيب العام لدالة الاستدعاء الذاتي




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

#main program
.......

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

تتكون دالة الاستدعاء الذاتي من حالتين:

- الحالة الأساسية (Base case).

هي الحالة التي تتوقف فيها الدالة عن استدعاء ذاتها، ويتم التحقق من الوصول للحالة الأساسية من خلال جملة شرطية، وبدون الحالة الأساسية ستكون عملية الاستدعاء الذاتي لا نهائية.

- حالة الاستدعاء الذاتي (Recursive case).

هي الحالة التي تستدعي فيها الدالة ذاتها عند عدم تحقق شرط التوقف، وتبقى الدالة في حالة الاستدعاء الذاتي لحين وصولها إلى الحالة الأساسية.

من أكثر الأمثلة الشائعة على استخدام الاستدعاء الذاتي عملية حساب مضروب رقم معين. يعرف مضروب الرقم بأنه ناتج ضرب الأعداد الصحيحة من 1 إلى هذا العدد، ويتم التعبير عن المضروب بالرقم متبوعًا بالرمز ! مثلًا، !5

مثال على المضروب			
0!	**0!=1**		
1!	**1!=1*1**	**or**	1!=1*0!
2!	**2!=2*1**	**or**	2!=2*1!
3!	**3!=3*2*1**	**or**	3!=3*2!
4!	**4!=4*3*2*1**	**or**	4!=2*3!
5!	**5!=5*4*3*2*1**	**or**	5!=5*4!

ما نلاحظه هنا هو أن حساب المضروب يعتمد على القاعدة أدناه:

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

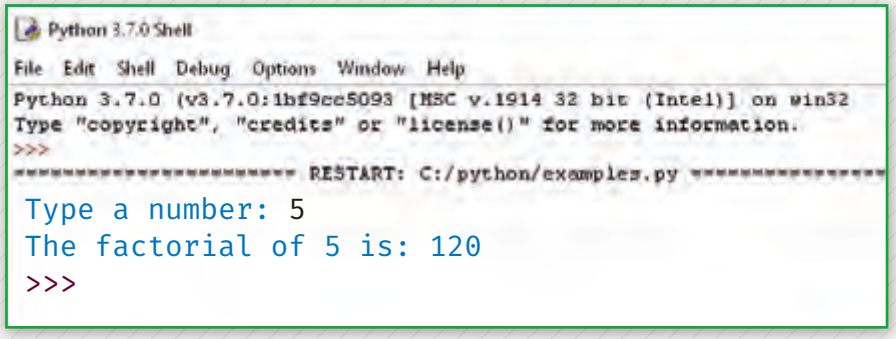
حالة الأساس

حالة الاستدعاء الذاتي

مثال 2

فلننشئ دالة factorialLoop تقوم بحساب مضروب الرقم باستخدام تكرار for.

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

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

    return result

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



قم بتشغيل البرنامج السابق وأكمل الجدول.

جرب كتابة الخوارزمية ومخطط التدفق لهذا البرنامج.

مثال على المضروب	
0!	1
7!	
8!!	
9!	
10!	

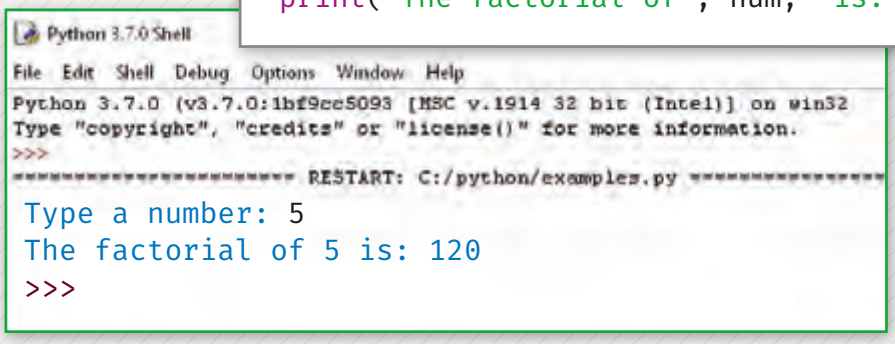
مثال 3

لنحسب الآن مضروب رقم باستخدام دالة factorial.

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

    return fact

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

حالة الأساس

حالة الاستدعاء الذاتي



الإيجابيات والتحديات لاستخدام دوال الاستدعاء	
الإيجابيات	**التحديات**
دوال الاستدعاء الذاتي تختزل المقطع البرمجي إلى عدد أصغر من التعليمات	في بعض الأحيان يكون من الصعب تتبع المنطق المتبع في دوال الاستدعاء الذاتي
يمكن تقسيم المهمة الصعبة إلى مجموعة من المشاكل الفرعية باستخدام الاستدعاء الذاتي	الاستدعاء الذاتي يتطلب المزيد من الذاكرة والوقت
في بعض الأحيان يكون من السهل استخدام الاستدعاء الذاتي بديلًا عن التكرارات المتداخلة	ليس من السهل تحديد الحالات التي يمكن فيها استخدام دوال الاستدعاء الذاتي

الاستدعاء الذاتي والتكرار

تشترك فكرة الاستدعاء الذاتي والتكرار في تنفيذ مجموعة من التعليمات مرارًا وتكرارًا، ويكمن الاختلاف الرئيسي بين الاستدعاء الذاتي والتكرار بالطريقة التي يتم بها إنهاء دالة الاستدعاء الذاتي، فدالة الاستدعاء الذاتي تستدعي نفسها وتقوم بإنهاء تنفيذها عند الوصول إلى حالة الأساس، ولكن من جهة أخرى يتم تنفيذ التكرار حذف بشكل متكرر حتى يتم استيفاء شرط معين أو في نهاية التسلسل الذي يتم تنفيذه لتكرار **for**.

نستعرض هنا بعض الاختلافات بين الاستدعاء الذاتي والتكرار في الجدول التالي.

الاستدعاء الذاتي والتكرار	
الاستدعاء الذاتي	**التكرار**
البطء في التنفيذ مقارنة بالتكرار	تنفيذه أسرع
يحتاج لمزيد من الذاكرة	يحتاج إلى ذاكرة أقل
حجم المقاطع البرمجية أقل	حجم المقاطع البرمجية أكبر
يتنهي عند الوصول إلى حالة الأساس	ينتهي باستكمال عدد التكرارات المحددة أو تحقق شرط

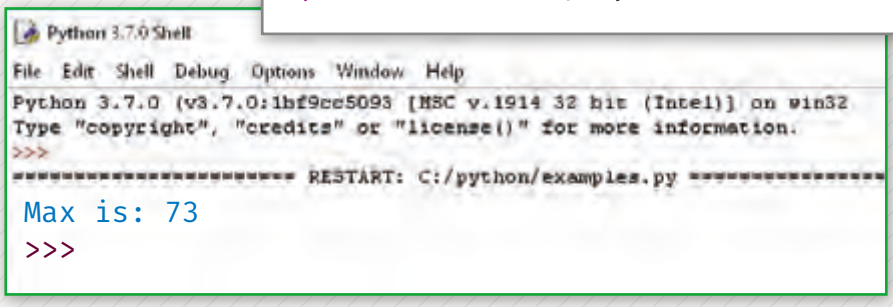
متى نستخدم الاستدعاء الذاتي؟

- في كثير من الحالات تعتبر كطريقة أكثر سهولة عند التعامل مع مشكلة ما.
- بعض تراكيب البيانات من السهل استكشافها باستخدام الاستدعاء الذاتي.
- بعض خوارزميات الفرز مثل الفرز السريع **quick sort** تستخدم الاستدعاء الذاتي.

مثال 4

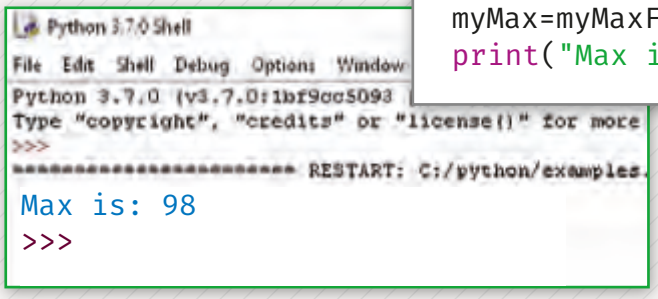
في المثال التالي سنعثر على أكبر رقم من بين قائمة أرقام باستخدام دالة استدعاء ذاتي.

```
def findMax(A,index):

    if index==1:
        m=A[index]
    else:
        m=max(A[index-1],findMax(A,index-1))
    return m

#main program
myList=[3,73,-5,42,7]
l=len(myList)
myMax=findMax(myList,l)
print("Max is:", myMax)
```



في المثال التالي سنعثر على أكبر رقم من بين قائمة أرقام باستخدام التكرار.

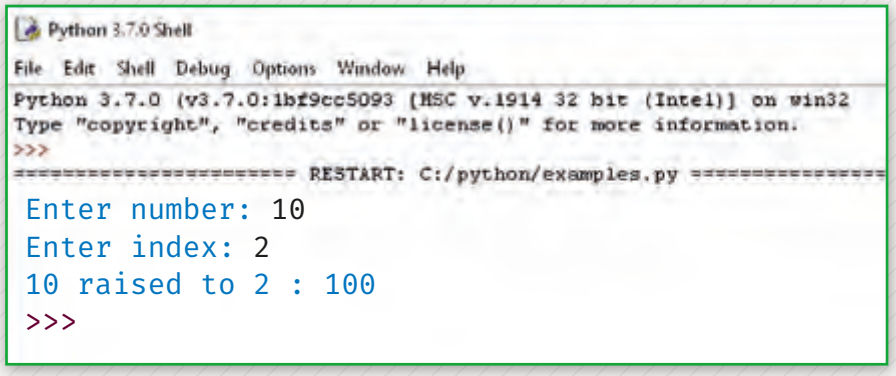
```
def myMaxFun(myList):
    myMax=myList[0]
    l=len(myList)
    for i in range(l):
        if myList[i]>myMax:
            myMax=myList[i]
    return myMax

#program section
myList=[89,88,98,95]
myMax=myMaxFun(myList)
print("Max is:",myMax)
```



مثال 5

في البرنامج التالي، سننشئ دالة استدعاء ذاتي لحساب قوة الرقم. سيقبل البرنامج رقمًا (الأساس) وفهرسًا (القوة) من المستخدم، وسنستخدم دالة الاستدعاء الذاتي powerFun() والتي ستستخدم هذين الوسيطين لحساب قوة رقم الأساس إلى الأس.

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

#main program
base=int(input("Enter number: "))
exp=int(input("Enter index: "))
numPower=powerFun(base,exp)
print( base, "raised to", exp, ":",numPower)
```



لمحة تاريخية

"الاستدعاء الذاتي Recursive" أصلها من الفعل اللاتيني "recurrere" ، والذي يعني "العودة ركضًا".

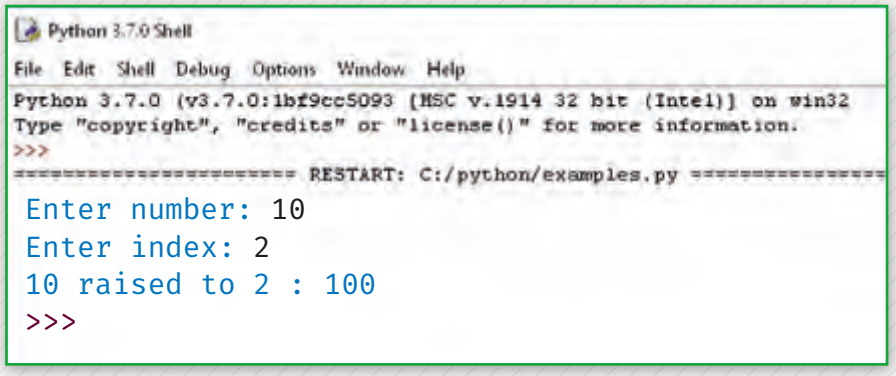
دالة الاستدعاء الذاتي اللانهائي

علينا أن نكون حذرين للغاية عند تنفيذ استدعاء دالة الاستدعاء الذاتي، فيجب أن نوفر طريقة لإيقاف التكرار بإيجاد شرط معين لتفادي حدوث التكرار غير المحدود. أثناء التكرار اللانهائي، يتوقف النظام عن الاستجابة بسبب استدعاءات العديد من الدوال الذي سيؤدي إلى تجاوز سعة الذاكرة وإنهاء التطبيق.

مثال 6

حساب قيمة الأس لرقم ما باستخدام التكرار.

```
base=int(input("Enter number: "))
exp=int(input("Enter index: "))

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

print( base, "raised to", exp, ":",numPower)
```

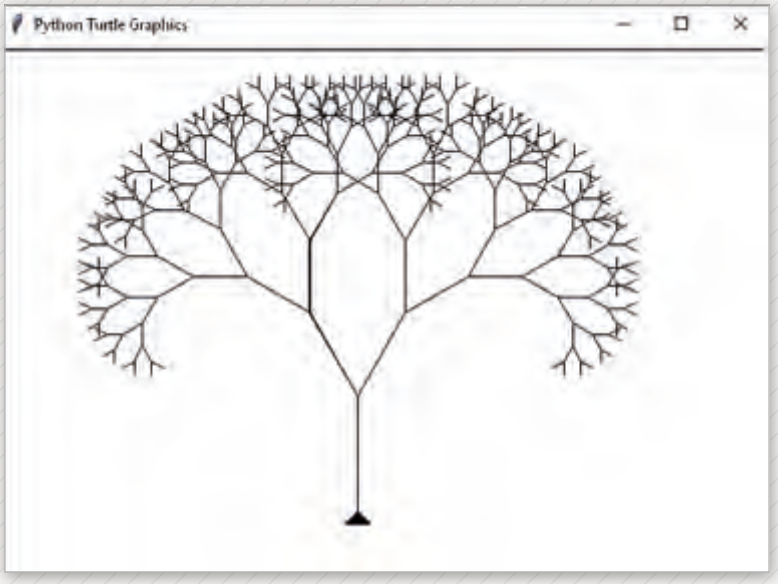


الرسم باستخدام الاستدعاء الذاتي

سنستخدم الاستدعاء الذاتي لرسم شجرة .

مثال 7

```
from turtle import*
pen=Turtle()
pen.shape("arrow")
pen.left(90)
pen.speed(20)

def draw(length):
    if (length<10):
        return
    else:
        pen.forward(length)
        pen.left(30)
        draw(3*length/4)
        pen.right(60)
        draw(3*length/4)
        pen.left(30)
        pen.backward(length)

#main programm
draw (100)
```



1

ضع علامة ✓ أمام العبارة الصحيحة وعلامة ✗ أمام العبارة الخطأ.

.1	تتكون دالة الاستدعاء الذاتي من حالتين .	
.2	تستدعي دالة الاستدعاء الذاتي دالة أخرى.	
.3	دوال الاستدعاء الذاتي أسرع في التنفيذ.	
.4	دوال الاستدعاء الذاتي تجعل المقطع البرمجي أصغر.	
.5	تتطلب كتابة الكود البرمجي الخاص بالتكرار قدرًا أقل من الاستدعاء الذاتي.	
.6	يمكننا استخدام الاستدعاء الذاتي لرسم الأشكال.	

2

ما أوجه الاختلاف بين التكرار والاستدعاء الذاتي؟

_____

_____

_____

_____

3

متى نستخدم الاستدعاء الذاتي؟

_____

_____

_____

4

نفذ يدويا المقطع البرمجي التالي واكتب النتيجة:

```
def myFunction(nums):
    if len(nums) == 0:
        return 0

    last_num = nums.pop()
    return last_num + myFunction(nums)

#main program
nums=[4,7,1,2]
s=myFunction(nums)
print(s)
```

> هل يوفر بايثون دالة جاهزة تمكن من الحصول على نفس النتيجة؟

_____

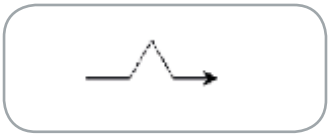
> اكتب اسم الدالة.

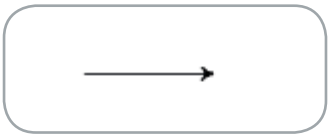
_____

5

أضف الأمر إلى البرنامج الرئيس ثم أوصل الأمر بالرسمة المناسبة له.

```
# Draw a Koch snowflake
from turtle import *

def koch(a, order):
    if order > 0:
        for t in [60, -120, 60, 0]:
            koch(a/3, order-1)
            left(t)
    else:
        forward(a)

# main programm
```





koch(100,0)

koch(100,1)

koch(100,2)

6

أنشئ دالة استدعاء ذاتي لإخراج أدنى رقم من بين مجموعة أرقام في قائمة.

7 اكتب مقطعًا برمجيًا لاحتساب الأسس، وذلك اعتمادًا على ما يلي:

> لاحتساب a^b يمكننا اعتماد الطريقة التالية:

$$a^b = \begin{cases} 1 & if\ b=0, \\ a*a^{b-1} & if\ b>0 \end{cases}$$

> باستخدام تقنية الاستدعاء الذاتي، أكتب الدالة power التي تمكن من احتساب a^b.

8 استنتاج دالة الاستدعاء الذاتي

لنفترض البرنامج الآتي:

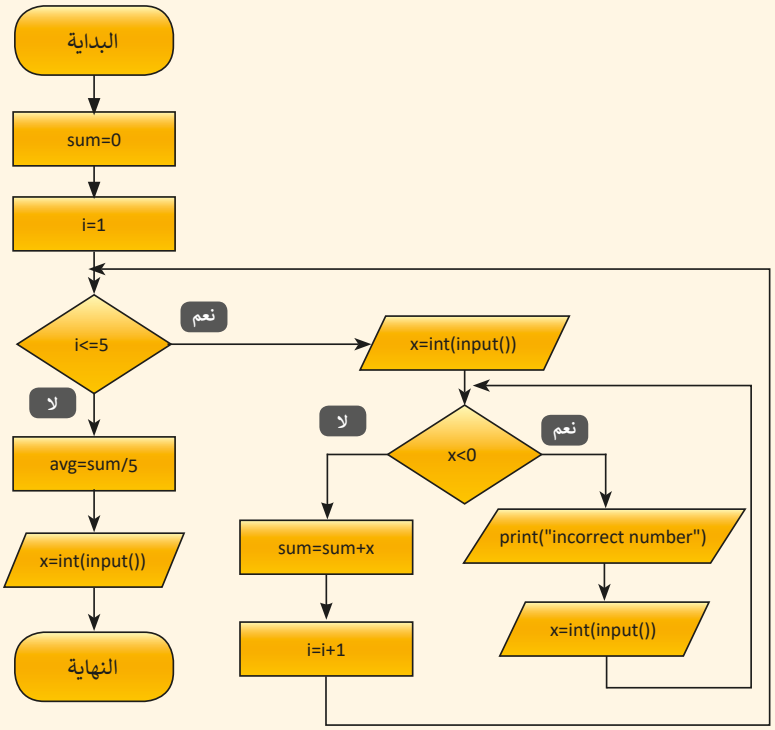
```
L=[3,4,3,4]
def sumList(List,n):
    s=0
    for i in range(n):
        s=s+List[i]
    return s

print sumList(L,4)
```

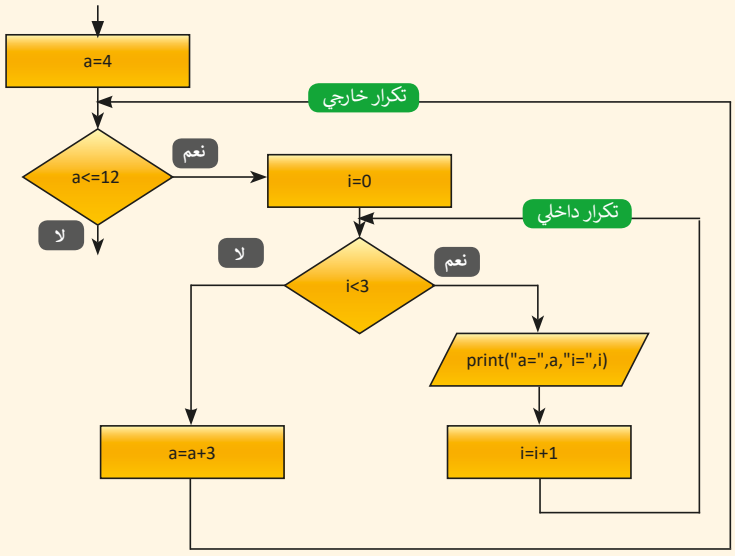
1. ماهي نتيجة تنفيذ البرنامج؟
2. هل يستخدم هذا البرنامج تقنية الاستدعاء الذاتي؟
3. قم بكتابة برنامج يمكن من تنفيذ نفس المهمة باستخدام تقنية الاستدعاء الذاتي:

9

أكتب المقطع البرمجي المناسب لكل مخطط انسيابي.

1.	المقطع البرمجي
2.	المقطع البرمجي





التعامل مع الملفات في Visual Basic

يعرض البرنامج التالي كيفية القراءة والكتابة لملف نصي txt. باستخدام Visual Basic.

```VB
Dim fileReader As String
fileReader = My.Computer.FileSystem.ReadAllText("C:\test.txt")
MsgBox(fileReader)
```

```VB
My.Computer.FileSystem.WriteAllText("C:\TestFolder1\test.txt",
"This is new text to be added.",True)
```

المصفوفة ثنائية الأبعاد 2D Array بلغة C

تُعرَف المصفوفة ثنائية الأبعاد (2D) في لغة برمجة C باسم المصفوفة (matrix).

يوضح هذا البرنامج كيفية تخزين العناصر التي أدخلها المستخدم في مصفوفة ثنائية الأبعاد وكيفية عرض عناصر المصفوفة ثنائية الأبعاد.

```c
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

ماذا تعلمت

العنوان: حساب الدرجات

الوصف: يحتفظ طالب بدرجاته في الرياضيات والعلوم والتاريـخ في ملفات نصية txt.، حيث تحتوي هذه الملفات على ثلاث درجات (اختبار، اختبار نصفي، واختبار نهائي) لكل مادة دراسية.

الأدوات: Python

خطوات التنفيذ:

> أنشئ 3 ملفات txt. ، واحد لكل مادة دراسية.

> قم بقراءة الدرجات من كل ملف وضعها في قائمة، أنشئ قائمة واحدة لكل مادة دراسية.

> قم بدمج القوائم الثلاث لإنشاء مصفوفة ثنائية الأبعاد، بحيث تكون كل مادة في سطر جديد.

> قم بحساب متوسط درجات الطالب.

> قم بحساب عدد مرات تكرار الدرجة 19 في المصفوفة.

> قم بتحديد المادة التي حصل فيها الطالب على أكبر درجة.

ماذا تعلمت

تعلمت في هذه الوحدة:

> فتح ملف بيانات باستخدام مقطع Python برمجي.

> القراءة والكتابة لملف بيانات باستخدام مقطع Python برمجي.

> طبيعة التكرار المتداخل وكيفية ا ستخد ا مه .

> كيفية طباعة أنماط الأرقام باستخدام التكرارات المتداخلة.

> ما هي القوائم المتداخلة.

> كيفية تمثيل المصفوفات ثنائية الأبعاد في Python باستخدام القوائم المتداخلة.

> وظيفة الاستدعاء الذاتي.

> إنشاء دوال تكرار مختلفة.

> معرفة وقت استخدام التكرار.

المصطلحات

الدرس 1	ملفات البيانات Data Files	ملفات نصية Text files	فتح Open
	قراءة Read	كتابة Write	إغلاق Close

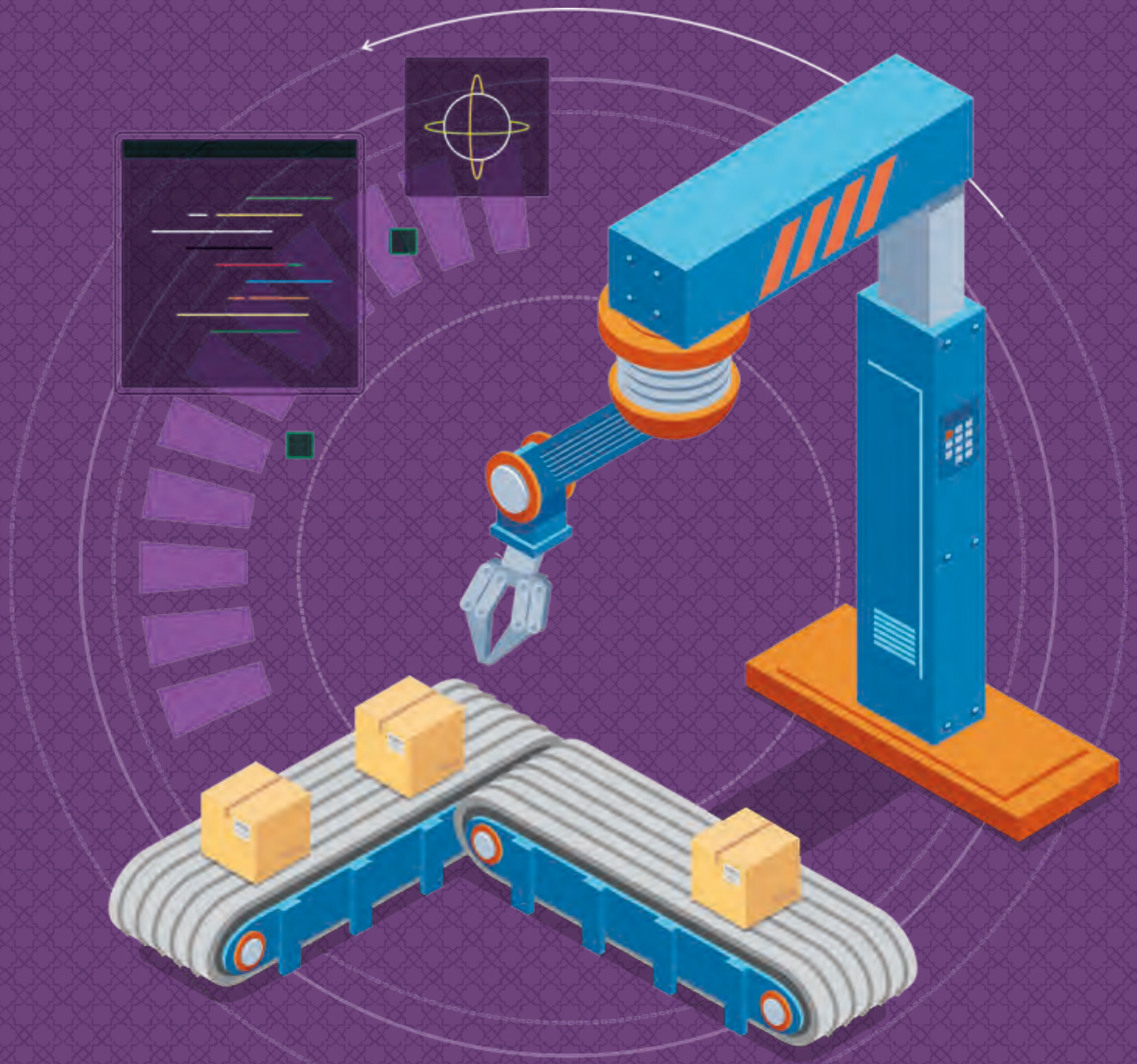
الدرس 2	تكرار Loop	تكرار متداخل Nested loop	نمط Pattern
	تكرار خارجي Outer loop	تكرار داخل Inner Loop	

الدرس 3	قائمة متداخلة Nested list	مصفوفة ثنائية الأبعاد 2D array	صف Row
	عمود Column		

الدرس 4	استدعاء ذاتي Recursion	الدوال التكرارية Recursive functions	حالة الأساس Base case
	حالة تكرارية Recursive case	مضروب Factorial	

3. برمجةالروبوت بـ Python

في هذه الوحدة سيقوم الطلبة ببناء ذراع آلية باستخدام روبوت **LEGO® EV3** والتحكم فيها من خلال لغة البرمجة **MicroPython**. سيحاكي النظام الناتج أجهزة فرز المواد في المصانع لإعادة تدويرها، وسيوظف الطلبة مفاهيم هياكل البيانات في برمجة الذراع الآلية.



ماذا سنتعلمْ؟

في هذه الوحِدة سنتعلم:

> تهيئة روبوت **EV3** لبرمجته في بيئة **MicroPython**.

> إنشاء المقاطع البرمجية بلغة **MicroPython** في **Visual Studio Code** لتحريك الروبوت في مسارات بسيطة .

> تصنيف الأنظمة الروبوتية.

> مجالات استخدام الأذرعة الروبوتية.

> إنشاء نموذج أولي للذراع الروبوتية.

> استخدام **MicroPython** للتحكم ومعايرة أجزاء الذراع الآلية.

> التروس وأثرها في التحكم بالسرعة والعزم.

> استخدام **MicroPython** لتحريك أجزاء الذراع الآلية يدويًا.

> برمجة مستشعر اللون باستخدام **MicroPython**.

> توظيف هياكل البيانات في برمجة الذراع الروبوتية.

مواضيع الوحدة

> برمجة **EV3** في بيئة **Micropython**

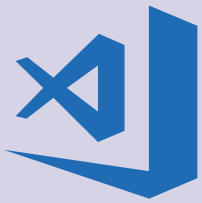
> تركيب وبرمجة قاعدة الروبوت

> ذراع الروبوت

> قابض الروبوت

الأدوات

> Visual Studio Code



هل تذكر؟

لبنة Move Steering يمكنها جعل الروبوت يتحرك إلى للأمام أو للخلف أو الدوران أو التوقف. يمكنك ضبط التوجيه لجعل الروبوت مستقيماً أو الحركة على شكل أقواس أو أخذ المنعطفات الضيقة. لبنة Move Tank تشبه لبنة Move Steering ولكن تختلف طريقة التحكم في الالتفاف لأننا نستطيع التحكم بكل محرك بشكل منفرد.

جهاز استشعار الموجات فوق الصوتية **Ultrasonic sensor**

يعتبر من المستشعرات الرقمية التي تقوم بقياس المسافة بينه وبين أي جسم أمامه من خلال إرسال موجات صوتية ذات تردد عالٍ ومن ثم قياس الزمن الذي استغرقته للرجوع إلى جهاز الاستشعار. يساعد جهاز استشعار الموجات فوق الصوتية الروبوت على تفادي الحواجز والعقبات الأخرى، أو تتبع جسم متحرك أو اكتشاف وجود دخيل في الغرفة، أو حتى إصدار صوت عند اقتراب جسم ما من الروبوت.

تقاس مسافة بُعد الجسم بوحدة السنتميتر، ويمكننا برمجة الروبوت للتوقف على بعد مسافة محددة من أي عائق. لا يمكن لجهاز الاستشعار اكتشاف أي عائق أمامه في مسافة أبعد من **250** سنتميتر. يمكن ضبط جهاز استشعار الموجات فوق الصوتية في وضع **Measure** (القياس) أو وضع **Compare** (المقارنة).

جهاز استشعار الألوان Colour Sensor

جهاز استشعار الألوان يمكنه اكتشاف لون أو شدة أي ضوء يمر من خلال عدساته. ويمكن استخدامه بثلاثة أوضاع مختلفة:

التكرارات Loops

في بعض الأحيان نحتاج إلى تكرار تنفيذ مجموعة أوامر برمجية عدة مرات داخل البرنامج، مما يستغرق وقتاً وجهداً كبيرين. تقدم لنا لغة بايثون مجموعة جمل للتحكم البرمجي تسمى بأوامر التكرار **loop** لمساعدتنا على تجنب إعادة كتابة الأوامر البرمجية مرةً أخرى. تسمح جمل التكرار بتنفيذ جملة برمجية واحدة أو مجموعة جمل برمجية عدة مرات. سوف نتعرف الآن على نوعين من هذه التكرارات وهما تكرار**for** و تكرار **while**.

تدعم لغة برمجة بايثون هذه الأنواع من الجمل التكرارية.	
تكرار **for**	**for loop_variable in range:** **statements**
تكرار **while**	**while condition:** **statements**

الأحداث Events

لنستطيع برمجة حركة الكرة داخل الملعب، نحتاج إلى تعلم مفهوم البرمجة المرتبطة بالأحداث **Events** والتي تعني إدخال المستخدم مجموعة من الأوامر للبرنامج باستخدام الفأرة أو لوحة المفاتيح مثلًا، ومن ثم مقابلة هذه الأحداث بعمليات برمجية محددة. مثلًا عند الضغط بزر الفأرة الأيسر في شاشة البرنامج (عملية إدخال) يتم رسم دائرة من خلال الدالة البرمجية المخصصة لذلك (عملية برمجية).

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** و**المعاملات** و**جسم الدالة**. عندما نريد تعريف دالة، فإننا نستخدم الكلمة الرئيسة "**def**". وعندما نريد استخدامها، فإننا نكتب اسمها متبوعًا بأقواس ونقطتي القول.

برمجة EV3 في بيئة Micropython

تثبيت بطاقة الذاكرة microSD

في المراحل السابقة كنا قد تعلمنا كيفية برمجة **EV3 Brick** من خلال دمج اللبنات البرمجية المختلفة معًا لإنشاء البرامج البسيطة والمتقدمة، كما تعلمنا في السابق كتابة برامج بلغة بايثون لحل بعض المشكلات في مجال الحوسبة وكذلك لرسم الأشكال وتصميم الألعاب وبرمجة الحواسيب المصغرة **Raspberry Pi**. وفي هذه الوحدة سنقوم بدمج هذه المعارف والمهارات لبرمجة **EV3 Brick** بلغة البرمجة بايثون مستخدمين **MicroPython** من **Visual Studio**. سنستخدم بطاقة ذاكرة **microSD** مثبت عليها برنامج **MicroPython**. في البداية، نحتاج إلى اتباع بعض الخطوات لتثبيت وتشغيل **MicroPython** على **EV3 Brick**.

الأدوات اللازمة لاستخدام MicroPython في EV3 Brick	
بطاقة ذاكرة **MicroSD**	سنحتاج إلى بطاقة ذاكرة بسعة لا تقل عن 4 جيجابايت ولا تزيد عن 32 جيجابايت.
وصلة **mini-USB**	يمكننا استخدام الوصلة الموجود لدينا في مجموعة EV3.

تثبيت بطاقة الذاكرة في EV3 Brick:

> أدخل بطاقة **microSD**، ❶ في مدخل **microSD** على **EV3 Brick**. ❷

> قم بتشغيل **EV3 Brick** بضغط زر البدء **Start** الرمادي الداكن في الوسط. ❸

تأكد من إيقاف تشغيل **EV3 Brick** قبل إدخال بطاقة **microSD** في مدخل البطاقة.

يمكننا إضافة لاصق لإزالة البطاقة بسهولة من مدخل **microSD**.

ضع البطاقة في مدخل **microSD** كما هو موضح في الصورة.

أدخل بطاقة الذاكرة **microSD** وقم بتشغيل **EV3 Brick**. قد تستغرق عملية التثبيت عدة دقائق ثم سيتحول ضوء الحالة إلى اللون البرتقالي ويومض بشكل متقطع، أيضًا ستظهر الكثير من النصوص على شاشة **EV3**.

عندما يُصبح **EV3 Brick** جاهزًا بعد الانتهاء من جميع الخطوات المذكورة أعلاه، سينطفئ ضوء الحالة.

النص على شاشة **EV3**

ضوء الحالة البرتقالي

بيئة MicroPython على EV3 Brick

تختلف القائمة الجديدة في **MicroPython** قليلًا عن تلك التي استخدمناها مع برنامج **LEGO® MINDSTORMS®**.

نصيحة ذكية

يمكننا دائمًا الرجوع إلى برنامج LEGO® الأصلي عن طريق إيقاف تشغيل EV3 Brick وإزالة بطاقة الذاكرة وتشغيل EV3 Brick مرة أخرى.

استخدام Visual Studio Code (VSC)

لاستخدام **Visual Studion Code (VSC)** لبرمجة **EV3 Brick** نحتاج إلى تثبيت بعض الإضافات **Extensions** إلى البرنامج، أولها هو **LEGO® MINDSTORMS® EV3 MicroPython**، والإضافة الثانية هي مستعرض أجهزة **EV3** من **VSC**. سنقوم بتثبيت الإضافات المطلوبة إلى **VSC**.

تثبيت الإضافات:

> افتح برنامج **Visual Studio Code**.

> من شريط **Activity**، اضغط علامة تبويب **Extensions** (الإضافات)، ❶ في صندوق البحث اكتب **EV3**. ❷

> اضغط زر **install** لتثبيت الإضافة **LEGO® MINDSTORMS®**. ❸

> اضغط زر **install** لتثبيت الإضافة **ev3dev-browser**. ❹

بعد الانتهاء من تثبيت **LEGO® MINDSTORMS® EV3 MicroPython** ستظهر هذه الأيقونة في شريط **Activity**.

نصيحة ذكية

يتم تثبيت الإضافات مرة واحدة فقط في بداية استخدام البرنامج، وفي حال عدم العثور على كلمة Install فهذا يعني أن الإضافات مثبتة مسبقًا.

ما هو MicroPython؟

كما ذكرنا في بداية هذا الدرس، سنستخدم **MicroPython** لبرمجة **EV3 Brick**، ولكن ما المقصود بـ **MicroPython**؟

MicroPython هو إصدار مصغر من لغة البرمجة **Python 3** وتضم مجموعة فرعية صغيرة من مكتبة **Python** القياسية، والتي تم تطويرها للعمل مع المتحكمات الدقيقة والأنظمة محدودة الموارد (الذاكرة، وسرعة المعالجة، ...).

ميزات MicroPython الفريدة	
الاتصال المباشر مع EV3 Brick	تمكن من الاتصال بـ EV3 Brick وتنفيذ المقاطع البرمجية دون الحاجة إلى تجميع أو تحميل.
مكتبة برامج واسعة النطاق	تتوفر في البرنامج العديد من المكتبات التي تدعم العديد من المهام.
قابلية العمل مع اللغات الأخرى	MicroPython يمكن أن تستدعي دوال C / C++ منخفضة المستوى بحيث يُمكن دمج مقطع MicroPython عالي المستوى مع المقاطع البرمجية منخفضة المستوى عند الحاجة إليها.

إنشاء برنامجنا الأول

خصص مجلدًا على حاسوبك لحفظ المشاريع التي ستقوم بإنشائها.

هيا بنا ننشئ برنامجنا الأول في **MicroPython**.

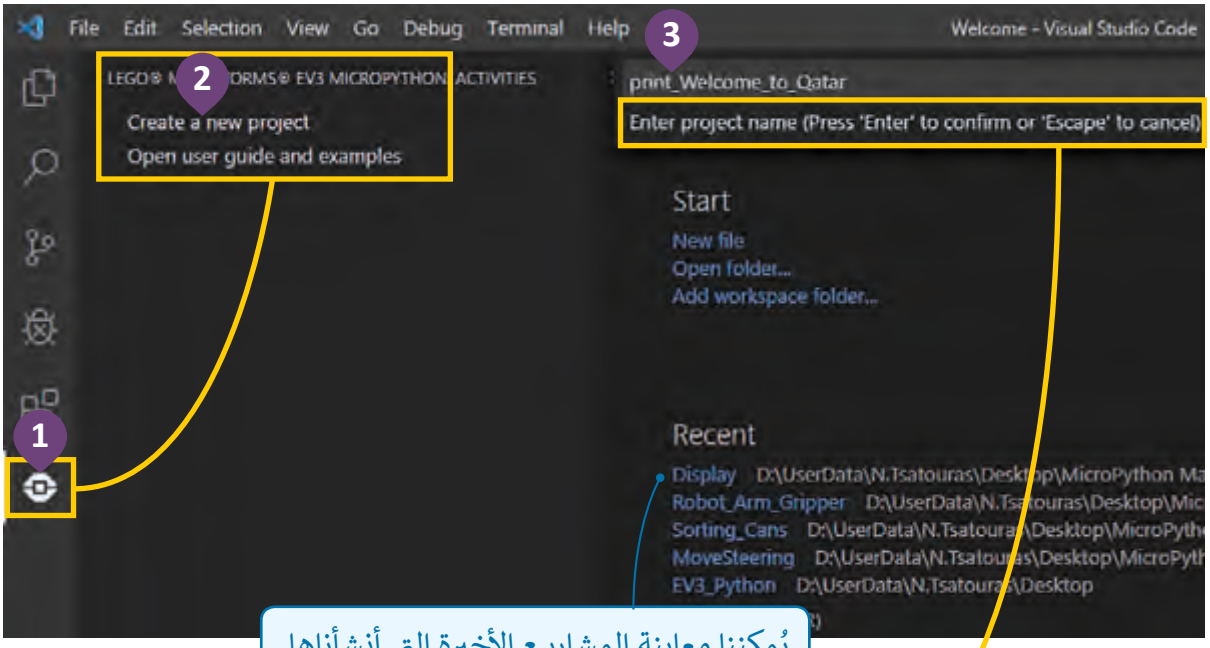
إنشاء مشروع جديد:

> من شريط الأنشطة، اضغط **LEGO®MINDSTORMS® EV3 MicroPython**. ❶

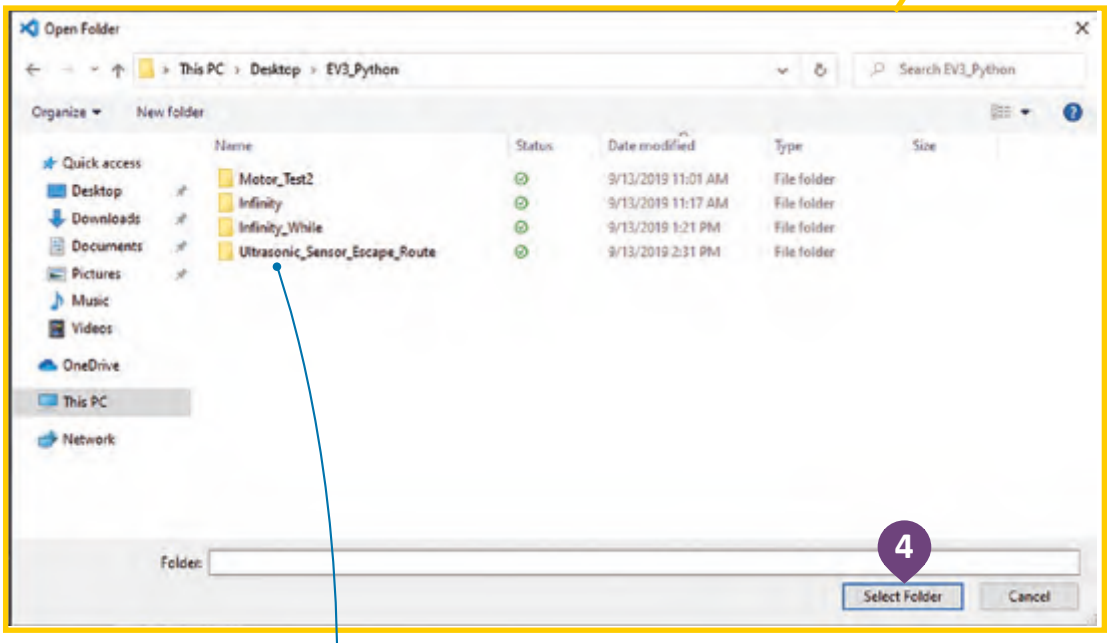
> من الشريط الجانبي، اضغط **Create a new Project** "إنشاء مشروع جديد". ❷

> قم بتسمية المشروع " **print _Welcome _to_Qatar**" واضغط "**Enter**". ❸

> اضغط **Select Folder** (اختر مجلد) لحفظ الملف في المجلد الذي قمت بإنشائه. ❹



يُمكننا معاينة المشاريع الأخيرة التي أنشأناها



مجلد المشروع هو مجلد على الحاسوب يحتوي على البرنامج الرئيس **'main.py'** وبعض الملفات الأخرى.

سيتم نسخ مجلد المشروع وجميع محتوياته إلى **EV3 Brick**، حيث سيتم تشغيل البرنامج الرئيس.

نصيحة ذكية

يُمكن MicroPython من التحكم في الأجهزة وتوصيلاتها، كما يُمكننا من التحكم في منافذ الإدخال والإخراج GPIO لوميض الأضواء وقراءة المفاتيح وغيرها.

بمجرد إنشاء المشروع يُمكننا إلقاء نظرة على المكتبات الافتراضية التي يستخدمها **MicroPython**.

معاينة المكتبات الافتراضية المستخدمة في البرنامج:

> من شريط الأنشطة اضغط علامة تبويب **Explorer** (المستكشف). ❶

> من الشريط الجانبي اضغط **main.py**. ❷

نصيحة ذكية

يتم إنشاء هذه الأسطر البرمجية تلقائيًا في كل مرة يتم فيها إنشاء برنامج جديد.

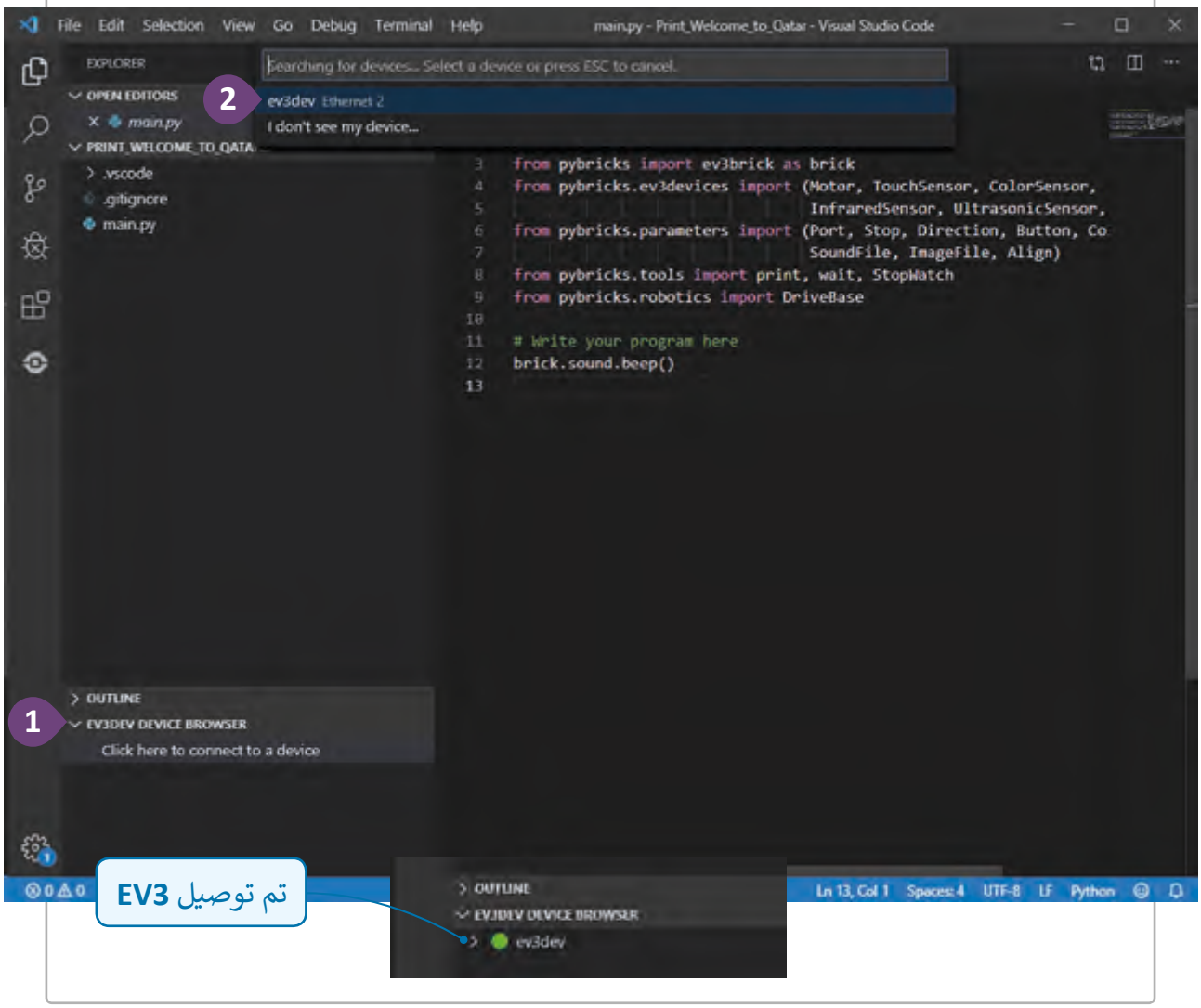
توصيل وتشغيل البرنامج

كما ذكرنا سابقًا فإن **MicroPython** يقوم بتهيئة البيئة للمبرمج حيث يقوم تلقائيا باستدعاء بعض المكتبات القياسية وتحميل بعض الدوال التي يمكن للمبرمج استخدامها، فما علينا إلا مواصلة كتابة برنامجنا ثم توصيل جهاز **EV3 Brick** بالحاسوب لتنزيل البرنامج عليه وتجربته. لا تنسَ أنك يجب أن تقوم بتوصيل **EV3 Brick** بمنفذ **USB** في الحاسوب قبل تنفيذ هذه الخطوة.

الاتصال مع جهاز EV3:

> اضغط فوق القائمة **EV3DEV DEVICE BROWSER** ، ثم اختر **Click here to connect to a device**. ❶

> من القائمة المنسدلة اختر **ev3dev**. ❷



نصيحة ذكية

في حال عدم ظهور ev3dev تأكد من تركيب وصلة USB بين الحاسوب والـ EV3 Brick.

تنفيذ وتجربة البرنامج

الآن، اضغط **F5** من لوحة المفاتيح لتشغيل البرنامج، ستسمع صوتًا لمدة ثانية.

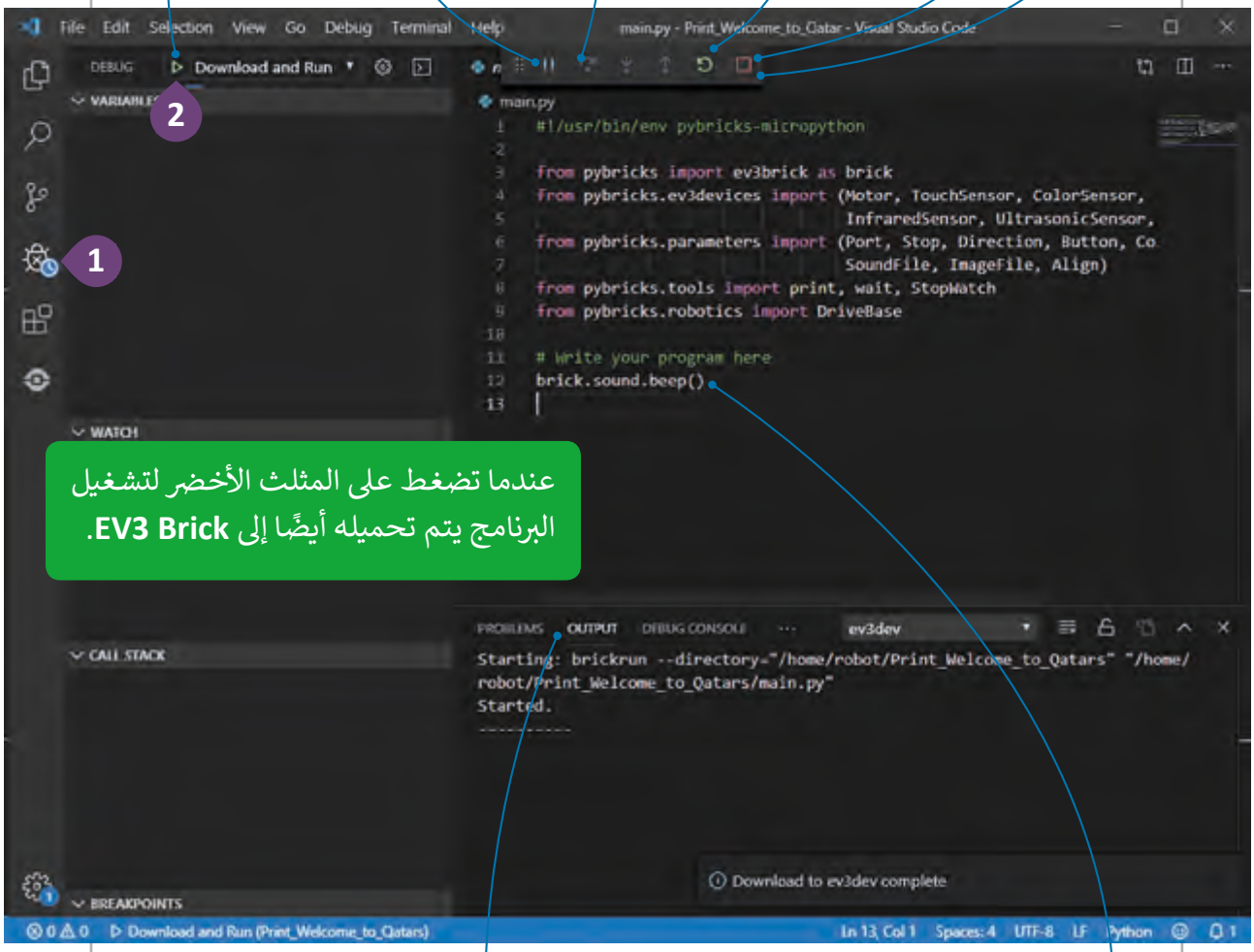
تشخيص الأخطاء وتنفيذ البرنامج:

> من شريط الأنشطة اضغط علامة تبويب تشخيص الأخطاء **Debug**. ❶

> من الشريط الجانبي اضغط على المثلث الأخضر لتشغيل البرنامج. ❷

عندما نضغط زر F5 تظهر قائمة التحكم

زر بدء التشخيص | إيقاف مؤقت | تخطي | إعادة تشغيل | إيقاف



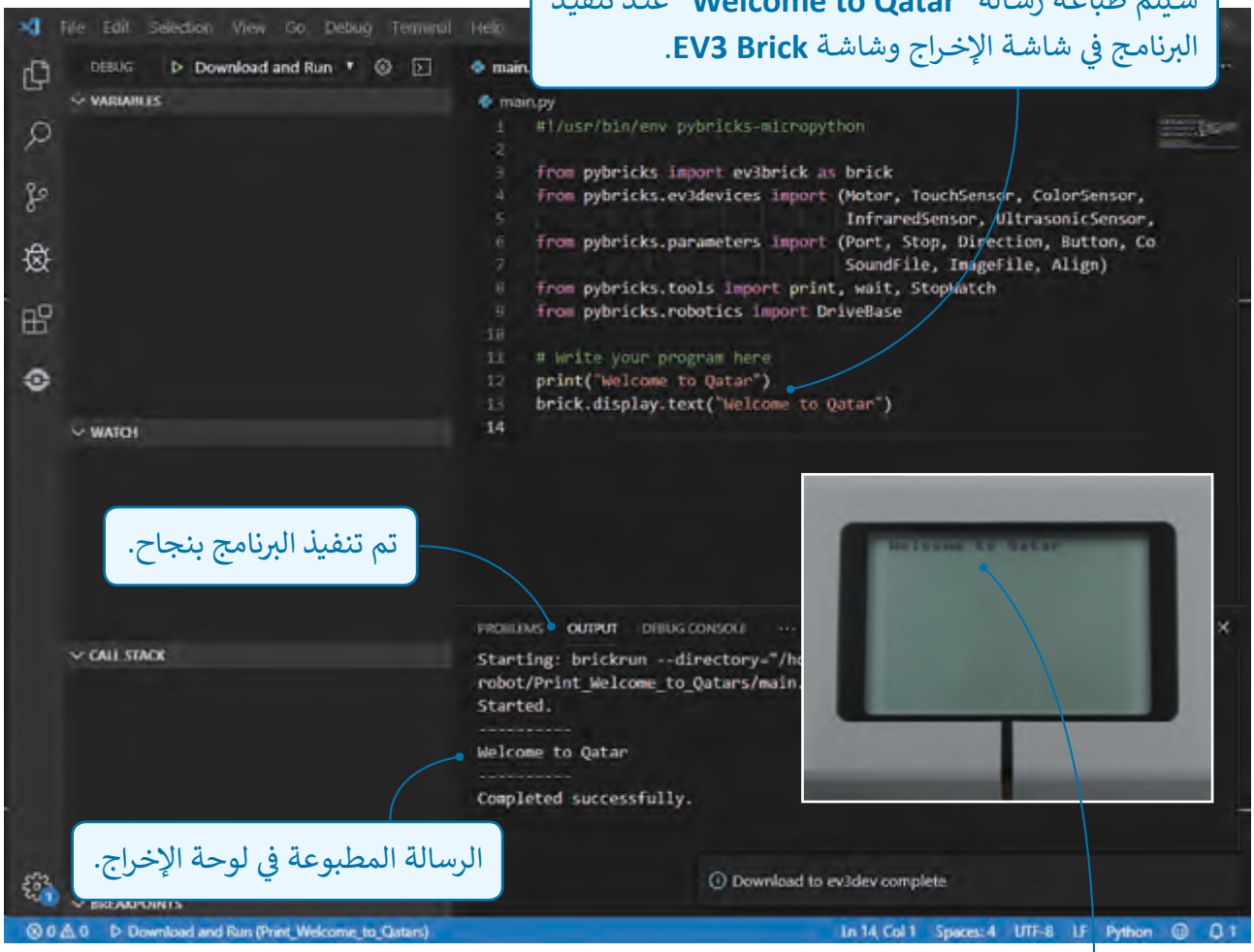
عندما تضغط على المثلث الأخضر لتشغيل البرنامج يتم تحميله أيضًا إلى **EV3 Brick**.

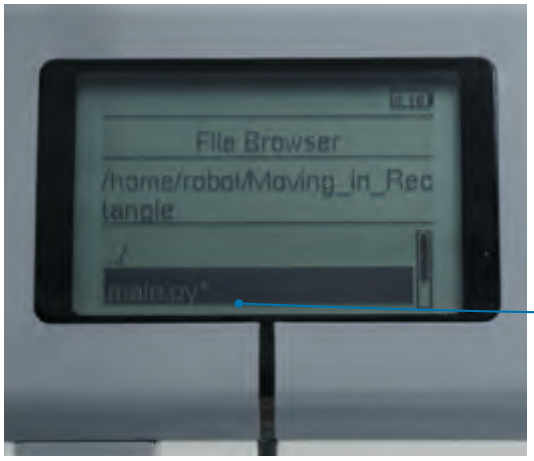
تعرض لوحة **OUTPUT** (الإخراج) رسائل الحالة الخاصة بالبرنامج، مثل: الأخطاء اللغوية، الأخطاء البرمجية، ومعلومات تكوين البرنامج والتي قد تحدث جميعها بمجرد بدء تشغيل المقطع البرمجي.

هذا السطر البرمجي يُنتج صوت تنبيه.

طباعة الرسالة

لنعدل البرنامج السابق بحيث يقوم بطباعة الرسالة "**Welcome to Qatar**".





إذا رغبنا في تشغيل البرنامج مباشرة من **EV3 Brick**، فإننا بحاجة إلى العثور على الملف في مستعرض الملفات، بعد ذلك الضغط على زر الوسط لتنفيذ **main.py**.

نصيحة ذكية

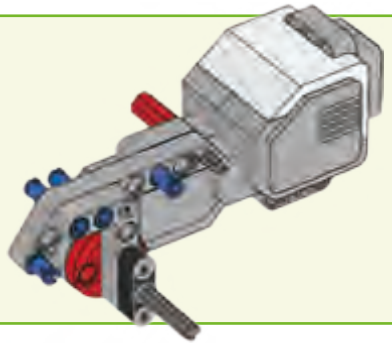
يمكن تشغيل البرنامج بالضغط على مفتاح F5 من لوحة المفاتيح.

تجميع قطع الروبوت

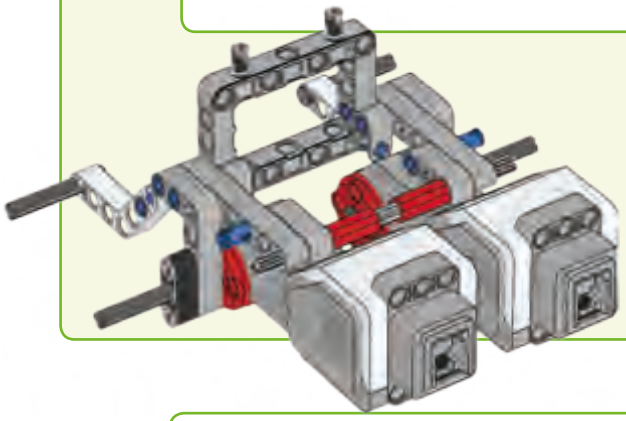
سنقوم الآن بإنشاء مشاريع مختلفة تنفذ بعض المهام التي كنا قد قمنا بها بواسطة **LEGO® MINDSTORMS® EV3** في المراحل الدراسية السابقة. يتعين علينا أولاً إنشاء قاعدة المحرك وكتابة بعض برامج **MicroPython** بحيث يمكن جعل الروبوت يتحرك حسب المهمة المطلوبة.

بناء قاعدة القيادة (Drive Base).

1 قم بتركيب قاعدة تثبيت المحرك.

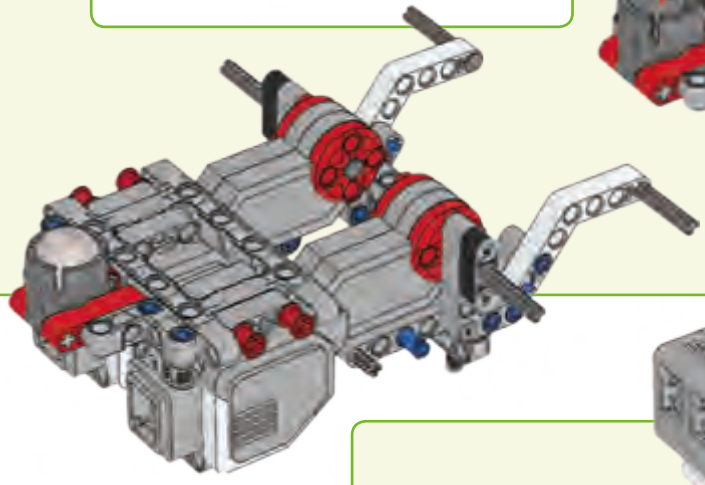


2 ضع المحركات جنبًا إلى جنب.

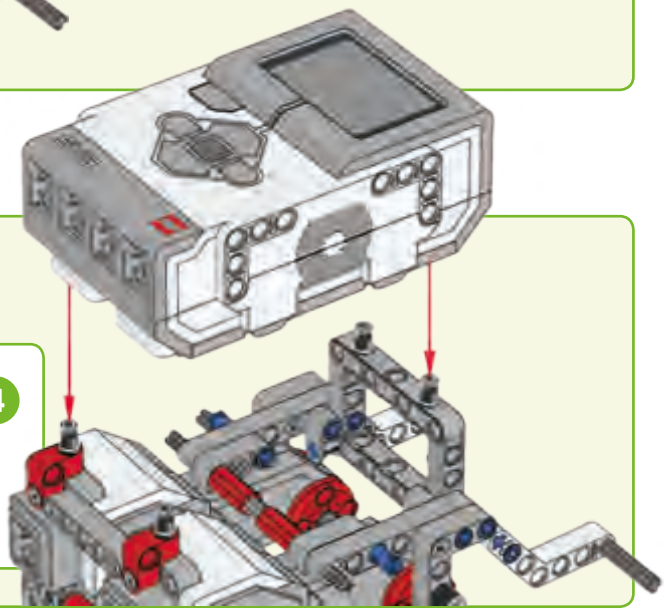


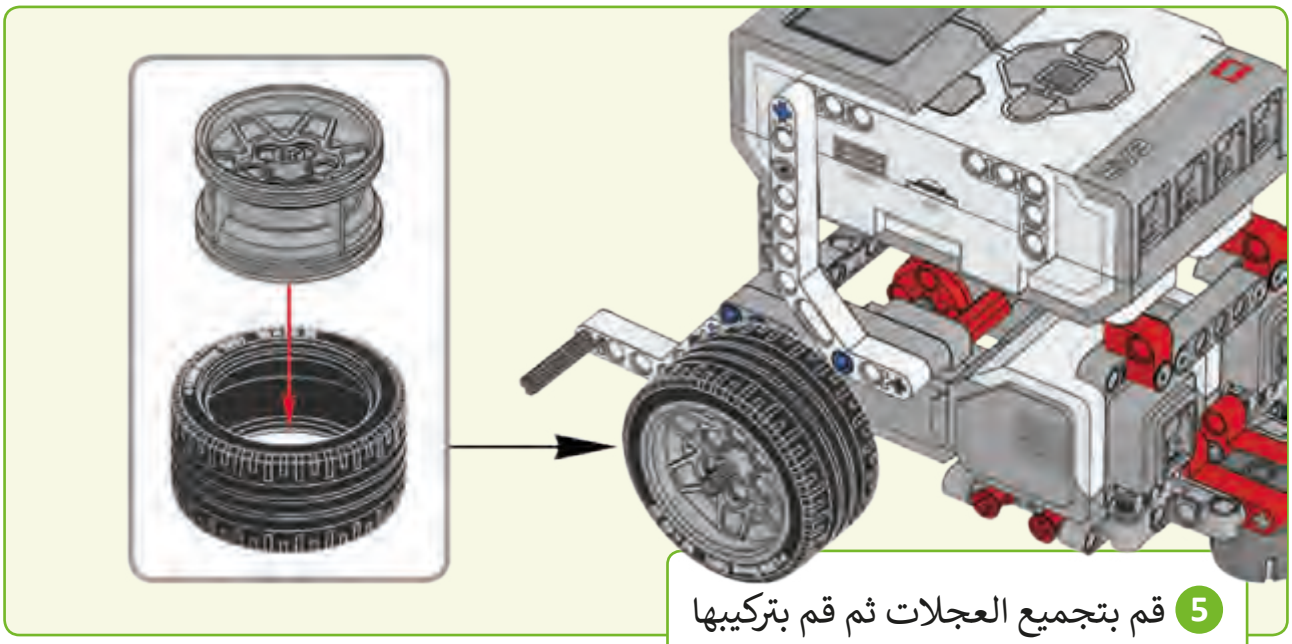
3 قم بتركيب هذا الجزء أسفل جهاز الروبوت حتى يتحرك الروبوت بسلاسة باستخدام الكرة.

ليبدو الجزء الأسفل من الروبوت بهذا الشكل.

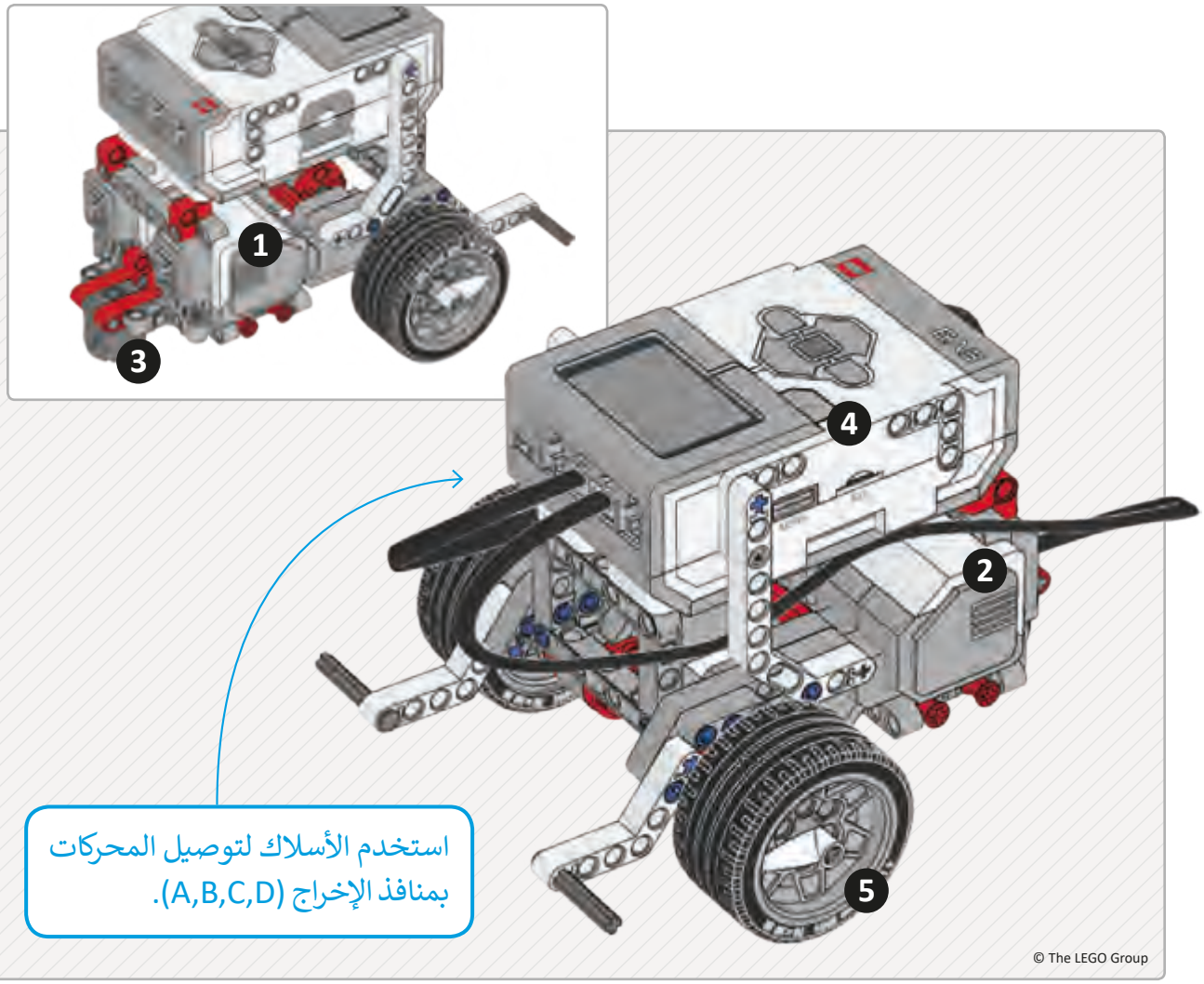


4 بعد أن قمت بإنشاء قاعدة الروبوت، حان الوقت لوضع وحدة التحكم على القاعدة.





5 قم بتجميع العجلات ثم قم بتركيبها بقاعدة الروبوت.



استخدم الأسلاك لتوصيل المحركات بمنافذ الإخراج (A,B,C,D).

التحرك في مسار مستطيل

هيا بنا نتعرف على أوجه التشابه والاختلاف بين **Mindstorms** و **MicroPython**. رأينا سابقًا أنه عند وضع لبنة توجيه الحركة في **LEGO® MINDSTORMS®** فإننا نحتاج إلى إعداد منافذ المحرك، ولكننا في **MicroPython** نحتاج إلى استيراد المكتبات المناسبة أولاً ثم إعداد منافذ المحرك.

الصيغة العامة لاستيراد وحدة التحكم في المحرك

يستورد العملية **MoveSteering**

SpeedPercent للتحكم في سرعة المحرك

يستورد المنفذ **OUTPUT_C**

يستورد المنفذ **OUTPUT_B**

```
from ev3dev2.motor import OUTPUT_B, OUTPUT_C, SpeedPercent, MoveSteering
```

صيغة ضبط إعدادات المحرك

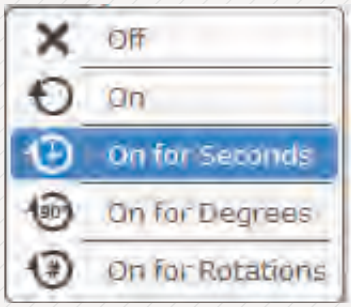
يُمكننا معاينة أوجه التشابه بين هاتين البيئتين في الصورة أدناه.

steering_drive هو الاسم الذي اخترناه لعملية **MoveSteering** لاستخدامه مع معاملاته.

```
steering_drive = MoveSteering(OUTPUT_B, OUTPUT_C)
```

إعداد اثنين من مخرجات المحركات وحفظهما في عملية التوجيه **MoveSteering**.

دعنا نلقي نظرة على القائمة المنسدلة للبنة التوجيه **MoveSteering**. الجدول التالي يعرض الخيارات الموجودة في لبنات التحكم ويمكن استعمالها مع **MicroPython** كمعاملات.



معاملات MoveSteering	
المعاملات	القيم
on_for_degrees	(steering, speed, rotations, brake=True, block=True)
on_for_rotations	(steering, speed, degrees, brake=True, block=True)
on_for_seconds	(steering, speed, seconds, brake=True, block=True)
on	(steering, speed)
off	(brake=True)

القيم التي يمكن أن تتخذها **Move Steering Block** هي [**100**, **-100**]. وكما نعلم بالفعل فعند تغيير هذه القيم يمكن أن يتحرك الروبوت في اتجاهات مختلفة.

قيم التوجيه من 100- إلى 100	
-100	أقصى قيمة سالبة توجه الروبوت إلى اليمين من الموقع الحالي.
0	قيمة صفر تدفع الروبوت في اتجاه مستقيم.
100	أقصى قيمة موجبة توجه الروبوت إلى اليسار من الموقع الحالي.

نصيحة ذكية

عند ذكر خصائص الحركة مع اللبنات تسمى خيارات وعند ذكرها أو استخدامها في البرمجة بـ MicroPython تسمى معاملات.

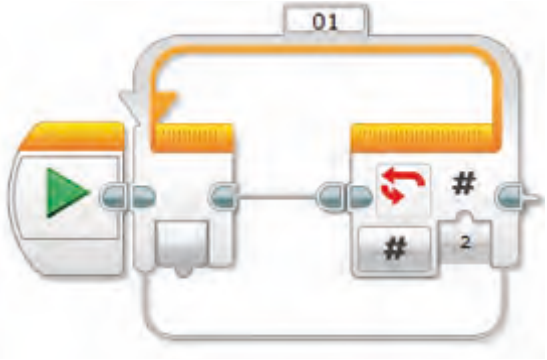
في السطر البرمجي أدناه، يمكننا اختيار المُعامل **on for seconds**.


```
steering_drive.on_for_seconds(0, SpeedPercent(40), 3)
```

- **Separate the** تفصل العملية عن معاملاتها بواسطة '.'
- التوجيه **Steering**
- السرعة **Speed**
- الوقت **Time**

لتجنب الحاجة الى تكرار العمليات البرمجية في **LEGO® MINDSTORMS®** نستخدم لبنة التكرار، وفي **MicroPython** نستخدم التكرارات البرمجية أيضًا. لقد تعرفنا سابقًا على كيفية استخدام تكرار**for** في **Python**.

```
for x in range(2):
```



نصيحة ذكية

يتم فصل المعاملات عن بعضها داخل الأقواس بواسطة الفاصلة. على سبيل المثال (steering, speed, seconds) (التوجيه ، والسرعة ، والثواني).

لننشئ برنامج بواسطة الأمر **MoveSteering** حيث سيتحرك الروبوت في مسار مستطيل.

مثال 1

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase

from ev3dev2.motor import OUTPUT_B, OUTPUT_C, SpeedPercent,
MoveSteering

# Write your program here
steering_drive = MoveSteering(OUTPUT_B, OUTPUT_C)

# ================== main program ==================
for x in range(2):
    steering_drive.on_for_seconds(0, SpeedPercent(40), 3)
    steering_drive.on_for_seconds(40, SpeedPercent(30), 3)
    steering_drive.on_for_seconds(0, SpeedPercent(40), 5)
    steering_drive.on_for_seconds(40, SpeedPercent(30), 3)
# ================== main program ==================
```

استخدم أسطر التعليقات البرمجية العريضة لفصل الأجزاء المختلفة للبرنامج.

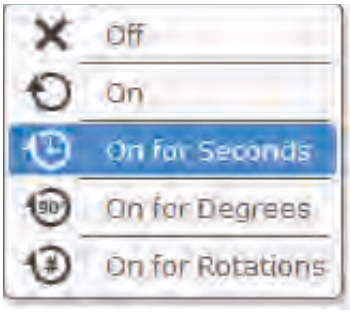
في **Mindstorms** نستخدم اللبنات البرمجية الموجودة في الصورة أدناه.

التحرك في مسار دائري

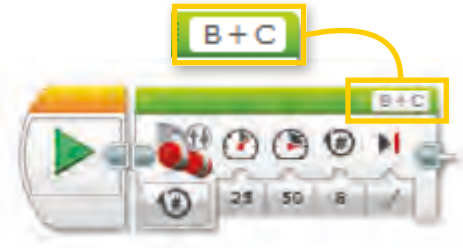
سنقوم الآن بإنشاء برنامج جديد، لكن هذه المرة سنستخدم العملية **MoveTank** والتي تُعتبر مماثلة للبنة **Move Tank** في **LEGO® MINDSTORMS®**.

في **LEGO® MINDSTORMS®** تتكرر نفس المعاملات التي تظهر عند فتح القائمة المنسدلة كما في لبنة **Move Steering**، ولكن هنا يوجد لدينا قيم مختلفة بسبب وجود خيار المحركين، وينطبق الأمر ذاته على **MicroPython**.

دعنا نلقي نظرة على الجدول حيث توجد المعاملات والقيم التي يمكننا استخدامها في **MicroPython**.



معاملات لبنة Move Tank	
المعاملات	القيم
on_for_degrees	(left_speed, right_speed, degrees, brake=True, block=True)
on_for_rotations	(left_speed, right_speed, rotations, brake=True, block=True)
on_for_seconds	(left_speed, right_speed, seconds, brake=True, block=True)
on	(left_speed, right_speed)
off	(brake=True)




```
move_Tank = MoveTank (OUTPUT_B, OUTPUT_C)
```

على سبيل المثال، إذا استخدمنا الأمر **on for rotations** وقمنا بضبط سرعة المحرك الأيسر بـ **25%**، وسرعة المحرك الأيمن بـ **50%** وضبطنا عدد الدورات إلى **8**، فإن السطر البرمجي سيصبح كما في الصورة أدناه.

```
move_Tank.on_for_rotations(SpeedPercent(25), SpeedPercent(50), 8)
```

لننشئ برنامجًا بعملية **MoveTank**، في المثال التالي سيتم تحريك الروبوت في مسار دائري.

مثال 2

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor,
ColorSensor,InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch

from pybricks.robotics import DriveBase
from ev3dev2.motor import OUTPUT_B, OUTPUT_C, SpeedPercent, MoveTank

# Write your program here

# Set Motors
move_Tank = MoveTank (OUTPUT_B, OUTPUT_C)

# ================== main program ==================
move_Tank.on_for_rotations(SpeedPercent(25), SpeedPercent(50), 8)
# ================== main program ==================
```

في **Mindstorms** سنستخدم اللبنات البرمجية الموجودة في الصورة أدناه.

إضافة مستشعر الموجات فوق الصوتية Ultrasonic Sensor

سنقوم بإضافة مستشعر الموجات فوق الصوتية **Ultrasonic** وتركيبه مع قاعدة القيادة.

إضافة مستشعر الموجات فوق الصوتية لقاعدة القيادة.

عملية DriveBase

تحتوي مكتبة **pybricks.robotics** على عملية تسمى **DriveBase** والتي تمنح القدرة على قيادة كلا المحركين في نفس الوقت، وكما أشرنا مسبقًا فهي مكتبة افتراضية تستخدمها **MicroPython**.

```
from pybricks.robotics import DriveBase
```

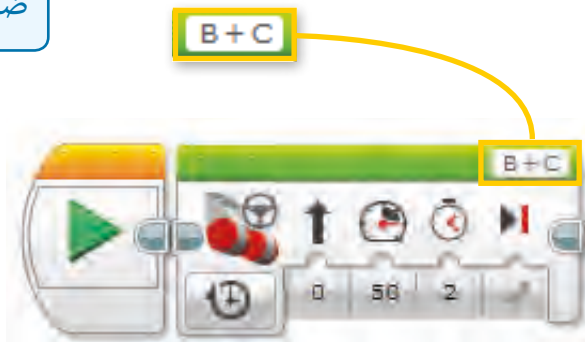
يمكن أن تُستخدم عملية **DriveBase** بشكل مشابه لاستخدام لبنة **Move Steering** ، حيث أن لدينا معاملات المحركين الأيمن والأيسر (**left_motor, right_motor**) المُخزنة على **DriveBase**، والتي تُمكننا من ضبط قطر العجلة ومسار المحور (**wheel_diameter, axle_track**). وهذه تعطي للمبرمج القدرة على التحكم وتحديد المسافة الفعلية التي يقطعها الروبوت، بالاضافة إلى زاوية الدوران، اعتمادًا على معرفة محيط العجلة والمسافة بين العجلتين.

```
DriveBase(left_motor, right_motor, wheel_diameter, axle_track)
```

قيم DriveBase	
left_motor(Motor)	المحرك الذي يقود العجلة اليمنى
right_motor(Motor)	المحرك الذي يقود العجلة اليسرى
wheel_diameter(dimension: mm)	قُطر العجلات
axle_track(dimension: mm)	المسافة بين مركزي العجلتين (طول المحور)

ضبط المحركين للمنفذين **B** و **C**.

```
left_motor = Motor(Port.B)
right_motor = Motor(Port.C)
```



تُمثل عملية **Drivebase** مركبة الروبوت مع عجلتين مزودتين بالطاقة.

قُطر العجلة ومسار المحور الخاص بعجلات **EV3**.

```
robot = DriveBase(left_motor, right_motor, 56, 114)
```

في سطر التعليمات البرمجية أدناه، يتحقق المستشعر فوق الصوتي من المسافة (**distance**).

```
ultra_sensor.distance()
```

يتم قياس المسافة من قِبل المستشعر بوحدة السنتميتر.

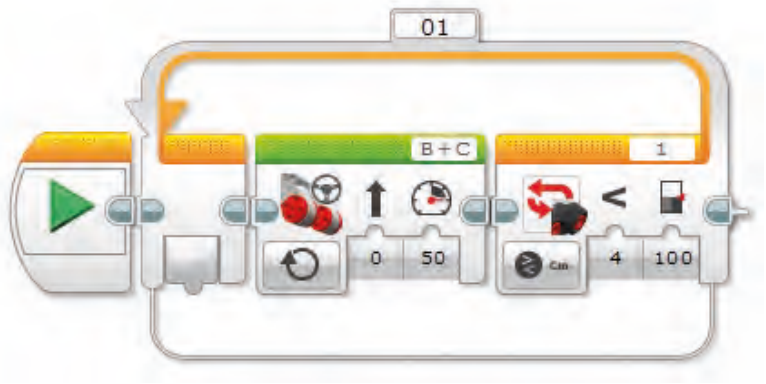
يمكننا تحريك الروبوت باستخدام عملية (**drive**).

```
robot.drive(speed, steering)
```

دعنا ننشئ برنامجًا لتحريك الروبوت للأمام بسرعة **150** حتى يكتشف مستشعر الموجات فوق الصوتية وجود عائق على مسافة أقل من **100** سم.

مثال 3

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase

# Write your program here

# Set Motors
left_motor = Motor(Port.B)
right_motor = Motor(Port.C)

# Set the default size of the wheels
robot = DriveBase(left_motor, right_motor, 56, 114)

# Set Ultrasonic sensor port
ultra_sensor = UltrasonicSensor(Port.S1)

# ================== main program ==================
while ultra_sensor.distance() > 100:
    robot.drive(150, 0)
# ================== main program ==================
```



في **Mindstorms** كنا سنستخدم اللبنة البرمجية أدناه.

1

اذكر ثلاث ميزات فريدة في **MicroPython** واشرح فائدة كل واحدة منها بشكل مفصل.

2

استخدم **MicroPython** لبرمجة الروبوت لكي يتحرك في مسار:

> مربع

> مثلث

> استخدم تكرار **for** لتفادي الحاجة إلى تكرار الأوامر.

3

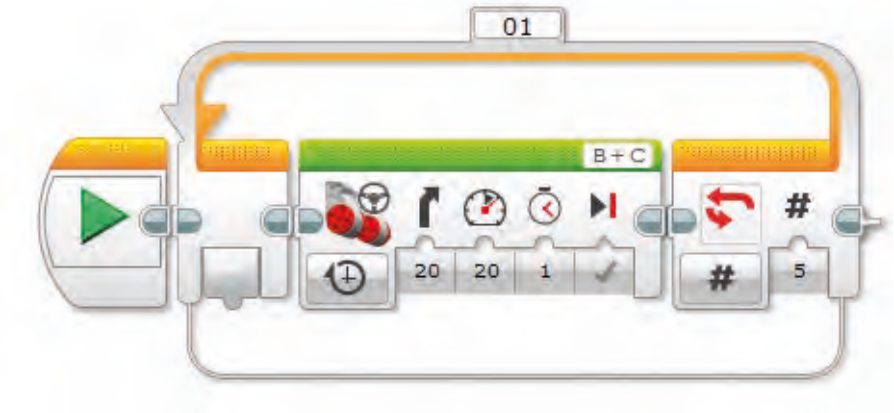
استخدم عملية **MoveTank** بحيث يستطيع الروبوت التقدم بسرعة **50%** بكلا المحركين حتى يعثر على عائق على مسافة أقرب من **50** سم، وعندها سيتوقف الروبوت ويصدر صوتًا ويعرض الرسالة **"Obstacle ahead"**.

4

مستخدمًا لغة **MicroPython**، اكتب الأسطر البرمجية التي تمثل وظيفة اللبنات أدناه:

_____

_____

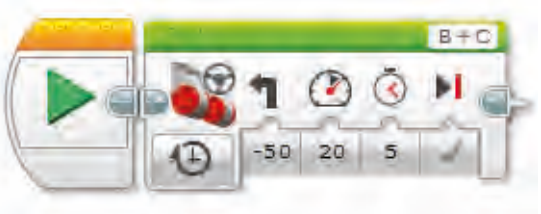


_____

_____

_____

_____



_____

_____

الدرس الثاني

تركيب وبرمجة قاعدة الروبوت

تصنيفات الأنظمة الروبوتية

يمكن تقسيم الأنظمة الروبوتية إلى قسمين: ثابتة **(Fixed)** ومتحركة **(Moving)**.

الروبوتات الثابتة

الأنظمة الثابتة هي أنظمة روبوتية لها قاعدة ثابتة وتقوم بكافة الوظائف المنوطة بها في البيئة المحيطة، مثل روبوتات مصانع تجميع السيارات.

الروبوتات المتحركة

الأنظمة المتحركة فهي أنظمة روبوتية قادرة على الحركة والتجوال لإنجاز المهام في البيئة المحيطة، وتكون عادة مزودة بأجزاء تساعدها على الحركة كالعجلات أو الأطراف والأرجل الصناعية، ومن تطبيقاتها روبوتات استكشاف الفضاء وإجراء الأبحاث العلمية.

روبوت متحرك

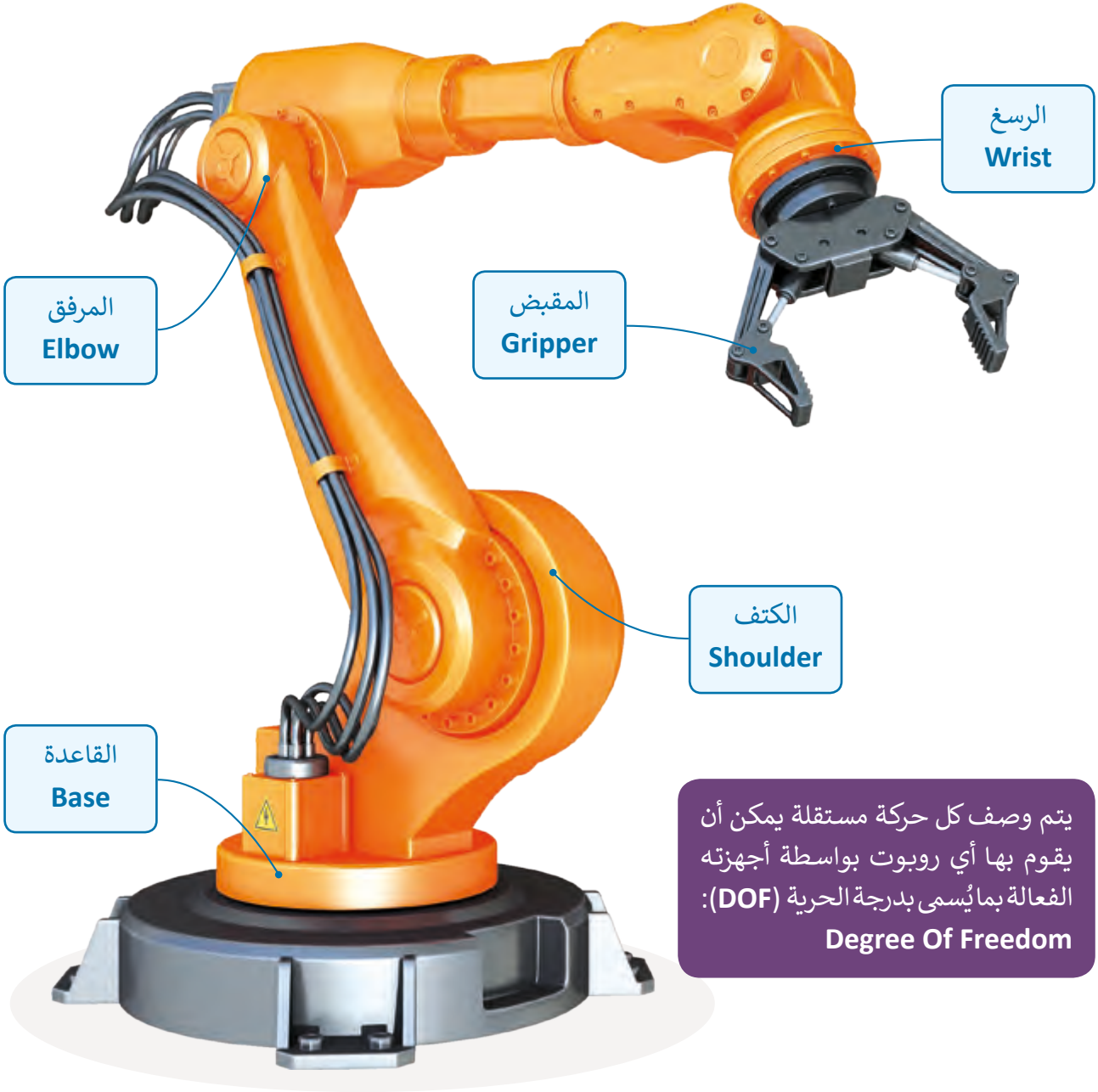
روبوت ثابت

تطبيقات استخدام الذراع الروبوتية

قبل أن نبدأ العمل سنتعرف قليلًا على الأذرع الروبوتية، ما المقصود بها وما هي تصنيفات استخدامها الحالية والمستقبلية؟

إن الذراع الروبوتية هي نوع من الأذرع الميكانيكية التي تكون عادةً قابلة للبرمجة للقيام بوظائف مماثلة للذراع البشري. قد تكون الذراع آلية متكاملة، أو قد تكون جزءًا من روبوت أكثر تعقيدًا، وترتبط وصلات الذراع بواسطة ما يُسمى المفاصل **(joints)** والتي تتيح الحركة الدورانية لأجزاء الذراع المختلفة.

تُظهر الصورة أدناه ذراعًا آليًا بخمسة مفاصل. سنتناول موضوع المفاصل وحركتها لاحقًا في الدرس الرابع عند الانتهاء من إنشاء ذراع الروبوت.



يتم وصف كل حركة مستقلة يمكن أن يقوم بها أي روبوت بواسطة أجهزته الفعالة بما يُسمى بدرجة الحرية **(DOF)**:
Degree Of Freedom

في الصناعة

يشيع استخدام أذرع الروبوت الآلية في صناعة السيارات، كما يستخدم الذراع الروبوتي في العديد من التطبيقات الصناعية مثل اللحام ومناولة المواد والطلاء الحراري والدهان والحفر. بشكل عام، تعمل معظم الروبوتات الصناعية في خطوط التجميع كإنتاج السيارات، وكما نجد أن هناك مزايا و استخدامات ضخمة ومتنوعة للأذرع الآلية في الصناعة.

مزايا استخدام الذراع الروبوتي في الصناعة
تعمل بكفاءة أكبر من البشر من حيث السرعة و الدقة.
تقلل من إصابات العمال، وبشكل خاص تلك الإصابات الناتجة عن الإجهاد وحوادث العمل التي قد تسبب أضرارًا كبيرة.
إخراج منتج ثابت في جودته بتكلفة رخيصة.

تسمح الروبوتات المستخدمة في صناعة السيارات بإنتاج أكثر دقة حيث أن الأذرع الروبوتية تثقب في نفس المكان تمامًا بدقة كبيرة جدًا، كما أنها تشد البراغي بنفس مقدار القوة، دون أي تفاوت في مقدار تلك الدقة أو القوة بصرف النظر عن عدد الساعات التي عملت بها.

أخيرًا، توفر الأذرع الروبوتية الكثير من تكلفة العمل:

يمكن للروبوتات العمل على مدار الساعة دون كلل مع الحاجة إلى الحد الأدنى من الإشراف البشري. تعتبر صناعة السيارات بشكل آلي من أفضل الأمثلة على استخدام الذراع الروبوتية.

تم تطوير العديد من الأذرع الروبوتية للقيام بمجموعة متنوعة من المهام الطبية المختلفة. يتم توظيف الروبوت في فئتين رئيستين متعلقتين بالمجال الطبي: الجراحة باستخدام الذراع الروبوتية، واستخدام الروبوت في إعادة التأهيل وتقديم العلاج، حيث تساعد الأذرع الروبوتية الآلية المرضى في إعادة التأهيل بعد تعرضهم لحالات مرضية خطيرة مثل السكتات الدماغية، كما تساهم الأذرع الروبوتية في رعاية الأفراد من كبار السن وذوي الاحتياجات الخاصة والإعاقات الجسدية والذهنية، كما تقوم الأذرع الروبوتية الصناعية بمجموعة متنوعة من المهام الروتينية مثل تعقيم غرف المستشفيات وإيصال الإمدادات الطبية والمعدات والأدوية.

استخدامات الأذرع الروبوتية الشائعة في المجال الطبي	
المساعدة الجراحية	وهي أذرع روبوت متطورة تستخدم في العمليات الجراحية ويتم التحكم فيها عن بُعد، حيث يمكن التحكم في تشغيلها من محطة عمل خارج غرفة العمليات.
إعادة التأهيل	تلعب الأذرع الروبوتية دورًا محوريًا في إعادة تأهيل الأشخاص ذوي الإعاقة، ويمكن برمجتها للتكيف مع الاحتياجات الفردية لحالة كل مريض.
الصحة العامة والتعقيم	يمكن أن تقوم الأذرع الروبوتية بعملية تعقيم غرف العمليات والمستشفيات من البكتيريا والفيروسات في غضون دقائق.
أنظمة صرف الوصفات آليًا	إن مزايا السرعة والدقة في هذه الأذرع الروبوتية تجعلها أدوات جيدة للاستخدام في الصيدليات، حيث يمكنها صرف الأدوية آليًا، بل والتعامل مع المساحيق والسوائل والمواد الطبية اللزجة.

يُمكن أن نجد استخدامًا مُميزًا للأذرع الروبوتية في مختبرات الأبحاث، ويرجع ذلك إلى الحاجة إلى المرونة والاستخدام الجيد للمساحات في المختبر مع توفير التكامل السلس بين الأجهزة المختلفة داخل المختبرات. يُمكن للأذرع الروبوتية مساعدة الباحثين في استكشاف العقاقير الجديدة والمواد النشطة والتعامل مع العينات والتعامل مع المزارع الحيوية في المختبرات.

تستخدم شركات الأدوية الروبوتات في نقل العينات البيولوجية أو الكيميائية لتركيب مواد كيميائية جديدة أو لاختبار جودة المواد الكيميائية الموجودة لديها.

مزايا استخدام الأذرع الروبوتية في مختبرات الأبحاث
نظرًا لصغر حجمها ووزنها يمكن توظيفها في تطبيقات في حيز مكاني ضيق.
تساعد العاملين في عمليات البحث والتقييم وبشكل خاص تلك العمليات التي تتطلب التكرار الكثير بهدف الحصول على نتائج أفضل.
يُمكن استخدام الذراع الروبوتي في القيام بالمهام المتكررة بحيث تتم أتمتة أي مهمة والقيام بعمليات التحليل والاختبار بسرعة وسهولة.

تسبب الحرائق عند حدوثها ظروفًا خطرة وقد تنتشر إلى مناطق تشكل فيها خطورة شديدة على الإنسان والبيئة، وبشكل خاص على رجال الإطفاء، وفي مناطق تخزين المواد السريعة الاشتعال أو المواد المتفجرة أو محطات الطاقة النووية. يمكننا استخدام أذرع روبوتية مثبتة على مركبات كبيرة الحجم ومزودة بخراطيم لإطلاق المياه في المكان الذي تشتعل فيه النيران، وذلك كبديل عن رجال الإطفاء.

يمكن لهذه الروبوتات أن تعمل بشكل مستقل على اكتشاف وإطفاء الحريق من خلال استخدام مستشعرات اللهب والاستفادة من قدرات الأذرع الروبوتية الآلية التي يمكنها أن تدور في اتجاهات متعددة، كما ويمكن التحكم في معظم تلك الروبوتات القائمة على إطفاء الحرائق بواسطة مشغلات التحكم عن بعد الموجودة على مسافة آمنة.

تتمثل إحدى الفوائد المهمة لاستخدام روبوتات إطفاء الحرائق في أن رجال الإطفاء يصبح بإمكانهم القيام بمهام أخرى غير إطفاء النيران، مثل إنقاذ الأشخاص أو الممتلكات الشخصية أو العامة كاللوحات الفنية في المتاحف، ولكن تكمن الأهمية الكبرى لها في حماية رجال الإطفاء من المخاطر الكبيرة التي تنطوي عليها هذه الوظيفة.

يمكن لروبوتات إبطال مفعول القنابل والأجسام المشبوهة، وبهذه الطريقة يتجنب الكثير من الناس التعرض لخطر الموت. يتم التحكم بهذا النوع من الروبوتات من قبل مشغليها من مسافة آمنة، حيث يمكنهم رؤية ما تنقله عدسات الكاميرات الملحقة بالروبوت على شاشة المشغل.

يعتبر ذراع الروبوت من أهم الميزات نظرًا للطبيعة الخاصة لعناصر المقبض وإمكانية دوران الرسغ في هذا الذراع الروبوتي، فيمكن لهذا الروبوت معالجة الكثير من الأشياء بما فيها رفع وإزالة العوائق الثقيلة واستخدام أدوات قطع الأسلاك لتجاوز أسوار الأسلاك التي قد تعوق تقدمه.

في المستقبل ، يمكن تحسين روبوتات التخلص من القنابل هذه. على سبيل المثال ، يمكننا تطوير روبوتات ذات ذراعين ، لتزويدهم بمهارة يدوية أكبر ، مثل السماح لهم بفتح حقيبة السيارة والنظر بالداخل.

أصبح بإمكاننا ملاحظة وجود الأذرع الروبوتية في كل مكان حولنا، وهذا باعتقادنا من شأنه أن يجعل المستقبل أفضل، فلنفكر في بعض الاستخدامات المستقبلية للروبوتات.

الأذرع الروبوتية المتعاونة

يتم وضع الأذرع الروبوتية التقليدية لتعمل في مناطق خاصة بالروبوتات فقط. قد يتغير هذا الأمر في المستقبل مع ظهور أذرع روبوتية تعاونية تسمح بدرجات مختلفة من التفاعل بين الإنسان والروبوت، ونظرًا لسعرها المعقول وأدائها العالي، فسوف تفتح مجالًا واسعًا للتشغيل الآلي للكثير من العمليات التي تقوم بها الشركات الصغيرة والمتوسطة، وستؤدي إلى ظهور العديد من التطبيقات التي يمكن للأفراد والروبوتات فيها العمل معًا.

ومن الأمثلة على ذلك، قيام الروبوتات بالأعمال اليدوية مثل رفع الأحمال الثقيلة، وتناول البضائع من البشر ونقلها إلى أماكنها الصحيحة، ... وغيرها، وهي بذلك تكمل القدرات البشرية وتريح الإنسان من المهام الشاقة دون أن تحل محله.

روبوتات الفضاء

تعتبر الأذرع الروبوتية ذات أهمية عظيمة في مهام الفضاء، فمعظم هذه الروبوتات لديها أذرع آلية للمساعدة في جمع الصخور والعينات لتحليل واكتشاف التركيب الجيوكيميائي لسطح الكواكب، حيث يتم التحكم بهذه الروبوتات عن بعد من قبل البشر على الأرض. من المتوقع أن يؤدي التطور الكبير والمتسارع للخوارزميات والبرمجيات وقدرات المعالجة الحاسوبية إلى منح الأنظمة الروبوتية بعض مستويات من الذكاء الاصطناعي و الاستقلال الذاتي. على سبيل المثال، لدى روبوت الفضاء **Curiosity** في كوكب مارس، ذراع روبوتية يمكنها التحرك مثل اليد البشرية. تتيح الذراع للروبوت العمل كما يعمل الجيولوجي البشري: من خلال حمل واستخدام أدوات العلوم بذراعه، وتقوم "الأدوات اليدوية" الخاصة بالروبوت بجمع العينات الصخرية والتقاط الصور المجهرية وفحص التركيب الأولي والمحتوى المعدني لتربة المريخ.

الروبوتات الجراحية

يتمحور مفهوم الجراحة التي تتم بمساعدة الروبوتات في أن الجراح يمكنه التحكم عن بعد وكذلك مراقبة الأذرع الروبوتية أثناء إجراء العمليات الجراحية للمرضى. يمكن اعتبار هذه الأذرع كنوع من الروبوتات التعاونية كونها على اتصال مباشر بالمريض. سيتمكن الأطباء من استخدام روبوت جراحي دقيق يعمل داخل تجويف ماسح التصوير بالرنين المغناطيسي لعمل الخزعات بشكل دقيق، وذلك من خلال استخدام أنظمة التحكم الإلكترونية وبرامج الحاسب للتحكم بذلك الروبوت.

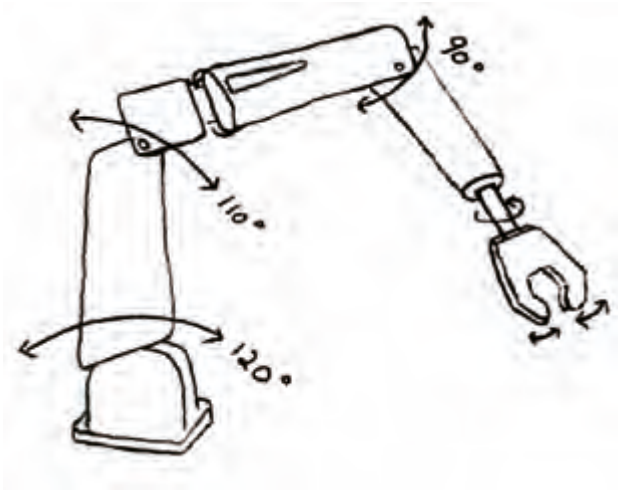
الأذرع الروبوتية في الصناعات الغذائية

تستخدم الأذرع الروبوتية على نطاق واسع في الصناعات الغذائية، وعلى سبيل المثال حلب المواشي وصناعة منتجات الحليب والألبان، وذلك باستخدام تجهيزات خاصة مثل كأس الشفط وأنظمة الليزر لضبط المسافة.

النموذج الأولي للذراع الروبوتية

عند وجود فكرة لإنشاء ذراع روبوتي، فإننا نحتاج إلى اتباع بعض الخطوات المحددة للقيام بذلك. فلنستعرض الآن أهمية هذه الخطوات وفاعلية اتباعها.

1.إنشاء مخطط الأفكار



كخطوة أولى للعمل، يجب تمثيل النموذج الأولي الذي نريد إنشاءه. يعتبر الرسم باستخدام الورقة والقلم أبسط وأسرع طريقة لتمثيل الأفكار، حيث يعتبر الرسم على الورق المرحلة الأكثر فعالية قبل البدء باستخدام أحد برامج الرسم الرقمي، وذلك لبلورة أفضل الأفكار بدلًا عن إضاعة الكثير من الوقت في إنشاء واتقان الرسم الرقمي.

2. تطوير النموذج الافتراضي (Virtual Prototype)

في مرحلة لاحقة؛ يصبح إنشاء الرسم الافتراضي لفكرتنا أمرًا هامًا وضروريًا للغاية. للقيام بذلك، نحتاج إلى استخدام أداة تصميم رقمية لمساعدتنا على التعرف على شكل النموذج المادي من التصميم. يجب أن تراعي عملية التصميم أن ذراع الروبوت ستكون متحركة، مما يوجب تمثيل كيفية حركتها في بيئتها (المكان الذي سيتم استخدامها فيه)، وما الذي سيحدث إذا تم وضعها في بيئة مختلفة، فمثلًا هل ستظل هذه الذراع قادرة على التحرك داخل هذه المساحة الجديدة؟

3. إنشاء النموذج المادي (Physical Prototype)

بمجرد جاهزية النموذج الافتراضي الأولي، تكون قد أصبحت جاهزًا لإنشاء النموذج المادي على أرض الواقع. قد تتطلب هذه الخطوة بناء الكثير من النماذج الأولية المكلفة ماديًا وزمنيًا، ولذلك يُنصح باختيار مواد منخفضة التكلفة كالخشب والبلاستيك خلال هذا العملية.

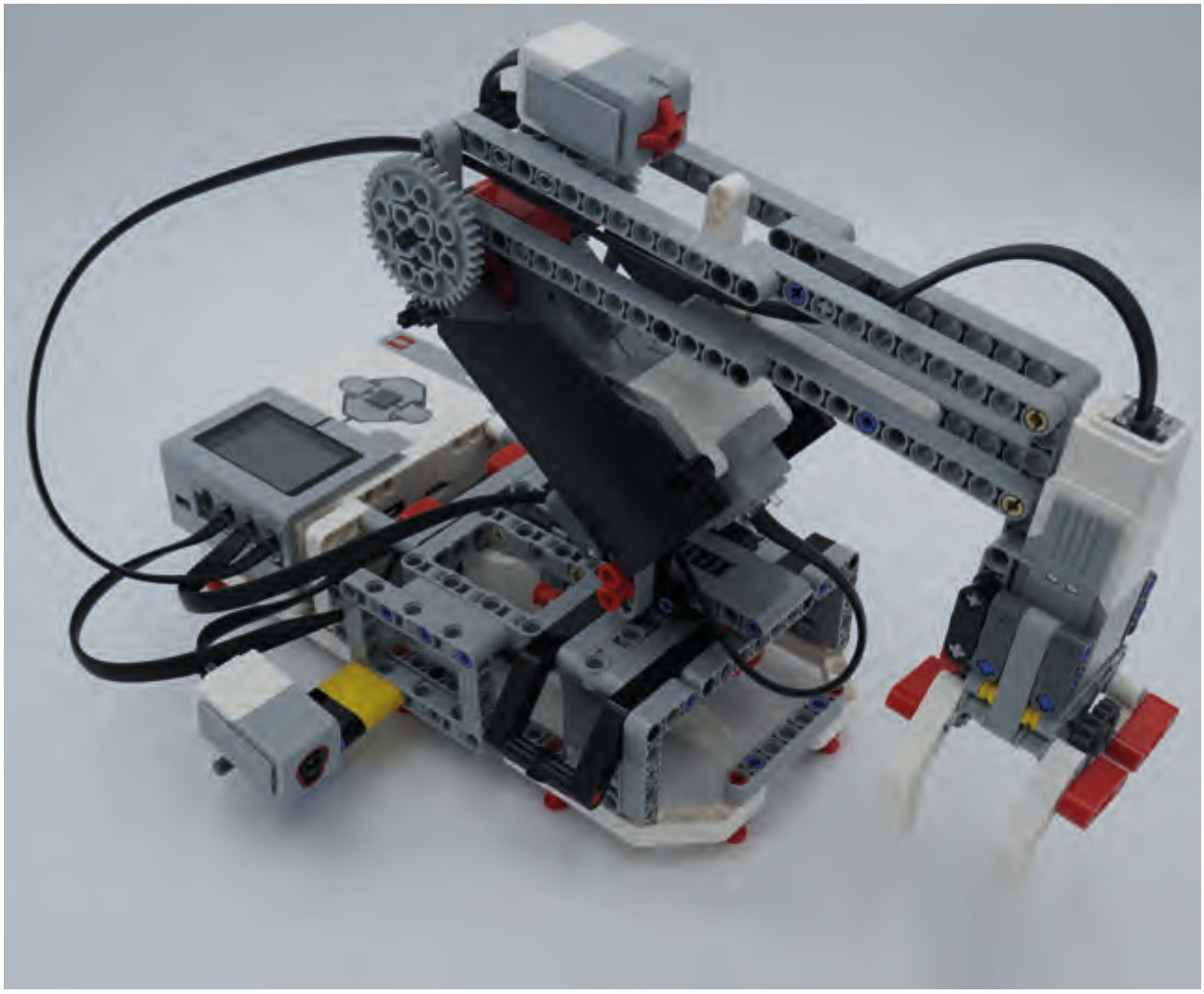
4. البرمجة والاختبار

الخطوة النهائية التي ستتم هي برمجة واختبار ذراع الروبوت، حيث يمكننا خلال هذه العملية مراقبة عمل ذراع الروبوت في البيئة المحيطة. قد يحتاج الأمر إلى وضع المزيد من أجهزة الاستشعار لتحقيق نسبة الذكاء المطلوبة ومن ثَم يُمكننا زيادة درجة التعقيد في الذكاء من خلال إضافة الأدوات والمفاتيح وبعض المكونات المستقلة حسب الحاجة.

أخيرًا نحتاج إلى تقييم عمل الذراع الروبوتية. إن أفضل طريقة للقيام بذلك هي عن طريق طرح بعض الأسئلة مثل:

الذراع الروبوتية

في هذه الوحدة سنقوم ببناء وبرمجة ذراع الروبوت (نظام روبوت ثابت) خاص بنظام فرز تلقائي يمكن استخدامه في عمليات إعادة التدوير.



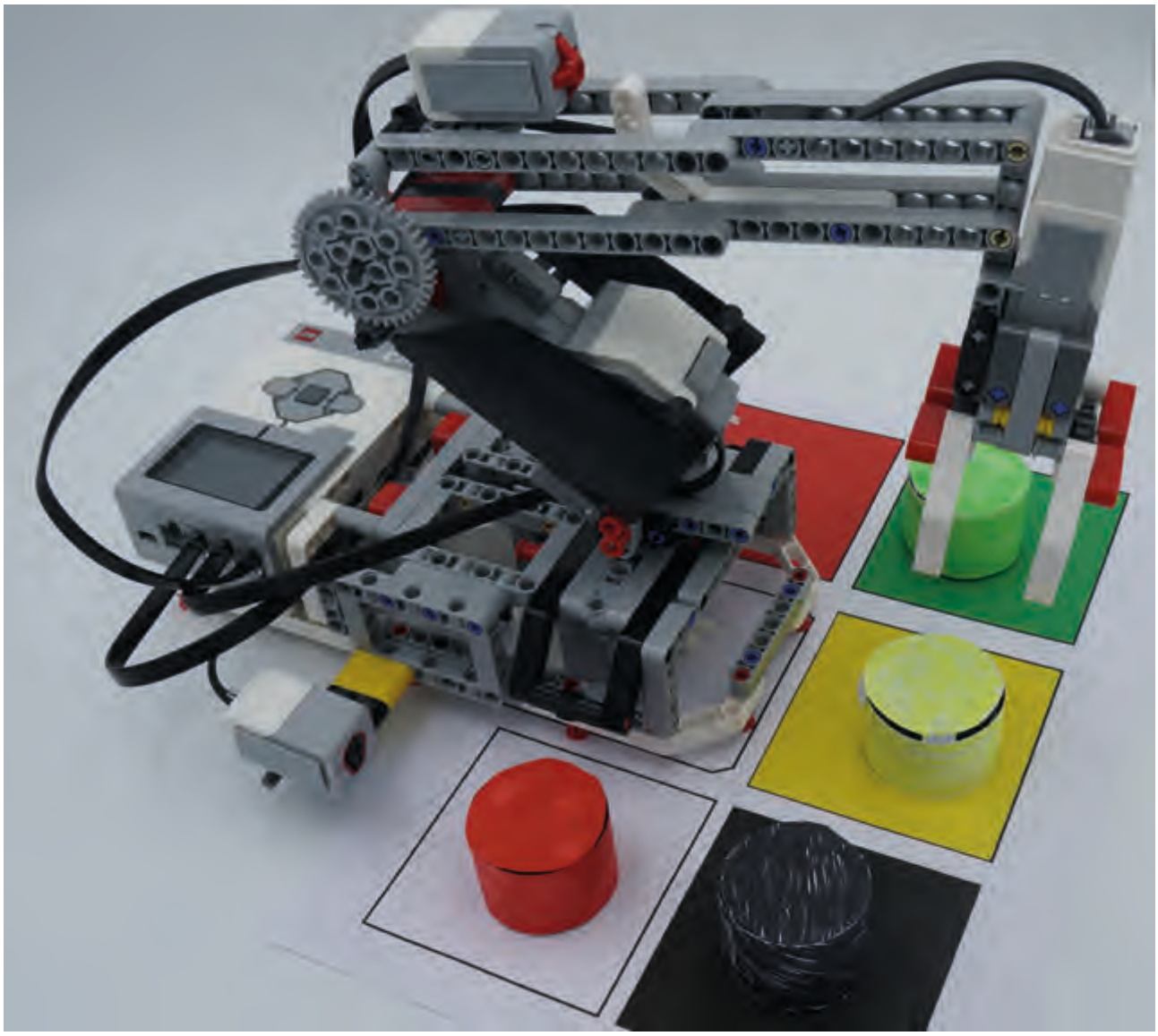
سنستخدم **LEGO® MINDSTORMS® EV3**، ثم سنقوم بعمل البرامج المطلوبة في **Python** بهدف إنشاء نظام ذراع روبوتية تستخدم كجهاز يقوم بفرز العلب بشكل تلقائي. لتصميم هذا الجهاز، نحتاج إلى وجود **3** محركات تتحكم بشكل منفصل في الوصلات الثلاثة المكونة لذراع الروبوت وباستخدام محرك واحد لكل منها.

في بداية هذا السيناريو سنتحكم في ذراع الروبوت يدويًا لكي يلتقط العلبة ثم نقوم بتحريكها من خلال استخدام مفاتيح الأسهم الموجودة في **EV3 Brick**.

في هذه الحالة، سنبرمج ذراع الروبوت لكي تعمل بشكل مستقل باستخدام مستشعرين، وهما: مستشعر اللمس **Touch Sensor** ومستشعر اللون **Color Sensor**، حيث سيعمل مستشعر اللمس عندما يتجه ذراع الروبوت لالتقاط العلبة، أي أنه عندما يصل ذراع الروبوت إلى موضع اختيار العلبة، فإن مكون **Lego** سوف يضغط على مستشعر اللمس وينشط الوصلتين العلويتين. يتموضع مستشعر اللون في قاعدة مدعومة بذراع الروبوت لكي يتمكن من الكشف عن نوع مادة العلبة حسب لونها.

سنستخدم هياكل البيانات مثل القاموس **Dictionary** لمطابقة لون العلبة ومادتها ورمزها المحدد لتسجيل جميع إجراءات الروبوت في سجلات داخل ملف نصي تسلسلي.

ستلتقط ذراع الروبوت العلبة وفقًا لمادتها ثم ستقوم بنقلها إلى الموقع المناسب لكي يتم إعادة استخدامها بشكل صحيح.



بناء قاعدة الروبوت

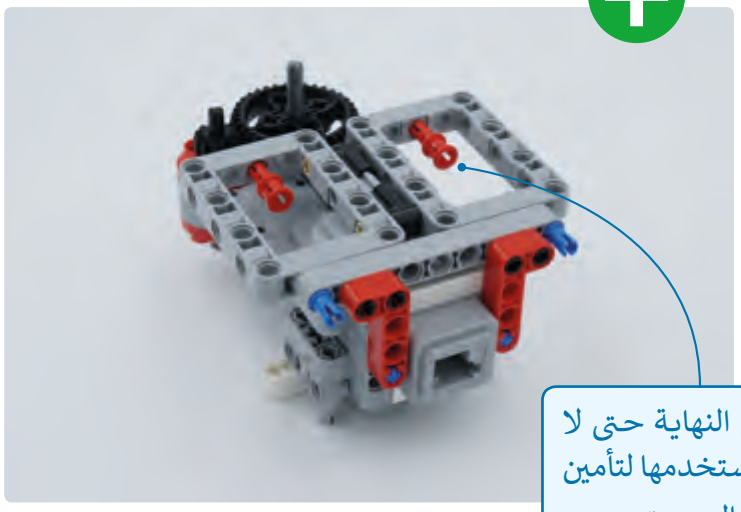
سنقوم بتقسيم الخطوات الخاصة بتجميع ذراع الروبوت إلى ثلاثة أجزاء رئيسة، حيث سنقوم في هذا الدرس ببناء قاعدة ذراع الروبوت. سنقوم أيضًا بتقسيم الخطوات الخاصة بتجميع قاعدة الروبوت إلى ثلاثة مراحل. بعد الانتهاء من كل مرحلة سنجمعها معًا لتجميع قاعدة الروبوت.

المرحلة الأولى

الجزء الأول من بناء قاعدة الروبوت هو لربط وحدة تحكم **EV3**.

المرحلة الثانية

الجزء الثاني من بناء قاعدة الروبوت سيتصل بالجزء الأول لإنشاء قاعدة كاملة للذراع.



المرحلة الثالثة

إنشاء حامل للمحرك الكبير وجهاز استشعار اللمس لتركيبها مع بناء المرحلة الأولى.

لا تدفع إسفين الربط إلى النهاية حتى لا يتم ربطها بالإطارات. سنستخدمها لتأمين جزء واحد من بناء مرفق الروبوت.

سنجمع القطع التي حصلنا عليها في المراحل الثلاثة السابقة للوصول إلى الشكل النهائي للقاعدة الروبوتية، ثم سنقوم بتوصيل مستشعر اللمس.

سنستخدم سلكين بطول **25** سم. سلك لتوصيل مستشعر اللمس بالمنفذ **1** وسلك آخر لتوصيل المحرك المتوسط بالمنفذ **C**.

الشكل النهائي للقاعدة الروبوتية

نصيحة ذكية

تذكر عدم ثني الأسلاك كثيرًا واتبع الإرشادات لتوصيلها بشكل صحيح.

برمجة قاعدة الروبوت

الآن وبعد تجميع قاعدة الروبوت، فقد حان الوقت لبرمجتها. في البداية سنقوم بإعداد المحرك المتوسط الذي سنستخدمه لتدوير قاعدة الروبوت ومن ثم مستشعر اللمس، ثم سنقوم أيضًا بتوجيه المحرك تجاه عقارب الساعة، وبالتالي فإن قيم السرعة السالبة ستجعل الذراع يتحرك بعيدًا عن مستشعر اللمس.

ونظرًا لأن المقطع البرمجي لدينا سيكون كبيرًا جدًا، فإننا نحتاج إلى الفصل بين كل قسم مختلف بخطوط التعليق، وهذا من شأنه أن يساعدنا في تتبع كل سطر من الأسطر البرمجية.

قبل أن نستخدم محرك معين، من المهم معرفة حجم الترس المتصل به، في حالتنا سنستخدم اثنين من التروس، أحدهما بـ 12 سن والآخر بـ 36 سن.

تعيين المحرك **Motor** إلى المنفذ C

تعليق خاص بكيفية إعداد المنافذ

قسم ضبط المنافذ

```
# Write your program here

# ================== Set Ports ================== #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])

# Set sensors
touch_sensor = TouchSensor(Port.S1)
# ================== Set Ports ================== #
```

تعيين مستشعر اللمس **TouchSensor** إلى المنفذ **1**

المحرك المتوسط يدور تجاه عقارب الساعة

تشير الأرقام هنا إلى التروس.

معايرة القاعدة

من أجل اختصار برنامجنا، نحتاج إلى إنشاء عدة دوال. إن الدالة الأولى التي سنستخدمها ستكون وظيفتها معايرة القاعدة، وتعني المعايرة هنا بأنه عندما نحتاج وجود كائن في نقطة الصفر، فسيتم تعريف هذه النقطة كنقطة البداية.

لتهيئة القاعدة عليك في البداية أن تقوم بتشغيل المحرك الأساسي في اتجاه عقارب الساعة إلى حين الضغط على مستشعر اللمس (**Touch Sensor**)، وبعدها سيتوقف المحرك مع الحفاظ على وضعه وإعادة ضبط الزاوية على "**0**"، وهذا يعني أنه عندما تستدير للخلف إلى "**0**" فيما بعد، فإنها ستعود إلى موضع البداية.

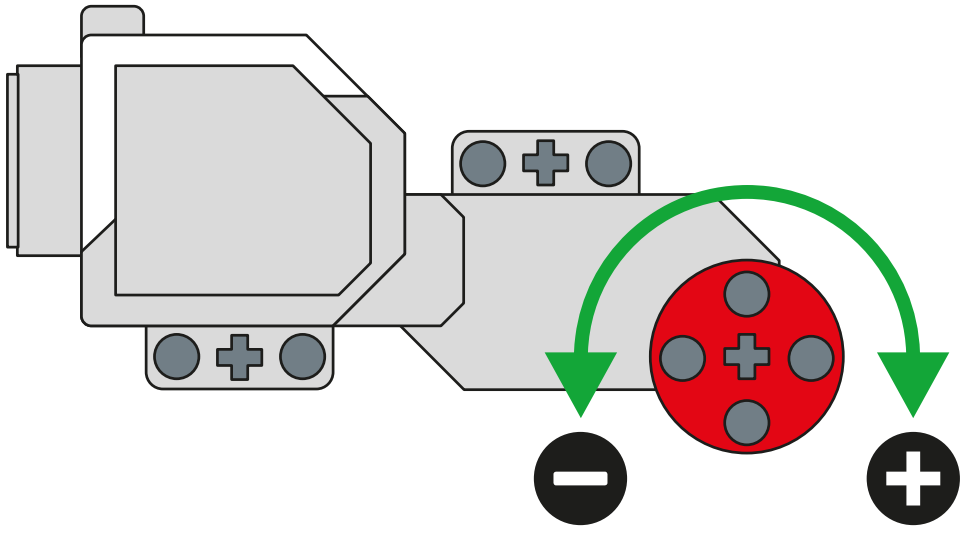
معايرة الروبوت هي تقنية تستخدم لتحسين دقة نظام تموضع الروبوت.

قم بتسمية الدالة **calibrate_base**

ضبط سرعة محرك القاعدة على **60**

دالة معايرة Calibration القاعدة

```
# Write your program here

# ================== Functions ================== #
def calibrate_base():
    base_motor.run(60)
    while not touch_sensor.pressed():
        wait(10)
    base_motor.stop(Stop.HOLD)
    base_motor.reset_angle(0)
# ================== Functions ================== #
```



نحتاج إلى استدعاء الدالة التي أنشأناها، وبها سيكون شكل البرنامج بصورته النهائية كما هو أدناه.
يمكنك الآن أن تقوم بتوصيل لبنة **EV3 Brick** بالحاسوب وتشغيل البرنامج لاختبار عمله.

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase

# Write your program here

# ================ Functions ================ #
def calibrate_base():
    base_motor.run(60)
    while not touch_sensor.pressed():
        wait(10)
    base_motor.stop(Stop.HOLD)
    base_motor.reset_angle(0)
# ================ Functions ================ #

# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])

# Set sensors
touch_sensor = TouchSensor(Port.S1)
# ================ Set Ports ================ #

# ================ main program ================ #

# Call Functions for calibration
calibrate_base()
# ================ main program ================ #
```

استدعاء الدالة الخاصة بمعايرة قاعدة الروبوت.

1

صنف أقسام الأنظمة الروبوتية واشرحها.

2

أي فئة من تطبيقات الروبوت تعتقد أنها الأكثر أهمية، ولماذا؟

3

ما هي الفائدة من استخدام الذراع الآلي في الصناعة؟

_____

_____

_____

_____

4

أجرِ التغييرات اللازمة على البرنامج الذي أنشأناه في الدرس السابق بحيث يتم توجيه المحرك للجهة المعاكسة.

5

بعد التغيير على البرنامج السابق، قم باستبدال أمر الاتجاه مع عقارب الساعة CLOCKWISE بالأمر عكس عقارب الساعة COUNTERCLOCKWISE. ما الذي ستلاحظه؟

6

قم بتغيير عدد أسنان التروس مثل الموجود بالأسطر البرمجية أدناه، الأول [12,80]، ثم [40,80]، ثم قم بتشغيل البرنامج لاختبار المقطع البرمجي.

```
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 80])
```

```
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [40, 80])
```

7

نريد في البرنامج التالي تغيير المنافذ، حيث سيتم تغيير المحرك الأوسط إلى المنفذ A ومستشعر اللمس إلى المنفذ 3. ما التغييرات التي علينا القيام بها لكي يعمل البرنامج بشكل صحيح؟

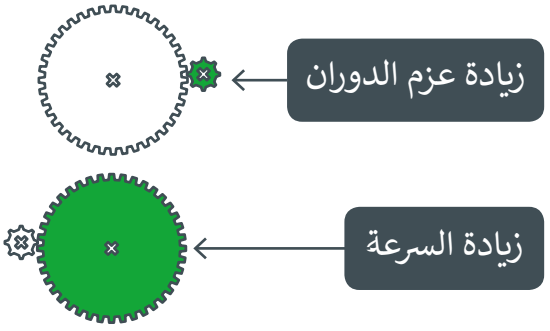
```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase

# Write your program here

# ================ Functions ================ #
def calibrate_base():
    base_motor.run(60)
    while not touch_sensor.pressed():
        wait(10)
    base_motor.stop(Stop.HOLD)
    base_motor.reset_angle(0)
# ================ Functions ================ #

# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])

# Set sensors
touch_sensor = TouchSensor(Port.S1)
# ================ Set Ports ================ #

# ================ main program ================ #

# Call Functions for calibration
calibrate_base()
# ================ main program ================ #
```

التروس

قبل أن نبدأ في بناء ذراع الروبوت في هذا الدرس، سنقضي بعض الوقت لنتعرف على التروس المستخدمة في الروبوتات وأنواعها وكيفية عملها واستخداماتها في الذراع الروبوتية. يُعرَّف الترس (**Gear**) بأنه عبارة عن عَجَلة بأسنان تندمج مع ترس مشابه آخر، عادةً ما يكون أكبر أو أصغر منه. تُستخدم التروس في كل مكان حولنا وهي جزء أساسي في الكثير من الآلات، كما أنها تأتي بأحجام وأنواع مختلفة. قد تعمل بعض التروس أفضل من غيرها، وتعتمد فعالية عمل التروس حسب الموضع الذي تستخدم فيه.

يُمكن أن يُستخدم الترس في تغيير السرعة أو الاتجاه أو عزم الدوران.

إن زيادة عزم الدوران (**Torque**) في آلة (مثل دراجة أو سيارة) ينتج عنه سرعات منخفضة، بينما يؤدي عزم الدوران الأقل إلى سرعات أعلى دون الحاجة إلى زيادة قوة المحرك، وهذا يرجع إلى القانون التالي:



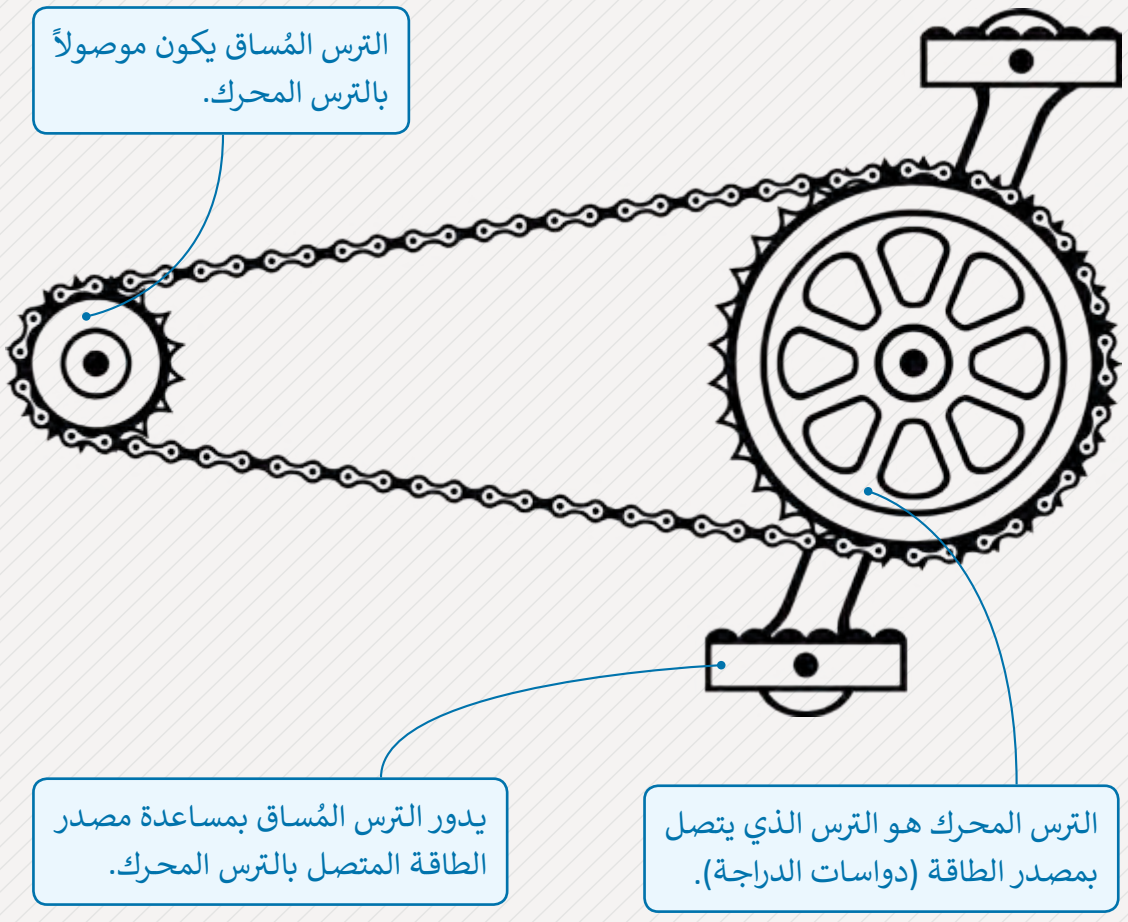
القوة = عزم الدوران × السرعة

عزم الدوران = القوة / السرعة

فيما يلي بعض الأشياء الأساسية التي يجب تذكرها بشأن التروس:

- يمكنك قياس الترس عن طريق عد أسنانه.
- نجمع بين التروس وفقًا لنصف القطر والسُمك.
- تحتوي التروس على ثقب متقاطع في الوسط بحيث يمكن وضعها على محاور ونقل الدوران من محور إلى آخر.
- يتصل الترس السائق عادة بمصدر حركة (محرك **Motor** على سبيل المثال).
- يتحرك الترس المساق بناء على حركة الترس السائق، ويتصل بوجهة الحركة (المحور أو القطع).

مثال على كيفية عمل التروس في الدراجة الهوائية.

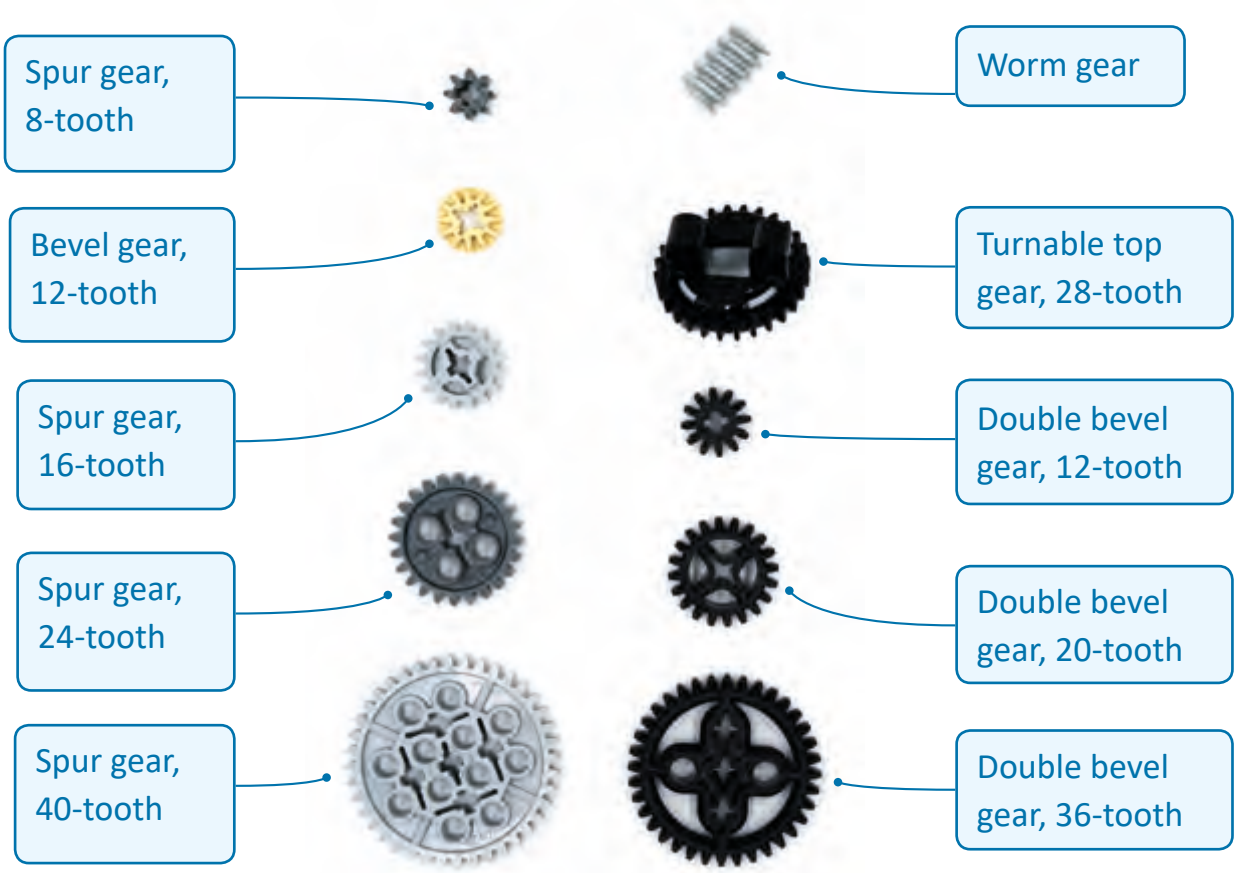


أنواع التروس

توجد العديد من أنواع التروس التي يتم تصنيفها حسب المحاور (**Axes**).

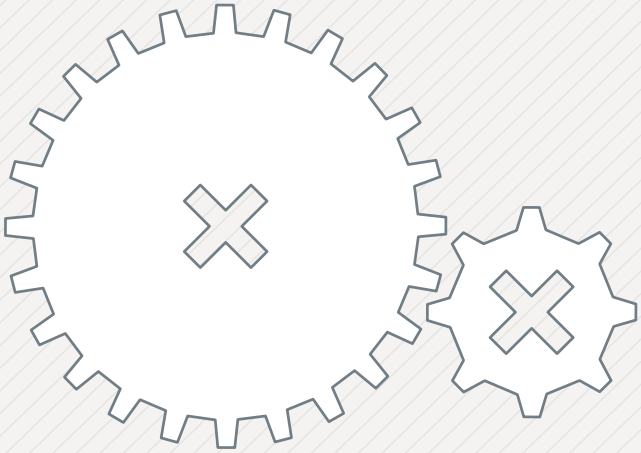
تصنيف التروس حسب اتجاه محورها	
المحاور المتوازية	ترس ذو أسنان مستقيمة (**Spur gear**)، ترس داخلي (**Internal gear**)، الترس الحلزوني (**Helical gear**)، قطاع مسنن (**Gear rack**).
المحاور المتقاطعة	الترس المائل (**Miter gear**)، الترس المخروطي (**Straight bevel gear**)، الترس المخروطي الحلزوني الأسنان (**Spiral bevel gear**).
محاور غير متقاطعة وغير متوازية	ترس لولبي (**Screw gear**)، ترس دودي (**Worm gear**)، عجلة دودية (**Worm wheel**).
أخرى	عمودي مخدد ومسطح (**Involute Spline Shaft and Bushing**)، ترس وصلة (**Gear Coupling**)، ترس ذو سقاطة (**Pawl and Ratchet**).

لنلقِ نظرة على نوع التروس الموجودة في مجموعة **EV3**.



العلاقة بين التروس يعبر عنها من حيث النسبة بين عدد أسنانهم.

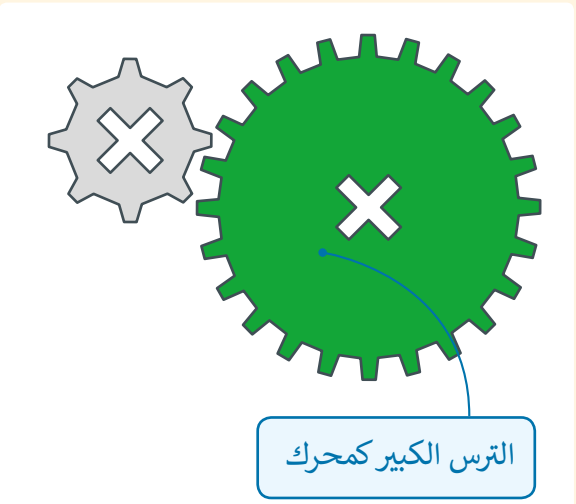
على سبيل المثال، إذا كان لدينا ترسان واحد له **8** أسنان وآخر له **24** سنًا، تكون النسبة بينهم **8:24** = **1:3**. النتيجة هي أن الترس ذو **24** سن تكون سرعته ثلث سرعة الترس ذو الأسنان الثمانية.

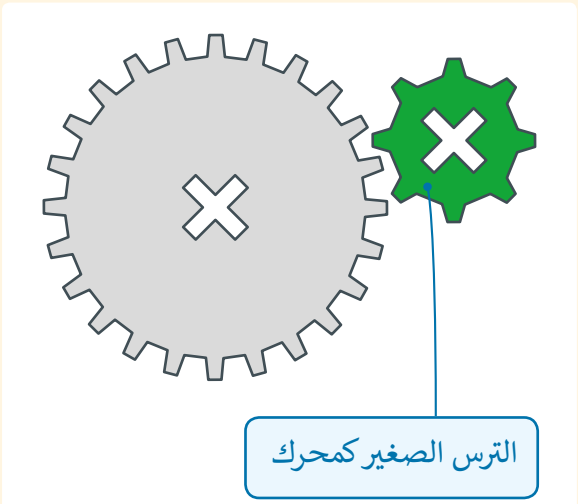


إن ميزة مجموعة التروس هذه هي أنه في حال تشغيل الترس المحرك ذي **8** أسنان، فإن الترس ذي **24** سن يكون لديه عزم دوران أكثر بثلاث مرات من الترس المحرك.

من المهم ذكر أن:

> الترس السائق هو الجهاز المتصل بالمحرك
> إذا أردنا زيادة عزم الدوران، فسنستخدم الترس الصغير كمحرك، وإذا أردنا زيادة السرعة، فسنستخدم الترس الكبير كمحرك.





على سبيل المثال، قد تحتاج بعض الآلات إلى السرعة، بينما قد تحتاج الآلات الأخرى إلى القوة، لذلك من المهم معرفة ما إذا كانت السرعة أو العزم أكثر أهمية عند تصميم التروس الخاصة بآلة معينة. لذلك، إذا كان الترس المحرك أصغر من الترس المُساق، فسوف تنخفض سرعة الترس المُساق، وسيزيد عزم الدوران الخاص به. من ناحية أخرى، إذا كان الترس المحرك هو الأكبر، فسوف تزداد سرعة الترس المُساق ولكن عزم الدوران الخاص به سينخفض.

يمكن تقسيم التروس المستديرة لمجموعة أدوات **EV3** إلى مجموعتين:

> التروس العادية ذات الأسنان المربعة **(Spur Gears)** والتروس المخروطية ذات الأسنان المستديرة **(Bevel Gears)**.
> عملياً، يمكن استخدام أي ترس من المجموعة الأولى مع أي ترس من المجموعة الثانية.

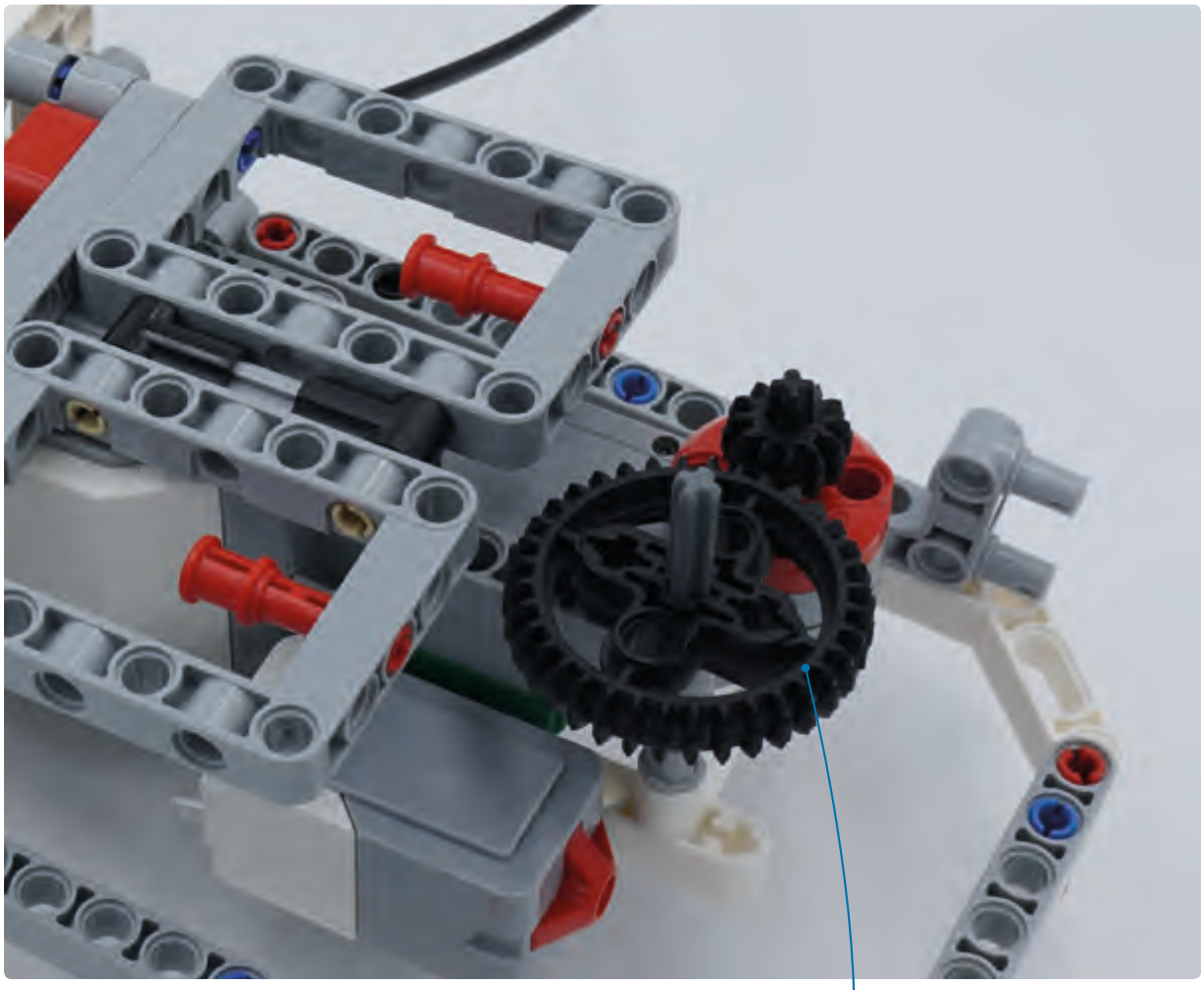
الميزة الفريدة للتروس المخروطية **Bevel Gear** هي أنه يمكن دمجها بطريقة متوازية وعمودية. كما أنها أكثر ملاءمة للاستخدام في الذراع الرافعة بسبب حجمها.

تروس الذراع الروبوتية

تحتوي الذراع الروبوتية على تسعة تروس. فلنستعرض التروس التي سيتم استخدامها في تجميع الروبوت.

قاعدة الذراع الروبوتية

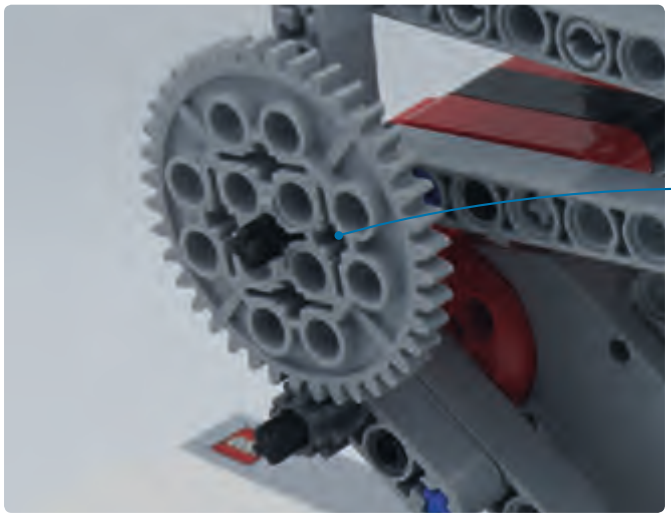
الترس المستخدم
الترس المخروطي المسنن المزدوج، **12** سن، أسود.
الترس المخروطي المسنن المزدوج، **36** سن، أسود.



يحتوي الترس المحرك في القاعدة الروبوتية على 12 سن، في حين أن الترس المُساق يحتوي على 36 سناً. نستخدم هذه المجموعة للحصول على مزيد من عزم الدوران للحصول على القوة اللازمة لتغيير اتجاه مرفق الروبوت.

الذراع الروبوتية

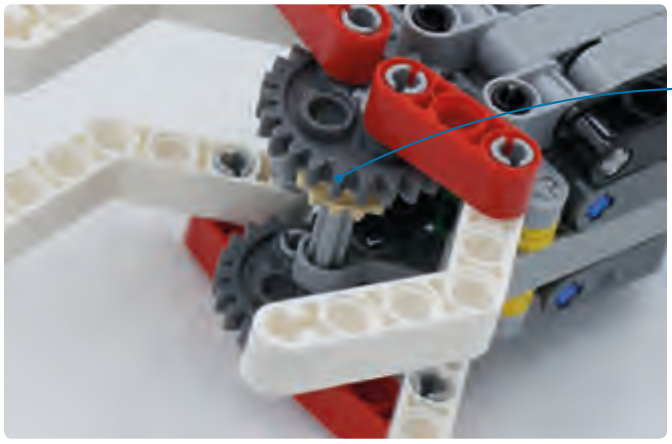
الترس المستخدم
الترس ذو أسنان مستقيمة (ترس مستقيم)، **8** أسنان، رمادي غامق.
الترس ذو أسنان مستقيمة (ترس مستقيم)، **40** سن، رمادي.



يحتوي الترس المحرك للمرفق الروبوتي على 8 أسنان والترس المُساق يحتوي على 40 سنًا. نستخدم هذه المجموعة للحصول على مزيد من عزم الدوران بهدف الحصول على القوة المطلوبة للرفع لأعلى وأسفل الجزء العلوي من مرفق الروبوت.

قابض الروبوت

الترس المستخدم
الترس المخروطي المسنن، **12** سن، بني فاتح (اللون البيج)
الترس المخروطي المسنن المزدوج، **12** سن، أسود
ترس ذو أسنان مستقيمة (ترس مستقيم)، **24** سن، رمادي غامق



محرك ترس المقبض الروبوتي يحتوي على 12 سناً والترس المُساق فيه يحتوي على نفس عدد الأسنان.

بناء ذراع ومرفق الروبوت (Building a robot elbow)

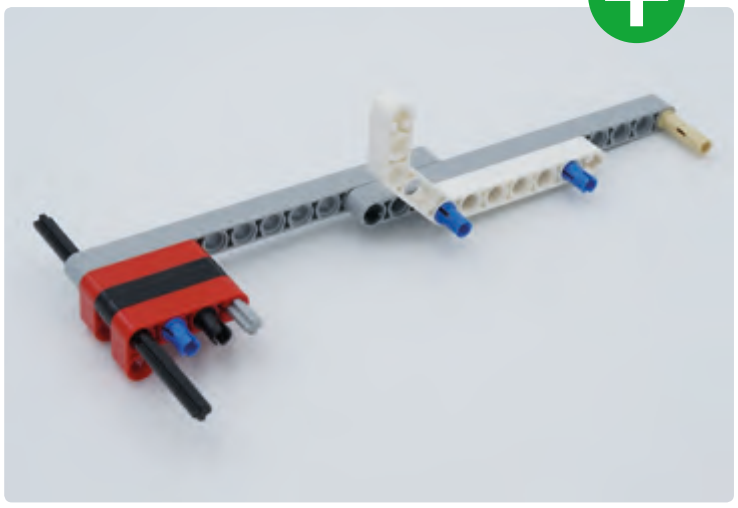
في هذا الدرس سنقوم ببناء الذراع الروبوتية. سنقوم بتقسيم هذه العملية إلى **3** مراحل. بعد الانتهاء من كل مرحلة، سنجمعها ونضيف بعض الأجزاء الإضافية لإكمال مرفق الروبوت.

المرحلة الأولى

سنبني قاعدة المرفق ونركبها على قاعدة الروبوت التي أنشأناها في الدرس السابق.

المرحلة الثانية

إنشاء حامل للمحرك الكبير ليتم تثبيته. بحيث يتصل بالبناء من المرحلة الأولى.



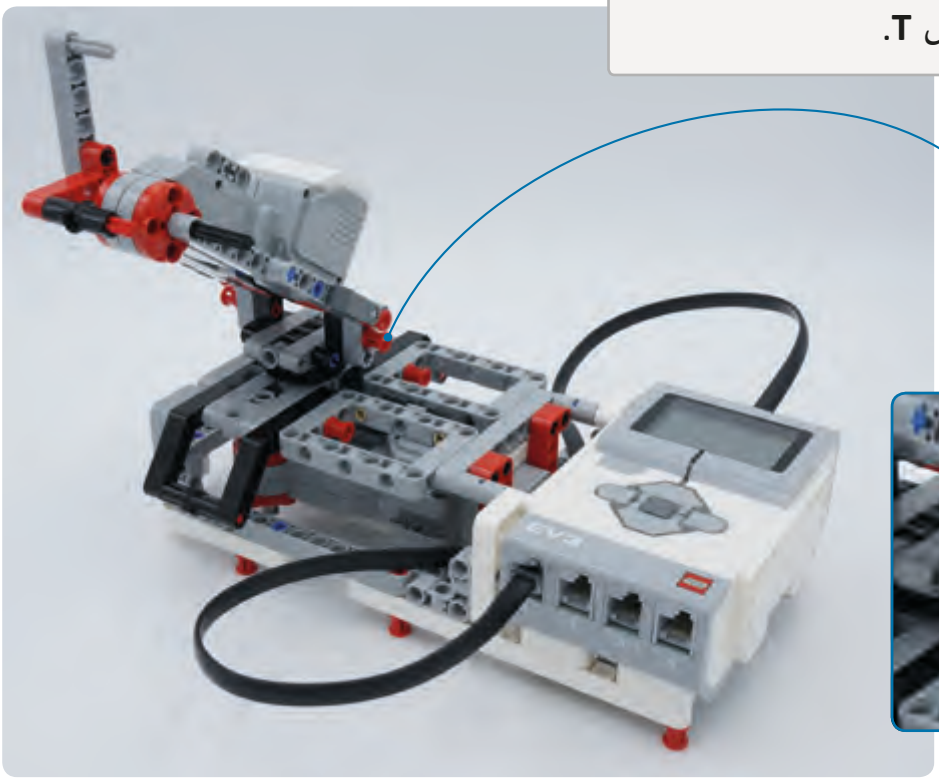
المرحلة الثالثة

بناء الجزء العلوي للذراع حيث سيعلق القابض لاحقًا.

لنقم الآن بتثبيت القطع التي قمنا بتجميعها في المراحل الثلاث السابقة على قاعدة الروبوت الرئيسة.

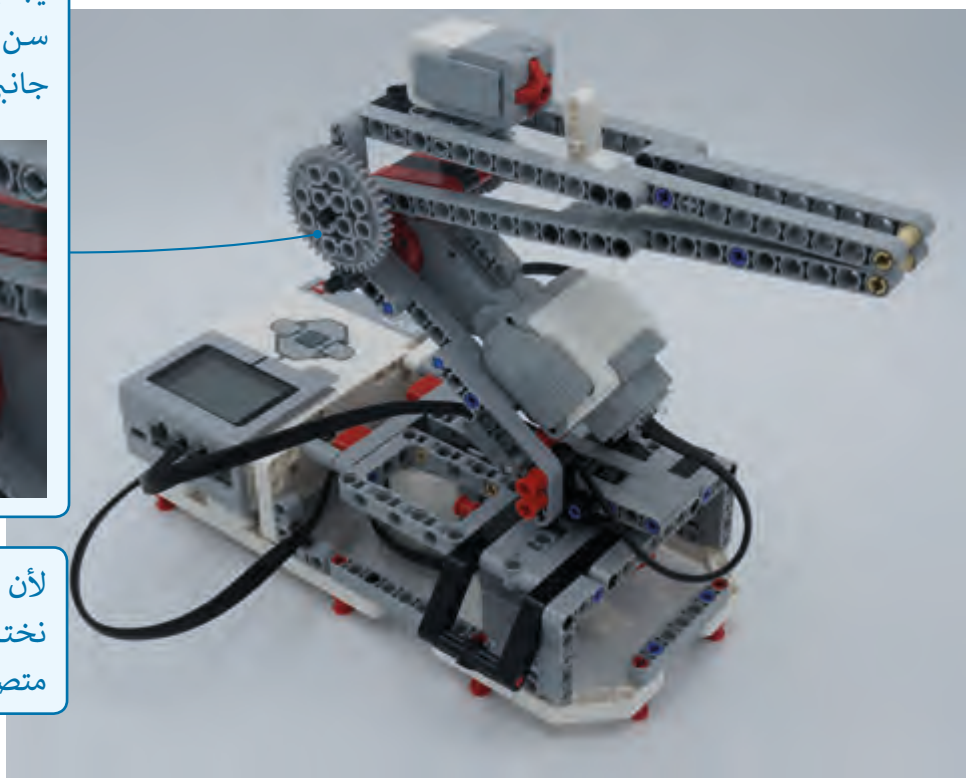
نقوم بتثبيت قاعدة المرفق التي حصلنا عليها في المرحلة الأولى من خلال دفع قطعة الربط داخل عارضة الزاوية.

نقوم بتثبيت حامل المحرك الذي حصلنا عليه من المرحلة الثانية من خلال دفع قطعة الربط داخل العارضة على شكل **T**.



نثبت الجزء العلوي من الذراع والذي قمنا بتركيبه في المرحلة الثالثة، لنحصل على الجانب الأول للذراع.

يجب أن نضع ترس ذو 40 سن وترس ذو 8 أسنان على جانبي المرفق.



لأن الكوع ثقيل جداً للرفع، نختار له ترس ذو 8 أسنان متصل بالمحرك.

نقوم بإنشاء الجانب الآخر من الذراع ونثبته للحصول على ذراع الروبوت بصورتها المكتملة.

سنستخدم تعليمات بناء **LEGO®** لتجميع ذراع الروبوت. استعن بمعلمك لفهم عملية التركيب وإتمامها بنجاح.

سنستخدم سلكين بطول **35** سم. سلك لتوصيل مستشعر اللمس بالمنفذ **3** وسلك ثاني لتوصيل المحرك الكبير بالمنفذ **B**.

استخدم الألواح المنحنية لتحقيق تصميم أفضل لذراع الروبوت.

الشكل النهائي لمرفق الروبوت

برمجة الذراع الروبوتية

بعد أن قمنا بتجميع ذراع الروبوت، حان الوقت لبرمجته. وسنقوم ببرمجة الوظائف الآتية والمتعلقة بالذراع:

1. إظهار صور على شاشة EV3 Brick توضح اتجاهات الحركة.
2. تحريك الذراع في الاتجاهات المختلفة.
3. معايرة مواقع الذراع وتهيئتها.
4. ضبط السرعة القصوى وتسارع المحرك.

لنبدأ بالتعرف على بعض المكتبات المطلوبة للبرمجة، فمثلًا:

الاستيراد من المكتبة

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase
from time import sleep
```

سنحتاج إلى استيراد الوظيفة **Sleep** من مكتبة **Time**، ويجب أن يتم استخدام تأخير زمني Delay بين خطوات التحكم اليدوي للذراع.

إظهار الصور على شاشة EV3 Brick

من أجل التعامل مع زر **EV3 Brick** وعرض صورة، نحتاج إلى التعرف على الأزرار المختلفة للروبوت وحالات استخدامها.

الضغط على زر key press

لأزرار **EV3 Brick** كلماتها المفتاحية الخاصة، وإذا تم دمجها مع زر **process** فيمكن الحصول على **6** حالات مختلفة. يتم تشغيل كل حالة عند الضغط على المفتاح الصحيح.

مثال

```
if button == Button.UP:
    # Execute this case when UP
    # Button is pressed
```

أسماء أزرار EV3 Brick
Button.UP
Button.DOWN
Button.LEFT
Button.RIGHT
Button.CENTER
Button.BACKSPACE

عرض ملف الصورة

يمكننا عرض العديد من الصور المختلفة على **EV3 Brick** باستخدام الكلمة المفتاحية المناسبة مع الأمر **brick.display.image()**.

مثال

```
brick.display.image(ImageFile.STOP_1)
```

قائمة ملف صور EV3 Brick	
STOP_1	ACCEPT
STOP_2	NO_GO
FORWARD	THUMBS DOWN
BACKWARD	QUESTION_MARK
LEFT	THUMBS_UP
RIGHT	DECLINE
TARGET	WARNING

STOP_1

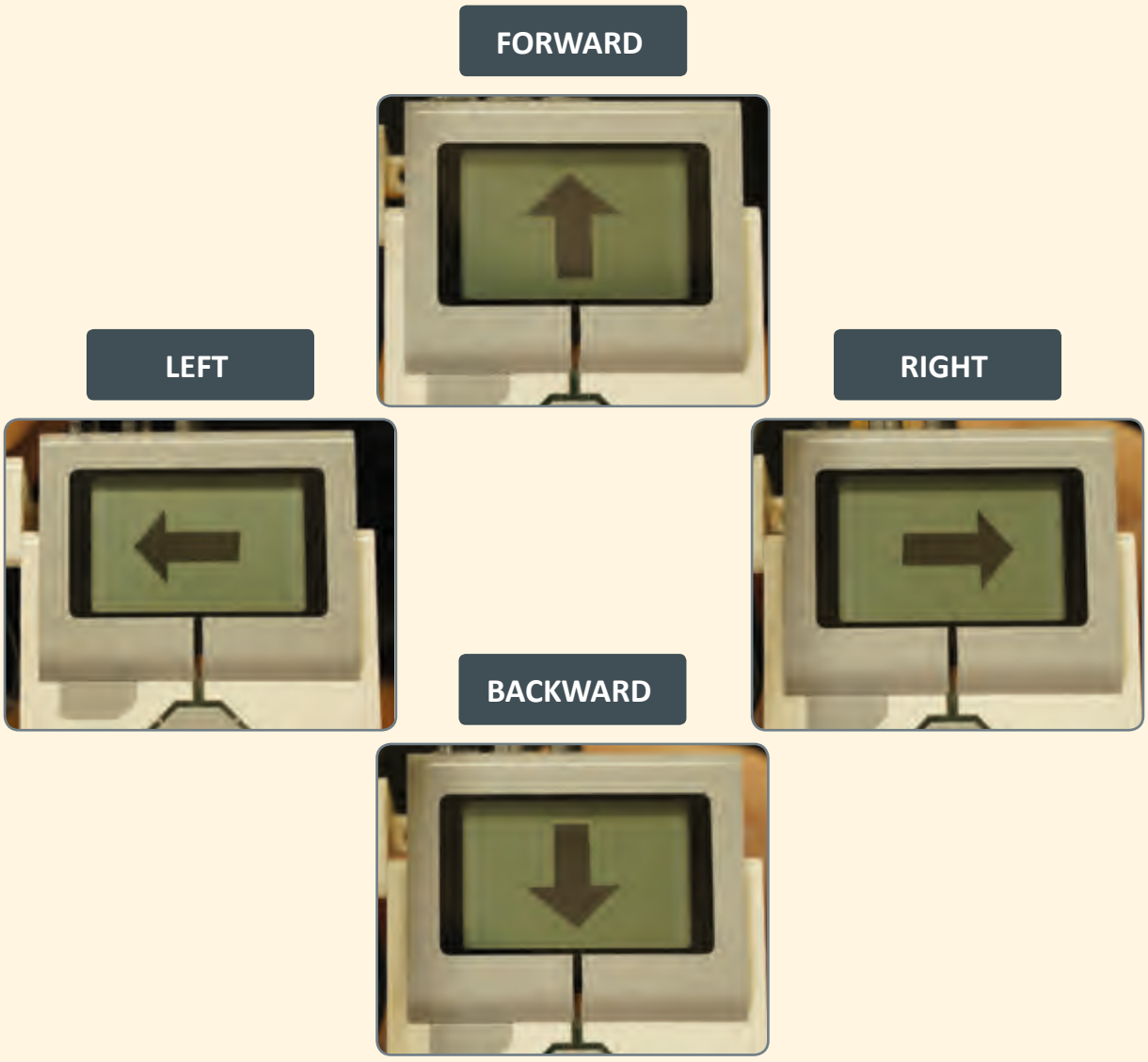
ACCEPT

TARGET

بعد ذلك، سنقوم بإنشاء دالة خاصة لاستخدام هذه الأوامر، ففي كل مرة يتم فيها الضغط على أحد أزرار الأسهم في **EV3 Brick** ستعرض الشاشة سهمًا مطابقًا بالاتجاه الذي ضغطنا عليه.

دالة عرض صورة عند الضغط على أحد الأزرار

```
def button_Direction():
    if button == Button.UP:
        brick.display.image(ImageFile.FORWARD)
    elif button == Button.DOWN:
        brick.display.image(ImageFile.BACKWARD)
    elif button == Button.LEFT:
        brick.display.image(ImageFile.LEFT)
    elif button == Button.RIGHT:
        brick.display.image(ImageFile.RIGHT)
```



دالة التحكم اليدوي بالذراع

سنقوم الآن بإنشاء دالة التحريك اليدوي للذراع، فمثلًا عند ضغط السهم في لبنة **EV3 Brick**، سينتقل الذراع الآلي إلى اتجاه السهم الذي نضغط عليه بسرعة **100** ولمدة ثانية واحدة. سيتم إيقاف طاقة المحرك وإجباره على التوقف باستخدام "**Stop. Coast**" في نفس الأمر، وستتحرك هذه الذراع بسلاسة نتيجة تنفيذ هذا الأمر.

دالة التحكم اليدوي

```
def manually_control():
    if Button.UP in brick.buttons():
        print("UP button")
        elbow_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.DOWN in brick.buttons():
        print("DOWN button")
        elbow_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.LEFT in brick.buttons():
        print("LEFT button")
        base_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.RIGHT in brick.buttons():
        print("RIGHT button")
        base_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
```

وهو جزء من الألف من الثانية

يقوم الأمر البرمجي **sleep**(0.5) بتأخير أي عملية لاحقة تلي الضغط على أزرار **EV3 Brick** لمدة نصف ثانية.

ستتم طباعة هذه الرسالة في الإخراج إذا ضغطنا على أزرار الاتجاه الأربعة على التوالي.

```
PROBLEMS  OUTPUT  D
Starting: brickrun
Started.
----------
DOWN button
LEFT button
RIGHT button
UP button
```

نصيحة ذكية

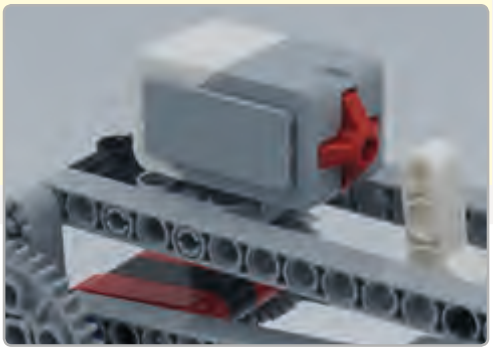
من المهم إضافة تأخير بين الخطوات، وإلا فقد يتوقف البرنامج عن العمل عند تلقي الكثير من الأوامر في نفس الوقت.

معايرة وضع الذراع الروبوتية

من أجل معايرة موضع ذراع الروبوت كما فعلنا مع القاعدة من قبل، سنقوم بإنشاء دالة أخرى. لإعداد الذراع، يجب أولًا التحرك لأسفل لمدة ثانية واحدة ثم التحرك ببطء للأعلى حتى يتم الضغط على مستشعر اللمس الثاني، وبعدها سيتوقف المحرك المتوسط محافظًا على وضعيته، مع إعادة ضبط الزاوية على "**0**".

دالة تهيئة الذراع

```
def calibrate_elbow():
    elbow_motor.run_time(30, 1000)
    elbow_motor.run(-15)
    while not touch_sensor_2.pressed():
        wait(10)
    elbow_motor.stop(Stop.HOLD)
    elbow_motor.reset_angle(0)
```



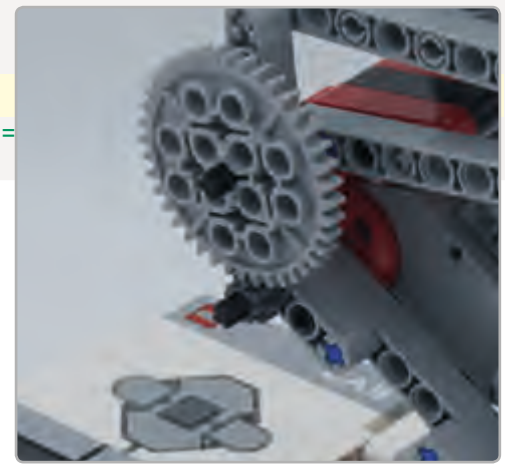
قمنا بإعداد المنافذ الخاصة بالمحرك ومستشعر اللمس الجديد، ولكن هذه المرة استخدمنا **4** تروس، اثنان منهما بـ **8** أسنان والآخران مع **40** سن.

تعيين المنفذ للمحرك الأوسط **B (Port B)**

إعداد المنفذ

```
# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
elbow_motor = Motor(Port.B, Direction.CLOCKWISE, [8, 40])

# Set sensors
touch_sensor = TouchSensor(Port.S1)
touch_sensor_2 = TouchSensor(Port.S3)
# ================ Set Ports ================
```

تعيين المنفذ **(Port 3)** لمستشعر اللمس الثاني



ضبط السرعة القصوى وتسارع المحرك

من أجل الحصول على حركة ذراع معقولة ومقبولة للروبوت، يجب تعيين السرعة القصوى وتسارع المحرك المتوسط.

ضبط التسارع

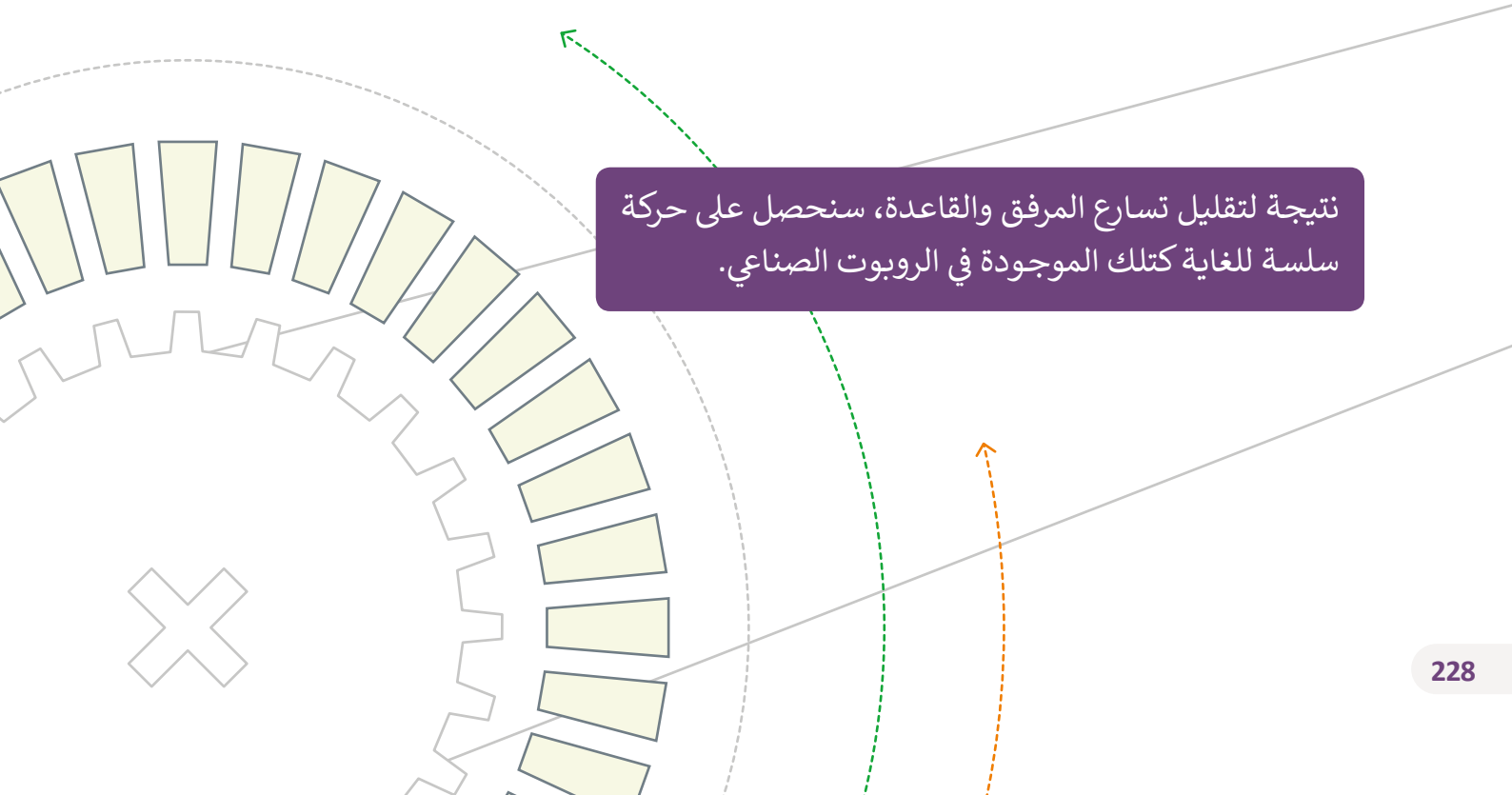
إعداد التسارع والسرعة القصوى/ تعريف المحرك لتشغيل جميع الأوامر.

```
base_motor.set_run_settings(max_speed, acceleration)
```

إعداد التسارع Acceleration

```
# Set accelerations
base_motor.set_run_settings(60, 120)
elbow_motor.set_run_settings(60, 120)
```

اكتب الحد الأقصى للسرعة وقيمة التسارع لمحركات تشغيل المرفق والقاعدة.



نتيجة لتقليل تسارع المرفق والقاعدة، سنحصل على حركة سلسة للغاية كتلك الموجودة في الروبوت الصناعي.

سنقوم الآن بكتابة التعليمات البرمجية الرئيسة الخاصة بالبرنامج. نحتاج أولاً إلى استدعاء دالة معايرة المرفق، ثم سنقوم بإنشاء تكرار يعرض صورة معينة حتى يتم الضغط على زر السهم في **Brick EV3**، وعند الضغط على أي زر فيها، سيتم تخزين قيمته (الزر الذي تم الضغط عليه).

تخزين القيمة

تخزين الزر الذي تم الضغط عليه

```
button = brick.buttons()[0]
```

في النهاية، سيتم تحريك الذراع الآلية وفقًا للزر المضغوط، وسيتم على **EV3 Brick** عرض سهم بالاتجاه المكافئ على الشاشة.

استدعاء الدالة **calibrate_elbow()**

عرض الصورة **STOP_1** على شاشة **EV3 Brick** حتى يتم الضغط على زر.

البرنامج الرئيس

```
# ================ main program ================ #

# Call Functions for calibration
calibrate_base()
calibrate_elbow()

while True:
    brick.display.image(ImageFile.STOP_1)

    if any(brick.buttons()):
        button = brick.buttons()[0]
        button_Direction()
        manually_control()
# ================ main program ================ #
```

قائمة لتخزين الأزرار التي تم الضغط عليها.

استدعاء الدالة **manually_control()**

استدعاء الدالة **button_direction()**

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase
from time import sleep

# Write your program here

# ================ Functions ================ #

def button_Direction():
    if button == Button.UP:
        brick.display.image(ImageFile.FORWARD)
    elif button == Button.DOWN:
        brick.display.image(ImageFile.BACKWARD)
    elif button == Button.LEFT:
        brick.display.image(ImageFile.LEFT)
    elif button == Button.RIGHT:
        brick.display.image(ImageFile.RIGHT)
```

```
def manually_control():
    if Button.UP in brick.buttons():
        print("UP button")
        elbow_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.DOWN in brick.buttons():
        print("DOWN button")
        elbow_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.LEFT in brick.buttons():
        print("LEFT button")
        base_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.RIGHT in brick.buttons():
        print("RIGHT button")
        base_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
```

```
def calibrate_elbow():
    elbow_motor.run_time(30, 1000)
    elbow_motor.run(-15)
    while not touch_sensor_2.pressed():
        wait(10)
    elbow_motor.stop(Stop.HOLD)
    elbow_motor.reset_angle(0)

def calibrate_base():
    base_motor.run(60)
    while not touch_sensor.pressed():
        wait(10)
    base_motor.stop(Stop.HOLD)
    base_motor.reset_angle(0)
# ================ Functions ================ #

# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
elbow_motor = Motor(Port.B, Direction.CLOCKWISE, [8, 40])

# Set sensors
touch_sensor = TouchSensor(Port.S1)
touch_sensor_2 = TouchSensor(Port.S3)
# ================ Set Ports ================ #

# Set accelerations
base_motor.set_run_settings(60, 120)
elbow_motor.set_run_settings(60, 120)

# ================ main program ================ #

# Call Functions for calibration
calibrate_base()
calibrate_elbow()

while True:
    brick.display.image(ImageFile.STOP_1)

    if any(brick.buttons()):
        button = brick.buttons()[0]
        button_Direction()
        manually_control()
# ================ main program ================ #
```

1

طابق صورة كل ترس مع اسمه.

Bevel gear, 12-tooth

Spur gear, 40-tooth

Double bevel gear, 36-tooth

Spur gear, 24-tooth

Spur gear, 8-tooth

Double bevel gear, 12-tooth

2

ما المقصود بالترس؟

_____

_____

3

ناقش العوامل المؤثرة على اختيار التروس في الآلات المختلفة، وذلك بالرجوع لمعادلة عزم الدوران.

4

في الدرس السابق تعلمنا تجميع قاعدة مرفق الروبوت وكان لدينا ترسين متصلين. ترس ذو 12 سن و ترس ذو 36 سن في الوجهة، كما يمكننا أن نرى في الصورة. لماذا قمنا بتوصيل الترس الصغير بدلاً من الترس الكبير بالمحرك؟

5

قم بتغيير أرقام السرعة القصوى وتسارع المحركات، واختبر البرنامج. قم بملاحظة ما تغير في كل محرك. ما هو السبب في التغييرات؟

```
# Set accelerations
base_motor.set_run_settings(60, 20)
elbow_motor.set_run_settings(20, 500)
```

6

في البرنامج الذي أنشأناه في هذا الدرس، قم بإزالة الأمر sleep (0.5) من دالة التحكم اليدوي ثم قم بتشغيل البرنامج وحرك ذراع الروبوت. ما الذي تغير؟ هل تعطل البرنامج في بعض الأحيان أيضًا؟

```
def manually_control():
    if Button.UP in brick.buttons():
        print("UP button")
        elbow_motor.run_time(-100, 1000, Stop.COAST)
    elif Button.DOWN in brick.buttons():
        print("DOWN button")
        elbow_motor.run_time(100, 1000, Stop.COAST)
    elif Button.LEFT in brick.buttons():
        print("LEFT button")
        base_motor.run_time(-100, 1000, Stop.COAST)
    elif Button.RIGHT in brick.buttons():
        print("RIGHT button")
        base_motor.run_time(100, 1000, Stop.COAST)
```

_____

_____

_____

7

استخدم أحد أسماء الصور في الجدول أدناه لتغيير الأيقونة الظاهرة على شاشة EV3 Brick خلال مرحلة الانتظار (حتى يتم الضغط على الزر).

قائمة ملف الصور في EV3 Brick	
STOP_1	ACCEPT
STOP_2	NO_GO
FORWARD	THUMBS DOWN
BACKWARD	QUESTION_MARK
LEFT	THUMBS_UP
RIGHT	DECLINE
TARGET	WARNING

الدرس الرابع

قابض الروبوت

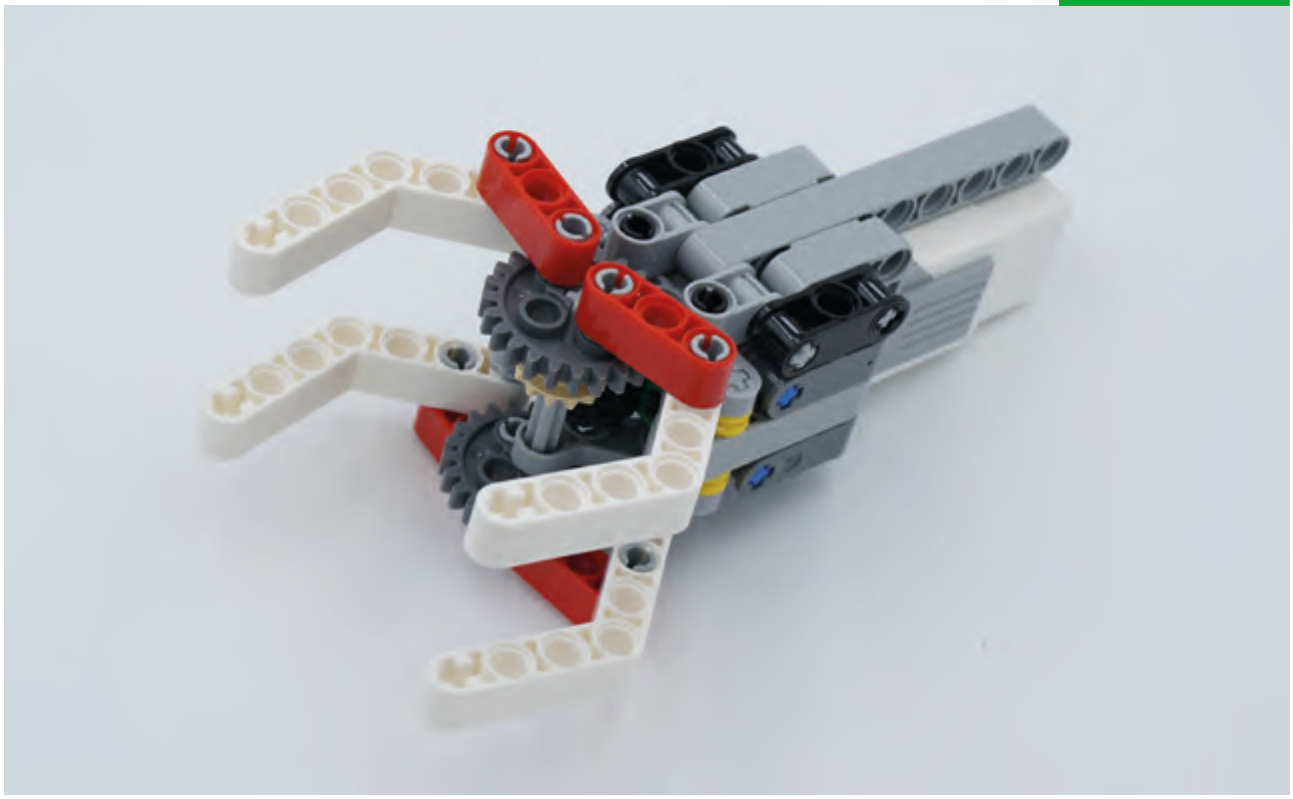
تجميع قابض ذراع الروبوت

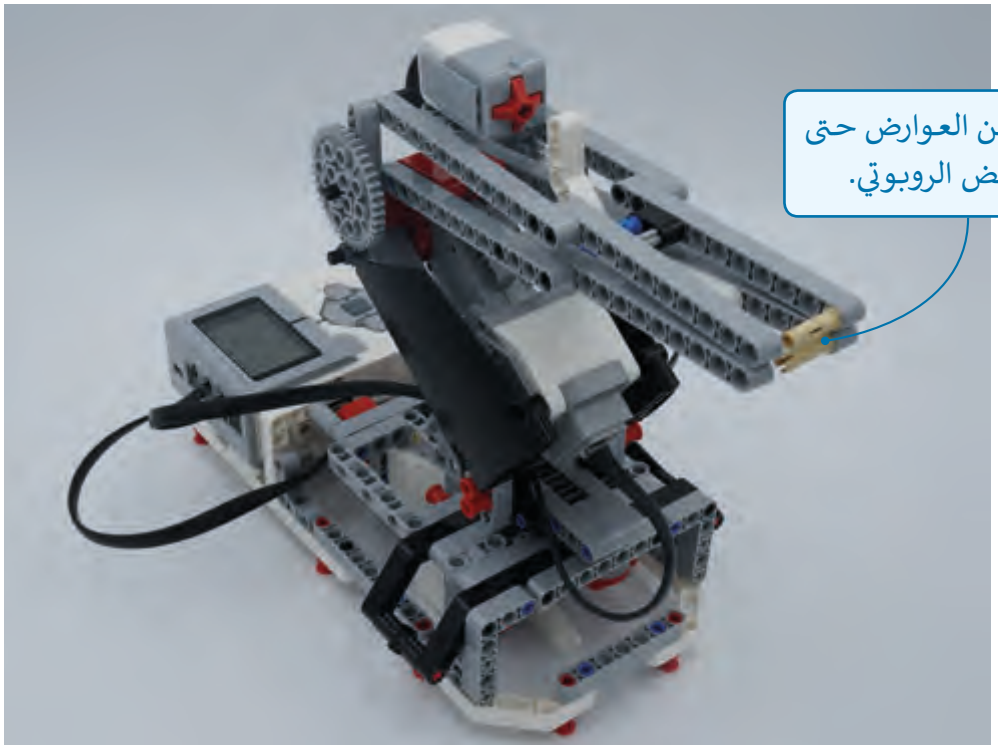
في هذا الدرس، سنقوم ببناء الجزء الأخير من ذراع الروبوت، وهو القابض. سننشئ أيضًا منصةً لقاعدة ذراع الروبوت لكي يتم إلحاق مستشعر اللون وذلك للكشف عن لون العلب المستعملة التي سيتم التقاطها.

بعد أن نقوم بتجميع الذراع الروبوتية، سنتناول بالتفصيل حركة كل مفصل من مفاصلها، وكيفية القيام بعملية المعايرة. هيا بنا لنستكمل عملنا.

بناء الجزء الميكانيكي من القابض الروبوتي.

القابض الروبوتي

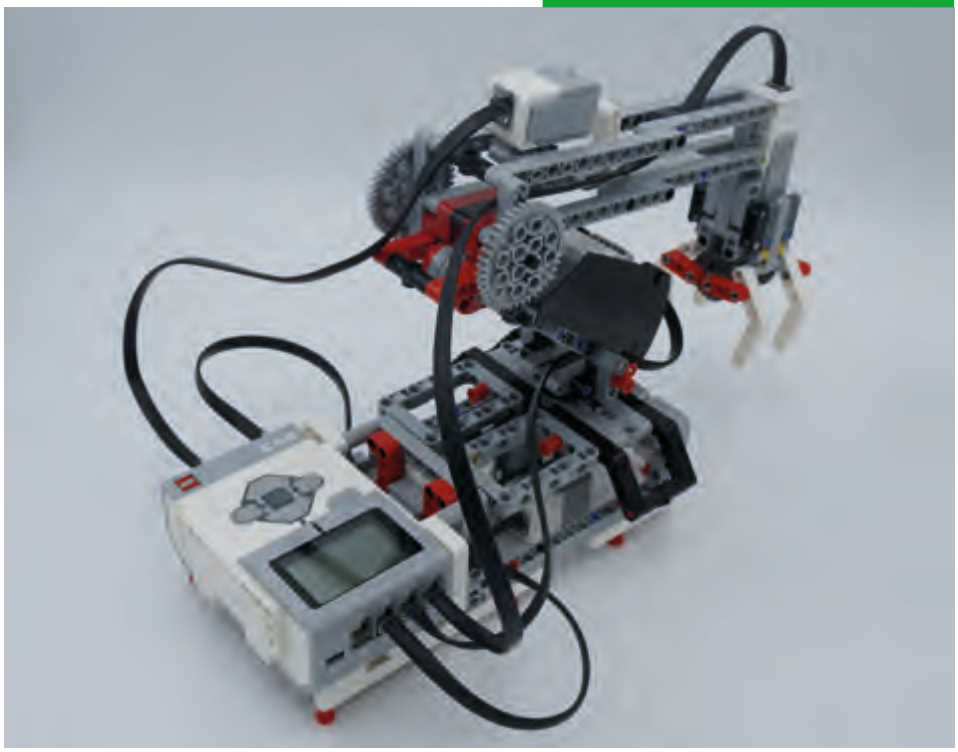




قم بإزالة قطع الربط من العوارض حتى نتمكن من تركيب القابض الروبوتي.

سنستخدم سلك بطول **50** سم لتوصيل المحرك المتوسط بالمنفذ **A**.

الشكل النهائي للقابض الروبوتي



قاعدة مستشعر اللون

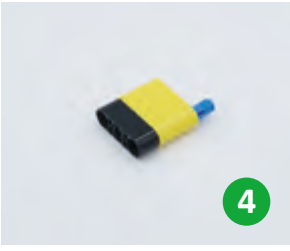
أخيرًا، سنحتاج إلى قاعدة لتركيب مستشعر اللون على جانب ذراع الروبوت.

هذه هي أجزاء **LEGO®** التي سنستخدمها لقاعدة مستشعر اللون.

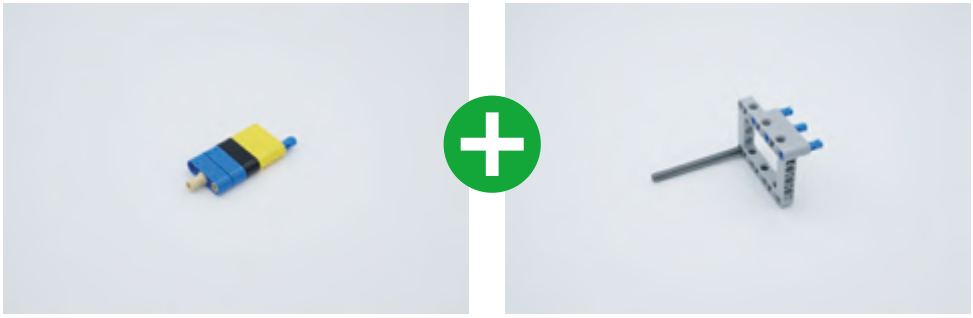



المرحلة الأولى

المرحلة الثانية

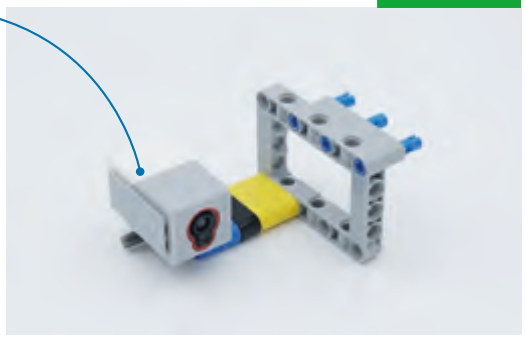


قم بجمع منتج المرحلة الأولى والمرحلة الثانية لإنشاء القاعدة.



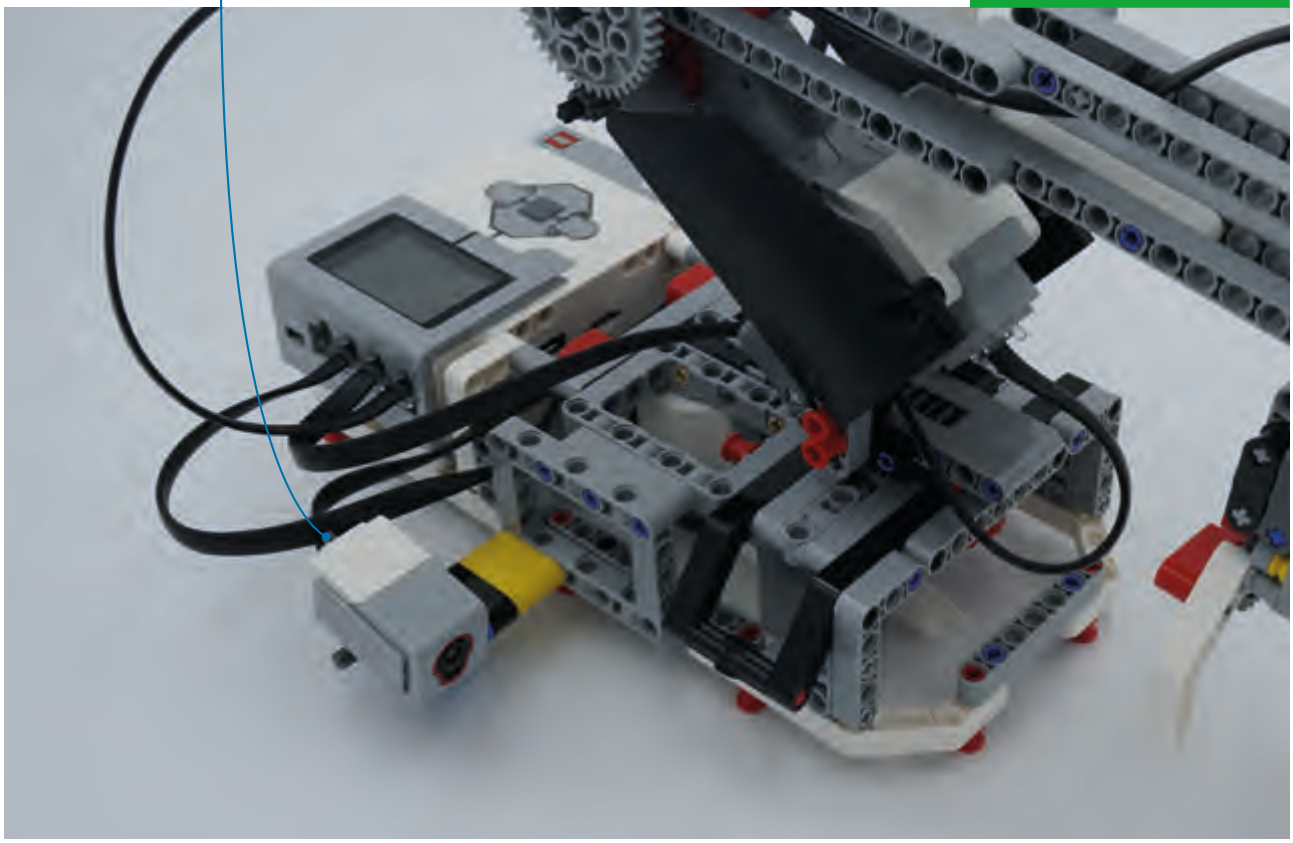
القاعدة

قم بتوصيل مستشعر اللون بقطعة ربط ومحور مزود بحاجز.



سنستخدم سلك بطول 25 سم لتوصيل المحرك المتوسط بالمنفذ 2.

قاعدة مستشعر اللون

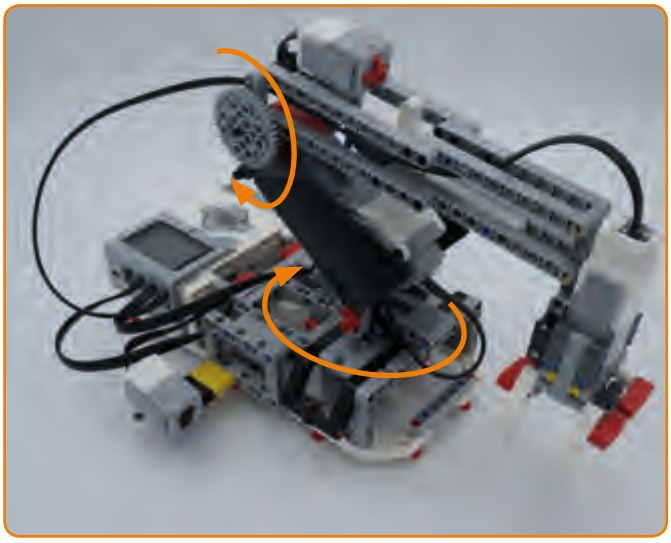


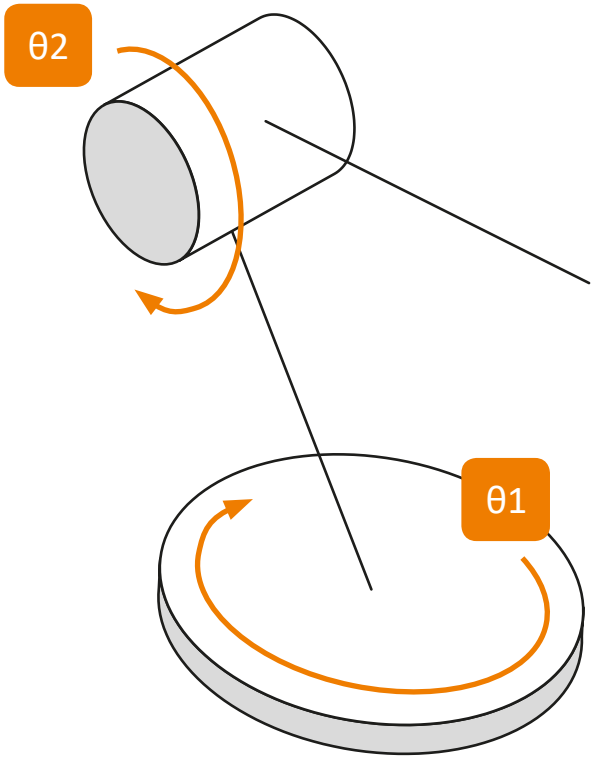
حركة الذراع الروبوتية

بعد أن انتهينا من تجميع الذراع الروبوتية، أصبحنا جاهزين لنتعلم كيفية تحريك كل مفصل من مفاصل الذراع الروبوتية. في الجدول أدناه، يمكننا رؤية الحركات الأساسية الخاصة بالأداء الحركي في معظم الروبوتات الصناعية.

الحركات الأساسية في الروبوتات الصناعية	
حركة دائرية **Rotational movement**	تتيح هذه الحركة للروبوت توجيه الذراع في أي اتجاه أفقي.
حركة قُطرية **Radial movement**	تُمكن هذه الحركة الروبوت من تحريك نهايته الفعالة بشكل قُطري للوصول إلى نقاط بعيدة.
حركة عمودية **Vertical movement**	تتيح للروبوت تحريك نهايته الفعالة إلى ارتفاعات مختلفة.

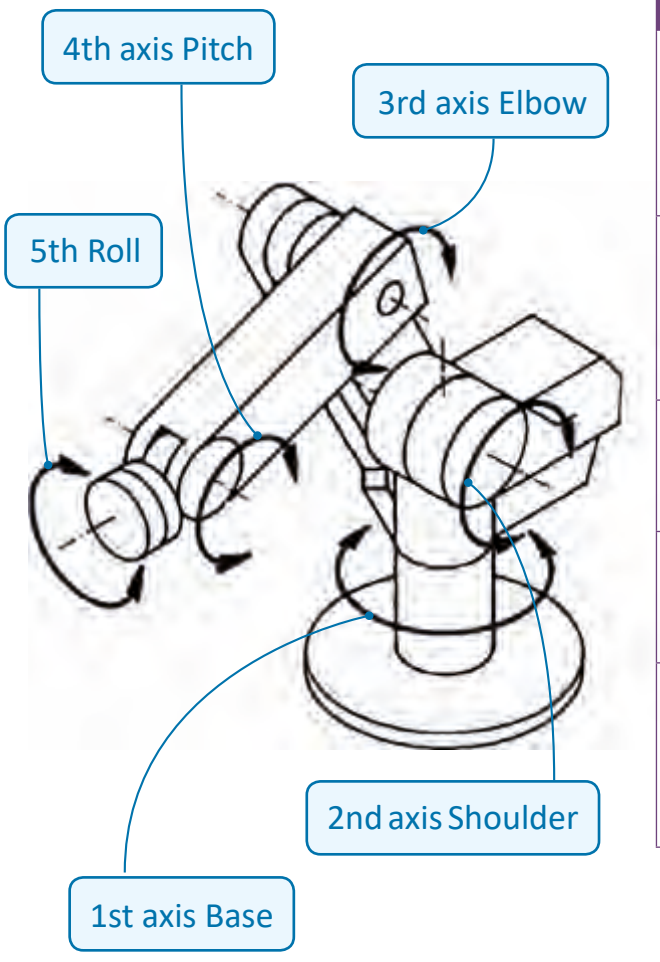
يُمكن للأذرع الروبوتية التحرك في مستوى **X** و **Y** و **Z**، ويمكنها أيضًا القيام بعملية الدوران حول محور ثيتا (**θ**) في نهاية مستوى **Z**، مما يمكنها من تدوير الأدوات الموجودة في نهاية الذراع.





يعتبر مصطلح درجات الحرية **Degree of Freedom (DoF)** مهمًا للغاية عند التعامل مع الأذرع الروبوتية، حيث أن كل درجة من درجات الحرية هي عبارة عن مفصل من مفاصل الذراع، وهي الموضع الذي تستطيع فيه الذراع أن تستدير أو تنحني فيه. يمكن تحديد عدد درجات الحرية حسب عدد المشغلات **(Actuators)** الموجودة في الذراع الروبوتية.

تمتلك أكثر الروبوتات شيوعًا ستة محاور، وعادةً ما يتم تثبيت هذه المحاور على قاعدة بحيث يمكنها التحرك للأمام والخلف، وللأعلى وللأسفل، كما يُمكنها تغيير اتجاهها أوالانحراف باتجاه معين أو التدحرج، وذلك لمنح تحكم أكبر في التوجيه. تعطي المحاور للروبوت القدرة على أداء الحركات التي تحاكي الذراع البشرية، فمثلًا يُمكن لذراع الروبوت التوجه للأسفل والتقاط شيء ما ووضعه على إحدى النواقل.



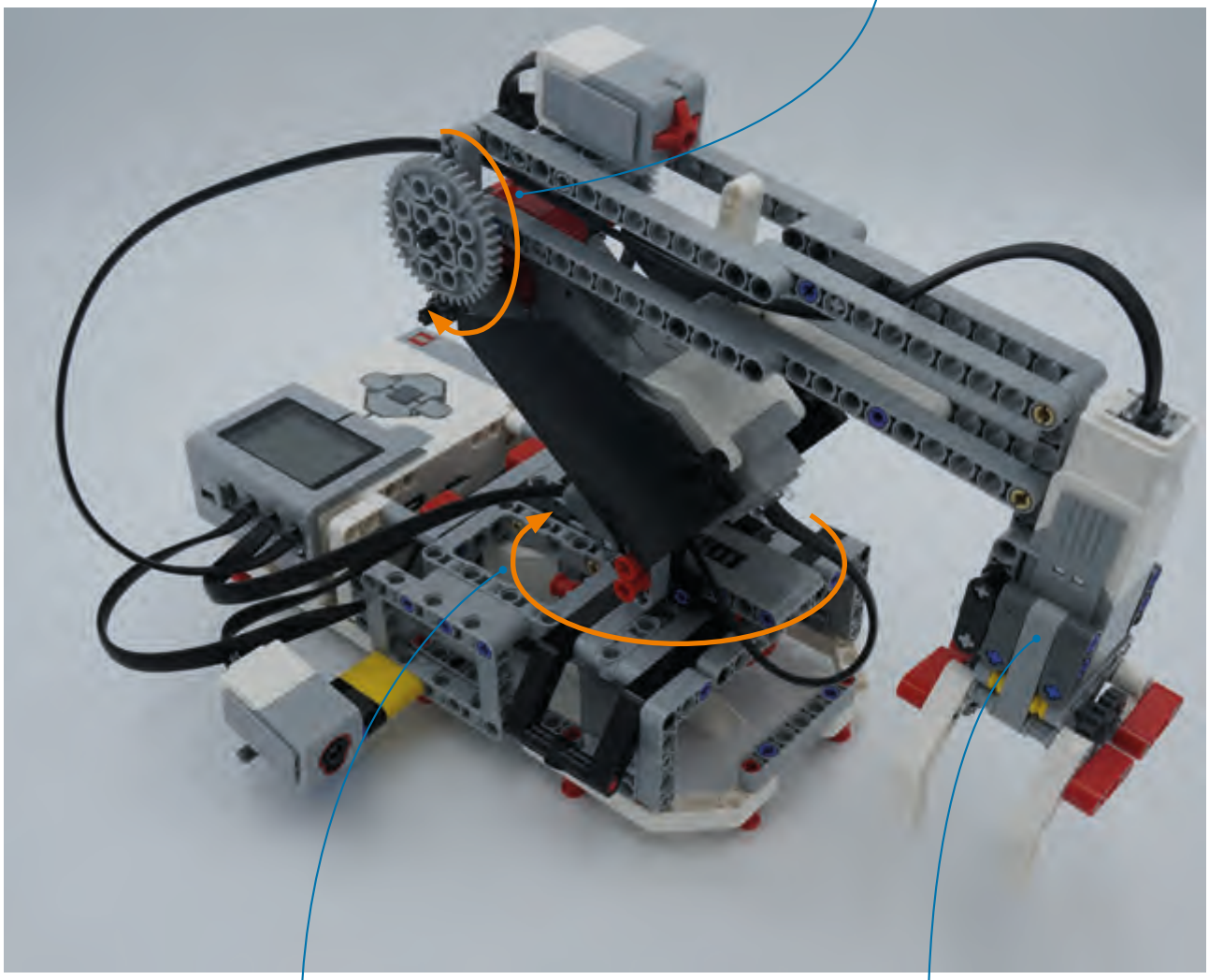
محاور حركة الذراع الروبوتية	
1st axis Base	يسمح باستدارة القاعدة بشكل أفقي.
2nd axis Shoulder	يُحرك جسم (جذع) الذراع إلى الأمام أو الخلف.
3rd axis Elbow	يحرك الذراع للأعلى أو الأسفل.
4th axis Pitch	يسمح باستدارة الذراع.
5th axis Roll	يسمح بتحرك الجزء العلوي من الذراع للأعلى أو للأسفل.

نصيحة ذكية

المشغل هو جزء من ألية مسؤول عن تحريكها والتحكم بنظامها.

تعتبر الذراع الروبوتية التي أنشأناها مُسبقًا "روبوت بدرجتي حرية" حيث يمكن لهذه الذراع التحرك في محورين كما هو موضح في الصورة أدناه:

يسمح هذا المحور لذراع الروبوت بالامتداد بشكل عمودي، مما يسمح للمرفق بالارتفاع والانخفاض.



يسمح المحور الموجود في قاعدة الذراع للروبوت بالاستدارة من اليسار لليمين.

يستخدم مقبض الروبوت في التقاط الأشياء، والتي ستكون عبارة عن العلب المعدنية المستعملة. بما أن عملية التقاط الأشياء لا تعتبر حركة، فلا يُمكن اعتبارها كمحور مستقل.

الآن وبعد تجميع الذراع الروبوتية، حان الوقت لبرمجتها. في البداية سنقوم بإعداد البرنامج الخاص بالقابض (**Gripper**)، وسيتم إنشاء أول دالة خاصة به وهي دالة المعايرة (**Calibration Function**).

لتهيئة القابض، يتم تشغيل المحرك للأمام لأقصى مسافة ممكنة حتى يتوقف، سيدور المحرك للخلف بزاوية **90** درجة من موضع القابض المغلق للذراع، وبالتالي سيفتح القابض وستعتبر هذه البداية نقطة الانطلاق.

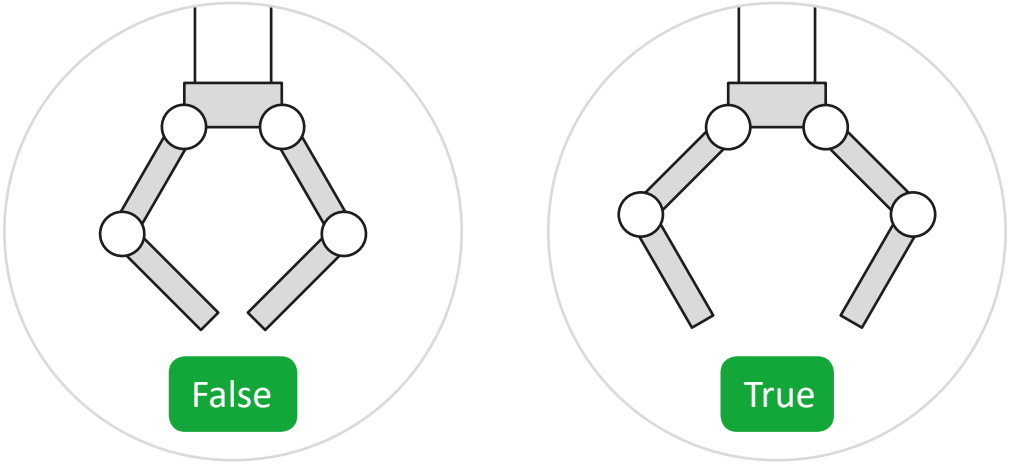
```
def calibrate_gripper():
    gripper_motor.run_until_stalled(200, Stop.COAST, 50)
    gripper_motor.reset_angle(0)
    gripper_motor.run_target(200, -90)
```

المتغير المنطقي Boolean

تعلمت سابقًا أنه يمكن للمتغير المنطقي أخذ قيمة من اثنتين: **True** أو **False** (صواب أو خطأ)، وفي **Python** يطلق على هذا النوع من المتغيرات اسم **bool**. سننشئ متغيرًا منطقيًا خاص بالقابض باسم **gripperVar** وسنمنحه قيمة الصواب **True**.

ضبط قيمة المتغير gripperVar إلى True

```
# ================ main program ================ #
# Set variable of Gripper to True
gripperVar = True
```



المتغيرات العامة Global Variables

عندما تقوم بتعريف المتغيرات ضمن دالة، فإنها لا ترتبط بأي شكل من الأشكال بالمتغيرات الأخرى التي لها نفس الأسماء المستخدمة خارج هذه الدالة، أي أن أسماء المتغيرات تكون محلية خاصة بالدالة. وهذا ما يسمى نطاق المتغير. كلمة **Global** هي كلمة أساسية تسمح لنا بتعديل متغير خارج النطاق الحالي. يتم استخدامها لإنشاء متغيرات عامة من نطاق محلي، أي نطاق ضمن دالة.

بعض المتغيرات يتم استخدامها في البرنامج الأساسي ولكنها أيضًا تُستخدم في دوال مختلفة. في هذه الحالات، يجب تعريف هذه المتغيرات على أنها عامة **Global**.

في برنامجنا نريد لمتغير **gripperVar** أن يكون عامًا، لذلك سنقوم بوضعه ضمن دالة وسنضع الكلمة **"Global"** قبل المتغير.

إعداد المتغير gripperVar ليكون عامًا

```
def manually_control():
    global gripperVar
```

متغير عام ضمن دالة.

تستخدم الكلمة المفتاحية **Global** (عام) في **Python** داخل دالة فقط عندما نريد القيام بمهام معينة، أو عندما نرغب بتغيير قيمة أحد المتغيرات.

قواعد المتغير العام
إذا تم تعيين قيمة للمتغير في أي مكان داخل الدالة، فسيتم اعتباره متغيرا محليًا ما لم يُعلن صراحةً أنه عام.
تعتبر المتغيرات الموجودة خارج الدوال متغيرات عامة.
نستخدم الكلمة المفتاحية **Global** لتعريف متغير عام داخل دالة.
ليست هناك حاجة لاستخدام كلمة **Global** خارج الدالة.

نصيحة ذكية

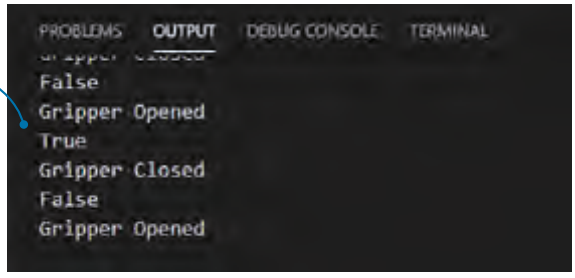
إذا تم تعيين قيمة جديدة لمتغير في أي مكان ضمن الدالة، فمن المفترض أن يكون هذا المتغير محلي.

للقيام بعملية التحكم اليدوي للقابض، سنستخدم المتغير الذي أنشأناه، وبواسطة الضغط على الزر الأوسط في **EV3 Brick** سيتم إغلاق القابض ومن ثم سيفتح وتتغير حالة المتغير من **True** إلى **False**.

إنشاء حالتين للقابض

```
def manually_control():
    global gripperVar
    if Button.UP in brick.buttons():
        print("UP button")
        elbow_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.DOWN in brick.buttons():
        print("DOWN button")
        elbow_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.LEFT in brick.buttons():
        print("LEFT button")
        base_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.RIGHT in brick.buttons():
        print("RIGHT button")
        base_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.CENTER in brick.buttons():
        if gripperVar == True:
            print(gripperVar)
            print("Gripper Closed")
            gripper_motor.run_until_stalled(200, Stop.COAST, 50)
            gripper_motor.reset_angle(0)
            sleep(0.5)
            gripperVar = False
        elif gripperVar == False:
            print(gripperVar)
            print("Gripper Opened")
            gripper_motor.run_target(200, -90)
            sleep(0.5)
            gripperVar = True
```

الرسائل التي ستظهر عند الضغط على الزر الأوسط.



ثم سنقوم بتعيين المنافذ للمحرك القابض ومستشعر اللون.

تعيين المنافذ

```
# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
elbow_motor = Motor(Port.B, Direction.CLOCKWISE, [8, 40])
gripper_motor = Motor(Port.A)

# Set sensors
touch_sensor = TouchSensor(Port.S1)
touch_sensor_2 = TouchSensor(Port.S3)
color_sensor = ColorSensor(Port.S2)
# ================ Set Ports ================ #
```

تعيين المنفذ port A للمحرك الأوسط.

تعيين المنفذ port 2 لمستشعر اللون.

برمجة مستشعر اللون

لكي يقوم مستشعر اللون بالتعرف على الألوان، يجب أن نقوم في البداية بإعداد الوضع أو الحالة **Mode** لذلك المستشعر، والذي يتطابق تمامًا مع خيارات المستشعر في **EV3 Mindstorms**.

حالات مستشعر اللون	
COL-COLOR	اللون المكتشف بواسطة المستشعر مصنفًا حسب القيمة الإجمالية.
COL-REFLECT	الضوء المنعكس كنسبة مئوية، والضوء على المستشعر باللون الأحمر.
COL-AMBIENT	شدة الضوء المحيط، الضوء على المستشعر باللون الأزرق الباهت.
RGB-RAW	لون مكتشف من اللون الأحمر والأخضر والأزرق في نطاق **0 – 1020**.

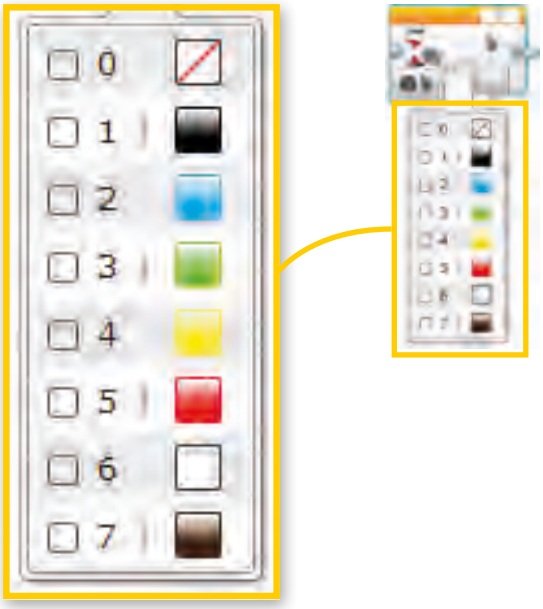
ضبط حالة مستشعر اللون

```
# Set accelerations
base_motor.set_run_settings(60, 120)
elbow_motor.set_run_settings(60, 120)

# Set color mode
color_sensor.mode = 'COL - COLOR'
```

ضبط حالة مستشعر اللون إلى **'COL - COLOR'**

يحتوي وضع **COL-COLOR** على **8** قيم تمثل **7** ألوان مختلفة، أما القيمة **0** فلا تمثل أي لون، وهذا مماثل لما نقوم به في بيئة **EV3 Mindstorms** إذا أردنا ضبط اللون الذي سيقوم المستشعر باكتشافه.

قيم الألوان في حالة COL-COLOR	
بدون لون	**0**
أسود	**1**
أزرق	**2**
أخضر	**3**
أصفر	**4**
أحمر	**5**
أبيض	**6**
بُني	**7**



في المثال التالي سننشئ برنامجًا باستخدام الوضع **COL-COLOR** لاكتشاف **4** ألوان مختلفة وطباعة اسم اللون المكتشف.

مثال 1

```
# Write your program here

color_sensor = ColorSensor(Port.S3)

color_sensor.mode = 'COL - COLOR'

while True:
    colorVar = color_sensor.color()
    if colorVar == 1:
        print("Black")
    elif colorVar == 3:
        print("Green")
    elif colorVar == 4:
        print("Yellow")
    elif colorVar == 5:
        print("Red")
```

تمرير رقم اللون المكتشف إلى المتغير **colorVar**.

جرب تمرير بطاقات بألوان مختلفة أمام مستشعر اللون ولاحظ المطبوع على الشاشة بعد تمرير كل بطاقة.

لنعد إلى الذراع الروبوتية، ولننشيء الدالة الخاصة بمستشعر اللون:

حالات اكتشاف الألوان

```
def check_color(colorVar):
    # Color of the material is Black
    if colorVar == 1:

    # Color of the material is Green
    elif colorVar == 3:

    # Color of the material is Yellow
    elif colorVar == 4:

    # Color of the material is Red
    elif colorVar == 5:
```

سنستكمل هذه الدالة بعد تعيين دوال الالتقاط والإفلات.

في السيناريو التالي، نريد من مستشعر اللون أن يكتشف لون العلبة المعدنية المستعملة وذلك بالقيام بالخطوات التالية:

> استخدام قاموس **Dictionary** لطباعة نوع المادة التي يتم التقاطها.

> تحريك الذراع الروبوتية إلى موقع الالتقاط.

> الانتقال إلى المنطقة المحددة لإفلات العلبة المستعملة.

> استخدام قاموس لطباعة نوع المواد التي تم نقلها.

في البداية نحتاج إلى إنشاء قاموس خاص بالمواد.

إنشاء القاموس

```
# ================ main program ================ #
# Set variable of Gripper to True
gripperVar = True

# Dictionary
Materials = {"Black":"Metal", "Green":"Glass", "Yellow":"Plastic",
"Red":"Paper"}
```

المفتاح للمواد المعدنية **Metal** هو الأسود **Black**

سنقوم الآن بتحديد **6** وجهات مختلفة نستخدمها في المقطع البرمجي للحصول على الأماكن التي يتم إفلات العلب فيها.

تعيين 6 وجهات للإفلات

```
# ================== Define the 6 destinations ==================

# One for the center.
CENTER = -120
# One for picking up.
PICK = 0

# Four locations for dropping the Recyclable products.
BLACK = -55
GREEN = -125
YELLOW = -185
RED = -240
# ================== Define the 6 destinations ==================
```

سيتم استخدام جميع هذه الوجهات مع عملية **run_target**. يعمل المحرك بواسطة هذه العملية بسرعة ثابتة وباتجاه زاوية مستهدفة معينة. سيتسارع المحرك بالسرعة المطلوبة مع ضبط دورة التشغيل تلقائيًا للحفاظ على سرعة ثابتة، وحتى في وجود بعض الحمل فإنه سيبدأ في التباطؤ في الوقت المناسب وصولًا للتوقف عند زاوية الهدف المحددة.

```
run_target(speed, rotation_angle, stop_type=Stop.COAST, wait=True)
```

لنستعرض مثالًا عن كيفية استخدامنا لهذه العملية.

مثال 2

```
base_motor.run_target(60, PICK, Stop.HOLD)
```

بواسطة هذا السطر البرمجي، ستنتقل القاعدة إلى الموضع المحدد الذي نختاره، وستكون عملية الالتقاط PICK بسرعة 60 ثم ستتوقف.

من أجل تجنب التكرار وتبسيط المقطع البرمجي، سننشئ دالتين، واحدة لجمع العلب المستعملة والأخرى لإفلاتها. هذه الدالة تدير القاعدة إلى الموضع المطلوب، وهناك ستقوم بخفض الذراع، وتقوم بإغلاق القابض، ثم يتم رفع الذراع لالتقاط العلب المعدنية المستعملة.

إعداد دالة الالتقاط

```
def pick_position():
    base_motor.run_target(60, PICK, Stop.HOLD)
    elbow_motor.run_time(30, 2500)
    elbow_motor.run(-15)
    gripper_motor.run_until_stalled(200, Stop.COAST, 50)
    elbow_motor.run_time(-30, 2500)
```

لننتقل الآن إلى دالة الإفلات، حيث تقوم هذه الدالة بتدوير القاعدة وصولًا إلى موضع الإفلات، وهناك تقوم بخفض الذراع ، ثم بفتح القابض لإفلات العلبة المستعملة، وأخيرًا رفع الذراع مرة أخرى.

إعداد دالة الإفلات

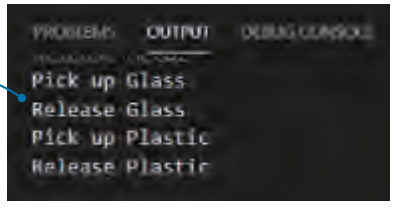
```
def release_position():
    elbow_motor.run_time(30, 2500)
    elbow_motor.run(-15)
    gripper_motor.reset_angle(0)
    gripper_motor.run_target(200, -90)
    elbow_motor.run_time(-30, 2000)
```

لنستكمل الآن دالة مستشعر اللون، لتحديد موقع الإفلات حسب اللون الذي يلتقطه المستشعر.

دالة مستشعر اللون

```
def check_color(colorVar):
    # Color of the material is Black
    if colorVar == 1:
        print("Pick up " + Materials.get('Black'))
        pick_position()
        base_motor.run_target(60, BLACK, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Black'))
    # Color of the material is Green
    elif colorVar == 3:
        print("Pick up " + Materials.get('Green'))
        pick_position()
        base_motor.run_target(60, GREEN, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Green'))
    # Color of the material is Yellow
    elif colorVar == 4:
        print("Pick up " + Materials.get('Yellow'))
        pick_position()
        base_motor.run_target(60, YELLOW, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Yellow'))
    # Color of the material is Red
    elif colorVar == 5:
        print("Pick up " + Materials.get('Red'))
        pick_position()
        base_motor.run_target(60, RED, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Red'))
```

سيتم طباعة المادة المصنوعة منها العلبة المستعملة عند التقاطها وكذلك عندما يتم إفلاتها.



بالإضافة إلى المادة المصنوعة منها العلب المستعملة، نريد أيضًا طباعة لون هذه المادة، لذلك سنقوم بإنشاء مصفوفة تضم الألوان، وطباعة اللون من هناك.

مصفوفة الألوان

```
# ================ main program ================ #
# Set variable of Gripper to True
gripperVar = True

# Dictionary
Materials = {"Black":"Metal", "Green":"Glass", "Yellow":"Plastic",
"Red":"Paper"}
# Table with EV3 colors
colors = ["No color ", "Black", "Blue", "Green" , "Yellow", "Red",
"White", "Brown"]

print(Materials.values())
```

تذكر أن المصفوفة تبدأ دائمًا بالعد من رقم "0"، لذلك يجب أن نضيف كافة الألوان الموجودة في الجدول التالي.

0	1	2	3	4	5	6	7
بلا لون	أسود	أزرق	أخضر	أصفر	أحمر	أبيض	بُني

سنستمر الآن في كتابة المقطع البرمجي داخل تكرار، بحيث أنه عند تشغيل المقطع وعدم الضغط على أي زر أو عدم اكتشاف أي من الألوان المعرفة فإننا نريد عرض "**STOP_1**" على شاشة **EV3**.

عرض الصورة

```
while True:
    # Display image 'STOP_1'
    brick.display.image(ImageFile.STOP_1)
```

سنقوم لاحقًا بإنشاء حالة **elif** تحت حالة **if**، بحيث ستكون جملة **elif** عبارة عن أي حالة يتم فيها الكشف عن أي لون وذلك لتنفيذ الحالة المقابلة، ثم تخزين قيمة اللون ليتم طباعتها.

حالة اكتشاف اللون

```
if any(brick.buttons()):
    # Store which button was pressed
    button = brick.buttons()[0]
    button_Direction()
    manually_control()

elif color_sensor.color() != None:
    # Set color variable
    colorVar = color_sensor.color()
    # Print Color of the material
    print(colors[colorVar])
    # Display image 'ACCEPT'
    brick.display.image(ImageFile.ACCEPT)
    # Call Functions for calibration
    calibrate_base()
    calibrate_elbow()
    calibrate_gripper()
```

وفقًا لقيمة اللون سيتم طباعة اللون المكافئ.

سنقوم بإزالة المعايرة الروبوتية بشكل كامل داخل حالة مستشعر اللون، وهذا لأننا نريد أن يتجه الذراع الروبوتي إلى موقع محدد فقط عند اكتشافه للون ما.

لإنهاء عملية معايرة الذراع الروبوتية، سنقوم بإرسالها إلى المركز **CENTER**.

```
    # Position Robot Arm at the Center
    base_motor.run_target(60, CENTER, Stop.HOLD)

    # Display image 'TARGET'
    brick.display.image(ImageFile.TARGET)
    check_color(colorVar)
```

تعيين موضع الذراع الروبوتية إلى المركز قبل البدء بالتحرك لموضع الالتقاط.

البرنامج الرئيس

```
# ================ main program ================ #
# Set variable of Gripper to True
gripperVar = True

# Dictionary
Materials = {"Black":"Metal", "Green":"Glass", "Yellow":"Plastic",
"Red":"Paper"}
# Table with EV3 colors
colors = ["No color ", "Black", "Blue", "Green" , "Yellow", "Red",
"White", "Brown"]

print(Materials.values())

while True:

    # Display image 'STOP_1'
    brick.display.image(ImageFile.STOP_1)

    if any(brick.buttons()):
        # Store which button was pressed
        button = brick.buttons()[0]
        button_Direction()
        manually_control()

    elif color_sensor.color() != None:
        # Set color variable
        colorVar = color_sensor.color()
        # Print Color of the material
        print(colors[colorVar])
        # Display image 'ACCEPT'
        brick.display.image(ImageFile.ACCEPT)
        # Call Functions for calibration
        calibrate_base()
        calibrate_elbow()
        calibrate_gripper()

        # Position Robotic Arm at the Center
        base_motor.run_target(60, CENTER, Stop.HOLD)

        # Display image 'TARGET'
        brick.display.image(ImageFile.TARGET)
        check_color(colorVar)
# ================ main program ================ #
```

الآن وبعد اكتمال كافة جزئيات البرنامج، لنلق نظرة عامة على المقطع البرمجي المتكامل للذراع الروبوتية:

المقطع النهائي

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor,
                                 GyroSensor)
from pybricks.parameters import (Port, Stop, Direction, Button,
                                 Color, SoundFile, ImageFile, Align)
from pybricks.tools import print, wait, StopWatch
from pybricks.robotics import DriveBase
from time import sleep

# Write your program here

# ================ Functions ================ #

def pick_position():
    base_motor.run_target(60, PICK, Stop.HOLD)
    elbow_motor.run_time(30, 2500)
    elbow_motor.run(-15)
    gripper_motor.run_until_stalled(200, Stop.COAST, 50)
    elbow_motor.run_time(-30, 2500)

def release_position():
    elbow_motor.run_time(30, 2500)
    elbow_motor.run(-15)
    gripper_motor.reset_angle(0)
    gripper_motor.run_target(200, -90)
    elbow_motor.run_time(-30, 2000)
```

```
def check_color(colorVar):
    # Color of the material is Black
    if colorVar == 1:
        print("Pick up " + Materials.get('Black'))
        pick_position()
        base_motor.run_target(60, BLACK, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Black'))
    # Color of the material is Green
    elif colorVar == 3:
        print("Pick up " + Materials.get('Green'))
        pick_position()
        base_motor.run_target(60, GREEN, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Green'))
    # Color of the material is Yellow
    elif colorVar == 4:
        print("Pick up " + Materials.get('Yellow'))
        pick_position()
        base_motor.run_target(60, YELLOW, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Yellow'))
    # Color of the material is Red
    elif colorVar == 5:
        print("Pick up " + Materials.get('Red'))
        pick_position()
        base_motor.run_target(60, RED, Stop.HOLD)
        release_position()
        print("Release " + Materials.get('Red'))
```

```
def button_Direction():
    if button == Button.UP:
        brick.display.image(ImageFile.FORWARD)
    elif button == Button.DOWN:
        brick.display.image(ImageFile.BACKWARD)
    elif button == Button.LEFT:
        brick.display.image(ImageFile.LEFT)
    elif button == Button.RIGHT:
        brick.display.image(ImageFile.RIGHT)
```

```
def manually_control():
    global gripperVar
    if Button.UP in brick.buttons():
        print("UP button")
        elbow_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.DOWN in brick.buttons():
        print("DOWN button")
        elbow_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.LEFT in brick.buttons():
        print("LEFT button")
        base_motor.run_time(-100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.RIGHT in brick.buttons():
        print("RIGHT button")
        base_motor.run_time(100, 1000, Stop.COAST)
        sleep(0.5)
    elif Button.CENTER in brick.buttons():
        if gripperVar == True:
            print(gripperVar)
            print("Gripper Closed")
            gripper_motor.run_until_stalled(200, Stop.COAST, 50)
            gripper_motor.reset_angle(0)
            sleep(0.5)
            gripperVar = False
        elif gripperVar == False:
            print(gripperVar)
            print("Gripper Opened")
            gripper_motor.run_target(200, -90)
            sleep(0.5)
            gripperVar = True

def calibrate_base():
    base_motor.run(60)
    while not touch_sensor.pressed():
        wait(10)
    base_motor.stop(Stop.HOLD)
    base_motor.reset_angle(0)

def calibrate_elbow():
    elbow_motor.run_time(30, 1000)
    elbow_motor.run(-15)
    while not touch_sensor_2.pressed():
        wait(10)
    elbow_motor.stop(Stop.HOLD)
    elbow_motor.reset_angle(0)
```

```
def calibrate_gripper():
    gripper_motor.run_until_stalled(200, Stop.COAST, 50)
    gripper_motor.reset_angle(0)
    gripper_motor.run_target(200, -90)

# ================ Functions ================ #

# ================ Set Ports ================ #
# Set motors
base_motor = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
elbow_motor = Motor(Port.B, Direction.CLOCKWISE, [8, 40])
gripper_motor = Motor(Port.A)

# Set sensors
touch_sensor = TouchSensor(Port.S1)
touch_sensor_2 = TouchSensor(Port.S3)
color_sensor = ColorSensor(Port.S2)
# ================ Set Ports ================ #

# Set accelerations
base_motor.set_run_settings(60, 120)
elbow_motor.set_run_settings(60, 120)

# Set color mode
color_sensor.mode = 'COL - COLOR'

# ================== Define the 6 destinations ==================
# One for the center.
CENTER = -120
# One for picking up.
PICK = 0

# Four locations for dropping the Recyclable products.
BLACK = -55
GREEN = -125
YELLOW = -185
RED = -240
# ================== Define the 6 destinations ==================
```

```
# ================ main program ================ #
# Set variable of Gripper to True
gripperVar = True

# Dictionary
Materials = {"Black":"Metal", "Green":"Glass", "Yellow":"Plastic",
"Red":"Paper"}
# Table with EV3 colors
colors = ["No color ", "Black", "Blue", "Green" , "Yellow", "Red",
"White", "Brown"]

print(Materials.values())

while True:
    # Display image 'STOP_1'
    brick.display.image(ImageFile.STOP_1)

    if any(brick.buttons()):
        # Store which button was pressed
        button = brick.buttons()[0]
        button_Direction()
        manually_control()

    elif color_sensor.color() != None:
        # Set color variable
        colorVar = color_sensor.color()
        # Print Color of the material
        print(colors[colorVar])
        # Display image 'ACCEPT'
        brick.display.image(ImageFile.ACCEPT)
        # Call Functions for calibration
        calibrate_base()
        calibrate_elbow()
        calibrate_gripper()

        # Position Robotic Arm at the Center
        base_motor.run_target(60, CENTER, Stop.HOLD)

        # Display image 'TARGET'
        brick.display.image(ImageFile.TARGET)
        check_color(colorVar)
# ================ main program ================ #
```

1

ما المقصود بدرجة الحرية للذراع الروبوتية (DOF)؟ قم بتحديد عدد درجات الحرية للذراع الروبوتية التي أنشأناها في هذه الوحدة ثم قم بتفسير إجابتك؟

2

ميز بين المتغير المحلي والمتغير العام، ووضح متى نحتاج لتعريف المتغير كمتغير عام Global؟

3

بالنسبة للبرنامج الذي أنشأناه في هذا الدرس، قم بتغيير الوجهات وفقًا للصورة أدناه ثم قم بتشغيل البرنامج لمعرفة ما سيفعله الروبوت عند اكتشافه لأحد الألوان وسجل ملاحظاتك بهذا الخصوص.

```
# ================== Define the 6 destinations ==================

# One for the center.
CENTER = -100
# One for picking up.
PICK = -240

# Four locations for dropping the Recyclable products.
BLACK = -55
GREEN = -125
YELLOW = -185
RED = 0
# ================== Define the 6 destinations ==================
```

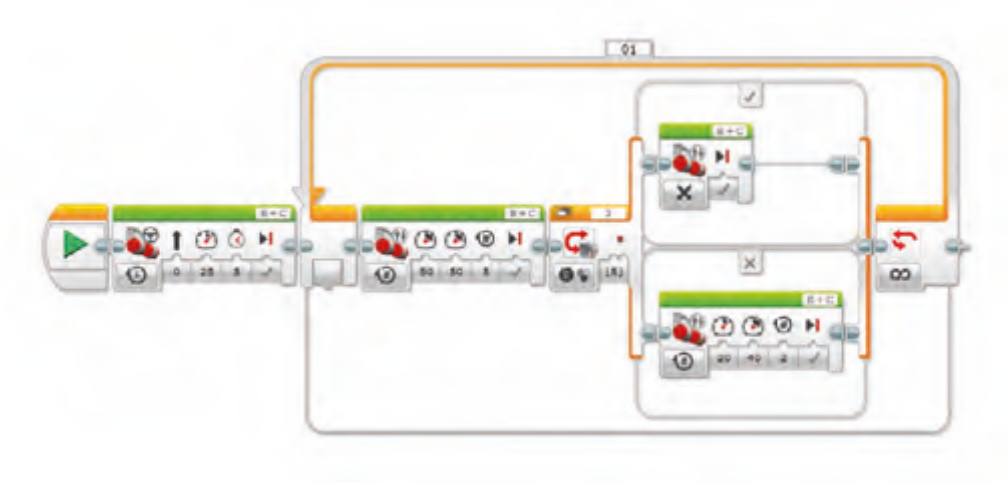
4

بالنسبة للبرنامج الذي أنشأناه في هذا الدرس، قم بتغيير قيم سرعة المحركات في الدالتين أدناه، ثم قم بتشغيل البرنامج لمعاينة النتيجة. وسجل ملاحظاتك بهذا الخصوص.

```
# ================ Functions ================ #

def pick_position():
    base_motor.run_target(120, PICK, Stop.HOLD)
    elbow_motor.run_time(60, 1250)
    elbow_motor.run(-15)
    gripper_motor.run_until_stalled(200, Stop.COAST, 50)
    elbow_motor.run_time(-60, 1250)

def release_position():
    elbow_motor.run_time(60, 1250)
    elbow_motor.run(-15)
    gripper_motor.reset_angle(0)
    gripper_motor.run_target(200, -90)
    elbow_motor.run_time(-60,, 1000)
```

LEGO® MINDSTORMS® EV3

LEGO® MINDSTORMS® هو نظام للأجهزة والبرامج لتطوير روبوتات قابلة للبرمجة تعتمد على عناصر بناء **LEGO®**. وبيئة البرمجة **LabView**.



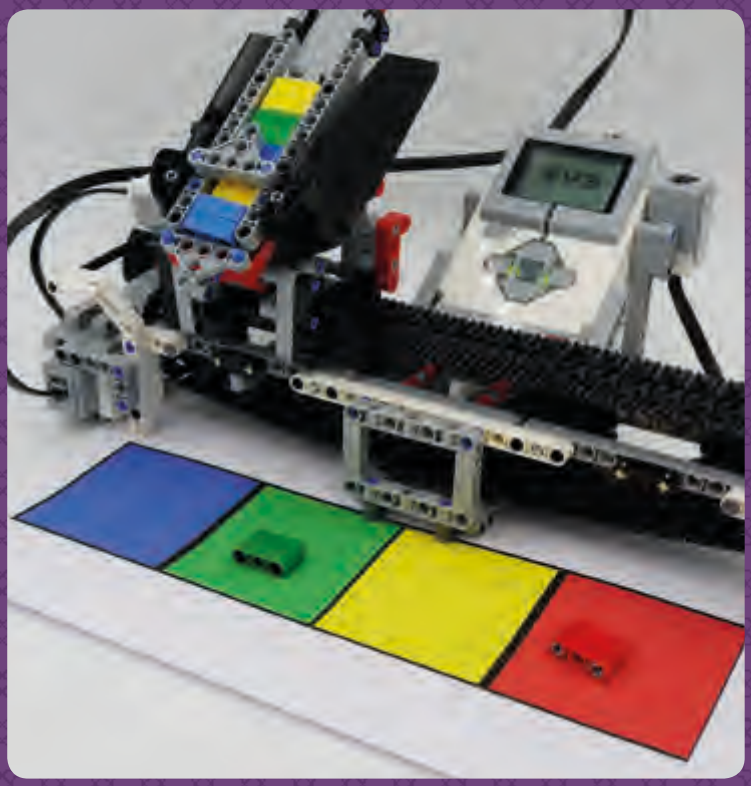
برنامج LEGO® MINDSTORMS® EV3 و Microsot MakeCode

يمكننا برمجة **LEGO® MINDSTORMS® Education EV3** من خلال **Microsoft MakeCode** الذي يوفر برامج تحرير النصوص واللبنات. من خلال جهاز المحاكاة المضمّن، يمكننا الحصول على تعليقات فورية حول كيفية تشغيل برنامجنا وتصحيح الرمز قبل تطبيقه على **EV3 Brick**. عندما نصبح جاهزين، يمكننا الانتقال من اللبنات إلى **JavaScript** باستخدام مقتطفات من التعليمات البرمجية واكتشاف الأخطاء.

مشروع الوحدة

العنوان: آلات الفرز

الوصف: قم ببناء واستخدام **MicroPython** لبرمجة آلة يمكنها تصنيف قطع الليجو بحسب ألوانها (مستشعر اللون **Color Sensor**) وترتيبها في مواقع مختلفة.



الأدوات: **MicroPython**

خطوات التنفيذ: اتبع الخطوات لإنشاء وبرمجة آلة الفرز.

> قم ببناء آلة الفرز واستخدم المسارات لبناء قاعدة متحركة لتركيب منحدر.

> قم بمعايرة موضع المنحدر على القاعدة المتحركة باستخدام مستشعر اللمس.

> قم بمعايرة موضع وحدة التغذية. يتم ذلك عن طريق تشغيل المحرك للأمام لأقصى مسافة ممكنة حتى يتوقف.

> قم بفرز كل كائن في المجموعة الصحيحة باستخدام محرك كبير لتحريك المنحدر عن طريق تحريك قاعدة المسار في الموضع الصحيح، ثم استخدم المحرك المتوسط لتحريك وحدة التغذية لرمي الكائن في موقعه.

ماذا تعلمت

تعلمت في هذه الوحدة:

- كيفية توصيل **EV3 Brick** مع **VSC** وكيفية برمجتها باستخدام **MicroPython**.
- كيفية ضبط المحركات واستخدام **Micropython** لبرمجة نموذج قاعدة القيادة لتتحرك بطرق مختلفة.
- كيفية استخدام **MicroPython** لبرمجة أجهزة استشعار اللمس.
- معايرة أجزاء الذراع الروبوتية.
- كيفية إنشاء نموذج أولي للذراع الروبوتية.
- كيفية برمجة الذراع الروبوتية للتحكم يدويًا بحركتها.
- كيفية استخدام **MicroPython** لبرمجة جهاز استشعار الألوان.
- كيفية بناء الذراع الروبوتية.

الدرس 1	MicroPython	متصفح الجهاز **Device Browser**	
الدرس 2	**Fixed** الثابتة	**Moving** المتحركة	الذراع الروبوتية **Robotic arm**
	نظام فرز تلقائي **Automatic sorting system**	**Joins** المفاصل	النموذج الافتراضي **Virtual prototype**
	Calibrate معايرة		
الدرس 3	**Gears** تروس	**Torque** عزم الدوران	ترس ذو أسنان مستقيمة **Spur gear**
	ترس مخروطي **Bevel gear**		
الدرس 4	حركة دائرية **Rotational movement**	حركة قُطرية **Radial movement**	حركة عمودية **Vertical movement**
	Theta axis محور ثيتا	درجات الحرية **Degrees of freedom**	

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

الصف	الصفحة	الملاحظة

علوم الحاسب المستوى الحادي عشر المسار التكنولوجي
الفصل الدراسي الثاني 2021 - 2020
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: 978-618-05-5243-0 N2009002030-18110