

مذكرة الوظائف الاشرافية



العام الدراسي 2025/2026م

Information Communication Technology

التوجيه الفني العام للحاسوب

جميع الحقوق محفوظة 2025/2026م



دولة الكويت
State of Kuwait



التوجيه الفني العام للحاسوب
General Supervisory of Computing



✉ ictgeneralsupervisor@moe.edu.kw

🌐 hasob.moe.edu.kw

الصفحة	الموضوع
3	لغة البرمجة Python
11	المتغيرات Variables
23	السلاسل النصية String
40	الشروط Conditions
56	التكرار Loops
73	تصحيح الأخطاء والاستثناءات Debugging and Exceptions
96	القوائم Lists
115	القواميس Dictionaries
133	الدوال Functions
159	المكتبات Libraries

لغة البرمجة بايثون

تُعتبر لغة Python من لغات البرمجة عالية المستوى، مفتوحة المصدر، قابلة للتوسع، سهلة التعلم وقوية الاستخدام. تُستخدم على نطاق واسع في مختلف المجالات منها تطوير تطبيقات الويب، برامج سطح المكتب، أدوات تحكم الأتمتة (مجموعة من الإجراءات لإنجاز مهمة ما بأقل قدر من التدخل البشري)، تطبيقات الشبكة، علم البيانات، الذكاء الاصطناعي، والتعلم الآلي، تحليل البيانات الضخمة، وإنشاء نماذج تنبؤية، وذلك لسهولة استخدامها ووجود العديد من المكتبات المتخصصة. كذلك يتمتع مجتمع Python بدعم كبير من المطورين والمبرمجين، مما يسهل الحصول على المساعدة والدعم المستمر.

كذلك تُعتبر Python لغة كائنية التوجه Object-Oriented Programming، وتُمثل (الأرقام، النصوص، القوائم، القواميس، الدوال)، وغير ذلك ككائنات Objects. كما تمتاز لغة Python بالعديد من المميزات التي جعلتها شائعة الاستخدام وواسعة الانتشار منها:

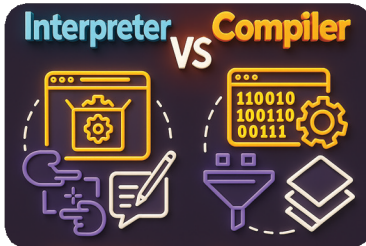
- سهولة التعلم والاستخدام، مما يجعلها اللغة المفضلة لتعلم البرمجة.
- يُمكن تطوير برمجيات معقدة من خلالها.
- تُستخدم في مجالات متعددة.
- تحتوي على العديد من المكتبات الجاهزة.
- تتوافق مع العديد من أنظمة التشغيل.
- يتم تحديثها بصورة مستمرة.

الفرق بين Interpreter و Compiler

تعتمد لغات البرمجة على مبدأ تحويل الأوامر البرمجية من تمثيلٍ عالي المستوى إلى تعليماتٍ تنفيذية بلغة الآلة يستطيع المعالج فهمها، والتعامل معها وتنفيذها. من خلال نظام معالجات لغات البرمجة Programming languages processing system، والذي يشير إلى الخطوات المطلوب تنفيذها من أجل تحويل الأوامر البرمجية إلى لغة الآلة، ويتم عبر برامج وسيطة كالمترجم Compiler، والمفسر Interpreter، ومن أهم ما يميزهما ما يلي:

العنصر	Compiler	Interpreter
طريقة العمل	يُترجم البرنامج بالكامل دفعة واحدة قبل التنفيذ.	يُترجم البرنامج سطراً بسطراً أثناء التنفيذ.
سرعة الترجمة	أبطأ في الترجمة. (يأخذ وقتاً أكبر في البداية)	أسرع في الترجمة. (فوري لكل سطر)
سرعة التنفيذ	أسرع في التنفيذ. (بعد الترجمة)	أبطأ في التنفيذ. (لأن الترجمة تتم أثناء التشغيل)
اكتشاف الأخطاء	تظهر جميع الأخطاء بعد الترجمة الكاملة.	تظهر الأخطاء مباشرة عند السطر الذي يحتويها.
سهولة التصحيح Debug	أصعب لأن الأخطاء تظهر دفعة واحدة بعد الترجمة.	أسهل لأن البرنامج يتوقف عند السطر الخاطئ.
مخرجات الترجمة	ملف تنفيذي مستقل.	لا يُنتج ملف تنفيذي، بل ينفذ التعليمات البرمجية مباشرة.

جدول يوضح الفرق بين Interpreter و Compiler



مع العلم بأن لغة Python تجمع بين المترجم والمفسر، حيث تُترجم التعليمات البرمجية أولاً إلى Byte Code¹، ثم تفسرها وتنفذها.

¹ Byte Code تمثيل وسيط للبرنامج يتم إنشاؤه بعد ترجمة التعليمات البرمجية، وتكون جاهزة للتنفيذ من قبل آلة افتراضية.

- يمكن استخدام تطبيقات معالجة النصوص مثل Notepad، لكتابة التعليمات البرمجية بلغة Python ومن ثم حفظ الملف بامتداد .py.
- يمكن استخدام إحدى بيئات التطوير المتكاملة (Integrated Development Environment (IDE)، والتي توفر الأدوات اللازمة لكتابة التعليمات وتحريرها واختبارها وتصحيح أخطاءها في مكان واحد مثل:
 - العديد من مواقع تحرير التعليمات البرمجية على الإنترنت مثل:

onlinepython.com و repl.it و onlinegdb.com

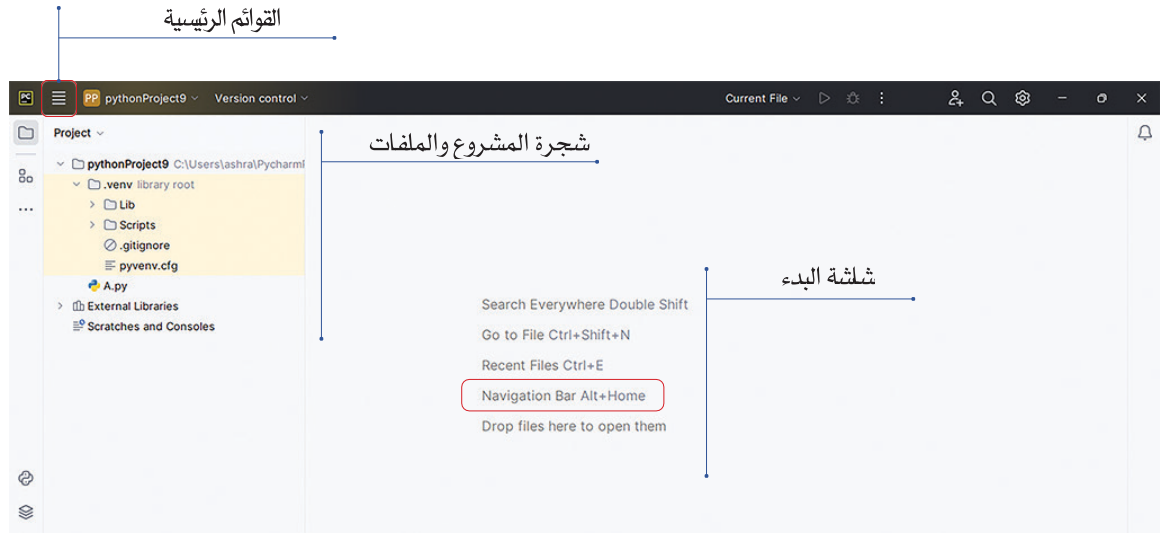
- محرر Visual Studio Code: الخاص بشركة Microsoft.
- محرر PyCharm: الذي سيتم شرحه في المذكرة.

بيئة التطوير المتكاملة PyCharm

- تحميل البرنامج.
- يمكن الحصول على آخر إصدار من محرر PyCharm من الموقع الرسمي jetbrains.com/pycharm
- اختيار نظام التشغيل المناسب .

Windows macOS Linux

- اختيار إحدى النسخ التالية:
 - الأولى: PyCharm Professional وهي نسخة غير مجانية؛ متقدمة للمحترفين.
 - الثانية: PyCharm Community Edition وهي نسخة مجانية؛ تُستخدم للتعليم.
- تثبيت البرنامج.

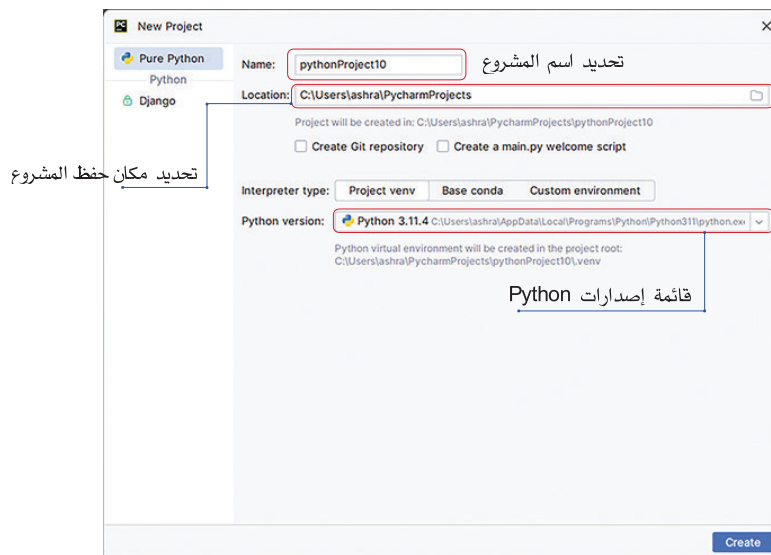


PyCharm Editor شاشة البدء

- القوائم الرئيسية Main Menu: لإظهار شريط قوائم البرنامج.
- شجرة المشروع والملفات Project: يظهر مسار المشروع الحالي وملفات بايثون التي يحتويها.
- شاشة البدء Welcome Screen: تظهر مفاتيح روابط لبعض العمليات الشائعة.

إنشاء مشروع جديد

لإنشاء مشروع جديد، من قائمة File > new project

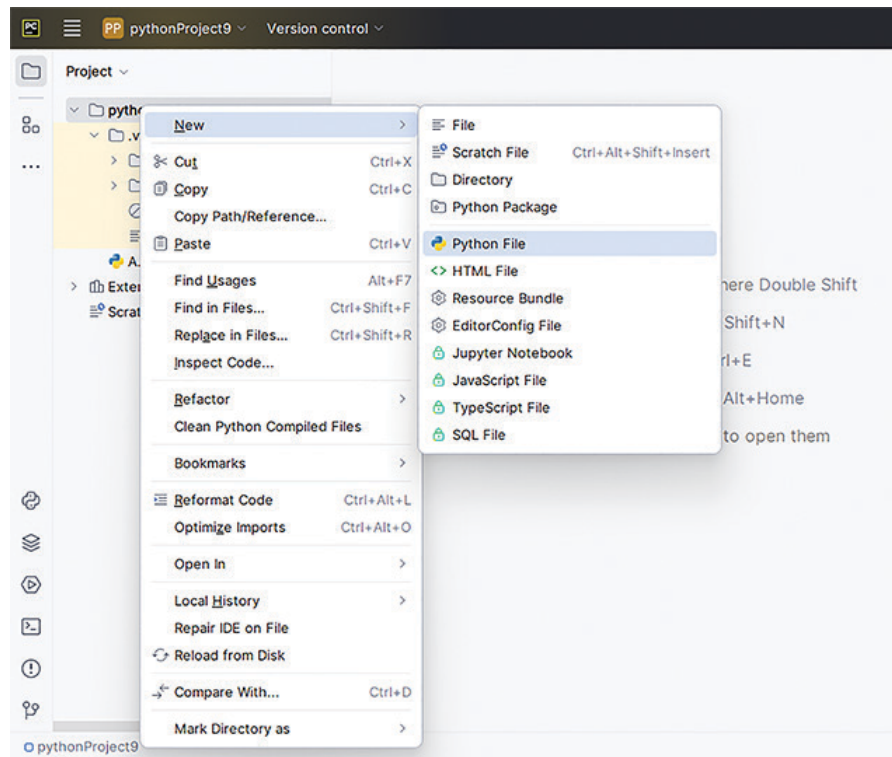


PyCharm Editor - New Project شاشة

إنشاء ملف Python جديد

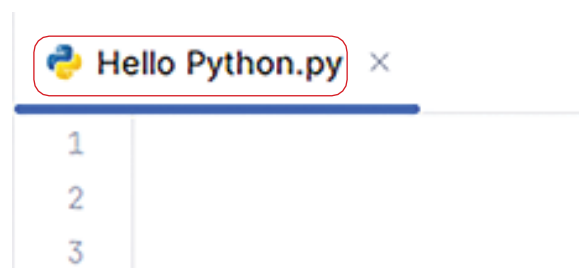
يُمكن أن يحتوي المشروع على العديد من ملفات التعليمات، ولإنشاء ملف Python جديد:

1. يتم الضغط بالزر الأيمن على اسم المشروع في منطقة Project (شجرة المشروع والملفات).
2. اختيار الأمر Python File من قائمة New.



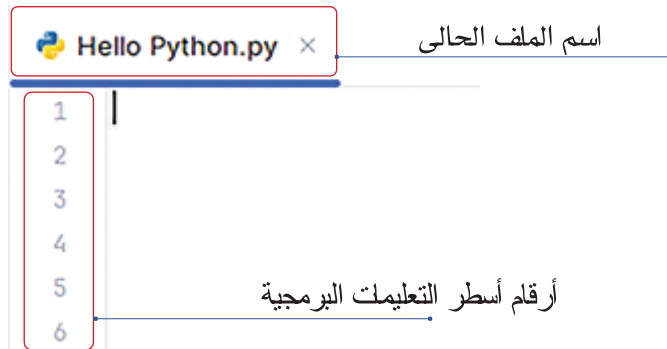
PyCharm Editor - New Python File شاشة

3. كتابة اسم الملف، ثم الضغط على مفتاح Enter.



PyCharm Editor — File Name شاشة

تظهر منطقة تحرير التعليمات البرمجية، ويظهر في الأعلى علامة تبويب باسم الملف الحالي، ويظهر المؤشر في السطر البرمجي الأول.



PyCharm Editor — Code Editor شاشة

كتابة البرنامج الأول

في نافذة كتابة التعليمات البرمجية، وفي السطر الأول اكتب التعليمات البرمجية التالية، مع مراعاة الدقة.

1 `print("I am a Python Developer")`

حيث التعليمة `print` هي من الدوال المدمجة في لغة بايثون والتي عند تنفيذ البرنامج تطبع المحتويات بين القوسين في نافذة التشغيل `RUN`.

لاحظ كتابة النص بين علامتي تنصيص، والتي سوف نوضح وظيفتها لاحقاً.

تشغيل البرنامج RUN

يمكن تشغيل البرنامج بإحدى الطرق التالية:

- القائمة المختصرة لمنطقة كتابة التعليمات البرمجية واختيار الأمر `RUN`.
- استخدام مفاتيح الاختصار `Ctrl + Shift + F10` (تشغيل الملف الحالي).
- منطقة شريط الأدوات `Toolbar`، يتم اختيار الملف المراد تنفيذه، أو اختيار `Current File` لتشغيل الملف الحالي، ثم الضغط على الأداة `RUN` الموضحة في الشكل التالي:



تظهر نافذة جديدة وبها نتيجة تنفيذ البرنامج كالتالي:

```
Run Hello Python x
C:\Users\ashra\PycharmProjects\pythonProject9\
I am a Python Developer
Process finished with exit code 0
```

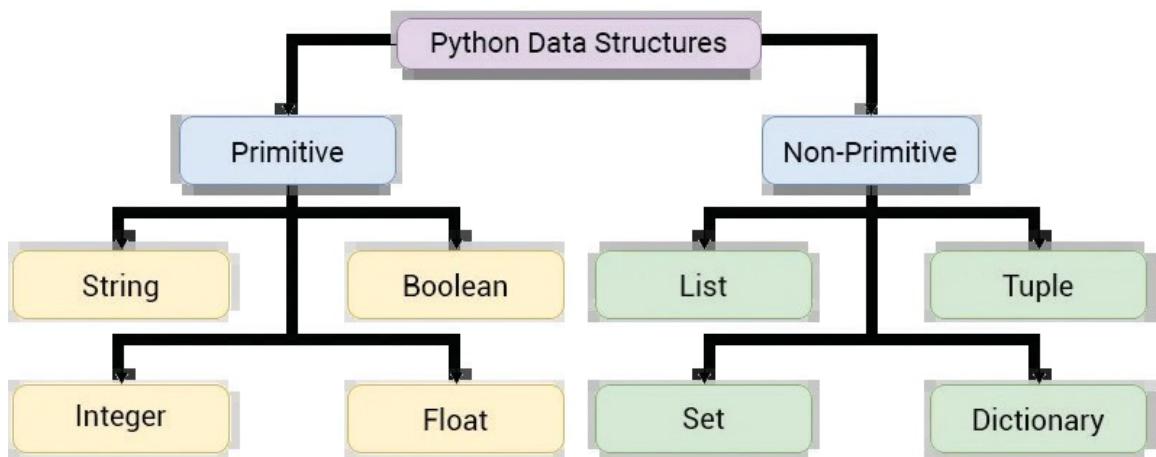
PyCharm Editor — RUN شاشة

هياكل البيانات Data Structure

طريقة لتنظيم وتخزين البيانات بشكل يسمح بالوصول إليها ومعالجتها بكفاءة. تُستخدم هياكل البيانات في جميع أنواع البرامج، من أبسطها إلى أكثرها تعقيداً.

وتنقسم هياكل البيانات إلى قسمين أساسيين:

- أولاً: هياكل البيانات الأولية (Primitive).
- ثانياً: هياكل البيانات غير الأولية (Non-Primitive).



بعض أنواع هياكل البيانات

هياكل البيانات الأولية (Primitive):

- البنية التي تسمح بتخزين قيم لنوع واحد فقط من البيانات.
- يعتمد حجم هياكل البيانات الأولية على نوعها.
- تُعد الأعداد الصحيحة والعشرية والسلاسل النصية (سلسلة من الحروف والأرقام والرموز الخاصة)، والبيانات المنطقية (True – False) من هياكل البيانات الأولية.

أنواع هياكل البيانات الأولية:

- الأعداد الصحيحة Integer.
- الأعداد العشرية Float.
- السلاسل النصية String.
- البيانات المنطقية Boolean.

هياكل البيانات غير الأولية (Non-Primitive):

- البنية التي تسمح بتخزين قيم أنواع بيانات متعددة.
- حجم بنية البيانات غير ثابت.
- تعد القائمة والمجموعة والقاموس والصف والمصفوفة من هياكل البيانات غير الأولية.

أنواع هياكل البيانات غير الأولية:

- القائمة List: تستخدم لتخزين مجموعة من القيم. يمكن أن تحتوي على أي نوع من البيانات، بما في ذلك الأعداد والسلاسل النصية، والقوائم الأخرى.
- المجموعة Set: تستخدم لتخزين مجموعة من القيم الفريدة Unique values. يمكن أن تحتوي عناصرها على أي نوع من البيانات.
- القاموس Dictionary: تستخدم لتخزين مجموعة من المفاتيح والقيم. ويمكن أن تكون القيم من أي نوع من البيانات.
- الصف Tuple: تستخدم لتخزين مجموعة من القيم الغير قابلة للتعديل، حيث يمكن أن تحتوي على عناصر من أنواع مختلفة.

المتغيرات Variables

المتغير هو مكان يتم حجزه في الذاكرة لتخزين البيانات بصورة مؤقتة أثناء تنفيذ البرنامج، وله اسم، وقيمة، ونوع (الأعداد والنصوص والقوائم والقواميس...) ، ويمكن لغة Python من تغيير نوع وقيمة المتغير أثناء تنفيذ البرنامج.

الإعلان عن المتغيرات Declaration

تتميز Python بالتعرف على نوع المتغير بشكل مباشر حسب القيمة المخصصة تلقائيًا، ويمكن تخصيص القيم للمتغيرات باستخدام رمز التخصيص (=).

أنواع المتغيرات Data Types

نوع المتغير	نوع البيانات	استخداماته	نوع البيانات
String وتُكتب str	نص	تخزين سلسلة من الأحرف والأرقام التي يتعامل معها كنصوص، ولا بد أن تكتب بين علامتي التنصيص المزدوجة " " أو المفردة ' '.	my_name = "Ali" my_country = "Kuwait" my_address = "15 Beirut street"
Integer وتُكتب int	عدد صحيح	تخزين قيم عددية صحيحة.	months_count = 12
Float وتُكتب float	عدد عشري	تخزين قيم تحتوي على كسور عشرية.	my_salary = 1930.750 my_weight = 50.00
Boolean وتُكتب bool	منطقية	يأخذ إحدى القيمتين True (صحيح) أو False (خطأ)، وغالبًا ما تنتج عن عمليات المقارنة.	var_num1 = 10 var_num2 = var_num1 > 5 #True var_num3 = var_num1 < 5 #False

أمثلة على أنواع البيانات

كما يوجد أنواع أخرى من البيانات سيتم التعرف عليها في وقتها مثل القوائم والصفوف.

ملاحظة: تُستخدم # لإضافة التعليقات، ولا يُنفذها البرنامج كتعليمة برمجية.

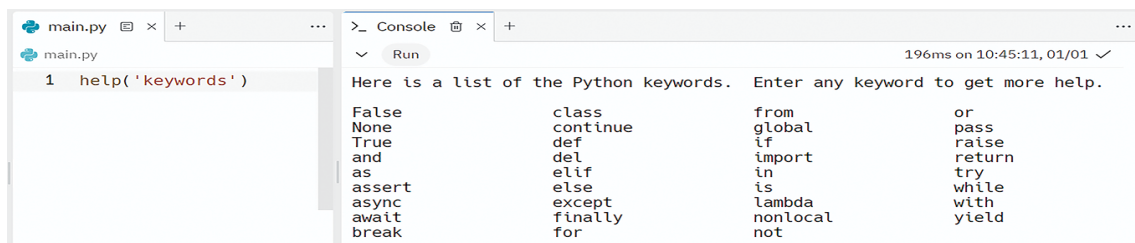
قواعد تسمية المتغيرات Naming Variables

يجب أن تتبع المتغيرات في Python قواعد معينة للتسمية:

- يجب أن يبدأ اسم المتغير (بحرف أو شرطة سفلية) ولا يبدأ برقم.
- يمكن أن يحتوي اسم المتغير على أحرف أبجدية وأرقام.
- لا يمكن أن يحتوي اسم المتغير على رموز خاصة (*&^%\$#@!).
- باستثناء الشرطة السفلية (_).
- لا يمكن استخدام الكلمات المحجوزة keywords reserved لتسمية المتغيرات.

ملاحظات:

- يراعى في تسمية المتغير حالة الأحرف (Small/Capital) sensitive for letters case.
- استخدام أسماء للمتغيرات وصفية ذات دلالة ومعنى.
- يُمكن التعرف على الكلمات المحجوزة في Python باستخدام الأمر `help('keywords')`.



الكلمات المحجوزة في Python

دالة (type): تستخدم للتعرف على نوع المتغير.

مثال 1:

Code

```
1 var_name = "Abdulla Rahman Al-Smait"
2 var_count = 29
3 print(type(var_name))
4 print(type(var_count))
```

Output

```
<class 'str'>
<class 'int'>
```

ملاحظات:

المتغير الأول `var_name`: من نوع نصي `<class 'str'>`.
المتغير الثاني `var_count`: من نوع عدد صحيح `<class 'int'>`.

دالة الإخراج print

دالة مدمجة في Python، تُستخدم لعرض البيانات والمخرجات على الشاشة. يمكن لها التعامل مع مختلف أنواع البيانات، كالنصوص، والأرقام، والقوائم، وغيرها.

استخدام الفاصلة (,) comma مع دالة print:

تُستخدم لتنسيق المخرجات حيث تُضيف مسافات بشكل تلقائي بين العناصر، مما يجعل النص أكثر وضوحًا وتنظيمًا دون الحاجة إلى إدخال المسافات يدويًا.

مثال 2: الربط بين قيم المتغيرات.

Code

```
1 x = "Free"
2 y = "Kuwait"
3 print(x,y)
```

Output

Free Kuwait

مثال 3: ربط النصوص، وقيم المتغيرات.

Code

```
1 age = 18
2 print("Your age = " , age)
```

Output

Your age = 18

تستخدم Python العمليات الحسابية الأساسية، وفي الجدول التالي رموز العمليات الحسابية ووظيفتها.

الرمز	العمليات الحسابية	مثال
+	الجمع Addition	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1+var2)</pre> 28
-	الطرح Subtraction	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1-var2)</pre> 18
*	الضرب Multiplication	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1*var2)</pre> 115
/	القسمة Division	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1/var2)</pre> 4.6
//	ناتج القسمة الصحيح بدون الكسور Floor division	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1 // var2)</pre> 4
%	باقي القسمة Modulus	<pre> 1 var1 = 23 2 var2 = 5 3 print(var1 % var2)</pre> 3
**	رفع العدد إلى أس محدد Exponentiation	<pre> 1 var2 = 5 2 print(var2 ** 2) 3 print(var2 ** 3)</pre> 25 125

رموز العمليات الحسابية ووظيفتها

أولويات تنفيذ العمليات الحسابية

1. العمليات داخل الأقواس.
2. رفع الأسس.
3. الضرب والقسمة.
4. الجمع والطرح
5. يتم تنفيذ العمليات المتكافئة من اليسار إلى اليمين.

مثال 4:

```
main.py  Format  Run
1 print(5 + (5 * 3 ** 2 / 5) - 6) 8.0
```

دالة input()

دالة مدمجة في Python تُستخدم لاستقبال البيانات المُدخلة من المستخدم، مثل أسماء المستخدمين وعناوين البريد الإلكتروني وأرقام الهواتف.

صيغة الدالة Syntax

تكتب صيغة الدالة input() كالتالي:

input([prompt])

سلسلة نصية تُستخدم لعرض رسالة إلى المستخدم (اختياري).

الدالة input() تُرجع أي قيمة مُدخلة من المستخدم كسلسلة نصية String.

مثال 5:

Code

```
1 name = input("Enter your name: ")
2 print("Hello, " + name + "!")
```

Output

Enter your name: Ahmed

Hello, Ahmed!

تفسير النص البرمجي:

- يعرض رسالة للمستخدم: Enter your name:
- ينتظر البرنامج حتى يُدخل المستخدم اسمًا ويضغط على مفتاح Enter.
- تخصيص القيمة المدخلة إلى المتغير name.
- طباعة العبارة Hello, [Name].

ملاحظات:

- يجب الانتباه إلى أن دالة input() تُرجع سلسلة string. ويمكن تحويلها إلى عدد صحيح أو عدد عشري إذا كانت المُدخلات عددية، باستخدام الدالة int() أو الدالة float().

مثال 6: إدخال قيم عددية لمتغيرين باستخدام دالة input، وإجراء عملية الجمع.

Code

```
1 num1 = input("Enter First number: ")
2 num2 = input("Enter Second number:")
3 print(num1 + num2)
```

Output

Enter First number: 10

Enter Second number: 12

1012

لاحظ أن عملية جمع الأعداد لم تُنفذ بشكل صحيح، حيث تم التعامل مع القيم المُدخلة على أنها نصوص، وتم ربطها بجوار بعضها البعض بدلاً من جمعها حسابياً.

مثال 7: إجراء عملية حسابية بجمع المتغيرين الأول والثاني كأرقام.

Code

```
1 num1 = input("Enter first number: ")
2 num1 = input("Enter second number: ")
3 num1 = int(num1)
4 num1 = int(num2)
5 print(num1 + num2)
```

Output

Enter first number: 10

Enter second number: 12

22

كما يمكن التحويل مباشرة أثناء عملية الإدخال كالتالي:

```
score = int(input("Enter player score: "))
```

ناتج استخدام الدالة `input()`، والدالة `int()` تعيد العدد المدخل إلى عدد صحيح.

اكتب برنامج لحساب (السرعة العددية) بمعلومية قيمتي المسافة والزمن:
1. يطلب من المستخدم إدخال قيمتي المسافة والزمن، ولتنفيذ ذلك باستخدام دالة `input()`.
أ. التعليمة البرمجية لإدخال المسافة `distance`.

`distance = input("Enter the distance (km): ")`

ب. التعليمة البرمجية اللازمة لإدخال قيمة الزمن `time`.

`time = input("Enter the time (hours): ")`

- تظهر رسالة للمستخدم عند تشغيل البرنامج تطلب منه إدخال قيمة المسافة، ثم رسالة أخرى تطلب منه إدخال قيمة الزمن.
- تستقبل دالة `input()` جميع القيم كسلاسل نصية.

2. حساب السرعة تُستخدم معادلة السرعة القياسية التالية:

`speed = distance / time`

- التعليمة البرمجية اللازمة لحساب السرعة:

`speed = float(distance) / float(time)`

- يتم الإعلان عن متغير جديد باسم `speed`، بتخصيص قيمه له وهي (ناتج قسمة المسافة على الزمن).
- المعامل (/) يمثل عملية القسمة، راجع جدول العمليات الحسابية.
- استخدام دالة `float()` لتحويل قيمة المسافة والزمن المُدخلة من نصية إلى عشرية.

3. طباعة السرعة.

مثال 8: برنامج
حساب السرعة
العددية بمعرفة
قيمتي المسافة
والزمن.

Code

```
1 distance = input("Enter the distance
2 (km): ") time = input("Enter the time
3 (hours): ")
4 speed = float(distance) / float(time)
   print("Speed = ", speed, "km/h")
```

Output

Enter the distance (km): 151

Enter the time (hours): 2

Speed = 7.5 km/h

اكتب برنامج لحساب تكلفة رحلة مدرسية لأبراج الكويت بمعلومية تكلفة المواصلات، دخول أبراج الكويت والوجبة.

- اطلب من المستخدم ادخال سعر دخول أبراج الكويت (ticket) و سعر الوجبة (meal) و سعر المواصلات (transportation).
- احسب إجمالي تكلفة الرحلة total_cost باستخدام المعادلة:
$$\text{total_cost} = \text{ticket} + \text{meal} + \text{transportation}$$
- اعرض إجمالي تكلفة الرحلة total_cost.

**مثال 9: حساب
تكلفة رحلة
مدرسية لأبراج
الكويت.**

Code

```
1 # Enter ticket , meal , transportation price:
2
3 ticket_price = float(input("Enter ticket price: "))
4 meal_price = float(input("Enter meal price: "))
5 trans_price = float(input("Enter transportation price: "))
6
7 # Calculate the total cost
8 total_cost = ticket_price + meal_price + trans_price
9
10 # print the total cost on the screen
11 print (total_cost)
```

Output

```
Enter ticket price: 3
Enter meal price: 1.500
Enter transportation price: 2
6.5
```


- اكتب برنامج لحساب الشغل المبذول بمعلومية القوى المؤثرة، والإزاحة في اتجاهها.
- استقبل من المستخدم القوة المؤثرة **Force** بوحدة (النيوتن).
 - استقبل من المستخدم الإزاحة **Displacement** بوحدة (المتر). (أكبر من الصفر).
 - احسب قيمة الشغل **Work** بوحدة (الجول) باستخدام المعادلة :
الشغل $W = \text{القوة } F \times \text{الإزاحة } D$.
 - اطبع الشغل **Work** على الشاشة.

مثال 10: حساب
الشغل المبذول

Code

```
1 # استقبال القوة والإزاحة من المستخدم
2 var_force = float(input("Enter the force in Newtens: "))
3 # التأكد من أن الإزاحة أكبر من الصفر
4 var_Displacement = float(input("Enter the Displacement in kg (must be greater than zero): "))
5 # حساب الشغل باستخدام القانون
6 var_Work = var_force * var_Displacement
7 # عرض الشغل
8 print("The Work is:", var_Work, "Joules")
```

Output

```
"C:\Users\HP\PycharmProjects\pythonProject 10-1\.venv\Scripts
Enter the force in Newtens: 65
Enter the Displacement in kg (must be greater than zero): 14
The Work is: 910.0 Joules
Process finished with exit code 0
```

ملاحظة:

الدالة المدمجة (input) ترجع دائماً قيمة نصية، فيتوجب تحويل هذه القيمة النصية إلى قيمة عددية لإجراء العمليات الحسابية.

اكتب برنامجاً يطلب من المستخدم إدخال بياناته (الاسم، العمر، الوزن، الدرجة)، ثم ينفذ التالي:

- يخزن البيانات في متغيرات بأنواع مناسبة.
- يطبق عمليات حسابية على الأعداد.
- يستخدم التحويل بين الأنواع (من نص إلى عدد صحيح أو عشري).
- يطبع النتائج بطريقة منسقة.
- يستخدم متغير منطقي لتحديد النجاح أو الرسوب.

مثال 11:

Code

```
1  # إدخال بيانات المستخدم
2  name = input("Enter your name: ")      # نص (String) لتخزين الاسم
3  age = int(input("Enter your age: "))    # تحويل إلى عدد صحيح (Integer) لتخزين العمر
4  weight = float(input("Enter your weight: ")) # تحويل إلى عدد عشري (Float) لتخزين الوزن
5  score = float(input("Enter your exam score: ")) # درجة الطالب (float)
6
7  # العمليات الحسابية
8  next_age = age + 1                      # عملية جمع: العمر في السنة القادمة
9  bmi = weight / (1.70 ** 2)              # حساب مؤشر كتلة الجسم (BMI) على طول ثابت 1.70 م
10
11 # متغير منطقي (Boolean)
12 passed = (score >= 50)                  # يعيد True إذا الدرجة  $\geq 50$  وإلا False
13
14 # المخرجات باستخدام print
15 print("\n--- Student Report ---")      # طباعة عنوان التقرير
16 print("Name:", name)                   # عرض الاسم
17 print("Current Age:", age, "-> Next year:", next_age) # العمر الحالي والقادم
18 print("Weight:", weight, "kg")          # الوزن بالكيلو
19 print("BMI (approx):", round(bmi, 2))   # BMI مع تقريب إلى منزلتين
20 print("Exam Score:", score)             # عرض درجة الطالب
21 print("Passed?", passed)                # عرض حالة النجاح أو الرسوب
```

Output

```
Enter your name: Ahmed
Enter your age: 17
Enter your weight: 65.5
Enter your exam score: 82

--- Student Report ---
Name: Ahmed
Current Age: 17 -> Next year: 18
Weight: 65.5 kg
BMI (approx): 22.65
Exam Score: 82.0
Passed? True
```

مثال 12: حساب فاتورة

- يكتب برنامجاً لحساب فاتورة مشتريات بسيطة:
- يطلب من المستخدم: سعر القطعة، الكمية ونسبة الخصم.
- يحسب المجموع الفرعي ، الخصم ، المبلغ الخاضع للضريبة، الضريبة 5٪، الإجمالي النهائي، ومتغير منطقي لتحديد الشحن المجاني (True إذا كان الإجمالي ≤ 20).

Code

```

1  # ثوابت عامة
2  TAX_RATE = 0.05          # نسبة الضريبة 5%
3  FREE_SHIP_THRESHOLD = 20.0 # حد الشحن المجاني
4
5  # إدخال بيانات من المستخدم
6  price = float(input("Enter item price (KWD): ")) # (float) سعر القطعة
7  qty = int(input("Enter quantity: ")) # (int) الكمية
8  disc = float(input("Enter discount percent (0-100): ")) # نسبة الخصم %
9
10 # الحسابات الأساسية
11 subtotal = price * qty # المجموع الفرعي قبل الخصم والضريبة
12 discount_amount = subtotal * (disc / 100) # قيمة الخصم
13 taxable_amount = subtotal - discount_amount # المبلغ بعد الخصم
14 tax = taxable_amount * TAX_RATE # قيمة الضريبة
15 total = taxable_amount + tax # الإجمالي النهائي
16
17 # متغير منطقي لتحديد أهلية الشحن المجاني
18 eligible_free_shipping = (total >= FREE_SHIP_THRESHOLD) # True أو False
19
20 # عرض الأنواع (اختياري)
21 print("Types:", type(price).__name__, type(qty).__name__, type(total).__name__) # float int float
22
23 # طباعة تقرير منسق
24 print("\n--- Invoice ---")
25 print("Unit Price      :", f"{price:.2f}", "KWD")
26 print("Quantity       :", qty)
27 print("Subtotal        :", f"{subtotal:.2f}", "KWD")
28 print("Discount         :", f"{discount_amount:.2f}", "KWD", f"({disc:.1f}%)")
29 print("Taxable Amount   :", f"{taxable_amount:.2f}", "KWD")
30 print("Tax (5%)         :", f"{tax:.2f}", "KWD")
31 print("TOTAL           :", f"{total:.2f}", "KWD")

```

Output

```

Enter item price (KWD): 6.5
Enter quantity: 3
Enter discount percent (0-100): 10
Types: float int float

```

```

--- Invoice ---
Unit Price      : 6.50 KWD
Quantity       : 3
Subtotal        : 19.50 KWD
Discount        : 1.95 KWD (10.0%)
Taxable Amount  : 17.55 KWD
Tax (5%)       : 0.88 KWD
TOTAL          : 18.43 KWD
Free Shipping? : False

```

السلاسل النصية Strings

هي مجموعة من الأحرف والأرقام والرموز تُميز بعلامات اقتباس، تخزن السلاسل في متغيرات يتم التعامل معها كقيمة نصية، وهي غير قابلة للتغيير **Immutable**، لذا فإن جميع العمليات التي تُجرى عليها لا تؤثر على السلسلة الأصلية، بل تُنتج نسخة جديدة معدلة تحتوي على التغييرات المطلوبة.

(1) العمليات الأساسية على السلاسل النصية:

تعريف تخصيص وطباعة السلاسل النصية:

يتم تعريف، وتخصيص السلاسل النصية بعلامات اقتباس فردية ' ' أو مزدوجة " " .

مثال 1: إنشاء سلسلة نصية باستخدام علامتي اقتباس فردية.

Code

```
1 var_x = 'Free Kuwait'
2 print(var_x)
```

Output

Free Kuwait

مثال 2: إنشاء سلسلة نصية باستخدام علامتي اقتباس مزدوجة.

Code

```
1 var_x = "Free Kuwait"
2 print(var_x)
```

Output

Free Kuwait

الفهرسة [Indexing]:

الوصول إلى موقع الأحرف في سلسلة نصية [x] حيث تبدأ عملية الفهرسة من رقم صفر.

Index Number	0	1	2	3	4	5	6	7	8	9	10
String	F	r	e	e		K	u	w	a	i	T

طريقة الوصول إلى موقع الأحرف في السلسلة النصية

مثال 3:

Code

```
1 var_x = 'Free Kuwait'
2 print(var_x[0])
3 print(var_x[1])
```

Output

F
R

مثال 4:

Code

```
1 var_x = 'Free Kuwait'
2 print(var_x[-1])
3 print(var_x[-4])
```

Output

t
w

القطع Slicing:

يمكن الحصول على جزء من السلسلة النصية بأرقام الفهارس $[x : y]$.

الوصف	الصيغة
تُعيد النص بداية من الموقع x وحتى الموقع ما قبل y	<code>str[x : y]</code>
تُعيد النص بداية من الموقع 0 وحتى الموقع ما قبل y	<code>str[: y]</code>
تُعيد النص بداية من الموقع x وحتى نهاية السلسلة النصية	<code>str[x :]</code>

أمثلة على عملية القطع

مثال 5:

Code

```
1 var_x = 'Abdul-Rahman Hamoud Al-Sumait'
2 print('First Name: ', var_x[ : 12])
3 print('Middle Name: ',var_x[13 : 19])
4 print('Last Name: ',var_x[20 : ])
```

Output

First Name: Abdul-Rahman

Middle Name: Hamoud Last

Name: Al-Sumait

مثال 6:

استخدام القطع Slicing للحصول على بيانات محددة (تقسيم الرقم المدني للحصول على [شهر، يوم] من تاريخ الميلاد).

Index Number	0	1	2	3	4	5	6	7	8	9	10	11
Civil ID	3	1	6	0	1	2	2	0	0	6	4	1

مثال على عملية الفهرسة

Code

```
1 civil_id = '316012200641'
2 month = civil_id[3 : 5]
3 day = civil_id[5 : 7]
4 print('Month = ',month)
5 print('Day = ',day)
```

Output

Month = 01

Day = 22

جمع السلاسل النصية Concatenation:

يستخدم معامل عملية الجمع (+) لإضافة سلسلتين نصيتين لإنشاء سلسلة نصية جديدة.

مثال 7:

Code

```
1 var_x='Free'
2 var_y = 'kuwait'
3 print(var_x + ' ' + var_y)
```

Output

Free Kuwait

لاحظ:

يُمكن استخدام ' '، لإضافة مسافة بين المتغيرين.

مثال 8:

Code

```
1 print("Liberation" + "Day: " + "26" + " Feb")
```

Output

LiberationDay: 26 Feb

لاحظ:

تم إضافة مسافة بعد كلمة Day: وقبل كلمة Feb.

تكرار السلاسل النصية:

تكرار سلسلة نصية لأي عدد باستخدام معامل الضرب (*).

مثال 9:

Code

```
1 print("#"*10)
```

Output

#####

المقارنة Comparison:

مقارنة السلاسل النصية باستخدام معاملي المقارنة ==, !=.

مثال 10:

Code

```
1 prophet = 'Muhammad'
2 prophet_input = input('Who is the last prophet? ')
3 print(prophet_input == prophet)
```

Output

Who is the last prophet? *Eissa*

False

Who is the last prophet? *Muhammad*

True

مثال 11:

Code

```
1 prophet = 'Muhammad'
2 prophet_input = input('Who is the last prophet? ')
3 print('Correct answer!' * (prophet_input == prophet) + 'Wrong answer!' *
      (prophet_input != prophet))
```

Output

Who is the last prophet? *Muhammad*

Correct answer!

Who is the last prophet? *Mosa*

Wrong answer!

(2) دوال السلاسل النصية:

الدوال هي مجموعة من الأوامر الجاهزة تُنفذ مهامًا معينة عند استدعائها.

- الدوال المدمجة (Built-in Functions):
هي دوال تُنفذ مهامًا معينة وتتوفر في Python بشكل افتراضي، ومن أمثلتها:

دالة len():

تُستخدم للحصول على طول السلسلة النصية.

مثال 12:

Code

```
1 var_x = 'Free Kuwait'
2 print(len(var_x))
```

Output

11

- الأساليب المدمجة (Built-in Method):
هي دوال تُنفذ مهامًا معينة، مُرتبطة بكائن object، ويفصل بينهما (.) عند الاستدعاء، ومن أمثلتها:

دالة capitalize():

تُستخدم لتحويل الحرف الأول إلى حرف كبير .
string.capitalize()

مثال 13:

Code

```
1 var_x = 'hello, cybersecurity!'
2 print(var_x.capitalize())
3 print(var_x)
```

Output

Hello, cybersecurity!

hello, cybersecurity!

دالة upper():

string.upper() تحول جميع الأحرف في السلسلة النصية إلى أحرف كبيرة .

مثال 14:

Code

```
1 var_x = 'Hello, Cybersecurity!'
2 print(var_x.upper())
```

Output

HELLO, CYBERSECURITY!

دالة lower():

string.lower() تحول جميع الأحرف في السلسلة النصية إلى أحرف صغيرة .

مثال 15:

Code

```
1 var_x = 'COMPUTER SCIENCE'
2 print(var_x.lower())
```

Output

computer science

دالة find():

string.find(sub,start,end) تستخدم لإرجاع رقم فهرس لأول ظهور لسلسلة فرعية في السلسلة المحددة.

sub: النص أو الحرف الذي تبحث عنه.

start: الموضع الذي يبدأ منه البحث (اختياري).

end: الموضع الذي ينتهي عنده البحث (اختياري).

مثال 16:

Code

```
1 text = 'Free Kuwait'
2 index_chr = text.find('K')
3 index_word = text.find('Kuwait')
4 index = text.find('Emirates')
5 print(index_chr)
6 print(index_word)
7 print(index)
```

Output

```
5
5
-1
```

لاحظ: إذا لم يتم العثور على النص، فستُرجع دالة find() العدد -1.

دالة replace():

string.replace(old value,new value) دالة تستخدم لاستبدال نص بآخر في السلسلة النصية.

مثال 17:

Code

```
1 text = 'The computer is new. I like my computer'
2 new_text = text.replace('computer', 'computer')
3 print(new_text)
```

Output

```
The computer is new. I like my computer.
```

التعليمة f-string

تُستخدم لكتابة النصوص مع القيم و المتغيرات، بحيث يُكتب حرف f قبل النص وتكتب المتغيرات داخل الأقواس المعقوفة { } Curly Brackets.

مثال 18:

Code

```
1 name = 'Muhammed' age= 63
2 info = f'Name: {name} and Age: {age}'
3 print(info)
```

Output

Name: Muhammed and Age: 63

ويمكن استخدام التعليمة البرمجية f-string مباشرة مع دالة print.

```
1 print(f'Name: {name} and Age: {age}')
```

يُمكن التحكم في تنسيق الأرقام العشرية.

مثال 19:

Code

```
1 x = 11
2 y = 3
3 z = x / y
4 print(f'{x} / {y} = {z}')
3 print(f'{x} / {y} = {z : .3f}')
```

Output

11 / 3 = 3.6666666666666665

11 / 3 = 3.667

دالة int():

تُستخدم لتحويل السلاسل النصية إلى أعداد صحيحة.

ملاحظة:

دالة () input تُرجع البيانات المُدخلة كسلسلة نصية string، ولتحويلها إلى integer تستخدم دالة () int.

مثال 20:

Code

```
1 n1 = input('Enter First Number: ')
2 n2 = input('Enter Second Number: ')
3 print(int(n1) + int(n2))
```

Output

```
Enter First Number: 4
Enter Second Number: 6
10
```

ويُمكن استخدام الطريقة التالية:

مثال 21:

Code

```
1 n1 = int(input('Enter First Number: '))
2 n2 = int( input('Enter Second Number: '))
3 print(n1 + n2)
```

Output

```
Enter First Number: 4
Enter Second Number: 6
10
```

دالة float():

تُستخدم لتحويل السلاسل النصية (أرقام numeric) أو الأعداد الصحيحة إلى أعداد عشرية.

مثال 22:

Code

```
1 n1 = float(input('Enter First Number: '))
2 n2 = float(input('Enter Second Number: '))
3 print(n1 + n2)
```

Enter First Number: 5.6
Enter Second Number: 4
9.6

دالة count():

تُستخدم لإيجاد عدد مرات ظهور سلسلة نصية.

مثال 23:

Code

```
1 sura_alnas = '''
2 بسم الله الرحمن الرحيم
3 قُلْ أَعُوذُ بِرَبِّ النَّاسِ
4 مَلِكِ النَّاسِ
5 إِلَهِ النَّاسِ
6 مِنْ شَرِّ الْوَسْوَاسِ الْخَنَّاسِ
7 الَّذِي يُوَسْوِسُ فِي صُدُورِ النَّاسِ
8 مِنَ الْغِيَةِ النَّاسِ
9 '''
10 count=sura_alnas.count("النَّاس")
11 print(f"The word 'النَّاس' appears {count} times.")
```

The word 'النَّاسِ' appears 5 times.

ملاحظة:

تستخدم علامات اقتباس ثلاثية "" لكتابة سلاسل نصية متعددة الأسطر.

دالة center():

تُعيد أحرف السلسلة مزاحة نحو الوسط ضمن سلسلة ذات طول محدد.

مثال 24:

Code

```
1 var_x = 'Security'
3 print(var_x.center(16 , '#'))
```

Output

```
#####Security#####
```

عند استخدام الدالة center()، إذا كانت القيمة المحددة لطول السلسلة المحددة أقل من أو تساوي طول النص الأصلي، فلن تتم إضافة أي رموز للتوسيط وسيُعرض النص كما هو.

اكتب برنامجاً يعكس ترتيب الأحرف في السلسلة النصية لتصبح (كلمة السر) مؤمنة وذلك من خلال الخطوات:

- يطلب من المستخدم إدخال كلمة المرور (password).
- يخزن الكلمة في متغير (password)
- ينشئ متغيراً جديداً (reversed_password) لتخزين الكلمة المعكوسة.
- يستخدم التقطيع Slicing لعكس الكلمة [::-1].
- يطبع الكلمة المعكوسة على أنها كلمة مرور محمية.

مثال 25: تأمين كلمة المرور

هذه الطريقة تعتمد على [start: end: step] للوصول إلى أجزاء معينة من السلسلة النصية.

مع تعيين قيمة step بقيمة -1 لعكس النص بأكمله. يمكن إضافة حل آخر باستخدام الـ loops.

Code

```
1 # Ask the user to enter the password
2 password = input("Enter Password: ")
3
4 # Store the reversed word using slicing
5 reversed_password = password[::-1]
6
7 # Print the reversed password as a secured version
8 print("Secured password: ", reversed_password)
```

Output

```
Enter Password: Kuwait
Secured password: tiawuK
```

اكتب برنامجاً يستخرج الجزء الصحيح من السلسلة (دينار)، والجزء العشري (فلس) من مبلغ مالي عددي مُدخل من المستخدم وذلك من خلال الخطوات:

- يستقبل قيمة المبلغ المالي مثال (500.750) من المستخدم.
- يحدد موضع الفاصلة العشرية للسلسلة النصية المُدخلة.
- يحدد جزء قبل الفاصلة المبلغ بالدينار.
- يحدد جزء بعد الفاصلة المبلغ بالفلس.
- يطبع المبلغ بكتابة بالتنسيق التالي: 500 KD 750 fils.

مثال 26: تقطيع
السلسلة النصية
(مبلغ مالي)

Code

```
1 amount = input("Enter amount e.g (500.750) : ")
2 position= amount.split(".")
3 kd = position[0]
4 fils = position [-1]
5 print(f"{kd} KD And {fils} fils")
```

Output

```
Enter amount e.g (500.750) : 1000.500
1000 KD And 500 fils
```


اكتب برنامجاً يقسم الاسم الكامل full name (باللغة الانجليزية) للحصول على الاسم الأول والأوسط والأخير.

وذلك من خلال الخطوات:

- يستقبل الاسم الكامل (ثلاثي) باللغة الإنجليزية من المستخدم، ويحفظه في متغير full_name.
- يطبع عدد أحرف الاسم الكامل باستخدام دالة len().
- يستخدم دالة find() لإيجاد موضع (Position) المسافة الأولى بين الاسم الأول والثاني، ويحفظه في متغير position_space1.
- يستخدم دالة find() لإيجاد موضع (Position) المسافة الثانية بين الاسم الثاني والثالث، ويحفظه في متغير position_space2.
- يستخدم الفهرسة والتقطيع للحصول على الاسم الأول، الثاني، والثالث ويحفظهم في متغيرات first_name و second_name و third_name.
- يطبع:
الاسم الأول والاسم الثاني والاسم الثالث مع تحويل أول حرف فقط من كل جزء في الاسم الكامل إلى حرف كبير Capital letter.

مثال 27: تقطيع
السلسلة النصية
(الاسم الكامل)

Code

```
1 full_name = input("Enter your full name (Three Parts): ")
2 print("Characters in your name:", len(full_name))
3
4 position_space1 = full_name.find(" ")
5 position_space2 = full_name.find(" ", position_space1 + 1)
6
7 first_name = full_name[:position_space1]
8 second_name = full_name[position_space1 + 1:position_space2]
9 third_name = full_name[position_space2 + 1:]
10
11 print(first_name.capitalize())
12 print(second_name.capitalize())
13 print(third_name.capitalize())
```

Output

```
Enter your full name (Three Parts): hamad fahed salem
Characters in your name: 17
Hamad
Fahed
Salem
```

اكتب برنامجاً يقرأ الاسم الكامل (ثلاثي) من المستخدم ثم ينفذ ما يلي:

- يطبع عدد أحرف الاسم باستخدام الدالة `len()`.
- يجد موضع أول مسافة.
- يجد موضع المسافة الثانية.
- يستخدم التقطيع للحصول على الأسماء الثلاثة.
- يطبع كل اسم بحيث يكون أول حرف كبير.

مثال 28:

Code

```
1  # إدخال الاسم الكامل من المستخدم (ثلاثي)
2  full_name = input("Enter your full name (Three Parts): ")
3
4  # حساب طول السلسلة النصية
5  print("Length:", len(full_name))    # عدد الأحرف
6
7  # إيجاد موضع أول مسافة
8  pos1 = full_name.find(" ")          # موضع أول مسافة
9
10 # إيجاد موضع ثاني مسافة
11 pos2 = full_name.find(" ", pos1+1)  # البحث ابتداءً من بعد أول مسافة
12
13 # الحصول على الأجزاء باستخدام التقطيع
14 first = full_name[:pos1]             # من البداية حتى أول مسافة
15 middle = full_name[pos1+1:pos2]     # من بعد أول مسافة حتى الثانية
16 last = full_name[pos2+1:]           # من بعد ثاني مسافة حتى النهاية
17
18 # طباعة النتائج مع جعل أول حرف كبير
19 print("First name :", first.capitalize())
20 print("Middle name:", middle.capitalize())
21 print("Last name  :", last.capitalize())
```

Output

```
Enter your full name (Three Parts): ahmed ali hussein
Length: 17
First name : Ahmed
Middle name: Ali
Last name  : Hussein
```

مثال 29:

- اكتب برنامجاً ينفذ ما يلي:
- يعرف المتغير 'civil_id' = '316012200641'.
- يستخرج شهر الميلاد من الرقم المدني ويطبعه.
- يستخرج يوم الميلاد من الرقم المدني ويطبعه.
- يعرف سلسلتين ويجمعهما.
- يطبع تكرار رمز # عشر مرات.
- يطلب من المستخدم إدخال اسم آخر نبي، ويقارن مع 'Muhammad' ويطبع النتيجة (True/False).

Code

```

1  # تعريف سلسلة نصية (رقم مدني)
2  civil_id = '316012200641'
3
4  # استخراج الشهر باستخدام التقطيع
5  month = civil_id[3:5] # السلسلة النصية من الموقع 3 إلى 4
6  print("Month =", month)
7
8  # استخراج اليوم باستخدام التقطيع
9  day = civil_id[5:7] # السلسلة النصية من الموقع 5 إلى 6
10 print("Day  =", day)
11
12 # الجمع بين سلسلتين نصيتين
13 var_x = "Free"
14 var_y = "Kuwait"
15 print(var_x + " " + var_y) # الناتج: Free Kuwait
16
17 # التكرار باستخدام *
18 print("#" * 10) # #####
19
20 # المقارنة بين النصوص
21 prophet = "Muhammad"
22 prophet_input = input("Who is the last prophet? ")
23 print(prophet_input == prophet) # True إذا تطابق النصوص

```

Output

```

Month = 01
Day  = 22
Free Kuwait
#####
Who is the last prophet? Muhammad
True

```

"I love Kuwait, kuwait! Python 2025."

اكتب برنامجاً ينفذ ما يلي على الجملة السابقة:

- يطبع الجملة كامله بأحرف كبيرة (upper()).
- يطبع الجملة كامله بأحرف صغيرة (lower()).
- يستبدل كل ظهور لكلمة "Kuwait" إلى "الكويت" (بحالاتها المختلفة).
- يزيل علامات الترقيم البسيطة (!, ., ,) ثم يقسم الجملة إلى قائمة كلمات باستخدام split.
- يحسب عدد مرات ظهور كلمة "kuwait" بدون حساسية حالة الأحرف.
- يطلب من المستخدم إدخال البريد الإلكتروني ويفصله إلى اسم مستخدم ونطاق. باستخدام split('@')

مثال 30:

Code

```
1 # الجملة المحددة في نص السؤال (بدون إدخال من المستخدم)
2 s = "I love Kuwait, kuwait! Python 2025."
3
4 # تحويل الحالة (1)
5 print("UPPER :", s.upper()) # upper() يحول كل الحروف إلى كبيرة
6 print("lower :", s.lower()) # lower() يحول كل الحروف إلى صغيرة
7
8 # استبدال كلمة Kuwait (بأشكالها المختلفة) بـ "الكويت"
9 # ملاحظة: replace حساس لحالة الأحرف، لذا نطبق أكثر من شكل منها
10 s_replaced = (s.replace("Kuwait", "الكويت")
11               .replace("kuwait", "الكويت")
12               .replace("KUWAIT", "الكويت"))
13 print("Replaced ->", s_replaced)
14
15 # إزالة علامات الترقيم ثم التقسيم إلى كلمات (3)
16 clear = (s.replace(",", " ")
17          .replace(".", " ")
18          .replace("!", " ")) # remove بعض العلامات
19 words = clear.split() # split() تقسيم إلى قائمة كلمات
20 print("Words list:", words)
21
22 # عد تكرار "kuwait" بدون حساسية حالة الأحرف (4)
23 count_kw = (clear.lower().split()).count("kuwait")
24 print("Count of 'kuwait' (case-insensitive):", count_kw)
25
26 # مثال إضافي على split: تفكيك بريد إلكتروني يُدخله المستخدم (5)
27 email = input("Enter an email: ")
28 # split('@') يفصل جزأين: قبل @ وبعدها
29 if "@" in email:
30     user, domain = email.split("@", 1) # لضمان تقسيم مرة واحدة
31     print("Email user :", user)
32     print("Email domain :", domain)
33 else:
34     print("Invalid email format (no @ found)")
```

Output

```
UPPER : I LOVE KUWAIT, KUWAIT! PYTHON 2025.
lower : i love kuwait, kuwait! python 2025.
Replaced -> I love الكويت, الكويت! Python 2025.
Words list: ['I', 'love', 'Kuwait', 'kuwait', 'Python', '2025']
Count of 'kuwait' (case-insensitive): 2
Enter an email: student@example.com
Email user : student
Email domain: example.com
```

الشروط Conditions

بُنِيَّة تَحْكَم تستخدم لاتخاذ القرارات وتنفيذ التعليمات بناءً على تحقق شرط محدد، والتحكم في مسار البرنامج، وفق العمليات التالية:

عمليات المقارنة Comparison Operators

العمليات التي تقارن بين عاملين (متغيرات، قيم، ...)، والنتيجة تكون من نوع boolean إما True أو False.

المعامل	الوصف
==	يساوي
!=	لا يساوي
>	أصغر من
<	أكبر من
>=	أصغر من أو يساوي
<=	أكبر من أو يساوي

معاملات المقارنة

لاحظ:

معامل = يُستخدم لتخصيص قيمة للمتغير Assignment.

معامل == يُستخدم للمقارنة بين عنصرين Comparison.

مثال 1:

Code

```
1 # Assignment
2 var_number = 50
3 # Comparison
4 print(var_number == 70) # false
5 print(var_number == 50) # true
```

x	y	x == y	x != y	x < y	x > y	x <= y	x >= y
1	1	True	False	False	False	True	True
1	2	False	True	False	True	False	True
2	1	False	True	True	False	True	False

جدول نتائج عمليات المقارنة

مثال 2:

Code

```
1 x = 10
2 y = 5
3 print(x == y) # False
4 print(x != y) # True
5 print(x > y) # True
6 print(x < y) # False
7 print(x >= y) # True
8 print(x <= y) # False
```

Output

```
C:\Users\HP\PycharmProjects\
False
True
True
False
True
False
```

العمليات المنطقية Logical Operators

تُستخدم لإجراء عمليات منطقية على قيم المتغيرات:

- **العملية المنطقية and:** تُرجع True إذا كانت كلتا القيمتين المنطقيتين صحيحتين، وإلا تُرجع False.
- **العملية المنطقية or:** تُرجع True إذا كانت أي من القيمتين المنطقيتين صحيحتان، وإلا تُرجع False.
- **العملية المنطقية NOT:** تُرجع False إذا كانت القيمة المنطقية صحيحتان، وإلا تُرجع True.

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	
False	True	False	True	True
False	False	False	False	

جدول العمليات المنطقية

مثال 3:

Code

```
a = 5
b = 6
c = 7
print (a < b and b < c)      # True
print (a > b or b < c)      # True
print (a > b and b < c)     # False
print (a > b or b > c)     # False
print (not (a < b and b < c)) # False
print (not (a > b or b > c)) # True
```

التعليقات الشرطية Conditional Statements

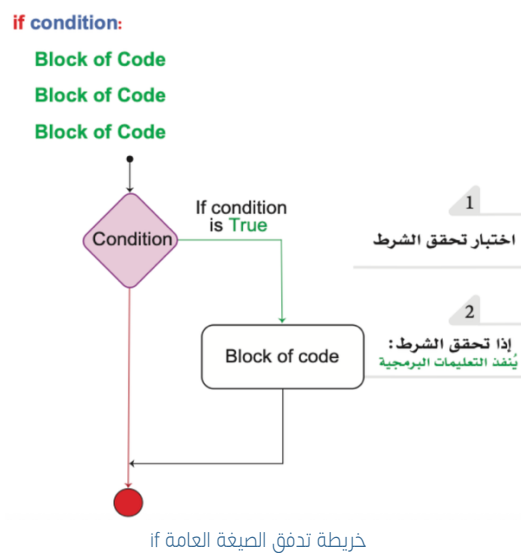
يمكن استخدام التعليقات الشرطية if و elif و else لاتخاذ القرارات وتنفيذ كتلة من التعليمات البرمجية بناءً على نتائج تلك القرارات.

الكتلة البرمجية (Block of code): هي سطر أو أكثر من التعليمات البرمجية، لها نفس المسافة البادئة (indentation).

التعليمة الشرطية if

تُستخدم التعليمة الشرطية if لاختبار شرط منطقي، وتنفيذ تعليمات برمجية إذا تحقق الشرط.

الصيغة العامة Syntax:



لاحظ:

يستخدم Python المسافة البادئة بينما تستخدم بعض لغات البرمجة الأخرى الأقواس والعبارات للدلالة على بداية ونهاية الكتل البرمجية.

تبدأ الكتلة البرمجية بعد علامة (:) colon وتنتهي عندما تنتهي المسافة البادئة Indentation.

Code

```

1  # برنامج يختبر ما إذا كانت الإجابة المُدخلة صحيحة بإظهار رسالة محددة.
2  password = input ("Enter your password: ")
3  if password == "admin":
4      print('Correct Password')
5
6  # برنامج يختبر ما إذا كانت الإجابة المُدخلة غير صحيحة بإظهار رسالة محددة.
7  prog_lang = input("What is your favorite programming language? ")
8  if prog_lang != "python":
9      print("Try Python. You'll love it.")
10
11 # برنامج يظهر تقدير الطالب (Excellent) عند حصوله على درجة الامتياز في أحد الاختبارات
12 student_score = int (input("Enter student score: "))
13 if student_score >= 90:
14     print ("Excellent")
15 print ("Good luck :) ")

```

Output

```

Enter your password: admin
Correct Password
What is your favorite programming language? C++
Try Python. You'll love it.
Enter student score: 93
Excellent
Good luck :)

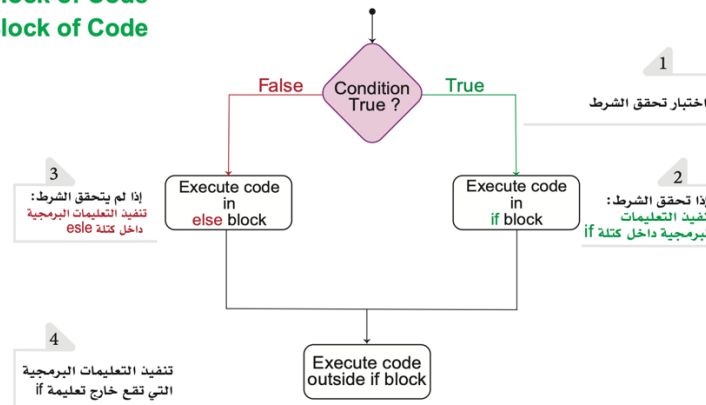
```

ملاحظات:

- عدم وجود مسافة بادئة للتعليمة البرمجية للسطر 15 يدل على أنها خارج نطاق الكتلة البرمجية للتعليمة الشرطية if.
- إذا تحقق الشرط سيطبع عبارة Excellent ثم عبارة Good luck.
- إذا لم يتحقق الشرط سيتجاهل تعليمة if، وينتقل للتعليمة البرمجية التالية print ليطبع عبارة (: Good Luck).

التعليمة الشرطية else

if condition:
Block of Code
Block of Code
else:
Block of Code
Block of Code



خريطة تدفق الصيغة العامة else

مثال 5:

Code

```

1  # برنامج يُظهر رسالة محددة في حال تحقق الشرط، وإظهار رسالة أخرى في حال عدم تحققه.
2  password = input('Enter your password: ')
3  if password == 'admin':
4      print('Correct Password')
5  else:
6      print('Wrong Password')
7
8  print("-----")
9
10 # برنامج يُظهر تقدير الطالب (Excellent) إذا الدرجة أكبر أو تساوي 90
11 score = int(input('Enter exam score: '))
12 if score >= 90:
13     print('You got an excellent grade')
14 else:
15     print('Work hard to get an excellent grade')
16
17 print("-----")
18
19 # برنامج يُظهر نتيجة الطالب (ناجح / راسب) بناءً على الدرجة المدخلة إذا كانت أكبر أو تساوي 50
20 score = int(input('Enter student score: '))
21 if score >= 50:
22     print('Congratulations, you succeeded!')
23 else:
24     print('Sorry, you failed')

```

Output

Enter your password: *admin*

Correct Password

Enter exam score: *97*

You got an excellent grade

Enter student score: *49*

Sorry, you failed

تطوير البرنامج السابق للتأكد من أن الدرجة المُدخلة تتراوح بين صفر و100، وتُظهر رسالة خطأ عند إدخال درجة غير ذلك

مثال 6:

Code

```
18 score = int(input('Enter student score: '))
19 if score >= 0 and score <= 100:
20     if score >= 50:
21         print('Congratulations, you succeeded!')
22     else:
23         print('Sorry, you failed')
24 else:
25     print('Wrong number!')
```

Output

Enter student score: *101*

Wrong number!

التعليمة الشرطية elif

وهي اختصار لجملته (else if) وتستخدم لإضافة شروط متعددة إلى التعليمة الشرطية if. ويتم تنفيذ النص البرمجي الموجود داخل كتلة التعليمات البرمجية إذا تحقق أحد هذه الشروط.

if condition1:

Block of code condition1 is true.

elif condition2:

Block of Code condition2 is true.

elif condition3:

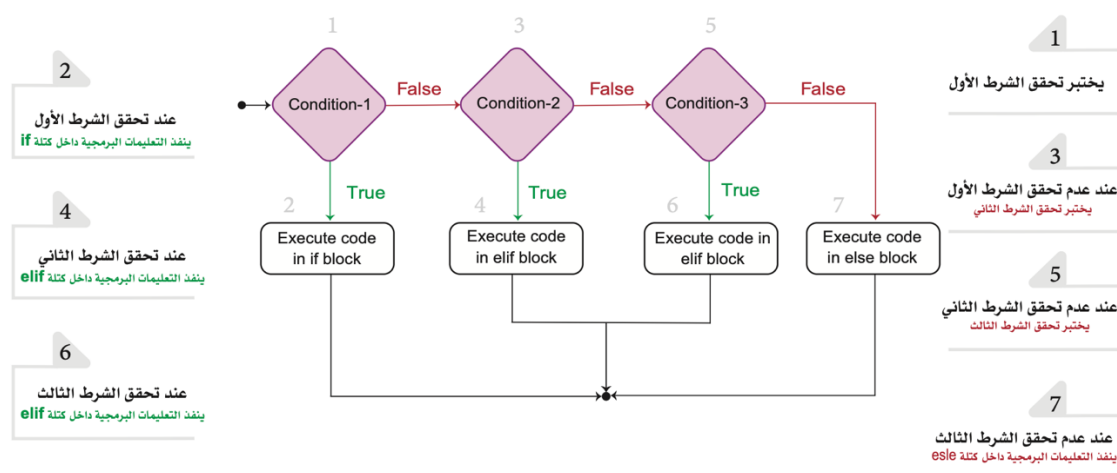
Block of Code condition3 is true.

...

else:

Block of Code all conditions are false.

الصيغة العامة Syntax:



خريطة تدفق الصيغة العامة elif...if

مثال 7:

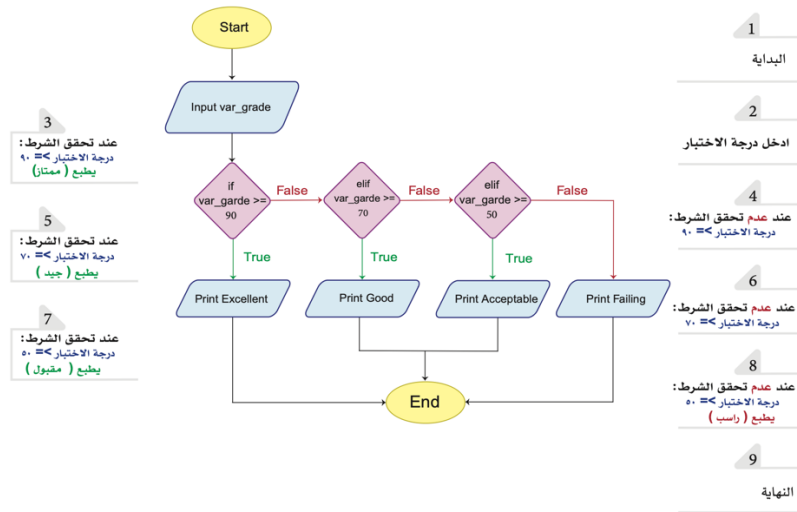
Code

```
1 #برنامج يظهر تقدير الطالب في أحد الاختبارات بناءً على الدرجة المخلطة.
2 student_score = int(input('Enter student score: '))
3
4 if student_score >= 90: # student_score >= 90 and student_score <= 100
5     print('Excellent grade !')
6 elif student_score >= 70: # student_score >= 70 and student_score <= 89
7     print('Good grade !')
8 elif student_score >= 50: # student_score >= 50 and student_score <= 69
9     print('Acceptable grade !')
10 else:
11     print('You have failed !')
```

Output

Enter student score: 70
Good grade !

في المثال السابق إذا تحقق الشرط الأول سينفذ البرنامج التعليمات الخاصة به ويتخطى اختبار باقي الشروط، وإذا لم يتحقق سيختبر الشرط الثاني، فإذا تحقق سينفذ التعليمات الخاصة به، وهكذا، وإذا كان ترتيب الشروط غير صحيح، يؤدي إلى نتائج غير صحيحة.



خريطة تدفق برنامج تقدير الطالب

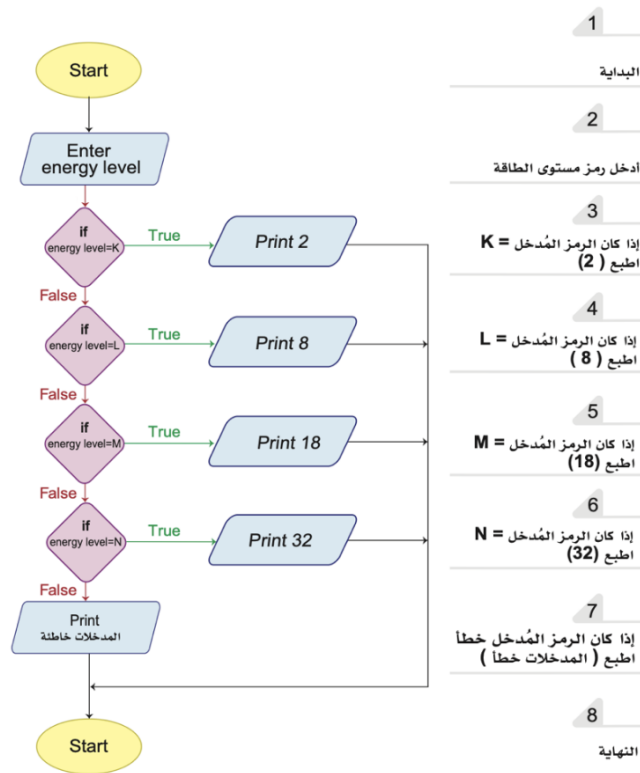
مثال 8:

من خلال دراستك لمادة الكيمياء، لمعرفة العدد الأقصى من الإلكترونات التي توجد في كل مستوى طاقة في الذرة من الجدول التالي:

الرقم مستوى الطاقة	الأول	الثاني	الثالث	الرابع
الرمز	K	L	M	N
أقصى عدد من الإلكترونات	2	8	18	32

العدد الأقصى من الإلكترونات التي توجد في كل مستوى طاقة في الذرة

بعد دراستك لخريطة التدفق التالية: اكتب برنامج يُظهر العدد الأقصى للإلكترونات حسب رمز مستوى الطاقة، ليعمل بشكل صحيح.



خريطة تدفق برنامج التعرف على العدد الأقصى للإلكترونات

Code

```

1 energylevel=input("Enter energy level code: " )
2 if energylevel=="K":
3     print("Energy level K ->",2)
4 elif energylevel=="L":
5     print("Energy level L ->", 8)
6 elif energylevel == "M":
7     print("Energy level M ->",18)
8 elif energylevel == "N":
9     print("Energy level N ->", 32)
10 else:
11     print("Invalid input.")
  
```

Output

```

Enter energy level code: N
Energy level N -> 32
  
```

طور البرنامج ليستقبل الحرف الأول من النص المُدخل ويحوّله إلى حرف كبير.

- صمم برنامج يحسب التكلفة النهائية لشراء منتجات، وفقًا لقائمة الخصومات المُعلنة.
- يستقبل من المستخدم السعر الإجمالي للمنتجات.
 - يتحقق من قيمة السعر الإجمالي:
 - إذا كان سعر المنتج أكبر من 1000: - احسب الخصم بنسبة 25%.
 - إذا كان السعر الإجمالي يتراوح ما بين 500 و 1000: - احسب الخصم بنسبة 15%.
 - إذا كان السعر الإجمالي أقل من 500: - لا يوجد خصم.
 - يحسب السعر النهائي بعد تطبيق الخصم.
 - يطبع السعر النهائي.

مثال 9: برنامج
حساب
الخصومات على
المنتجات

Code

```

1  #Get total price from the user
2  total_price = float(input("Enter the total price of the products: "))
3
4  #Determine the discount percentage
5  if total_price > 1000:
6      discount = 0.25
7  elif total_price >= 500:
8      discount = 0.15
9  else:
10     discount = 0.0
11
12 #Calculate the final price after discount
13 final_price = total_price - (total_price * discount)
14
15 # Print the final price
16 print("Final price after discount:", final_price)

```

Output

Enter the total price of the products: 1500
Final price after discount: 1125.0

صمم برنامج يتحقق من حالة رصيد حساب المستخدم:

- يستقبل من المستخدم قيمة الرصيد كرقم عشري.
- يتحقق من قيمة الرصيد:
 - إذا كان الرصيد يساوي صفرًا: طباعة "لا يوجد رصيد."
 - إذا كان الرصيد أكبر من صفر: طباعة "رصيد دائن [الرصيد]."
 - إذا كان الرصيد أقل من صفر: طباعة "رصيد مدين [الرصيد]."

مثال 10: برنامج

لمعرفة حالة

الرصيد الائتماني

Code

```
1 #Ask the user to enter their account balance
2 balance = float(input("Enter your account balance: "))
3
4 #Check the balance status
5 if balance == 0:
6     print("No balance.")
7 elif balance > 0:
8     print("Creditor balance: ", balance)
9 else:
10    print("Debtor balance: ", balance)
```

Output

Enter your account balance: 0

No balance.

صمم برنامج يستخدم المعاملات الحسابية (+ , - , * , /) في تنفيذ العمليات الحسابية (الجمع، الطرح، الضرب، القسمة).

1. يستقبل من المستخدم العدد الأول، ثم العدد الثاني.
2. يستقبل من المستخدم المعامل الحسابي المطلوب.
3. يستخدم الشروط **Conditions**:
 - إذا كان المعامل المُدخل +، يجمع العددين الأول والثاني.
 - إذا كان المعامل المُدخل -، يطرح العدد الثاني من العدد الأول.
 - إذا كان المعامل المُدخل *، يضرب العددين الأول والثاني.
 - إذا كان المعامل المُدخل /، يقسم العدد الأول من العدد الثاني.
4. يطبع ناتج العملية الحسابية.

مثال (II): برنامج
تصميم آلة
حاسبة

Code

```
1 #Conditions and math. operator
2
3 num1 = float(input("Enter First Number: "))
4 num2 = float(input("Enter Second Number: "))
5 operator = input("Enter Math. Operator(+, -, *, /): ")
6 if operator == '+':
7     print(num1+ num2)
8 elif operator == '-':
9     print (num1-num2)
10 elif operator == '*':
11     print(num1 * num2)
12 elif operator == '/':
13     if num2 != 0:
14         print(num1/num2)
15     else:
16         print ("Error Division by zero is not allowed.")
17
18 else:
19     print("Invalid operator")
```

Output

```
Enter First Number: 100
Enter Second Number: 2
Enter Math. Operator(+, -, *, /): /
50.0
```

- إضافة جزء للمعالجة (اجراء العملية الحسابية).
- تليها خطوة الطباعة لنتيجة العملية.

في المثال التالي إضافة متغير result ليتم تخزين ناتج العملية الحسابية.

Code

```
1  #Conditions and math. operator
2
3  num1 = float(input("Enter First Number: "))
4  num2 = float(input("Enter Second Number: "))
5  operator = input("Enter Math. Operator(+, -, *, /): ")
6  if operator == '+':
7      result = num1+ num2
8      print("result:",result)
9  elif operator == '-':
10     result = num1- num2
11     print("result:",result)
12  elif operator == '*':
13     result = num1* num2
14     print("result:",result)
15  elif operator == '/':
16     if num2 != 0:
17         result = num1/num2
18         print("result:", result)
19     else:
20         print ("Error Division by zero is not allowed.")
```

Output

```
Enter First Number: 10
Enter Second Number: 20
Enter Math. Operator(+, -, *, /): *
result: 200.0
```

صمم برنامج يطلب من المستخدم إدخال العمر، ثم يحدد سعر التذكرة كالتالي:

- أقل من 5 سنوات: مجاناً.
- من 5 إلى 11: نصف السعر (50%).
- من 12 إلى 17: خصم (25%).
- من 18 إلى 59: السعر كامل (0.4 د.ك.).
- من 60 فأكثر: خصم (40%).

مثال 12:

Code

```
1 # السعر الأساسي للتذكرة
2 BASE = 4.00
3
4 # إدخال عمر المستخدم
5 age = int(input("Enter your age: "))
6
7 # استخدام if / elif / else لتحديد السعر
8 if age < 5:
9     price = 0.0 # مجاني
10 elif 5 <= age <= 11:
11     price = BASE * 0.5 # نصف السعر
12 elif 12 <= age <= 17:
13     price = BASE * 0.75 # خصم 25%
14 elif 18 <= age <= 59:
15     price = BASE # السعر كامل
16 else:
17     price = BASE * 0.60 # خصم 40%
18
19 # إخراج السعر
20 print("Ticket price:", price, "KWD")
```

Output

```
Enter your age: 9
Ticket price: 2.0 KWD
```

- اكتب برنامج يحدد أهلية خصم الطلاب وفقاً لما يلي:
- يستقبل من المستخدم العمر وهوية الطالب (yes/no) وسداد الرسوم (yes/no) وموافقة ولي أمر الطالب إذا كان العمر $18 >$ (yes/no).
- شروط الأهلية:
- العمر بين 12 و 35.
 - موافقة ولي الأمر إذا كان العمر $18 >$.
 - وجود هوية الطالب وسداد الرسوم.
- اطبع تقريراً ورسالة مختصرة باستخدام الشرط المختصر.

مثال 13:

Code

```

1  # إدخال المستخدم
2  age = int(input("Enter age: "))
3  student_id = input("Student ID (yes/no): ").lower()
4  paid = input("Fees paid (yes/no): ").lower()
5  consent = input("Parental consent (yes/no): ").lower() if age < 18 else "yes"
6  # (1) فحص المدى
7  if not (0 <= age <= 120):
8      print("Invalid age range (0..120).")
9      eligible = False
10 else:
11     # تحويل إلى منطقية
12     has_id = (student_id == "yes")
13     has_paid = (paid == "yes")
14     has_consent = (consent == "yes")
15     # (2) شرط العمر للخصم
16     in_range = (12 <= age <= 35)
17     # (3) تداخل: موافقة ولي الأمر لمن هم دون 18
18     if age < 18:
19         consent_ok = has_consent
20     else:
21         consent_ok = True
22     # (4) المزج بين الشروط
23     eligible = in_range and consent_ok and has_id and has_paid
24     # تنبيهات مختصرة
25     if not in_range:
26         print("Age not in discount range [12..35].")
27     if age < 18 and not has_consent:
28         print("Parental consent required for minors.")
29     if not has_id:
30         print("Student ID is required.")
31     if not has_paid:
32         print("Fees must be paid.")
33     # (5) الشرط المختصر
34     message = "Eligible for student discount" if eligible else "Not eligible"
35     # تقرير موجز
36     print("\n--- Result ---")
37     print("Final eligibility:", eligible)
38     print("Message:", message)

```

Output

```
Enter age: 17
Student ID (yes/no): yes
Fees paid (yes/no): yes
Parental consent (yes/no): yes

--- Result ---
Final eligibility: True
Message: Eligible for student discount
```

التكرار Loop

هو تنفيذ مجموعة من التعليمات البرمجية بشكل متكرر. وهناك نوعان رئيسيان من التكرار وهما:

● الحلقة التكرارية: `while`.

● الحلقة التكرارية: `for`.

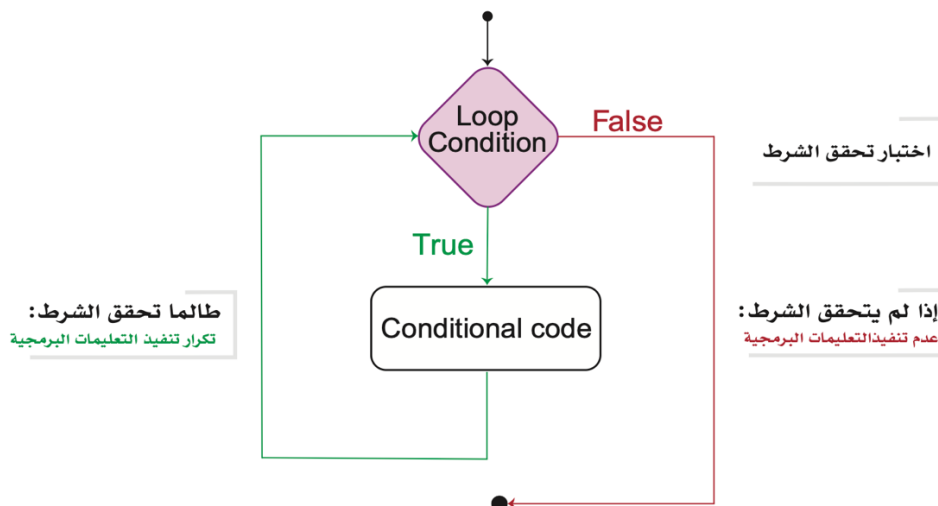
أولاً: الحلقة التكرارية: `while`.

تُستخدم لتكرار تنفيذ كتلة برمجية طالما تحقق شرط مُعين.

الصيغة العامة Syntax :

while condition:

code block to be executed until the condition becomes False.



خريطة تدفق الصيغة العامة while

مثال 1: استخدام الحلقة التكرارية while لطباعة الأعداد من 1 إلى 5:

Code

```
1  # تحديد المتغير
2  number = 1
3  # while مازال العدد أقل أو يساوي 5 شرط التحقق لتنفيذ الحلقة
4  while number <= 5:
5      print("Number is:", number)
6      # معادلة المتغير لضمان الوصول إلى شرط التوقف
7      number += 1
8  print("End Loop")
```

Output

```
Number is: 1
Number is: 2
Number is: 3
Number is: 4
Number is: 5
End Loop
```

لاحظ: كل من `number = number + 1` و `number += 1` يؤديان نفس الوظيفة، وهي زيادة قيمة المتغير بمقدار واحد.

عند استخدام الحلقة التكرارية **while** هناك نقاط يجب الانتباه لها:

- الشرط غالباً ما يتضمن متغير تم الإعلان عنه مسبقاً، ويخص له قيمة ابتدائية initial value.
- الخروج من التكرار: يتم الخروج من التكرار في حالتين:
 - عدم تحقق الشرط (تعديل قيمة متغير الشرط).
 - استخدام التعليمة البرمجية `break`.

مثال 2: التكرارات اللانهائية :Infinite loops

Code

تستخدم while True لتكرار تنفيذ التعليمات البرمجية إلى ما لا نهاية، على النحو التالي:

```
1 while True:
2     print ("Hello World!!")
```

Output

```
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!
```

- ستؤدي هذه التعليمات البرمجية إلى طباعة Hello World!! دون توقف.
- يمكن إيقاف تنفيذ التعليمة البرمجية باستخدام أداة Stop من شريط الأدوات.
- عدل التعليمات البرمجية في مثال 1 لطباعة قيمة المتغير number بصورة لا نهائية

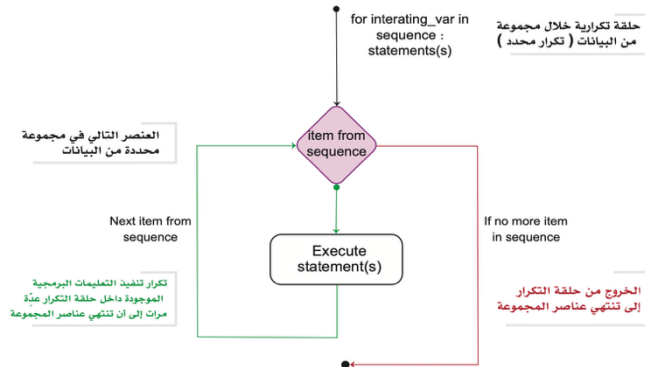
ثانياً: الحلقة التكرارية for:

تستخدم الحلقة التكرارية **for** للتكرار خلال مجموعة من البيانات (تكرار محدد).

الصيغة العامة Syntax:

for **variable** in iterable:

code block to be executed for each element.



خريطة التدفق للصيغة العامة for

قابل للتكرار **iterable**: كائن يمكن الانتقال عبر عناصره بشكل متعاقب.

السلاسل النصية Strings تعتبر كائنات قابلة للتكرار.

مثال 3:

التكرارات

اللانهاية

:Infinite loops

استخدم دالة input لإدخال الاسم ثم طباعة كل حرف من السلسلة النصية للمتغير في أسطر مستقلة.

Code

```
1 name = input('Enter your country name: ')
2 for ch in name:
3     print(ch)
```

Output

Enter your country name: *Kuwait*

K
u
w
a
i
t

يمكن استخدام الدالة `range()` لإنشاء سلسلة من الأعداد الصحيحة:

الصيغة	الوصف	مثال	عناصر السلسلة
<code>range(stop)</code>	الأعداد الصحيحة من صفر إلى أقل من <code>stop</code> بواحد	<code>range(10)</code>	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
<code>range(start, stop)</code>	الأعداد الصحيحة من <code>start</code> إلى أقل من <code>stop</code> بواحد.	<code>range(3, 8)</code>	3, 4, 5, 6, 7
<code>range(start, stop, -step)</code>	سلسلة من الأعداد الصحيحة من <code>start</code> إلى أقل من <code>stop</code> بزيادة <code>step</code> في كل مرة.	<code>range(1, 11, 2)</code>	1, 3, 5, 7, 9

مثال 4: استخدام الحلقة التكرارية `for` مع `range` لطباعة الأعداد من 0 إلى 3:

Code

```
1 for x in range(4):  
2     print(x)
```

Output

```
0  
1  
2  
3
```

مثال 5: استخدام الحلقة التكرارية `for` لطباعة الأعداد من 6 إلى 10:

Code

```
1 for x in range(6, 11):  
2     print(x)
```

Output

```
6  
7  
8  
9  
10
```

مثال 6:

لدينا مجموعة من الأرقام من (1 - 10) ، المطلوب استخدام التكرار لطباعة الأرقام الأكبر من 5:

Code	Output
1 <code>for number in range(1,11):</code>	6
2 <code>if number > 5:</code>	7
3 <code>print(number)</code>	8
	9
	10

مثال 7:

برنامج يطلب إدخال كلمة المرور حتى يتم مطابقتها مع كلمة المرور الصحيحة، ثم يعرض رسالة <Password Correct>.

Code	Output
1 <code>correctPassword = '1234'</code>	
2 <code>password = ' '</code>	
3 <code>while (password != correctPassword):</code>	
4 <code>password = input("Enter password: ")</code>	Enter password: 12
5 <code>print("Password Correct")</code>	Enter password: 1234
	Password Correct

حل آخر: باستخدام الحلقة التكرارية **for** ليتم تحديد عدد محاولات الإدخال بثلاث فقط:

Code

```
1  # الكلمة الصحيحة التي يجب على المستخدم إدخالها
2  correct_password = "1234"
3  # عدد المحاولات المسموح بها
4  for x in range(3):
5      password = input(" enter password : ")
6      if password == correct_password :
7          print (" password is correct ")
8      else :
9          print (" password is incorrect ")
10
```

Output

```
enter password : 555
password is incorrect
enter password : 1234
password is correct
enter password : 127
password is incorrect
```

ملاحظة: يكرر البرنامج طلب كلمة المرور من المستخدم حتى في حال إدخالها بشكل صحيح.

تطوير البرنامج بعدد لا نهائي من المحاولات.

Code

```
1  # تحديد كلمة المرور الصحيحة
2  correctpassword = "1234"
3  #تهيئة المتغير password سلسلة فارغة يتم تخزين كلمة المرور المدخلة من قبل المستخدم
4  password = ""
5  # التحقق من مطابقة كلمة المرور
6  while (password != correctpassword):
7      # إذا كانت كلمة المرور غير صحيحة، اطلب من المستخدم الإدخال مرة أخرى
8      password = input("Enter password : ")
9      # إذا تمت المطابقة، اطبع رسالة النجاح
10 print("Password Correct.")
```

Output

```
Enter password: 111
Enter password: 121
Enter password: 122
Enter password: 123
Enter password: 1234
Password Correct.
```

مثال 8: حل مثال 1 باستخدام الحلقة التكرارية for:

Code

```
1 # استخدام الحلقة for لتحديد المتغير ونطاق التكرار
2 for number in range(1, 6):
3     # طباعة الرقم
4     print("Number is :", number)
5     print("End Loop ")
```

Output

```
Number is: 1
Number is: 2
Number is: 3
Number is: 4
Number is: 5
End Loop
```

التكرارات المتداخلة nested loops:

تعلمت أن التكرار هو عملية إعادة تنفيذ مجموعة من التعليمات البرمجية أكثر من مرة، بدلاً من كتابتها. ويمكن أن تحتوي بداخلها على تكرارات أخرى.

مثال 9:

Code

```
1 x = 0
2 y = 0
3 for r in range(2):
4     x += 1
5     for c in range(3):
6         y += 1
7 print(x)
8 print(y)
```

Output

```
2
6
```

كم عدد مرات تخصيص المتغير x والمتغير y؟

التعليمة البرمجية break:continue

تُستخدم التعليمة البرمجية break و continue للتحكم في عملية التكرار.

التعليمة البرمجية break:

تُستخدم تعليمة break للخروج من الحلقة فورًا.

الصيغة العامة Syntax:

while condition:

loop code

if condition_to_break:

break

more loop code

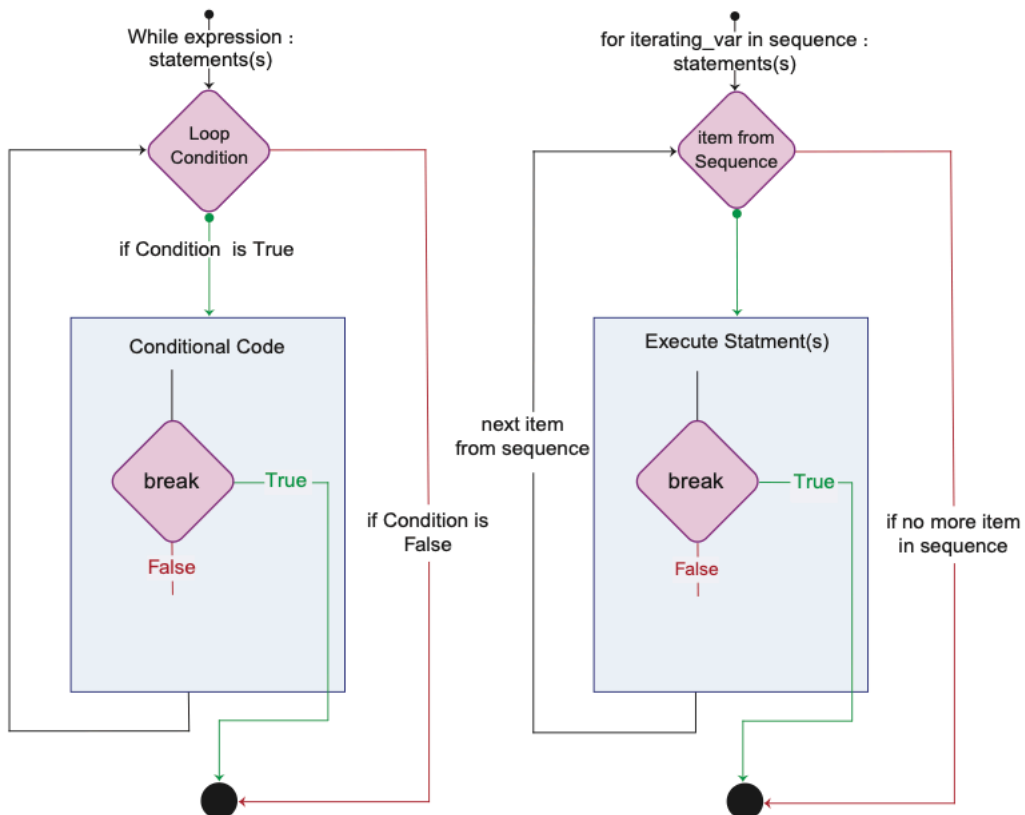
for variable in iterable:

loop code

if condition_to_break:

break

more loop code



خريطة تدفق الصيغة العامة break

- إذا كانت لدينا الحلقة التكرارية `while` تكرر من 0 إلى 10، ونريد الخروج من الحلقة عندما نصل إلى 5، يمكننا استخدام تعليمات `break` على النحو التالي:

مثال 10:

Code	Output
1 <i>i</i> = 0	0
2 while <i>i</i> <= 10:	1
3 print(<i>i</i>)	2
4 if <i>i</i> == 5 :	3
5 break	4
6 <i>i</i> += 1	5

سيؤدي هذا النص البرمجي إلى طباعة الأرقام من 0 إلى 5. عند الوصول إلى 5، ستؤدي تعليمة `break` إلى خروج الحلقة، ولن يتم طباعة أي شيء بعد ذلك.

التعليمة البرمجية `continue`:

تُستخدم تعليمة `continue` لتجاوز بقية النص البرمجي في الحلقة الحالية والانتقال إلى التكرار التالي:

الصيغة العامة Syntax:

while condition:

loop code

if condition_to_continue:

continue

more loop code

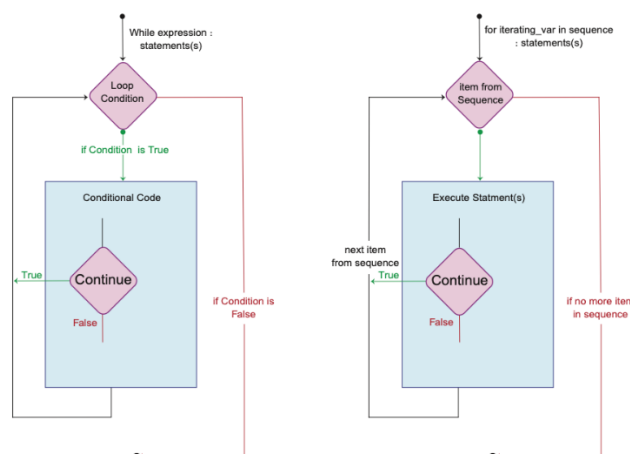
for variable in iterable:

loop code

if condition_to_continue:

continue

more loop code



خريطة تدفق `continue`

طباعة الأعداد الزوجية باستخدام الحلقة التكرارية :while

مثال 11:

Code

```
1 x = 0
2 while x < 10:
3     x+=1
4     if x%2==1:
5         continue
6     print (x)
```

Output

2
4
6
8
10

طباعة الأعداد الزوجية باستخدام الحلقة التكرارية :for

مثال 12:

Code

```
1 for x in range(1,11):
2     if x%2==1:
3         continue
4     print (x)
```

Output

2
4
6
8
10

الحلقة التكرارية :while ...else

يمكن تنفيذ الكتلة البرمجية else مع الحلقة التكرارية :while.

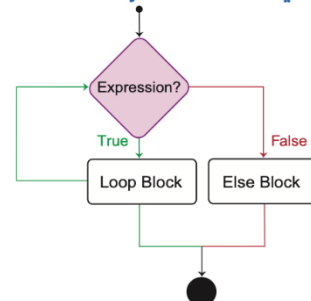
الصيغة العامة Syntax:

while condition:

loop body

else:

code to be executed when loop finishes-
normally



خريطة تدفق الصيغة العامة :while ...else

صمم برنامج يظهر رسالة خطأ عند محاولة إدخال المستخدم كلمة المرور ثلاث مرات بصورة خاطئة.

مثال 13:

Code

```
1 attempts = 1
2 correct_password = 'a1234'
3 while attempts < 4:
4     password = input("Enter password: ")
5     if password == correct_password:
6         break
7     attempts = attempts + 1
8 else:
9     print("You have entered the wrong password 3 times.")
10    exit() #Exit the program
```

Output

```
Enter password: abcd
Enter password: a1234567
Enter password: a1234
```

حل آخر:

Code

```
1 # تحديد كلمة المرور الصحيحة
2 correct_password = "1234"
3 # عدد المحاولات المسموح بها
4 attempts = 3
5 # تهيئة عدد المحاولات
6 attempt_count = 0
7 # التحقق من مطابقة كلمة المرور
8 while attempt_count < attempts :
9     # طلب ادخال كلمة المرور من المستخدم
10    password = input(" enter password : ")
11    # زيادة عدد المحاولات
12    attempt_count +=1
13    # التحقق من كلمة المرور
14    if password == correct_password :
15        print (" password is correct ")
16        break
17    else :
18        print (" All Ateempts finish ")
```

Output

```
enter password: 112
enter password: 668
enter password: 920
All attempts finish
```


صمم برنامج يخمن المستخدم عددا صحيحا ثم يقارنه البرنامج مع العدد المعلن عنه مسبقا، باتباع الخطوات التالية:

- يعلن عن متغير يساوي العدد المطلوب تخمينه.
- يستقبل عدد صحيح من المستخدم.
- يطابق العدد المدخل مع العدد المحدد في الحالات التالية:
 - إذا كان أكبر يطبع رسالة العدد أكبر ويستمر المستخدم في عملية الإدخال.
 - إذا كان أصغر يطبع رسالة العدد أصغر ويستمر المستخدم في عملية الإدخال.
- يطبع رسالة العدد المدخل مطابق للعدد المعرف وينتهي البرنامج.

مثال 14:

Code

```
1 correct_number = 8
2 player_number = 0
3 while player_number != correct_number:
4     player_number = int(input("Guess what's the number in my mind: "))
5     if player_number > correct_number:
6         print("The player number is higher.")
7     elif player_number < correct_number: #if the number less
8         print("The player number is lower.")
9 print("Correct! well done.")
```

Output

```
Guess what's the number in my mind: 9
The player number is higher.
Guess what's the number in my mind: 7
The player number is lower.
Guess what's the number in my mind: 8
Correct! well done.
```

مثال 15: تطوير البرنامج السابق لتوليد عدد عشوائي:

Code

```
1  # توليد رقم عشوائي من 1 - 30
2  import random
3  correct_number = random.randint(1,30)
4  # -----
5  player_number = 0
6  while player_number!= correct_number:
7      player_number = int (input("Enter a number between 1 and 30: "))
8      if player_number > 30 or player_number <1:
9          print("Please enter a number between 1 and 30")
10         continue
11     if player_number < correct_number:
12         print("The player number is low, Try again.")
13     elif player_number > correct_number:
14         print("The player number is high, Try again.")
15     else:
16         print("Congratulations! You guessed the correct number.")
```

Output

```
Enter a number between 1 and 30: 15
The player number is high, Try again.
Enter a number between 1 and 30: 10
The player number is low, Try again.
Enter a number between 1 and 30: 13
The player number is low, Try again.
Enter a number between 1 and 30: 14
Congratulations! You guessed the correct number.
```

- استقبال من المستخدم العدد المطلوب طباعة جدول الضرب الخاص به.
- اطبع رسالة (جدول ضرب العدد المُدخل).
- أنشئ حلقة تكرارية لطباعة ضرب العدد المدخل في الأعداد من 1 إلى 12.
 - اكتب التعليمات البرمجية لحساب:
الناتج = العدد المدخل * قيمة متغير التكرار.
- اطبع جدول الضرب.

**مثال 16: تصميم
برنامج يطبع
جدول الضرب
لعدد محدد.**

Code

```
1 #Ask the user to enter an integer number
2 number = int(input("Please enter an integer number: "))
3 #Print a message showing the table heading
4 print("Multiplication table for ", number)
5
6 #Use a loop to print the multiplication table from 1 to 12
7 for i in range(1, 13):
8     result = number * i
9     print(number, "x", i, "=", result)
```

Output

```
Please enter an integer number: 6
Multiplication table for 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
6 x 11 = 66
6 x 12 = 72
```

اكتب برنامج يطلب من المستخدم إدخال أعداد صحيحة متتالية (كل عدد في سطر) ثم يطبع done. ثم يحسب عدد القيم الصحيحة المدخلة ومجموعها ومتوسطها وأكبر قيمة وأصغر قيمة. يجب تجاهل الإدخالات الفارغة أو غير الصالحة مع رسالة تنبيه مختصرة (باستخدام continue).

مثال 17:

Code

```
1 # 'done' حتى إدخال كلمة 'done' باستخدام while
2 print("Enter integers one per line. Type 'done' to finish.")
3
4 count = 0 # عداد القيم الصحيحة
5 total = 0 # مجموع القيم الصحيحة
6 min_val = None # أصغر قيمة (سنهيئها عند أول إدخال صالح)
7 max_val = None # أكبر قيمة
8
9 while True: # حلقة لا نهائية مع شرط كسر داخلي
10     s = input("> ")
11
12     # شرط الإنهاء (1)
13     if s.lower() == "done":
14         break
15
16     # تجاهل الإدخالات الفارغة (2)
17     if s.strip() == "":
18         print("Empty input, ignored.")
19         continue
20
21     # محاولة تحويل الإدخال إلى int، وإن فشل نتابع (3)
22     try:
23         n = int(s) # قد يرفع ValueError إذا لم يكن عددًا صحيحًا
24     except ValueError:
25         print("Invalid integer, try again.")
26         continue
```

تابع / Code

```
27
28 # تحديث العداد والمجموع
29 count += 1
30 total += n
31
32 # تحديث الصغرى والكبرى (5)
33 if min_val is None or n < min_val:
34     min_val = n
35 if max_val is None or n > max_val:
36     max_val = n
37
38 # حساب المتوسط بحذر لتجنب القسمة على صفر (6)
39 if count > 0:
40     avg = total / count
41 else:
42     avg = 0
43
44 # طباعة تقرير نهائي (7)
45 print("\n--- Summary ---")
46 print("Count :", count)
47 print("Sum :", total)
48 print("Avg :", round(avg, 2))
49 print("Min :", min_val)
50 print("Max :", max_val)
```

Output

Enter integers one per line. Type 'done' to finish.

> 10

> 3

>

Empty input, ignored.

> hello

Invalid integer, try again.

> 25

> -2

> done

--- Summary ---

Count : 4

Sum : 36

Avg : 9.0

Min : -2

Max : 25

اكتب برنامج للتحقق من كلمة المرور، يمنح المستخدم 3 محاولات لإدخال الكلمة الصحيحة.
إذا أدخل الكلمة الصحيحة خلال المحاولات، اطبع رسالة نجاح و توقف.
إذا انتهت المحاولات دون نجاح استخدم جزء (while مع else) لطباعة رسالة "انتهت المحاولات".
اجعل كلمة المرور الصحيحة ثابتة مثل "python123".

مثال 18:

Code

```
1  # كلمة المرور الصحيحة (ثابتة)
2  SECRET = "python123"
3
4  # عدّاد المحاولات
5  attempt = 0
6  MAX_TRIES = 3
7
8  # حلقة while بمحاولات محدودة
9  while attempt < MAX_TRIES:
10     pwd = input("Enter password: ") # إدخال كلمة المرور
11     attempt += 1                     # زيادة العدّاد
12
13     if pwd == SECRET:                # تحقق من الصحة
14         print("Access granted.")    # نجاح
15         break                       # (لن يُنفَّذ else) خروج مبكر من الحلقة
16     else:
17         remaining = MAX_TRIES - attempt
18         if remaining > 0:
19             print("Incorrect. Tries left:", remaining)
20
21     # جزء else مع while:
22     # لا يُنفَّذ إلا إذا انتهت الحلقة طبيعيًا بدون break
23 else:
24     print("No tries left. Access denied.")
```

Output

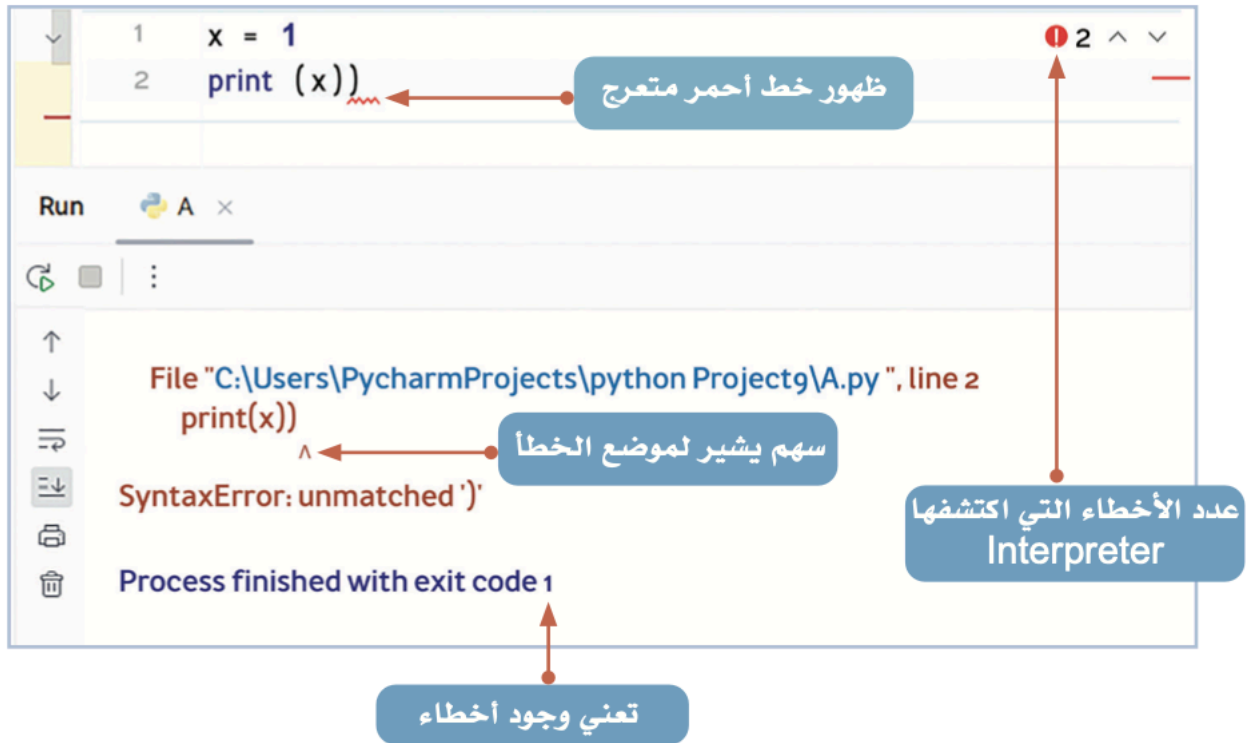
```
Enter password: abc
Incorrect. Tries left: 2
Enter password: python123
Access granted.

Enter password: 111
Incorrect. Tries left: 2
Enter password: 222
Incorrect. Tries left: 1
Enter password: 333
No tries left. Access denied.
```

تصحيح الأخطاء والاستثناءات Debugging and Exceptions

تعتبر الأخطاء والاستثناءات مفهوم أساسي في البرمجة، وتهدف إلى التعامل مع الأخطاء التي قد تحدث أثناء كتابة وتشغيل البرنامج بطريقة منظمة تمنع توقفه، وتتيح آلية معالجتها للمبرمجين التحقق من الأخطاء والتعامل معها بمرونة. كما تساعد في تحسين استقرار البرنامج وتقديم تجربة مستخدم User Experience أفضل من خلال عرض رسائل تنبيهية أو اتخاذ إجراءات بديلة دون تعطيل البرنامج بالكامل.

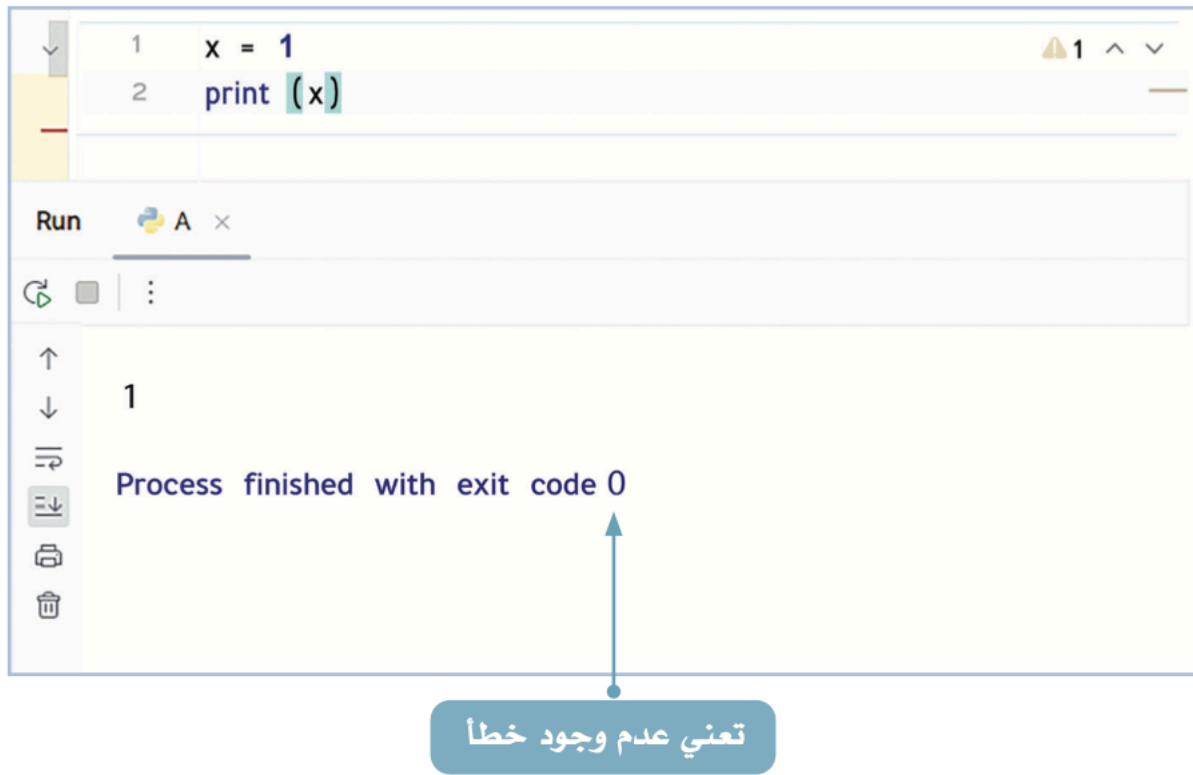
الأخطاء في التعليمات البرمجية في Python داخل بيئة PyCharm



الأخطاء في Python من خلال بيئة PyCharm

ملاحظة: يشير الرقم (2) إلى وجود خطأين في الكود:

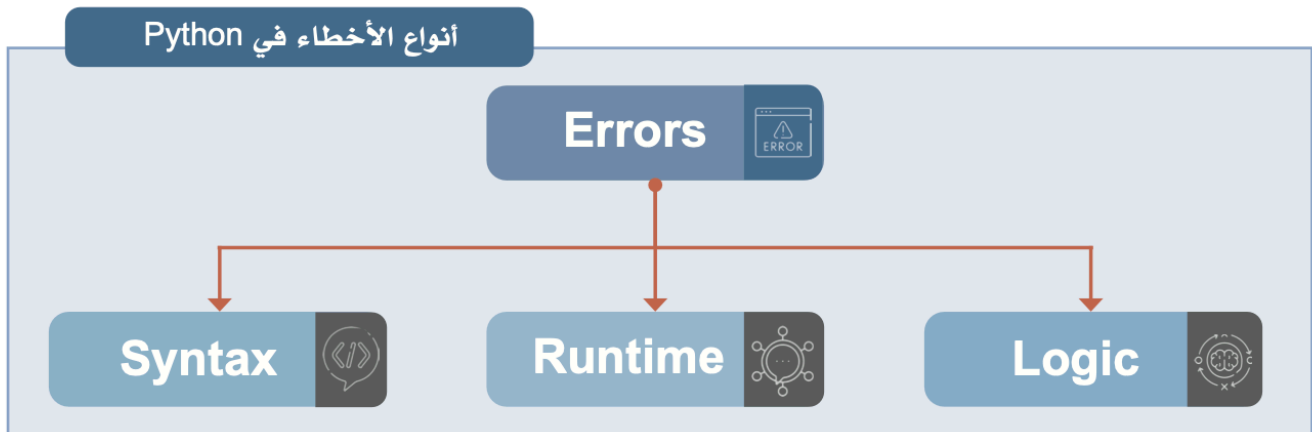
- وجود قوس زائد غير مطابق (في السطر print (x))، وهو ما تسبب في ظهور رسالة الخطأ SyntaxError: unmatched .
- خطأ في بناء الجملة البرمجية نتيجة القوس الزائد، مما جعل المفسر غير قادر على فهم تركيب الجملة بشكل صحيح.



تصحيح الأخطاء في Python من خلال بيئة PyCharm

تصحيح الأخطاء Debugging:

عند التعامل مع التعليمات البرمجية بلغة Python، قد نواجه بعض الأخطاء التي تؤثر على سير البرنامج. ولمعالجة هذه الأخطاء، يمكن استخدام مجموعة من التقنيات التي تساعد على اكتشافها وتصحيحها لضمان عمل البرنامج بشكل صحيح وتنقسم الأخطاء في Python إلى ثلاثة أنواع رئيسية، وهي:



1. الأخطاء اللغوية Syntax Errors:

أخطاء تحدث بسبب عدم الالتزام بقواعد كتابة التعليمات البرمجية، ولا يستطيع المفسر Interpreter فهم تلك التعليمات، مما يمنع البرنامج من العمل، (وهذه الأخطاء تظهر قبل بدء التنفيذ).

أخطاء في بناء الجملة :Syntax Errors

تحدث عند مخالفة قواعد كتابة التعليمة البرمجية، مثل نسيان الأقواس، أو النقطتين، أو علامات التنصيص، أو كتابة الكلمات المفتاحية بشكل خاطئ.

نوع رسالة الخطأ	مثال	نوع الخطأ
SyntaxError	if x > 0 print(x)	نسيان النقطتين (:) بعد الجملة الشرطية، أو مع الحلقات التكرارية For / While.
SyntaxError	print ("Hello, Kuwait!"	عدم إغلاق الأقواس.
IndentationError	for i in range (5): print (i)	أخطاء المسافات البادئة (Indentation Errors): بايثون تعتمد على المسافات البادئة لتحديد الكتل البرمجية. أي خطأ في ترتيب المسافات أو المزج بين المسافات والـ tab يؤدي إلى ظهور خطأ نحوي.

أمثلة على بعض الأخطاء في بناء الجملة

الخطأ: نسيان إغلاق القوس في دالة print() .

مثال 1:

Code

```
1 x = 5
2 y = 10
3 print(x + y
```

Output

```
File "C:\Users\HP\Documents\pythonpart 1\1-1.py", line 3
    print(x + y
          ^
SyntaxError: '(' was never closed
```


الخطأ: نسيان النقطتين (:) في الجمل الشرطية أو الحلقات (Loops).

مثال 2:

Code

```
1 if x > 5
2     print("x is greater than 5")
3
```

Output

```
if x > 5
      ^
SyntaxError: expected ':'
```

الخطأ في التعليمة البرمجية هو **Indentation Error**، ويحدث عندما يكون هناك مشكلة في المسافة البادئة (الإزاحة) في التعليمة البرمجية في Python.

مثال 3:

Code

```
1 if x > 5:
2     print("x is greater than 5")
```

Output

```
print("x is greater than 5")
      ^
IndentationError: expected an indented block after 'if' statement on line 1
```

2. أخطاء وقت التشغيل Runtime Errors:

- تظهر هذه الأخطاء أثناء تنفيذ البرنامج، رغم أن كتابته اللغوية سليمة، وغالبًا ما تحدث بسبب ظروف غير متوقعة.
- تظهر هذه الأخطاء أثناء تنفيذ البرنامج، بسبب مدخلات أو عمليات رغم أن التعليمات البرمجية مكتوبة بشكل صحيح.

من أشهر أخطاء وقت التشغيل Runtime Errors:

نوع الخطأ	مثال	رسالة الخطأ
محاولة قسمة عدد على صفر (ZeroDivisionError)	<code>print(5 / 0)</code>	<code>ZeroDivisionError: division by zero</code>
محاولة استخدام متغير غير معرف (NameError)	<code>x = 5</code> <code>print(z)</code>	<code>NameError: name 'z' is not defined</code>
إدخال نوع بيانات غير صحيحة (TypeError)	<code>print('5' + 10)</code>	<code>TypeError: can only concatenate str (not "int") to str</code>
إدخال قيمة غير مناسبة لنوع البيانات (ValueError)	<code>number = int('abc')</code>	<code>ValueError: invalid literal for int() with base 10: 'abc'</code>

أمثلة على بعض أخطاء وقت التشغيل

الخطأ وقت التشغيل: عندما يقوم المستخدم بإدخال قيمة عدد صحيح للمتغير الأول والثاني تتم عملية القسمة بشكل صحيح، ولكن عند ادخال قيمة 0 للمقسوم عليه (**var_y**) يحاول البرنامج تنفيذ عملية القسمة **var_x / var_y**، فيؤدي إلى خطأ في البرنامج بسبب القسمة على الصفر، وهو أمر غير مقبول رياضيًا. **النتيجة المتوقعة:** إذا لم يتم التعامل المستخدم مع هذا الخطأ بشكل مناسب بإدخال اعداد صحيحة (سيقوم البرنامج بإصدار **ZeroDivisionError**) وسيتوقف عن العمل.

مثال 4:
Zero Division Error

Code

```
1 var_x = int(input('Enter first number: '))
2 var_y = int(input('Enter second number: '))
3 var_z = var_x / var_y
4 print(var_z)
```

Output

```
↓ Enter first number: 6
Enter second number: 0
Traceback (most recent call last):
  File "C:\Users\HP\Documents\pythonpart 1\main.py", line 3, in <module>
    var_z = var_x / var_y
    ~~~~~^~~~~~
ZeroDivisionError: division by zero
```

الخطأ وقت التشغيل: استخدام متغير x و هو متغير غير مُعرف.
النتيجة المتوقعة: هنا سيحدث الخطأ وقت التشغيل عندما يكتشف المفسر أن المتغير لا يحتوي على قيمة.
يجب أن يكون هناك تعريف مسبق للمتغير x .

مثال 5:
Name Error

Code

```
1 print (x)
```

Output

Run: 1 x

Traceback (most recent call last):
File "C:\Users\ali\Desktop\python\daleel10-a1\.venv\1
print (x)
^
NameError: name 'x' is not defined

الخطأ وقت التشغيل: المتغير x هو سلسلة نصية (string)، والمتغير y هو عدد صحيح (integer). عملية الجمع + غير مدعومة بين هذين النوعين من البيانات، مما يؤدي إلى حدوث خطأ النوع (TypeError).
النتيجة المتوقعة: عملية الجمع + غير مدعومة بين هذين النوعين من البيانات، مما يؤدي إلى حدوث خطأ النوع (TypeError).
يجب عليك التأكد من أنك تستخدم العمليات المدعومة مع أنواع البيانات الصحيحة. في المثال أعلاه، يمكنك تحويل العدد الصحيح إلى سلسلة نصية قبل الجمع، أو يمكنك استخدام طريقة أخرى لمعالجة البيانات حسب الحاجة.

مثال 6:
Type Error

Code

```
1 x = "مرحبا"  
2 y = 5  
3 z = x + y  
4 print(z)
```

Output

Run: 1 x

Traceback (most recent call last):
File "C:\Users\ali\Desktop\python\daleel10-a1\.venv\1.py",
z = x + y
~~~~~  
TypeError: can only concatenate str (not "int") to str

الخطأ وقت التشغيل : "i" ليست عددًا صحيحًا.  
النتيجة المتوقعة: سيؤدي إلى ValueError لأن المستخدم أدخل المستخدم سلسلة نصية غير قابلة  
للتحويل إلى عدد صحيح.  
يجب كتابة قيمة قابلة للتحويل إلى عدد صحيح.

مثال 7 :  
Value Error

#### Code

```
1 int("i")  
2
```

#### Output

Run:  1 ×

```
C:\Users\ali\Desktop\python\daleel10-a1\.venv\Scripts\python  
Traceback (most recent call last):  
  File "C:\Users\ali\Desktop\python\daleel10-a1\.venv\1.py",  
    int("i")  
ValueError: invalid literal for int() with base 10: 'i'
```

### 3. الأخطاء المنطقية Logic Errors:

الأخطاء التي لا يتعرف عليها Python Interpreter، ولكنها قد تؤدي إلى نتائج غير متوقعة أو غير صحيحة (بمعنى أن التعليمات البرمجية تعمل بشكل صحيح، لكن نتيجة تنفيذها غير صحيحة). على سبيل المثال:

- استخدام شرط غير صحيح.
- استخدام العمليات الحسابية بشكل غير صحيح.

| نوع الخطأ                                                                                          | مثال خاطئ                                                                           | رسالة الخطأ                                                                                                                                                                                                | تصحيح الخطأ المقترح                                                                 |
|----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| خطأ في العمليات الحسابية<br><br>استخدام عملية رياضية غير مناسبة للهدف                              | length = 5<br>width = 4<br>rectangle_area = length + width<br>print(rectangle_area) | لا تظهر رسالة خطأ وتظهر النتيجة العدد 9<br><br><b>التفسير:</b><br>يقوم بجمع الطول والعرض، وهذا غير صحيح رياضياً لحساب مساحة المستطيل.                                                                      | length = 5<br>width = 4<br>rectangle_area = length * width<br>print(rectangle_area) |
| خلط في ترتيب العمليات الحسابية<br><br>عدم استخدام الأقواس مما يسبب تنفيذ العمليات بترتيب غير متوقع | a = b = c = 10<br>average = a + b + c / 3<br>print (average)                        | لا تظهر رسالة خطأ وتظهر النتيجة العدد 23.333333333333332<br><br><b>التفسير:</b><br>العمليات الحسابية لم تتبع أولوية التنفيذ لحساب المتوسط الحسابي.                                                         | a = b = c = 10<br>average = (a + b + c) / 3<br>print (average)                      |
| مقارنة منطقية خاطئة<br><br>خطأ في صياغة شرط if يؤدي لنتائج عكسية                                   | # Maximum final score = 100<br>score = 45<br>if score < 50:<br>print("Excellent")   | لا تظهر رسالة خطأ وتظهر النتيجة Excellent<br><br><b>التفسير:</b><br>الشرط score < 50 يعني أن الطالب راسب (أقل من 50)، ولكن الجملة التي ستطبع هي: "Excellent" (ممتاز) وهذا يتناقض مع نظام التقييم التعليمي. | # Maximum final score = 100<br>score = 45<br>if score < 50:print("Fail")            |

أمثلة على بعض الأخطاء المنطقية

**الخطأ المنطقي:** الشرط غير صحيح  $x < 5$  (وحسب البرنامج قيمة المتغير  $x$  يساوي 10 أي أكبر من 5 المذكورة في الشرط)، والبرنامج يطبع "x is greater than or equal to 5" وهو صحيح.

**مثال 8 :  
استخدام  
شرط غير  
صحيح**

**النتيجة المتوقعة:** إذا كان الهدف التحقق مما إذا كان  $x$  أقل من 10، يجب تعديل الشرط ليصبح  $x < 10$ .  
إذا كان الهدف هو التحقق مما إذا كان  $x$  أقل من 10، فيجب تعديل الشرط.

#### Code

```
1 x = 10
2 if x < 5:
3     print("x is less than 5")
4 else:
5     print("x is greater than or equal to 5")
```

#### Output

```
x is greater than or equal to 5
Process finished with exit code 0
```

**الخطأ المنطقي:** يفترض البرنامج يقوم بطباعة القيمة الأولية للمتغير `count` بشكل مستمر بدون توقف لأن المتغير لا يتم تحديثه، مما يؤدي إلى حلقة لا نهائية.  
**النتيجة المتوقعة:** كان يجب إضافة المتغير على أن يزيد بمعدل 1 في كل مره في الحلقة التكرارية  $count += 1$  ويتم تحديث المتغير في كل تكرار للحلقة.

**مثال 9:**

#### Code

```
1 count = 0
2 while count < 5:
3     print("Counting:", count)
4     # count += 1 # تم نسيان من قبل المبرمج تحديث المتغير
```

#### Output

```
Counting: 0
Counting: 0
Counting: 0
Counting: 0
Counting: 0
Counting: 0
```

بعد الضغط على أداة Stop أو Ctrl+F2 تظهر الشاشة التالية:

## Out put

```
File "C:\Users\HP\Documents\pythonpart 1\main.py", line 3, in <module>
    print("Counting:", count)
KeyboardInterrupt
```

### الخطأ المنطقي (Logic Error):

البرنامج يعمل بشكل صحيح، ولا يوجد أخطاء أثناء التنفيذ، ولكنه لا يعطي النتيجة المتوقعة بسبب خطأ في المنطق المستخدم في هذه الحالة، تم استخدام صيغة لحساب محيط المستطيل بدلاً من مساحة المستطيل.

النتيجة المتوقعة: الهدف هو حساب المساحة، فيجب تعديل الصيغة لتصبح:

( var\_area = var\_l \* var\_w )

مثال 10:

مقارنة منطقية  
خاطئة

## Code

```
1 var_l = 5
2 var_w = 6
3 # المطلوب حساب المساحة و العملية المحسوبة لقياس المحيط
4 var_area = 2* ( var_l + var_w)
5 # طباعة النتيجة
6 print('Area = ', var_area)
```

## Output

```
Area = 22
Process finished with exit code 0
```

## Code

```

1 x = input('Enter first number: ') # تخزين المدخل في المتغير x
2 if x == 0: # مقارنة x مع الصفر
3     print('x is equal to 0.')
4 else:
5     print('x is not equal to 0.')
```

## الخطأ المنطقي Logic Errors

- `x = input('Enter first number: ')` لم يتم تحويل المدخل في المتغير `x` من نص إلى قيمة عددية باستخدام أحد الدوال (`int`, `float`, ..).
- في السطر الذي يحتوي على جملة الشرطية `if x == 0`، يتم مقارنة المتغير `x` مع الرقم `0` المشكلة هنا هي أن المدخل `x` يتم استبداله باستخدام دالة `input()`، والتي تقوم بإرجاع القيمة كسلسلة نصية (`String`)، حتى لو أدخل المستخدم رقماً.
- عدم وجود : في عبارة `if`.
- عدم وجود مسافة بادئة في عبارة `print` الأولى.

## الاستثناءات Exceptions:

يمكن استخدام التعليمات البرمجية (`try/except`) لتحديد الجزء من البرنامج المتوقع حدوث خطأ به، لاستمرار تنفيذ البرنامج دون توقف.

تنقسم التعليمات البرمجية `try/except` إلى:

- `try`: تُستخدم كتلة `try` لكتابة التعليمة البرمجية الذي قد يسبب خطأ أثناء التنفيذ. إذا حدث خطأ في هذا الجزء، ينتقل البرنامج إلى كتلة `except`، أما إذا لم يحدث خطأ، فيستكمل تنفيذ البرنامج.
- `except`: تُستخدم كتلة `except` لمعالجة الأخطاء التي قد تحدث داخل `try`.
- `else`: تُنفذ كتلة `else` فقط إذا لم يحدث أي خطأ داخل `try`.
- `finally`: تُستخدم كتلة `finally` لتنفيذ أوامر يتم تنفيذها دائماً في نهاية البرنامج، سواء حدث خطأ في كتلة `try` أو لم يحدث.

## الصيغة العامة للاستثناءات Syntax:

```

try:
    # Code that might cause an error
except ErrorType:
    # Code to execute if an error occurs
else:
    # Code to execute if no error occurs
finally:
    # Code that always executes, whether an error occurred or not
```



## استخدام استثناءات محددة Specific Exceptions:

التعليمات البرمجية التالية تستخدم لحساب قيمة مشاركة كل طالب في تكلفة الرحلة المدرسية؛ وفق التالي:

### الكتلة try:

- استقبال تكلفة الرحلة.
- استقبال عدد الطلاب.
- حساب قيمة المشاركة.

### الكتلة except:

- عند حدوث خطأ من نوع ValueError، تعرض رسالة توضح بأن القيم المدخلة غير رقمية.
- عند حدوث خطأ من نوع ZeroDivisionError، (عدد الطلاب يساوي صفر) تعرض رسالة توضح بأنه لا يمكن إجراء القسمة على صفر.

### الكتلة else:

- عند نجاح جميع العمليات دون وقوع استثناء، يتم طباعة قيمة المشاركة.

### الكتلة finally:

- تنفذ دائماً بغض النظر عن وقوع أخطاء أم لا.
- تطبع رسالة تؤكد انتهاء تنفيذ البرنامج.

مثال 12:

#### Code

```
1  try:
2      print("School Trip Cost Sharing Calculator")
3      total_cost = float(input("Enter total trip cost (KWD): "))
4      students = int(input("Number of participating students: "))
5      share = total_cost / students
6  except ValueError:
7      print("Input error: Please enter valid numbers only!")
8  except ZeroDivisionError:
9      print("Error: Cannot divide by zero - please check student count")
10 else:
11     print(f"Each student's share: {share: .2f} KWD")
12 finally:
13     print("program ended!")
```

#### Output

```
School Trip Cost Sharing Calculator
Enter total trip cost (KWD): 1.5
Number of participating students: 0
Error: Cannot divide by zero - please check student count
program ended!
```

### استخدام استثناء شامل Catch-all Exceptions:

التعليمات البرمجية التالية تستخدم لحساب قيمة مشاركة كل طالب في تكلفة الرحلة المدرسية؛ وفق التالي:

#### الكتلة try:

- استقبال تكلفة الرحلة.
- استقبال عدد الطلاب.
- حساب قيمة المشاركة.
- طباعة قيمة المشاركة مباشرة.

#### الكتلة except:

- تلتقط أي استثناء يحدث من أي نوع عبر (Exception as e).
- تعرض رسالة عامة تحتوي على وصف الخطأ المخزن في المتغير e.

#### الكتلة else:

- عند نجاح جميع العمليات دون وقوع استثناء، يتم طباعة قيمة المشاركة.

#### الكتلة finally:

- تنفذ دائماً بغض النظر عن وقوع أخطاء أم لا.
- تطبع رسالة تؤكد انتهاء تنفيذ البرنامج.

مثال 13:

#### Code

```
1  try:
2      print("School Trip Cost Sharing Calculator")
3      total_cost = float(input("Enter total trip cost (KWD): "))
4      students = int(input("Number of participating students: "))
5      share = total_cost / students
6      print(f"Each student's share: {share: .2f} KWD")
7  except Exception as e:
8      print('Error is:', e)
9  finally:
10     print("Program ended!")
```

#### Output

```
School Trip Cost Sharing Calculator
Enter total trip cost (KWD): 10
Number of participating students: 0
Error is: float division by zero
Program ended!
```

## ملاحظات:

### 1. التعامل مع الاستثناءات Exceptions.

- يُفضل استخدام الاستثناءات المحددة (Specific Exceptions) مثل `ValueError` و `ZeroDivisionError`: لأنها تساعد في تحديد ومعالجة الخطأ بشكل واضح ومحدد
- يمكن استخدام الاستثناء الشامل (Catch-all Exception).

#### except Exception as e:

- تُستخدم لالتقاط ومعالجة جميع الأخطاء المحتملة، ويُفضل استخدام الاستثناء المحدد؛ لأنه يسهل تتبع وفهم الأخطاء.

### 2. التعامل مع كتلة else.

- تم الاستغناء عن الكتلة `else`، ونقل التعليمة التي كانت بداخلها إلى نهاية الكتلة `try`.

برنامج يستقبل أعداد صحيحة، ويُظهر رسالة تُحدد ما إذا كان العدد المدخل زوجيًا أم فرديًا، وعند إدخال بيانات نصية، يتم استثناء الخطأ وطباعة رسالة **You did not enter a valid number**، ويستمر البرنامج دون توقف.

مثال 14:

#### Code

```
1 while True:
2     try:
3         number = int(input('Enter a number:'))
4         if number % 2 == 0:
5             print('The number is even.')
6         else:
7             print('The number is odd.')
8     except ValueError:
9         print('You did not enter a valid number.')
```

#### Output

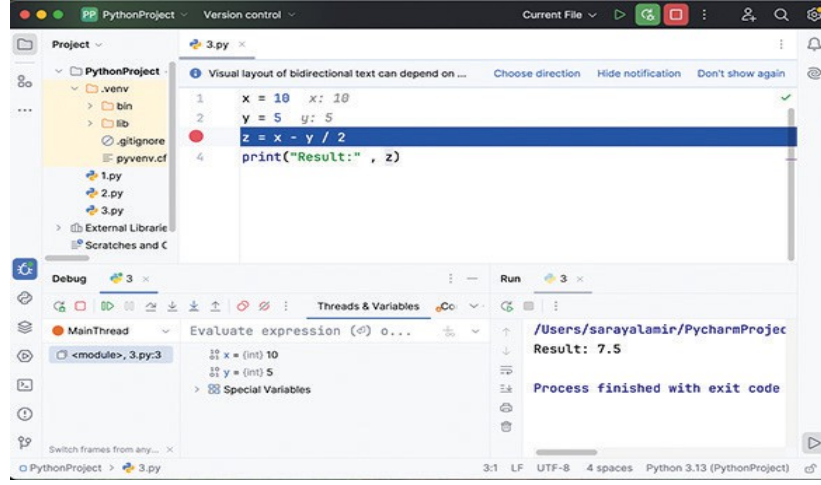
```
Enter a number:56
The number is even.
Enter a number:8.9
You did not enter a valid number.
Enter a number:GR11
You did not enter a valid number.
Enter a number:
```

## تصحيح الأخطاء Debugging:

عملية تتبع وتحليل التعليمات البرمجية خطوة بخطوة لاكتشاف الأخطاء، فهم سير التنفيذ، بهدف تصحيح النتائج غير المتوقعة.

## خطوات تتبع الأخطاء في PyCharm:

يتم تنفيذ الخطوات التالية بهدف تتبع وتحليل الأخطاء البرمجية وتصحيحها.



بعض أدوات تتبع الأخطاء

### 1. إضافة نقطة توقف (Breakpoint):

الضغط بجانب رقم السطر المطلوب إيقاف التنفيذ عنده، لتظهر دائرة حمراء (●) في محرر التعليمات البرمجية PyCharm. **لاحظ:** Breakpoint هي أداة نستخدمها لإيقاف البرنامج مؤقتاً عند سطر معين أثناء تنفيذه.

### 2. الضغط على أداة Debug:

- تنفيذ التعليمات البرمجية.
- توقف تنفيذ التعليمات البرمجية عند أول Breakpoint.
- ظهور شريط أدوات التصحيح Debug Toolbar.
- ظهور لوحة Threads Panel & Variables للقيام بالعديد من العمليات منها مراقبة المتغيرات وقيمها أثناء التصحيح.

### 3. التعامل مع أدوات التصحيح:

تُستخدم أدوات التصحيح لمتابعة تنفيذ التعليمات البرمجية خطوة بخطوة، أو للانتقال إلى السطر التالي، أو لإيقاف عملية التصحيح في أي وقت.



وهي مجموعة من الأدوات تُستخدم لتتبع تنفيذ البرنامج خطوة بخطوة، وتساعد على مراقبة القيم، واكتشاف الأخطاء المنطقية أو أخطاء أثناء التشغيل.

وتمكّن هذه الأدوات من إيقاف البرنامج مؤقتاً عند سطر معين (Breakpoint)، أو تنفيذ الأسطر واحداً تلو الآخر (Step Over/Into)، أو الدخول والخروج من الدوال، أو إعادة التشغيل، أو إنهاء التصحيح.

- **Resume Program**  : لاستكمال تشغيل البرنامج حتى نقطة التوقف التالية أو حتى النهاية.
- **Pause Program**  : لإيقاف تشغيل البرنامج مؤقتاً في أي لحظة.
- **Step Over**  : لتنفيذ السطر الحالي دون الدخول إلى داخل الدوال.
- **Step Into**  : للدخول داخل الدالة التي يتم استدعاؤها في السطر الحالي.
- **Step Out**  : للخروج من الدالة الحالية والعودة للسطر التالي في الدالة التي استدعتها.
- **Rerun**  : لإعادة تشغيل البرنامج من البداية بنفس إعدادات التصحيح.
- **Stop**  : لإيقاف جلسة التصحيح نهائياً.
- **View Breakpoints**  : لعرض جميع نقاط التوقف وإدارتها.
- **Mute Breakpoints**  : لتعطيل جميع نقاط التوقف مؤقتاً دون حذفها.

4. ظهور النتائج في Console:

بعد الانتهاء من تنفيذ البرنامج، تظهر النتائج أو المخرجات في نافذة Console.

### كيفية استخدام try/except :

- التعليمة البرمجية **try** تحتوي على التعليمات البرمجية الذي قد يحدث فيه خطأ.
- التعليمة البرمجية **except** تحتوي على التعليمات البرمجية الذي سيتم تنفيذه إذا حدث خطأ في الجزء الموجود داخل **try**.

```
Code
1 try:
2     x = 10 / 0 # ZeroDivisionError هذا سيؤدي إلى خطأ
3     print(x)
4     except ZeroDivisionError:
5         print("لا يمكن القسمة على صفر")
6 except:
7     print("حدث خطأ غير متوقع")
```

---

Out put

Run:  1 ×

  C:\Users\ali\Desktop\python\daleel10-a1\.venv\Scripts\python.exe  
 لا يمكن القسمة على صفر  
  Process finished with exit code 0

- سيتم تنفيذ كتلة `except ZeroDivisionError` لأن قسمة 10 على 0 تسبب في خطأ `ZeroDivisionError`.
- إذا تم تغيير الكود ليصبح `x = 10 / 2` ، فلن يتم تنفيذ أي من كتل `except` لأنه لن يحدث أي خطأ.

## تصحيح الأخطاء في بايثون Debugging :

تصحيح الأخطاء في بايثون (Debugging) هو عملية تحديد وإزالة الأخطاء من برنامج بايثون.

### خطوات تصحيح الأخطاء في بايثون:

1. تحديد نوع الخطأ: أول خطوة هي تحديد نوع الخطأ في البرنامج.
2. استخدام مصصح الأخطاء مثل:
  - PDB: يعد مصصح الأخطاء المدمج في بايثون (PDB) أداة قوية لتصحيح الأخطاء.
3. تتبع الأخطاء:
  - الخطوة الأولى: مثلاً استخدام PDB لتحديد مكان الخطأ في الكود.
  - الخطوة الثانية: فحص قيم المتغيرات في نقاط مختلفة من الكود لتحديد سبب الخطأ.
  - الخطوة الثالثة: إجراء التعديلات اللازمة على الكود لإصلاح الخطأ.
4. اختبار الأخطاء:
  - الخطوة الأولى: تشغيل الكود بعد إجراء التعديلات.
  - الخطوة الثانية: تأكيد أن الكود يعمل بشكل صحيح وأنه لا توجد أخطاء أخرى.

## نصائح لتصحيح الأخطاء:

- اجعل الكود بسيطاً : كلما كان الكود أبسط، كان من الأسهل تحديد الأخطاء.
  - استخدم التعليقات : التعليقات في الكود تساعد على فهمه وتسهيل عملية التصحيح.
  - اختبر الكود بشكل منتظم: لا تنتظر حتى يكتمل الكود لتصحيحه، اختبره بشكل دوري.
  - استخدم المصححات: استخدم PDB أو IDE لتصحيح الأخطاء، فهي أداة قوية.
  - ابحث عن الأخطاء الشائعة: تعرف على الأخطاء الشائعة في بايثون وحاول تجنبها.
- باستخدام هذه الأدوات والتقنيات، يمكنك أن تصبح مصحح أخطاء فعالاً في بايثون وتضمن تشغيل برامجك بسلاسة.

1. استخدم كتلة try لمحاولة تنفيذ الخطوات التالية:
  - a. اطلب من المستخدم إدخال price (السعر الأصلي).
  - b. اطلب من المستخدم إدخال discount (نسبة الخصم).
  - c. احسب السعر النهائي باستخدام:  
$$\text{net\_price} = \text{price} - (\text{price} * \text{discount} / 100)$$
  - d. اطبع "Net price after discount is: net\_price"
2. استخدم كتلة except بحيث إذا حدث خطأ في نوع البيانات (ValueError):  
اطبع : "Error: Please enter valid numeric values."
3. في جميع الأحوال، اطبع الرسالة داخل finally: "Program has ended."

مثال 16:  
حساب قيمة  
فاتورة  
المشتريات بعد  
الخصم.

## Code

```
discount.py x
1  try:
2      #Get the original price from the user
3      price = float(input("Enter the original price: "))
4
5      #Get the discount percentage from the user
6      discount = float(input("Enter the discount percentage (0 - 100): "))
7
8      #Calculate the net price
9      net_price = price - (price * discount / 100)
10
11     #Print the result
12     print("Net price after discount is:", net_price)
13
14 except ValueError:
15     #Handle invalid input
16     print("Error: Please enter valid numeric values.")
17
18 finally:
19     #Always print this message
20     print("Program has ended.")
```

## Output

```
"/Users/mirror/PycharmProjects/Grade 10/.venv/bin/python"
Enter the original price: 600g
Error: Please enter valid numeric values.
Program has ended.

Process finished with exit code 0
```

تطوير البرنامج:

## Code

```
discount_upgrade.py x
1  try:
2      #Get the original price from the user
3      price = float(input("Enter the original price: "))
4      #Check if price is valid
5      if price <= 0:
6          print ("Error: Price must be greater than zero!")
7      else:
8          #Get the discount percentage from the user
9          discount = float(input("Enter the discount percentage (0 - 100): "))
10         #Check if discount is valid
11         if discount < 0 or discount > 100:
12             print("Error: Discount must be between 0 and 100!")
13         else:
14             #Calculate the net price
15             net_price = price - (price * discount / 100)
16             #Print the result
17             print("Net price after discount is:", net_price)
18
19     except ValueError:
20         #Handle invalid input
21         print("Error: Please enter valid numeric values.")
22
23     finally:
24         #Always print this message
25         print("Program has ended.")
```

## Output

```
"/Users/mirror/PycharmProjects/Grade 10/.venv/bin/python"
Enter the original price: 1500
Enter the discount percentage (0 - 100): 105
Error: Discount must be between 0 and 100!
Program has ended.

Process finished with exit code 0
```



1. بداية التكرار اللانهائي وتنفيذ التالي:

#### الكتلة try:

- اطلب من المستخدم إدخال الوزن بالكيلو غرام وحفظه كعدد عشري.(float)
- اطلب من المستخدم إدخال الطول بالمتر وحفظه كعدد عشري.(float)
- احسب مؤشر كتلة الجسم BMI باستخدام المعادلة:
$$\frac{\text{الوزن بالكيلو غرام}}{\text{الطول بالمتر المربع}} = \text{(BMI)}$$
- اطبع قيمة الـ BMI بتنسيق عشري من خانتين.
- حدد الفئة الصحية بناءً على قيمة الـ BMI :
  - (a) إذا كان BMI أقل من 18.5، اطبع "You are underweight."
  - (b) إذا كان BMI بين 18.5 و 24.9، اطبع "You have a normal weight."
  - (c) إذا كان BMI بين 25 و 29.9، اطبع "You are overweight."
  - (d) إذا كان BMI غير ذلك اطبع "You are obese."

#### الكتلة except:

- إذا ادخل المستخدم قيمة غير رقمية (ValueError):
    - (a) طباعة رسالة: "Error: Please enter valid numeric values for weight and height."
  - إذا حدث خطأ وأدخل المستخدم الطول صفرًا (لا يمكن القسمة على صفر) (ZeroDivisionError)
    - (a) اطبع رسالة: "Error: You can't divide by zero!"
2. الرجوع إلى الخطوة 1.

مثال 17:

مؤشر كتلة

الجسم BMI

(Body mass

index).

## Code

```
BMI_Answer.py ×
1 # Program to calculate BMI with error handling
2 while True:
3     try:
4         # Get user input: weight and height
5         weight = float(input("Enter your weight in kilograms: "))
6         height = float(input("Enter your height in meters: "))
7         # Calculate BMI
8         bmi = weight / (height ** 2)
9         # Display the BMI
10        print(f"Your BMI is: {bmi:.2f}")
11        # Determine BMI category
12        if bmi < 18.5:
13            print("You are underweight.")
14        elif 18.5 <= bmi < 24.9:
15            print("You have a normal weight.")
16        elif 25 <= bmi < 29.9:
17            print("You are overweight.")
18        else:
19            print("You are obese.")
20
21    except ValueError:
22        print("Error: Please enter valid numeric values for weight and height.")
23    except ZeroDivisionError:
24        print("Error: You can't divide by zero!")
```

## Output

Enter your weight in kilograms: 70

Enter your height in meters: 1.5

Your BMI is: 31.11

You are obese.

Enter your weight in kilograms: 50

Enter your height in meters: 1.7

Your BMI is: 17.30

You are underweight.

Enter your weight in kilograms: 60

Enter your height in meters: 1.6

Your BMI is: 23.44

You have a normal weight.

Enter your weight in kilograms: |

برنامج آلة حاسبة بسيط يستقبل من المستخدم رقمين ثم يحدد نوع العملية الحسابية ثم يطبع النتيجة بعد التنفيذ، يحتوي هذا البرنامج على نوعين من الأخطاء:

مثال 18:

- الأخطاء النحوية (Syntax Errors).
- الأخطاء المنطقية (Logical Errors).

## Code

```
1 # آلة حاسبة بسيطة تجمع، تطرح، تضرب، وتقسم رقمين
2
3 # الحصول على الأرقام من المستخدم
4 num1 = input("أدخل الرقم الأول: ")
5 num2 = input("أدخل الرقم الثاني: ")
6
7 # اختيار العملية الحسابية
8 operation = input("أدخل العملية التي تريد القيام بها (+, -, *, /): ")
9
10 # تحويل الأرقام إلى float مع معالجة الأخطاء
11 try:
12     num1 = float(num1)
13     num2 = float(num2)
14 except ValueError:
15     print("خطأ: يرجى إدخال أرقام صحيحة.")
16     exit()
17
18 # إجراء العملية الحسابية وطباعة النتيجة
19 if operation == "+":
20     result = num1 - num2 # خطأ دلالي: استخدام الطرح بدلاً من الجمع
21 elif operation == "-":
22     result = num1 + num2 # خطأ دلالي: استخدام الجمع بدلاً من الطرح
23 elif operation == "*":
24     result = num1 * 2 # خطأ دلالي: يجب أن يكون num2 هو العدد المضروب
25 elif operation == "/":
26     if num2 == 0:
27         print("خطأ: لا يمكن القسمة على الصفر")
28     else:
29         result = num1 / num2 # خطأ دلالي: القسمة على نفس العدد بدلاً من num2
30 else:
31     print("خطأ: عملية غير صالحة")
32
33 # طباعة النتيجة
34 print("النتيجة:", result)
```

| رقم السطر | الخطأ                                                                                                                                                           | نوع الخطأ                               | التصحيح                                                                                                                                                 | ملاحظات                                                                    |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| 14        | نسيان النقطتين : بعد except<br>ValueError                                                                                                                       | الأخطاء النحوية<br>Syntax )<br>(Errors  | <b>except ValueError:</b>                                                                                                                               | عدم كتابة النقطتين :<br>بعد <b>except</b><br><b>ValueError</b>             |
| 34        | نسيان إغلاق القوس في نهاية<br>جملة الطباعة<br><b>print("النتيجة:", result)</b>                                                                                  | الأخطاء النحوية<br>Syntax )<br>(Errors  | <b>print("النتيجة:", result)</b>                                                                                                                        |                                                                            |
| 25        | القسمة على num2 بدون<br>التحقق مما إذا كان يساوي<br>صفرًا<br><b>elif operation:"/" ==</b><br><b>result = num1 /</b><br><b>num2</b>                              | الأخطاء المنطقية<br>(Logical<br>Errors) | <b>elif operation == "/"</b> :<br>if num2 == 0:<br><b>print (" خطأ: لا يمكن "</b><br><b>القسمة على الصفر</b> )<br>else:<br>result = num1 / num2         | إضافة شرط للتحقق<br>مما إذا كان<br>num2 يساوي<br>صفرًا قبل إجراء<br>القسمة |
| 31        | إذا كانت العملية المدخلة<br>( <b>operation</b> )صالحة إذا<br>أدخل المستخدم رمزًا غير<br>صالح<br><b>else:</b><br><b>print ("خطأ: عملية غير</b><br><b>صالحة")</b> |                                         | إضافة شرط if إضافي قبل الطباعة<br>للتحقق مما إذا كانت العملية صالحة<br>if operation not in:["/", "*", "-", "+"]<br><b>print("خطأ: عملية غير صالحة")</b> |                                                                            |
| 24        | في عملية الضرب، يتم ضرب<br>num1 بدلاً من 2 في<br>num2<br><b>elif operation == "*":</b><br><b>result = num1 * 2</b>                                              |                                         | <b>elif operation == "*":</b><br><b>result = num1 * num2</b>                                                                                            | استبدال 2 بـ<br>num2 لإتمام<br>عملية الضرب بشكل<br>صحيح                    |

## القوائم Lists

### البيانات غير الأولية Non-Primitive Data Structures:

تم عرض البيانات الأولية والبيانات غير الأولية، ومن البيانات غير الأولية الخطية Linear Data Structures:

1. القوائم Lists.

2. الصفوف Tuples.

### خصائص البيانات الخطية Linear Data Structures:

تتوافر هذه الخصائص في بعض أنواع البيانات الخطية:

- **الترتيب Ordering** : ترتيب العناصر مع كل عملية تشغيل Run.
- **قابلية للتغيير Mutability** : إمكانية تغيير عناصرها.
- **الفهرسة Indexing** : عنصر في القائمة يقابله رقم صحيح يمثل فهرس ذلك العنصر (فهرس العنصر الأول في القائمة = 0).
- **العناصر الفريدة Unique Elements** : إمكانية تكرار قيم البيانات.
- **أنواع مختلفة من البيانات Different of Data Types** : إمكانية توافر أنواع مختلفة من البيانات.

### 1. القوائم Lists:

تمثل مجموعة مرتبة من البيانات تحتوي على أي نوع من البيانات، (الأعداد الصحيحة والعشرية والسلاسل النصية والكائنات، ...)

| نوع البيانات<br>Data Type | قابلية للتغيير<br>Mutable | العناصر مرتبة<br>Ordered | الفهرسة<br>Indexing | العناصر<br>الفريدة<br>Unique Elements | أنواع مختلفة<br>من البيانات<br>Different of<br>Data Types | الصيغة<br>Syntax              |
|---------------------------|---------------------------|--------------------------|---------------------|---------------------------------------|-----------------------------------------------------------|-------------------------------|
| List                      | نعم                       | نعم                      | نعم                 | لا                                    | نعم                                                       | [ , , ]<br>Square<br>Brackets |

## إنشاء القوائم:

يتم إنشاء القوائم باستخدام الأقواس المربعة [ ]، ويفصل بين عناصرها علامة الفاصلة Comma ,.

### مثال 1:

إنشاء قائمة تحتوي العديد من أنواع البيانات مثل الأعداد الصحيحة والعشرية والسلاسل النصية.

```
main.py × + ...
main.py > ...
1 numbers = [1,2,3,4,5] # قائمة من الأعداد الصحيحة
2 decimals = [1.0,2.5,3.14] # قائمة من الأرقام العشرية
3 strings = ['Hello','world!','Iam','a Student'] # قائمة من السلاسل النصية
4 multi = ['Hello','world!','Iam',16,'years','old'] # قائمة من أنواع مختلفة من البيانات
```

### مثال 2:

إنشاء قائمة فارغة.

```
main.py × +
main.py > ...
1 list_1 = [ ]
```

### مثال 3:

إنشاء قائمة من عدة قوائم مختلفة.

```
main.py × + ... ↗
main.py > ... Format
1 name_degree = [['Fahed', 'Ahmed', 'Ali'], [13, 15, 20]]
```

## العمليات على القوائم:

### 1. فهرسة Indexing عناصر القائمة List.

`list_name[index]`

#### Code

```
1 x = [1,2,3,4,5, 'Hello', 'World']
2 print (x[0]) # الوصول إلى العنصر الأول
3 print (x[1]) # الوصول إلى العنصر الثاني
4 print (x[-1]) # الوصول إلى العنصر الأخير
```

#### Output

```
1
2
World
```

### 2. تعديل عناصر القائمة List.

#### Code

```
1 x = [2,3,4,5, 'Hello', 'World']
2 print (x)
3 x[0] = 10 # تغيير قيمة العنصر الأول
4 print (x)
```

#### Output

```
[2, 3, 4, 5, 'Hello', 'World']
[10, 3, 4, 5, 'Hello', 'World']
```

### 3. إضافة وتحرير عناصر القائمة List.

`list_name.append(value)`

#### Code

```
1 x = [10,3,4,5, 'Hello', 'Kuwait']
2 x.append(6) # إضافة العنصر 6 في نهاية القائمة
3 print(x)
4 x[-1] = "Kuwait" # تغيير قيمة العنصر الأخير
5 print (x)
```

#### Output

```
[10, 3, 4, 5, 'Hello', 'Kuwait', 6]
[10, 3, 4, 5, 'Hello', 'Kuwait', 'Kuwait']
```

## 4. حذف عناصر من القائمة List.

- حذف عنصر محدد حسب قيمته من القائمة.

### list\_name.remove(value)

#### Code

```
1 x = [10,3,4,5,'Hello','Kuwait', 1.5,-9]
2 x.remove('Hello') #حذف العنصر Hello من القائمة
3 print(x)
```

#### Output

```
[10, 3, 4, 5, 'Kuwait', 1.5, -9]
```

- حذف عنصر محدد حسب رقم indexing من القائمة.

### list\_name.pop(index)

#### Code

```
1 x = [10,2,3,4,5,'Hello','Kuwait', 1.5,-9]
2 x.pop(5)
3 print(x)
```

#### Output

```
[10, 2, 3, 4, 5, 'Kuwait', 1.5, -9]
```

- حذف العنصر الأخير من القائمة.

### list\_name.pop()

#### Code

```
1 x = [10,3,4,5,'Hello','Kuwait', 1.5,-9]
2 x.pop() #حذف العنصر الأخير من القائمة
3 print(x)
```

#### Output

```
[10, 3, 4, 5, 'Hello', 'Kuwait', 1.5]
```

### ملاحظة:

- لاحظ الفرق بين استخدام `pop` و `remove`.
- يمكن حذف جميع عناصر القائمة باستخدام دالة `clear()`.



## 5. الحصول على طول القائمة List – عدد عناصرها len().

### Code

```
1 countries = ['Kuwait','UAE','Bahrain','Oman', 'Saudi Arabia', 'Qatar']
2 print (len(countries))
```

### Output

6

مثال آخر:

### Code

```
1 numbers= [1,2,3,4,5]
2 print (len(numbers))
```

### Output

5

## 6. التأكد من وجود عنصر في القائمة List - if in :

### Code

```
1 fruits_list =['Apple', 'Orange', 'Banana', 'Strawberry','Grapes']
2 fruit_check = input ('Enter the name of the fruit: ')
3 if fruit_check in fruits_list:
4     print ('The fruit is in the fruits list.')
5 else:
6     print ('The fruit is Not in the fruits list.')
```

### Output

Enter the name of the fruit: *Orange*  
The fruit is in the fruits list.

## 7. تعبئة قائمة List من المستخدم.

- باستخدام تعليمة التكرار **while**:

### Code

```
1 students_list=[] #قيمة فارغة
2 students_name = 'stop' #القيمة الابتدائية للمتغير
3 while students_name != '': #عملية التكرار
4     students_name = input('Enter the name of the student: ') #طلب اسم الطالب
5     students_list.append(students_name) #إضافة الاسم إلى القائمة
6     students_list.pop() #حذف آخر عنصر (اسم) تم إدخاله
7 print(students_list) #طباعة القائمة
```

### Output

```
Enter the name of the student: Ameena
Enter the name of the student: Ahmed
Enter the name of the student: Ali
Enter the name of the student:
['Ameena', 'Ahmed', 'Ali']
```

**ملاحظة:**

- لاحظ سبب استخدام **pop()** في المقطع البرمجي السابق.

## 8. التعامل مع عناصر القائمة List من خلال عملية التكرار.

**طباعة جميع عناصر القائمة باستخدام for،**

### Code

```
1 animals = ['cat', 'tiger', 'lion']
2 for animal in animals:
3     print (animal)
```

### Output

```
cat
tiger
lion
```

**طباعة جميع عناصر القائمة باستخدام الفهرسة (موقع العنصر في القائمة).**

### Code

```
1 animals = ['cat', 'tiger', 'lion']
2 for n in range (len(animals)):
3     print (animals[n])
```

### Output

```
cat
tiger
lion
```

## 9. نسخ قائمة List:

يمكن نسخ القائمة باستخدام دالة `copy()`، مع ملاحظة أن التغيير في محتوى القائمة الجديدة لا يغير في محتوى القائمة الأصلية.

### Code

```
1 list = ['Ahmed', 'Ali', 'Mohammad']
2 new_list = list.copy()
3 new_list[1] = 'Ibrahim'
4 print(list)
5 print(new_list)
```

### Output

```
['Ahmed', 'Ali', 'Mohammad']
['Ahmed', 'Ibrahim', 'Mohammad']
```

مثال آخر:

### Code

```
1 original_list = ['Ahmed', 'Ali', 'Mohammad', 'Mona', 'Bader']
2 equal_list = original_list
3 copy_list = original_list.copy()
4 original_list.pop()
5 print(original_list)
6 print(equal_list)
7 print(copy_list)
```

### Output

```
['Ahmed', 'Ali', 'Mohammad', 'Mona']
['Ahmed', 'Ali', 'Mohammad', 'Mona']
['Ahmed', 'Ali', 'Mohammad', 'Mona', 'Bader']
```

## 10. عمليات الفرز على القوائم.

تستخدم لترتيب العناصر في القائمة.

`List_name.sort()`

فرز القائمة حسب الأعداد الصحيحة:

### Code

```
1 numbers = [2,1,4,3,5]
2 numbers.sort()
3 print(numbers)
```

### Output

```
[1, 2, 3, 4, 5]
```

### فرز القائمة حسب الأعداد العشرية:

#### Code

```
1 decimals = [1.5, 2.5, 3.14, 1.0]
2 decimals.sort()
3 print(decimals)
```

#### Output

```
[1.0, 1.5, 2.5, 3.14]
```

### فرز القائمة حسب السلاسل النصية:

#### Code

```
1 strings = ['c', 'a', 'B']
2 strings.sort()
3 print(strings)
```

#### Output

```
['B', 'a', 'c']
```

### ملاحظة:

- ظهور حرف B في بداية القائمة حيث يتم ترتيب الحروف Capital ثم ترتيب الحروف Small.
- يتم الترتيب فقط في حال كان نوع البيانات داخل القائمة من نفس النوع.

## 11. تقسيم السلسلة النصية إلى قائمة من العناصر.

### List\_name.split()

- تقسيم السلسلة النصية إلى قائمة من السلاسل النصية على أساس علامة ( \_ ).

#### Code

```
1 x = 'Cybersecurity_Python_Artificial_Intelligence'
2 print(x.split('_'))
```

#### Output

```
['Cybersecurity', 'Python', 'Artificial', 'Intelligence']
```

- يمكن استخدام الخاصية maxsplit لتقسيم القائمة لسلاسل نصية بعدد محدد.

#### Code

```
1 string = 'Learn*Python*with*Computer Technical Supervision'
2 split_string = string.split ("*",3)
3 print (split_string)
```

#### Output

```
['Learn', 'Python', 'with', 'Computer Technical Supervision']
```

12. تجميع قائمة من السلاسل النصية في سلسلة نصية واحدة (join()).

### string.join(iterable)

#### Code

```
1 x = ['Hello', 'Cybersecurity!']
2 print(' '.join(x))
3 print('_'.join(x))
```

#### Output

```
Hello Cybersecurity!
Hello_Cybersecurity!
```

### بعض الدوال التابعة للكائن list:

- list.extend () - إضافة مجموعة من العناصر إلى نهاية القائمة.
- list.insert () - إضافة عنصرًا إلى القائمة في الموقع الذي يحدده المستخدم.
- list.index() - تحديد موقع العنصر ضمن القائمة.
- list.count () - تحديد عدد مرات تكرار العنصر الذي يحدده المستخدم في القائمة.
- list.reverse() - قلب ترتيب عناصر القائمة في مكانها.

لديك قائمة تضم أسماء الطلاب وقائمة تضم درجات الطلاب المطلوب طباعة أسماء الطلاب الناجحين (الحاصلين على درجة أكبر من 50).

مثال 4:

#### Code

```
1 names = ['Ahmad', 'Ali', 'Muhammad']
2 degree = [70,90,30]
3 for x in degree:
4     if x>50:
5         print(names[degree.index(x)])
```

#### Output

```
Ahmad
Ali
```

### مثال 5:

لديك قائمة تتضمن درجات الطلاب مطلوب طباعة أقل / أكبر درجة بالقائمة.

#### Code

```
1 degrees = [70,80,50,90,95,30,45]
2 degrees.sort()
3 print ("min = ", degrees[0])
4 print("max = ", degrees[-1])
```

#### Output

```
min = 30
max = 95
```

### مثال 6:

لديك قائمة تتضمن اسم الطالب بالكامل، اطبع اسم الطالب بالكامل.

#### Code

```
1 student_name = ['Abdulrahman', 'Ahmed', 'Ali']
2 print(" ".join(student_name))
```

#### Output

```
Abdulrahman Ahmed Ali
```

### مثال 7:

احسب مجموع الدرجات والمتوسط الحسابي بعد إدخال مجموعة من درجات الطلاب.  
ملاحظة: لمعرفة انتهاء عملية إدخال الدرجات من خلال إدخال رقم سالب.

#### Code

```
1 degrees = [] # قائمة لتخزين الدرجات
2 d = 0 # قيمة ابتدائية للمتغير
3 # تستمر الحلقة طالما المدخل غير سالب
4 while d >= 0:
5     d = int(input("enter degree: "))
6     degrees.append(d)
7 degrees.pop() # إزالة آخر عنصر (لأنه رقم سالب أوقف الحلقة)
8 print(degrees)
9 # حساب المجموع
10 sum = 0
11
12 for d in degrees:
13     sum = sum + d
14 print("sum =", sum)
15 # حساب المتوسط مع معالجة القسمة على صفر
16 try:
17     average = sum / len(degrees)
18     print("average =", average)
19 except ZeroDivisionError:
20     print("Error: No valid degrees entered (cannot divide by zero).")
```

## Output

```
enter degree: 50
enter degree: 60
enter degree: 100
enter degree: -2
[50, 60, 100]
sum= 210
average= 70.0
```

مثال 8:

لديك سلسلة تتضمن مجموعة من درجات الطلاب، المطلوب حذف جميع الدرجات الأقل من 50 في السلسلة.

## Code

```
1 degrees = [20, 90, 50, 70, 30, 75, 18, 20, 100]
2 degreesNew = []
3 for d in degrees:
4     if d >= 50:
5         degreesNew.append(d)
6 degrees = degreesNew
7 print(degrees)
```

## Output

```
[90, 50, 70, 75, 100]
```

حل آخر:

## Code

```
1 degrees = [20, 90, 50, 70, 30, 75, 18, 20, 100]
2 n=0
3 while n < len(degrees):
4     if degrees[n] <50:
5         degrees.remove (degrees[n])
6     else:
7         n+=1
8 print(degrees)
```

## Output

```
[90, 50, 70, 75, 100]
```

## 2. الصفوف Tuples.

هي أحد أنواع البيانات الأساسية في بايثون. وهي عبارة عن صفوف مرتبة من البيانات يمكن أن تحتوي على أي نوع من البيانات، بما في ذلك الأعداد الصحيحة والأرقام العشرية والسلاسل النصية والكائنات.

| نوع البيانات<br>Data Type | قابلة للتغيير<br>Mutable | العناصر مرتبة<br>Ordered | الفهرسة<br>Indexing | العناصر الفريدة<br>Unique Elements | أنواع مختلفة من البيانات<br>Different of Data Types | الصيغة<br>Syntax   |
|---------------------------|--------------------------|--------------------------|---------------------|------------------------------------|-----------------------------------------------------|--------------------|
| Tuple                     | لا                       | نعم                      | نعم                 | لا                                 | نعم                                                 | ( )<br>Parentheses |

لذلك أي مجموعة من البيانات التي نحتاج إلى عدم تغير قيمتها بالخطأ أثناء تنفيذ البرنامج يستخدم tuple  
مثال أيام الأسبوع الأحد – الاثنين - ... يفضل حفظها في tuple.

```
1 weekdays = ('Sunday', 'Monday', 'Tuesday',
2             'Wednesday', 'Thursday', 'Friday', 'Saturday')
```

اكتب برنامج لتحديد العناصر الكيميائية المختلفة بتنفيذ التالي:

- الإعلان عن قائمة بأسماء العناصر الكيميائية باسم element\_name.
- الإعلان عن قائمة برموز العناصر الكيميائية باسم element\_symbol بنفس ترتيب القائمة الأولى.
- الإعلان عن متغير باسم degree وقيمه الابتدائية صفر لحساب درجة الطالب.
- الإعلان عن متغير باسم es لاستقبال رموز العناصر من المستخدم، وتكون قيمته الابتدائية فراغ.
- إنشاء حلقة تكرارية بعدد عناصر القائمة بحيث:
  - يظهر عنصر من القائمة element\_name ويطلب من المستخدم إدخال رمزه.
  - يقارن البرنامج بين الرمز المدخل والقائمة element\_symbol.
  - إذا كانت الإجابة صحيحة → يطبع "Correct" ويزيد 1 إلى قيمة المتغير degree.
  - إذا كانت الإجابة خاطئة → يطبع "Error" بدون تغيير قيمة degree.
- بعد انتهاء الحلقة يظهر الدرجة الكلية للطالب.

مثال 9:

تحديد رموز

العناصر

الكيميائية

المختلفة.

| اسم العنصر الكيميائي | Sodium | Potassium | Copper | Iron | Mercury | Lead |
|----------------------|--------|-----------|--------|------|---------|------|
| الرمز                | Na     | K         | Cu     | Fe   | Hg      | Pb   |

قائمة ببعض العناصر الكيميائية ورموزها



## Code

```
1 # القوائم
2 element_name = ['Sodium', 'Potassium', 'Copper', 'Iron', 'Mercury', 'Lead']
3 element_symbol = ['Na', 'K', 'Cu', 'Fe', 'Hg', 'Pb']
4 # المتغيرات
5 x = 0
6 es = ""
7 degree = 0
8 # حلقة التكرار
9 while x < len(element_name):
10     es = input("Enter element symbol " + element_name[x] + ": ")
11
12     if es == element_symbol[x]:
13         print("Correct")
14         degree += 1
15     else:
16         print("Error الصحيح هو:", element_symbol[x])
17     x += 1 # زيادة العداد
18 # طباعة الدرجة النهائية
19 print("الدرجة:", degree, "/", len(element_name))
```

## Output

```
Enter element symbol Sodium: Na
Correct
Enter element symbol Potassium: P
Error الصحيح هو: K
Enter element symbol Copper: Co
Error الصحيح هو: Cu
Enter element symbol Iron: Fe
Correct
Enter element symbol Mercury: Hg
Correct
Enter element symbol Lead: Pb
Correct
الدرجة: 6 / 4
```

اكتب برنامج لحساب المتوسط الحسابي باتباع التالي:

- أعلن عن قائمة فارغة باسم `numbers_list`.
- أعلن عن متغير باسم `sum` وعيّن قيمته الابتدائية صفر.
- أعلن عن متغير باسم `input_number` وعيّن قيمته الابتدائية صفر لاستقبال الأعداد.
- أنشئ حلقة تكرارية تستمر طالما كان المدخل أكبر من أو يساوي صفر، وبداخلها:
  - استقبل عددًا من المستخدم.
  - إذا كان أكبر من أو يساوي صفر:
    - أضف العدد إلى عناصر القائمة.
    - أضف العدد إلى قيمة المتغير `sum`.
  - وإلا (إذا كان سالبًا) أوقف الحلقة.
- احسب المتوسط الحسابي `sum =` مقسومًا على طول القائمة `numbers_list`.
- اطبع:
  - عبارة: "الأعداد التي قمت بإدخالها: " متبوعة بعناصر القائمة.
  - عبارة: "المتوسط الحسابي: " متبوعة بالقيمة التي تم حسابها.

مثال 10:

حساب  
المتوسط  
الحسابي

## Code

```
1 # 1) Empty list
2 numbers_list = []
3 # 2) Sum of numbers
4 sum = 0
5 # 3) Initial value for input
6 input_number = 0
7 print("Enter numbers greater than or equal to 0. Enter a negative number to stop.\n")
8 # 4) Loop
9 while input_number >= 0:
10     input_number = float(input("Enter a number (negative = stop): "))
11     if input_number >= 0:
12         numbers_list.append(input_number) # Add to list
13         sum += input_number                # Add to sum
14 # 5) Calculate average
15 if len(numbers_list) > 0:
16     avg = sum / len(numbers_list)
17 else:
18     avg = 0
19 # 6) Output
20 print("\nNumbers you entered:", numbers_list)
21 print("Average:", avg)
```

## Output

Enter numbers greater than or equal to 0. Enter a negative number to stop.

Enter a number (negative = stop): 10

Enter a number (negative = stop): 20

Enter a number (negative = stop): -2

Numbers you entered: [10.0, 20.0]

Average: 15.0

حل آخر مع استخدام try/except لتجنب الأخطاء:

## Code

```
1  # 1) Empty list
2  numbers_list = []
3  # 2) Sum of numbers
4  sum = 0
5  # 3) Initial value for input
6  input_number = 0
7  print("Enter numbers greater than or equal to 0. Enter a negative number to stop.\n")
8  # 4) Loop
9  while input_number >= 0:
10     try:
11         input_number = int(input("Enter a number (negative = stop): "))
12
13         if input_number >= 0:
14             numbers_list.append(input_number) # Add to list
15             sum += input_number               # Add to sum
16     except ValueError:
17         print("Invalid input, please enter a valid number.")
18 # 5) Calculate average
19 if len(numbers_list) > 0:
20     avg = sum / len(numbers_list)
21 else:
22     avg = 0
23 # 6) Output
24 print("\nNumbers you entered:", numbers_list)
25 print("Average:", avg)
```

## Output

Enter numbers greater than or equal to 0. Enter a negative number to stop.

Enter a number (negative = stop): 10

Enter a number (negative = stop): 20

Enter a number (negative = stop): f

Invalid input, please enter a valid number.

Enter a number (negative = stop): ,

Invalid input, please enter a valid number.

Enter a number (negative = stop): 5

Enter a number (negative = stop): -2

Numbers you entered: [10, 20, 5]

Average: 11.666666666666666

---

اكتب برنامج لإدارة المهام باتباع التالي:

1. أعلن عن متغير باسم choice وخصص قيمته الابتدائية فراغ ' '.
2. أعلن عن قائمة فارغة باسم mytask.
3. أنشئ حلقة تكرارية تستمر طالما أن قيمة المتغير choice لا تساوي '3' وتنفذ التالي:
  - اطبع اسم البرنامج.
  - يظهر للمستخدم ثلاث خيارات:
    1. إدخال مهمة جديدة.
    2. عرض المهام.
    3. خروج من البرنامج.
4. يستقبل من المستخدم الخيار المناسب ثم ينفذ الخيارات التالية:
  - إذا اختار المستخدم 1: يستقبل منه مهمة جديدة، ويضيفها إلى القائمة، ويطبع رسالة "تمت إضافة المهمة بنجاح". ثم يستمر بالتكرار.
  - إذا اختار المستخدم 2: يطبع محتوى القائمة على الشاشة بعد العنوان "مهامك هي:" وإذا كانت القائمة فارغة يطبع "لا توجد مهام بعد". ثم يستمر.
  - إذا اختار 3: يطبع: "تم الخروج من مدير المهام". ثم تنتهي الحلقة.
  - أما إذا أدخل خيارًا غير 1 أو 2 أو 3 فاطبع رسالة تحذير "خيار غير صحيح، حاول مرة أخرى".

مثال 11:

برنامج مدير  
المهام

## Code

```
1 mytask = []
2 choice = ''
3 while choice != '3':
4     print("*****")
5     print("[1] add new task : ")
6     print("[2] Display tasks")
7     print("[3] quit")
8     print("*****")
9     print()
10    choice = input("enter your choice >>> ")
11
12    if choice == '1':
13        print("\nWelcome! Add new task .....")
14        tsk = input("\nPlease enter your new task: ")
15        mytask.append(tsk)
16        print("New task added successfully")
17    elif choice == '2':
18        if len(mytask) == 0:
19            print("\nNo tasks yet!")
20        else:
21            print("\nDisplay list of all tasks:")
22            task_no = 1
23            for i in mytask:
24                print(f"Task no {task_no}\t{i}")
25                task_no += 1
26    elif choice == '3':
27        print("\nThank you for using task manager program..\n")
28        exit()
29    else:
30        print("Wrong menu choice")
31    print()
```

## Output

```
*****
```

```
[1] add new task :
```

```
[2] Display tasks
```

```
[3] quit
```

```
*****
```

```
enter your choice >>> 2
```

```
No tasks yet!
```

```
*****
```

```
[1] add new task :
```

```
[2] Display tasks
```

```
[3] quit
```

```
*****
```

```
enter your choice >>> 1
```

```
welcome add new task .....
```

```
Please enter your new task: Study ICT
```

```
new task added successfully
```

```
*****
```

```
[1] add new task :
```

```
[2] Display tasks
```

```
[3] quit
```

```
*****
```

```
enter your choice >>> 3
```

```
thank you for using task manager program..
```

---

## القواميس Dictionaries

تعتبر القواميس أحد أنواع البيانات غير الأولية Non-Primitive ، والتي يمكن أن تخزن أكثر من قيمة - مثل القوائم List- الصفوف Tuple- المجموعات Sets- وتمثل بياناتها كأزواج من المفاتيح Keys والقيم Values.

### خصائص القواميس:

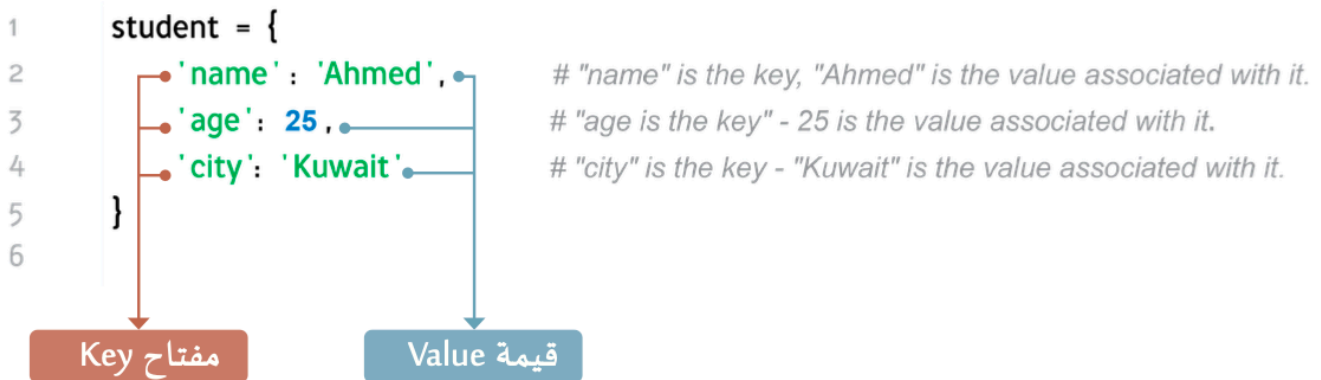
- تنحصر عناصر القواميس بين الأقواس المعقوفة { } .Curly Braces
- كل عنصر يتضمن مفتاح Key، والقيمة الخاصة به Value .
- عناصرها مرتبة Ordered، مما يعني أن عناصرها تحتفظ بترتيبها.
- المفاتيح keys يجب أن تكون فريدة Unique (غير مكررة)، وفي حال التكرار يتم استبدال القيمة القديمة بالقيمة الجديدة.
- المفاتيح Keys غير قابلة للتعديل Immutable، ويكون من نوع (string-integer-set-tuple)، ولا يمكن أن يكون من القوائم أو القواميس الأخرى.
- القيم Values تمثل بأي من أنواع البيانات Data Types.

| نوع البيانات | قابلة للتغيير | العناصر مرتبة | الفهرسة                | العناصر فريدة     | أنواع مختلفة من البيانات                 | الصيغة           |
|--------------|---------------|---------------|------------------------|-------------------|------------------------------------------|------------------|
| Data Type    | Mutable       | Ordered       | Indexing               | Unique Elements   | Different of Data Types                  | Syntax           |
| Dictionaries | نعم           | نعم           | باستخدام المفاتيح Keys | المفاتيح Keys فقط | القيم Values يمكن أن تكون متعددة الأنواع | {"key": "value"} |

خصائص القواميس

### صيغة القاموس Dictionary Syntax:

عند كتابة القاموس يجب أن يتضمن كل عنصر بالقاموس مفتاح Key، وقيمة Value.



## التعامل مع القواميس Dictionaries:

### 1. إنشاء القاموس:

في Python يمكننا إنشاء قاموس Dictionary من خلال تضمين قائمة مفصولة بفاصل " , " , لأزواج من القيمة Value ومفتاح Key بأقواس معقوفة {...}. Curly braces وتفصل النقطتان " : " Colon كل مفتاح عن القيمة المرتبطة به.

a. إنشاء القاموس باستخدام الأقواس المعقوفة {}.

مثال 1:

```
1 student = {  
2     "name": "Ahmed",  
3     "grade": "Tenth",  
4     "age": 15,  
5     "city": "Jahra"  
6 }  
7 |
```

لاحظ:

لاحظ: يتم وضع عناصر القاموس (المفاتيح والقيم) داخل الأقواس {}.

مثال 2:

```
dir0 = {'id':1, 'name':'Nora', 'size': 37}
```

b. إنشاء القاموس باستخدام الدالة dict().

يمكن أيضاً استخدام الدالة dict () لإنشاء Dictionary من خلال مجموعة متغيرات أو List أو Tuple .

مثال 3:

```
1 student = dict(name = "Ahmed", grade= "Tenth", age= 15, city= "Jahra")  
2 |
```

لاحظ:

يتم وضع عناصر القاموس (المفاتيح والقيم) داخل أقواس الدالة () بدون استخدام علامات تنصيب للمفاتيح.



## مثال 4:

### Code

```
1 d1 = dict(id=1, name="Noor", size=37)
2 d2 = dict({"id": 1, "name": "Noor", "size": 37})
3 print(d1)
4 print(d2)
```

### Output

```
{'id': 1, 'name': 'Noor', 'size': 37}
{'id': 1, 'name': 'Noor', 'size': 37}
```

**ملاحظة:** يعبر عن القاموس Dictionary الفارغ بدون أي عناصر مكتوب قوسين فقط، على النحو التالي: {}.

## 2. الوصول إلى العناصر (قيمة العنصر):

يوفر Python عدد من الدوال للوصول إلى عناصر القواميس Dictionaries حيث يتم الوصول إلى جميع القيم الموجودة في القاموس Dictionary باستخدام الدالة `values()` بينما يتم الوصول إلى المفاتيح بواسطة الدالة `keys()`. يمكن الوصول إلى قيمة العنصر في القاموس باستخدام المفتاح الخاص به.

## مثال 5: الوصول إلى عمر الطالب (15) باستخدام المفتاح به (age).

### Code

```
1 student = {"name": "Ahmed", "grade": "Tenth", "age": 15, "city": "Jahra"}
2 print(student["age"])
3 |
```

### Output

15

## مثال 6: الوصول إلى قيمة العنصر (Ahmed) باستخدام المفتاح به (name).

### Code

```
1 student = dict(name = "Ahmed", grade= "Tenth", age= 15, city= "Jahra")
2 print(student["name"])
```

### Output

Ahmed

**ملاحظة:** صيغة القاموس باستخدام الأقواس { } في المثال 5، وصيغة القاموس باستخدام الدالة dict في المثال 6.

**مثال 7:**

Code

```
1 Capitals={"France ":"Paris", "Germany":"Berlin", "Spain": "Madrid"}
2 print(Capitals["Germany"])
3
```

Output

Berlin

**مثال 8:**

Code

```
1 data = {
2     1: 'Admin',
3     2: 'Editor',
4     3: 'Reader'
5 }
6 # هنا قمنا بطباعة العنصر الذي يملك الرقم 1
7 print(data[1])
```

Output

Admin

**ملاحظة:** إذا لم يتم العثور على المفتاح في القاموس يرجع القيمة None من KeyError.

## مثال 9:

### Code

```
2 data = {
3     1: 'Admin',
4     2: 'Editor',
5     3: 'Reader'
6 }
7 # هنا قمنا بطباعة لعنصر الذي يحمل الرقم 4
8 print(data[4])
```

### Output

```
C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.12.exe D:\dictionaries\n.py
Traceback (most recent call last):
  File "D:\dictionaries\n.py", line 8, in <module>
    print(data[4])
    ~~~~~^
KeyError: 4

Process finished with exit code 1
```

## 3. إضافة وتعديل العناصر:

قواميس بايثون قابلة للتغيير وديناميكية. وبالتالي، يمكن تحديث عناصر القاموس، أو إضافة عناصر جديدة، أو حذف العناصر الموجودة. لتحديث عنصر قاموس، يتم تعيين قيمة جديدة لهذا العنصر من خلال الإشارة إليه بالمفتاح المرتبط.

### a. إضافة عنصر جديد.

يمكن إضافة أزواج قيمة-مفتاح إلى قاموس دون استخدام توابع أو دوال باستخدام الصيغة التالية:

`dict[Key] = value`

## مثال 10:

### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 print(student)
3 student["Country"] = "Kuwait"
4 print(student)
```

### Output

```
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra'}
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra', 'Country': 'Kuwait'}
```

## مثال 11:

### Code

```
1 Capitals={"France":"Paris", "Germany":"Berlin", "Spain":"Madrid"}
2 print(Capitals)
3 # إضافة عنصر جديد الى القاموس
4 Capitals["Italy"] = "Roma"
5 print(Capitals)
```

### Output

```
{'France': 'Paris', 'Germany': 'Berlin', 'Spain': 'Madrid'}
{'France': 'Paris', 'Germany': 'Berlin', 'Spain': 'Madrid', 'Italy': 'Roma'}
```

b. تعديل قيمة موجودة.

## مثال 12:

### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 print(student)
3 student["name"] = "Salem"
4 print(student)
```

### Output

```
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra'}
{'name': 'Salem', 'age': 15, 'city': 'Jahra'}
```

## مثال 13:

### Code

```
1 cars = {
2 "brand": "Ford",
3 "model": "Mustang",
4 "year": 1964
5 }
6 print(cars)
7 cars["brand"]="Gmc"
8 print(cars)
```

### Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Gmc', 'model': 'Mustang', 'year': 1964}
```

#### 4. حذف العناصر:

يمكن حذف عنصر من القاموس باستخدام الأمر (del) أو باستخدام الدالة (pop()).  
a. حذف عنصر باستخدام الأمر del.

مثال 14:

##### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 print(student)
3 del student["age"]
4 print(student)
```

##### Output

```
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra'}
{'name': 'Ahmed', 'city': 'Jahra'}
```

مثال 15:

##### Code

```
1 Dict = {'Tim': 18, 'Charlie': 12, 'Tiffany': 22, 'Robert': 25}
2 print(Dict)
3 # حذف العنصر الثاني في القاموس
4 del Dict['Charlie']
5 print(Dict)
```

##### Output

```
{'Tim': 18, 'Charlie': 12, 'Tiffany': 22, 'Robert': 25}
{'Tim': 18, 'Tiffany': 22, 'Robert': 25}
```

b. حذف عنصر باستخدام الدالة (pop()).

مثال 16:

##### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 print(student)
3 student.pop("age")
4 print(student)
```

##### Output

```
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra'}
{'name': 'Ahmed', 'city': 'Jahra'}
```

### مثال 17:

#### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 student_name = student.pop("name")
3 print("student_name:", student_name)
```

#### Output

student\_name: Ahmed

### مثال 18:

#### Code

```
1 cars = {
2 "brand": "Ford",
3 "model": "Mustang",
4 "year": 1964
5 }
6 print(cars)
7 cars.pop("model")
8 print(cars)
```

#### Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'year': 1964}
```

c. حذف العنصر الأخير بالقاموس باستخدام الدالة `.popitem()`.

### مثال 19:

#### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 print(student)
3 student.popitem() ←
4 print(student)
```

#### Output

```
{'name': 'Ahmed', 'age': 15, 'city': 'Jahra'}
{'name': 'Ahmed', 'age': 15}
```

**ملاحظة:** الأمر `(pop)` يحذف عنصر (مفتاح وقيمته) من القاموس، ويمكن إرجاع قيمة المفتاح المحذوف إلى متغير، بينما الأمر `(del)` لا يرجع أي قيمة.

## مثال 20:

### Code

```
1 cars = {
2 "brand": "Ford",
3 "model": "Mustang",
4 "year": 1964
5 }
6 print(cars)
7 cars.popitem()
8 print(cars)
```

### Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang'}
```

d. مسح جميع عناصر القاموس باستخدام الدالة `.clear()`.

## مثال 21:

### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 student.clear() ←
3 print(student)
```

### Output

```
{}
```

**ملاحظة:** الفرق بين استخدام `clear()` من التعليمة البرمجية:

`student.clear()` # مسح جميع عناصر القاموس

`del(student)` # حذف القاموس بالكامل من الذاكرة وأي محاولة للوصول إليه سيؤدي إلى خطأ

## مثال 22:

### Code

```
1 cars = {
2 "brand": "Ford",
3 "model": "Mustang",
4 "year": 1964
5 }
6 del cars
7 print(cars)
```

### Output

NameError: name 'cars' is not defined. Did you mean: 'vars'?

Process finished with exit code 1

- ظهر خطأ دلالي لأنه طلب طباعة القاموس والقاموس تم حذفه بالكامل

## مثال 23:

### Code

```
1 cars = {
2 "brand": "Ford",
3 "model": "Mustang",
4 "year": 1964
5 }
6 cars.clear()
7 print(cars)
```

### Output

{}



## 5. التكرار عبر عناصر القاموس باستخدام دالة ( item ):

ترجع قائمة تحتوي على المفتاح والقيمة للعناصر الموجودة في القاموس.  
التكرار عبر العناصر (المفاتيح والقيم).

يكنك استخدام حلقة for للتكرار على المفاتيح والقيم في القاموس.  
التكرار على المفاتيح والقيم

مثال 24:

```
for Key, value in my_dict.items():
 print(f"Key: {Key}, Value: {value}")
```

### Code

```
1 d1 = {'id': 1, 'name': 'Noor', 'size': 37, 'age': 26}
2 for x, y in d1.items():
3 print(x, y)
```

### Output

```
↑ name Noor
↓ size 37
age 26
```

مثال 25:

### Code

```
1 student = {"name": "Ahmed", "age": 15, "city": "Jahra"}
2 for key, value in student.items():
3 print(f' {key}: {value}')
```

### Output

```
name: Ahmed
age: 15
city: Jahra
```

## مثال 26: إنشاء قاموس، والوصول (القيمة) العنصر بمعلومية (المفتاح).

طباعة المفاتيح والقيم للقاموس (بيانات درجات الطالب)، وطباعة مجموع الدرجات للمواد الدراسية.

### Code

```
1 dic_student={"name": "Ahmed Salem", "arabic":90, "english":85, "science":95, "computer":98}
2 total_degrees=0
3 for key in dic_student:
4 # Print the element key in uppercase (upper() function)
5 print(key.upper(), ":", dic_student[key])
6 # To get the sum of numeric values only type () == int
7 if type(dic_student[key]) == int:
8 total_degrees += dic_student[key]
9 print(f"Total degrees = {total_degrees}")
```

انشاء قاموس يحتوي على اسم الطالب ودرجات بعض المواد

طباعة القاموس كالتالي : المفتاح بأحرف كبيرة ثم " : " ثم القيمة المرتبطة بالمفتاح

قارن نوع القيمة في حال كانت القيمة عددية (درجة) يتم اضافتها

طباعة قيمة المتغير مجموع الدرجات total\_degree

### Output

```
NAME : Ahmed Salem
ARABIC : 90
ENGLISH : 85
SCIENCE : 95
COMPUTER : 98
Total degrees = 368
```

يمكن استخدام for key in dic\_student. keys(): لأداء نفس المهمة.

## إنشاء قاموس متعدد الأبعاد Nested Dictionary

القاموس المتداخل **Nested Dictionary** في بايثون هو في الأساس قاموس داخل قاموس . وهي طريقة فعّالة لتخزين وتنظيم هياكل البيانات المعقدة في بايثون .

مثال 27 :

### Code

```
1 #Build the nested dictionaries
2
3 customers = {'customer1': {'id': 1, 'name': 'Noor', 'size': 37},
4 'customer2': {'id': 2, 'name': 'Mona', 'size': 36},
5 'customer3': {'id': 3, 'name': 'Sara', 'size': 38},
6 }
7
8 #OR
9
10 customer1 = {'id': 1, 'name': 'Noor', 'size': 37}
11 customer2 = {'id': 2, 'name': 'Mona', 'size': 36}
12 customer3 = {'id': 3, 'name': 'Sara', 'size': 38}
13
14 customers= {'customer1': customer1,
15 'customer2': customer2,
16 'customer3': customer3,
17 }
18
19 #To access items in nested dictionaries
20 print(customers['customer1']['name'])
```

تحليل بيانات الطلاب المخزنة في قواميس واستخراج نتائج معينة بناءً على درجاتهم في مواد مختلفة.

في البداية تم إنشاء 3 قواميس كل قاموس يخزن بيانات ودرجات طالب واحد ، **dict\_student1** ، **dict\_student2** ، **dict\_student3** ثم تم إنشاء قاموس متداخل باسم **All\_students** وتجميع القواميس فيه كقيمة والمفتاح لها ( key ) هو **one,two,three**

مثال إترائي 28:

إنشاء قاموس

متعدد الأبعاد

Nested

Dictionary:

## Code

```

1 dict_student1 = {"name": "Ahmed", "Arabic": 90, "English": 50, "Computer": 98}
2 dict_student2 = {"name": "Ali", "Arabic": 70, "English": 95, "Computer": 90}
3 dict_student3 = {"name": "Fahed", "Arabic": 40, "English": 92, "Computer": 98}
4
5 all_students = {
6 "one": dict_student1,
7 "two": dict_student2,
8 "three": dict_student3
9 }
10 # Printing students who failed in Arabic
11 count_arabic = 0
12 print("Students who failed in Arabic")
13 print("=====")
14 for x in all_students.values():
15 if x["Arabic"] < 50:
16 count_arabic += 1
17 print("name:", x["name"])
18 print("Number of students failing in Arabic =", count_arabic, "\n")
19
20 # Students with an excellent grade in English
21 print("Students who get excellent in English")
22 print("=====")
23 count_english = 0
24 for x in all_students.values():
25 if x["English"] >= 90:
26 count_english += 1
27 print("name:", x["name"])
28
29 print("Number of students who received an excellent grade in English =", count_english)

```

انشاء قاموس متداخل باسم all\_students

انشاء متغير لحساب عدد الطلبة الراسيين في العربي وتخصيص القيمة له صفر

تكرار مقارنة درجة العربي لجميع الطلبة في القاموس عن طريق المفتاح اذا كانت اقل من 50 , اصف واحد للمتغير واطبع اسم الطالب الراسي

اطبع عدد الطلبة الراسيين في العربي (قيمة المتغير)

انشاء متغير لحساب عدد الطلبة الممتازين في الانجليزي وتخصيص القيمة له صفر

تكرار مقارنة درجة الانجليزي لجميع الطلبة في القاموس عن طريق المفتاح اذا كانت أكبر من او تساوي 90 , اصف واحد للمتغير واطبع اسم الطالب المتميز

اطبع عدد الطلبة المتميزين في الانجليزي (قيمة المتغير)

## Output

```

Students who failed in Arabic
=====
name: Fahed
Number of students failing in Arabic = 1

Students who get excellent in English
=====
name: Ali
name: Fahed
Number of students who received an excellent grade in English = 2

```

طور البرنامج لطباعة أسماء الطلاب وعددهم الحاصلين على درجة أكبر من 95 في مادة الحاسوب.

مطلوب دراسة التعليمات البرمجية، ثم تنفيذ التالي:

مثال 29:

1- القاموس: `my_country()`

- تعديل قيمة العنصر «`population`» لتصبح «`5000000`»
- إضافة مفتاح جديد «`Currency`» ، وإدخال قيمته عن طريق المستخدم.

2- القاموس: `gcc_countries()`

- عدل ما يلزم ليتضمن القاموس دول مجلس التعاون الخليجي فقط.
- طباعة القاموس.

التعامل مع عناصر  
القاموس (الحذف،  
الإضافة، والتعديل).

### قبل التعديل Code

```
1 # قاموس دول مجلس التعاون الخليجي
2 gcc_countries = {
3 "Saudi Arabia": "Riyadh",
4 "Kuwait": "Kuwait City",
5 "United Arab Emirates": "Abu Dhabi",
6 "Qatar": "Doha",
7 "Oman": "Muscat",
8 "Bahrain": "Manama",
9 "Yemen": "Sana'a"
10 }
11 # قاموس معلومات عن الكويت
12 my_country = {
13 "Country": "Kuwait",
14 "Capital": "Kuwait",
15 "population": 5000000
16 }
17
18 # إضافة عملة إلى القاموس
19 my_country["currency"] = "Kuwaiti Dinar"
20 # تعديل عدد السكان
21 my_country["population"] = 4500000
22 # حذف عنصر من قاموس دول مجلس التعاون الخليجي
23 del gcc_countries["Yemen"]
24 # طباعة معلومات عن الكويت
25 print("\nInfo About Kuwait\n" + "-" * 40)
26 for key, value in my_country.items():
27 print(key, ":", value)
```

## بعد التعديل

```
1 # قاموس دول مجلس التعاون الخليجي
2 gcc_countries = {
3 "Saudi Arabia": "Riyadh",
4 "Kuwait": "Kuwait City",
5 "United Arab Emirates": "Abu Dhabi",
6 "Qatar": "Doha",
7 "Oman": "Muscat",
8 "Bahrain": "Manama"
9 }
10 # قاموس معلومات الكويت
11 my_country = {
12 "Country": "Kuwait",
13 "Capital": "Kuwait",
14 "population": 5000000
15 }
16 # طلب إدخال العملة من المستخدم
17 currency = input("Enter Kuwait Currency: ")
18 my_country["currency"] = currency
19 # طباعة معلومات عن الكويت
20 print("\nInfo About Kuwait")
21 print("*" * 40)
22 for key, value in my_country.items():
23 print(f"{key:<12} {value}")
24 # طباعة دول مجلس التعاون الخليجي
25 print("\nThe Gulf Cooperation Council Countries")
26 print("*" * 40)
27 for country, capital in gcc_countries.items():
28 print(f"{country:<25} {capital}")
```

## Output

Enter Kuwait Currency: *Kuwait Dinar*

Info About Kuwait

\*\*\*\*\*

|            |              |
|------------|--------------|
| Country    | Kuwait       |
| Capital    | Kuwait       |
| population | 5000000      |
| currency   | Kuwait Dinar |

The Gulf Cooperation Council Countries

\*\*\*\*\*

|                      |             |
|----------------------|-------------|
| Saudi Arabia         | Riyadh      |
| Kuwait               | Kuwait City |
| United Arab Emirates | Abu Dhabi   |
| Qatar                | Doha        |
| Oman                 | Muscat      |
| Bahrain              | Manama      |

مثال 30:

التعامل مع عناصر  
القاموس  
(Dictionaries)،  
والقوائم (Lists)

ادرس التعليمات البرمجية التالية ثم طور البرنامج بتنفيذ التالي:

• إنشاء قاموس باسم (student4) يحتوي بيانات متعلم)

• إضافة القاموس إلى القائمة all\_students باستخدام الدالة المناسبة

## قبل التعديل Code

```
1 # create dictionaries
2 student1 = {"name": "ali", "result": "success", "total_degrees": 650}
3 student2 = {"name": "khalid", "result": "success", "total_degrees": 700}
4 student3 = {"name": "ahmed", "result": "failed", "total_degrees": 250}
5
6 # list = some of the dictionaries
7 all_students = [student1, student2, student3]
8
9 # Create new Student's Dictionary
10
11
12 # Add Student's Dictionary to list (all_student)
13
14
15 # Print keys and values
16 for i in all_students:
17 for x, y in i.items():
18 print(f"{x:<20}{y}")
19 print("=" * 40)
```

## بعد التعديل Code

```
1 # إنشاء القواميس : لبيانات الطلاب
2 student1 = {"name": "ali", "result": "success", "total_degrees": 650} # بيانات الطالب الأول
3 student2 = {"name": "khalid", "result": "success", "total_degrees": 700} # بيانات الطالب الثاني
4 student3 = {"name": "ahmed", "result": "failed", "total_degrees": 250} # بيانات الطالب الثالث
5 # إنشاء قائمة تحتوي على القواميس
6 all_students = [student1, student2, student3] # قائمة تجمع بيانات الطلاب
7 # إضافة بيانات طالب جديد
8 # طلب إدخال البيانات من المستخدم
9 name = input("Enter new name: ") # إدخال اسم الطالب الجديد
10 # إدخال نتيجة الطالب (نجاح أو رسوب)
11 result = input("Enter student result(success/failed): ")
12 # إدخال مجموع الدرجات (يجب أن يكون رقماً)
13 total_degrees = float(input("Enter total degrees: "))
14 # إنشاء قاموس جديد للطالب وإضافته إلى القائمة
15 new_student = {"name": name, "result": result, "total_degrees": total_degrees}
16 # إضافة قاموس الطالب الجديد إلى القائمة
17 all_students.append(new_student)
18 # طباعة بيانات جميع الطلاب
19 # المرور عبر كل طالب (قاموس) في القائمة
20 # المرور عبر مفاتيح وقيم القاموس
21 for x, y in i.items():
22 # طباعة المفاتيح والقيم بتنسيق عرض 20 خانة
23 print(f"{x:<20}{y}")
24 # طباعة خط فاصل بين بيانات كل طالب
25 print("=" * 40)
```

## Output

```
Enter new name: Fahad
Enter student result(success/failed): success
Enter total degrees: 500
name ali
result success
total_degrees 650
=====
name khalid
result success
total_degrees 700
=====
name ahmed
result failed
total_degrees 250
=====
name Fahad
result success
total_degrees 500.0
=====
```

### شرح مثال 30 :

- تم في البداية انشاء قاموس خاص بكل طالب يحتوي على بياناته ( الاسم والنتيجة والدرجة الكلية )
- بعد ذلك تم انشاء قائمة باسم **all\_students** تتكون من القواميس الخاصة بالطلبة

( student1 ,student2,student3)

- باستخدام حلقة for يتم طباعة مفاتيح القاموس (name, result, total\_degrees) وقيمتها لكل طالب
- تم استخدام تنسيق نصوص لجعل الطباعة مرتبة كالتالي :

|         |                                                     |    |                                    |
|---------|-----------------------------------------------------|----|------------------------------------|
| {x:<20} | يعني أن المفتاح يأخذ عرض 20 خانة                    | <: | يحدد اتجاه محاذاة النص إلى اليسار. |
| x:      | تمثل القيمة التي سيتم طباعتها (المفتاح في القاموس). | 20 | العرض الإجمالي للحقل (20 مسافة).   |
| ( : )   | الفاصل بين اسم المتغير (x) وخيارات التنسيق.         |    |                                    |

- بعد طباعة بيانات كل طالب، يتم طباعة خط فاصل (40 \* =) للفصل بين طالب وآخر.

### مثال 31:

اكتب برنامج تقرير الطالب لتنفيذ مايلي:

- قراءة الاسم الكامل للطالب باللغة الإنجليزية وإظهار الاسم المختصر وتنسيقه بحيث يكون كل جزء .capitalized
- إدخال درجات متتالية باستخدام while حتى يكتب المستخدم done ويتجاهل الإدخالات الفارغة وغير الصحيحة.
- حساب العدد، والمجموع و متوسط الدرجات وأكبر درجة وأصغر درجة.
- استخدام for مع enumerate لطباعة حالة كل درجة (A/B/C/Fail).
- البحث عن الدرجة الكاملة 100 باستخدام for...else.
- حساب المتوسط باستخدام الشرط المختصر وطباعة رسالة Passed/Failed.



## Code

```
1 # إدخال ومعالجة الاسم (سلاسل نصية + متغيرات) (1)
2 full_name = input("Enter full name (e.g., ahmed ali hussein): ").strip()
3 parts = [p.capitalize() for p in full_name.split() if p.strip() != ""]
4 short_name = parts[0] + (" " + parts[-1][0] + "." if len(parts) >= 2 else "")
5
6 print("Formatted name:", " ".join(parts))
7 print("Short name :", short_name)
8
9 # إدخال الدرجات باستخدام while حتى 'done' (2)
10 print("\nEnter scores one per line (0..100). Type 'done' to finish.")
11 scores = []
12 while True:
13 s = input("> ").strip()
14 if s.lower() == "done":
15 break
16 if s == "":
17 print("Empty input ignored.")
18 continue
19 try:
20 v = float(s)
21 except ValueError:
22 print("Invalid number, try again.")
23 continue
24 if not (0 <= v <= 100):
25 print("Out of range (0..100), ignored.")
26 continue
27 scores.append(v)
28
29 if len(scores) == 0:
30 print("\nNo scores were entered.")
31 else:
32 print("\nScores captured:", scores)
33
```

```

34 # إحصاءات أساسية (3)
35 count = len(scores)
36 total = sum(scores) if count > 0 else 0
37 avg = total / count if count > 0 else 0
38 min_v = min(scores) if count > 0 else None
39 max_v = max(scores) if count > 0 else None
40
41 print("\n--- Stats ---")
42 print("Count:", count)
43 print("Sum :", round(total, 2))
44 print("Avg :", round(avg, 2))
45 print("Min :", min_v)
46 print("Max :", max_v)
47
48 # استعراض الدرجات باستخدام for + enumerate + شروط (4)
49 def grade_letter(x): 1 usage
50 if x >= 90: return "A"
51 elif x >= 80: return "B"
52 elif x >= 70: return "C"
53 elif x >= 60: return "D"
54 else: return "F"
55
56 print("\n--- Detailed list ---")
57 for i, sc in enumerate(scores, start=1):
58 print(i, sc, grade_letter(sc))
59
60 # بحث عن درجة كاملة باستخدام for...else (5)
61 print("\n--- Search for perfect score (100) ---")
62 for sc in scores:
63 if sc == 100:
64 print("Perfect score found.")
65 break
66 else:
67 print("No perfect score found.")
68
69 # الشرط المختصر (6)
70 status = "Passed" if avg >= 50 else "Failed"
71 print("\nFinal status:", status)

```

## Output

Enter full name (e.g., ahmed ali hussein): *ahmed ali hussain*

Formatted name: Ahmed Ali Hussain

Short name : Ahmed H.

Enter scores one per line (0..100). Type 'done' to finish.

> *30*

> *40*

> *0*

> *h*

Invalid number, try again.

> *50*

> *100*

> *done*

Scores captured: [30.0, 40.0, 0.0, 50.0, 100.0]

--- Stats ---

Count: 5

Sum : 220.0

Avg : 44.0

Min : 0.0

Max : 100.0

--- Detailed list ---

1 30.0 F

2 40.0 F

3 0.0 F

4 50.0 F

5 100.0 A

--- Search for perfect score (100) ---

Perfect score found.

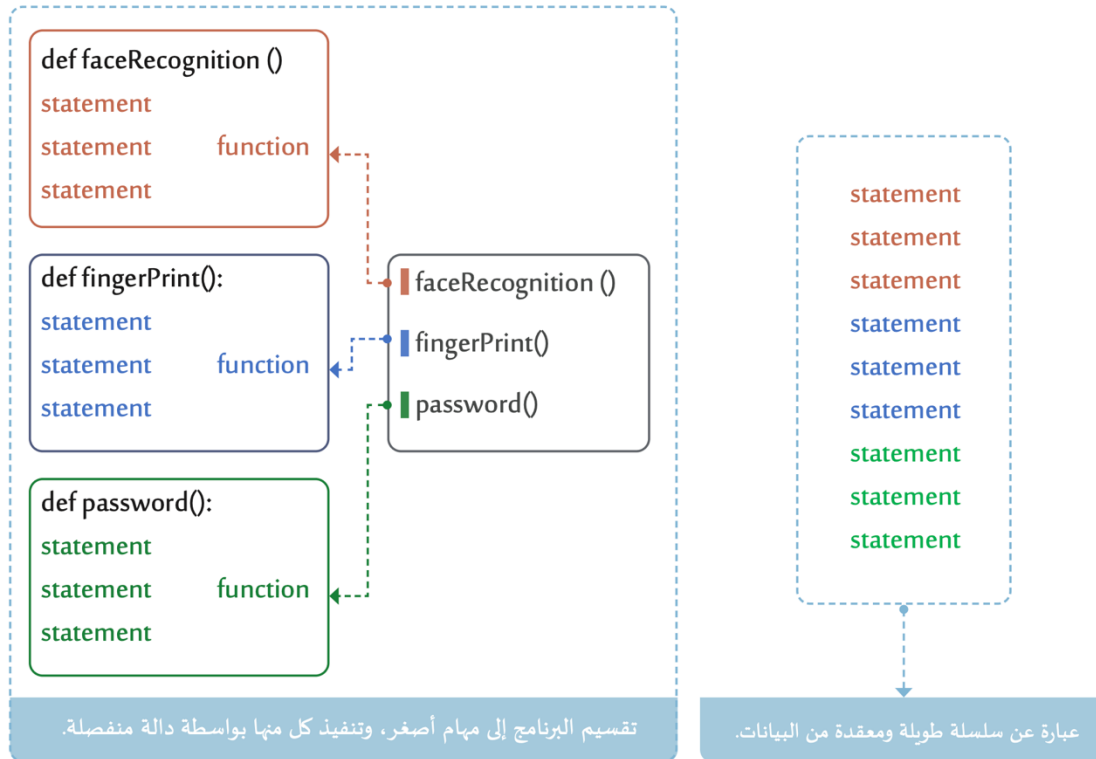
Final status: Failed

## الدوال Functions

تعتبر الدوال (Functions) أحد أدوات البرمجة الأساسية في Python، التي تساعد على كتابة التعليمات البرمجية بصورة منظمة لسهولة التعامل معها.

### مفهوم الدالة Function Concept

مجموعة من التعليمات البرمجية لها مهام محددة تنفذ عند استدعائها، وهناك بعض البرامج تؤدي مهامًا كبيرة بما يحتاج لتقسيمها إلى مهام فرعية (دوال)، وتكرر ها في عدة مواضع في البرنامج، مما يسهل تعديلها وتطويرها بشكل مستمر.



مفهوم الدالة (استخدام الدوال في كتابة وتنظيم البرنامج)

### أنواع الدوال Function Types:

- **الدوال المدمجة Built-in functions:**  
الدوال التي تأتي مع Python بشكل افتراضي، وتستخدم دون الحاجة إلى استيرادها مثل: `print()` – `input()` – `len()` – `max()` – `type()`.
- **الدوال المعرفة من المستخدم User-defined functions:** الدوال التي ينشئها المستخدم والتي تمثل مجموعة من الأسطر البرمجية التي تنفذ مهام محددة.  
**مثال:** عندما نريد أن نقول "مرحبًا" لكل أصدقائنا، بدلاً من كتابة "مرحبًا" 10 مرات، نكتب دالة نقول ذلك مرة واحدة ونستخدمها عدة مرات.
- **الدوال المجهولة Lambda functions:** الدوال التي تُستخدم لتنفيذ وظائف بسيطة ومباشرة، وتُكتب في سطر واحد دون استخدام `def` للتعريف عنها.

## مميزات استخدام الدوال:

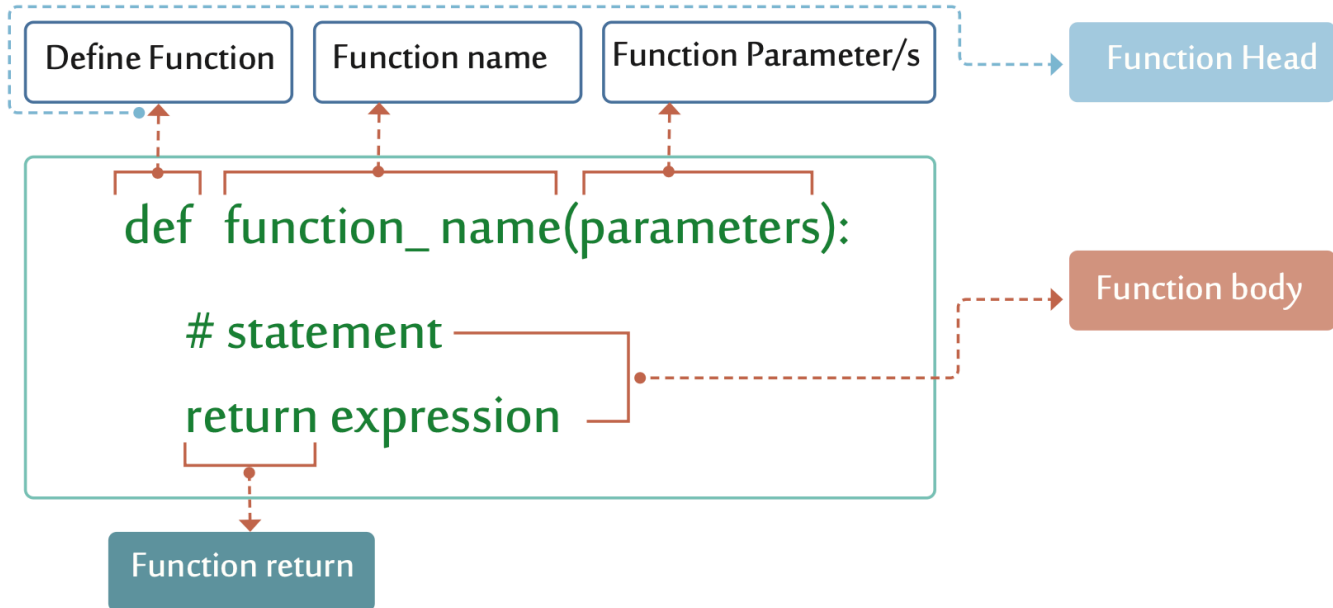
توفر الدوال في Python العديد من المميزات التي تساعد المستخدمين على:

1. تسهيل كتابة البرامج.
2. تحسين كفاءة المقطع البرمجي.
3. إعادة استخدام نفس التعليمات في أكثر من موضع.
4. تسهيل اكتشاف الأخطاء وإصلاحها.
5. الدوال تُساعد في تقسيم البرامج الكبيرة إلى أجزاء صغيرة (مهام فرعية) تسهل قراءتها، تعديلها، وتطويرها.
6. يتم استدعاء الدالة عند الحاجة إلى تنفيذها، مما يجعل الكود أكثر تنظيماً وأقل تكراراً.

## إنشاء الدوال Function :

إنشاء الدوال في Python يعد من المهارات الأساسية لتنظيم التعليمات البرمجية وإعادة استخدامها، والتي يمكن استدعاؤها في أي وقت عند الحاجة.

1. صيغة الدالة Function Syntax:



### Function Head

- تعريف الدالة Define a function: نستخدم الكلمة المفتاحية def وهي اختصار لكلمة (definition) لتعريف الدالة، وهي من الكلمات المحجوزة في Python.

- اسم الدالة Function name: اسم يُعبر عنها، ويعكس وظيفتها، ويتم اتباع قواعد تسمية المتغيرات عند تسمية الدوال في Python.

- معاملات الدالة / Function Parameter : هي متغيرات يتم تعريفها في الدالة، ويتم تعيين قيم لها عند استدعائها، والتي يتم تمريرها للدالة ويمكن أن تكون فارغة.

ويُلي اسم الدالة أقواس دائرية **Parentheses** حيث يُعرف معاملات الدالة بداخل الأقواس، ثم يُتبع بالنقطتين ( : ) **colon**.

Function body

يشمل مجموعة من التعليمات البرمجية داخل الدالة (لاحظ المحاذاة Indentation)، لأن مفسر Python Interpreter يستخدمها للإعلان عن بداية ونهاية التعليمات البرمجية.

يحتوي جسم الدالة على:

- تعليمات (statements) لتنفيذ المهام.
- تعليمات **Return Expression** لإرجاع قيمة عند الانتهاء.

- **return statement**: تُستخدم لإنهاء تنفيذ استدعاء الدالة و«إرجاع» نتيجتها (قيمة التعبير الذي يلي return expression).

### استدعاء دالة Function Calling:

- كتابة اسم الدالة متبوعاً بالأقواس الدائرية ( ) **Parentheses**.
- يمكن تمرير قيم (Arguments) للدوال التي تتضمن معاملات (Parameters).
- **Parameters (المعاملات)**: المتغيرات التي تعرف في رأس الدالة. وتمثل أماكن القيم التي يتم تمريرها إلى الدالة لاحقاً، ويتم تحديدها عند إنشاء الدالة.
- **Arguments (القيم المُرسلة)**: القيم الفعلية التي تُمرر إلى الدالة عند استدعائها. هذه القيم تحل محل Parameters داخل الدالة.
- تنفيذ التعليمات البرمجية الموجودة في جسم الدالة.
- الانتقال مرة أخرى إلى جزء البرنامج الذي استدعى الدالة.
- استئناف البرنامج التنفيذ عند هذه النقطة.

### مثال 1: إنشاء دالة (بدون معاملات)

#### Code

```
1 def greet(): # تعريف الدالة Function head
2 print("Hello, Kuwait!") # تعليمات داخل الدالة، تعرض رسالة ترحيبية عند استدعائها Function body
3
4 greet() # استدعاء الدالة لتنفيذ ما بداخلها Function Calling
5
```

#### Output

Hello, Kuwait!

## شرح المثال:

1. `def greet():`

○ رأس الدالة **Function Head**

- `def:` لتعريف الدالة - **greet** اسم الدالة.
- ( ) الأقواس الفارغة تشير إلى أن هذه الدالة لا تحتوي على أي معاملات.
- النقطتان: تُستخدم لتحديد بداية جسم الدالة.

2. `print("Hello, Kuwait!")`

○ جسم الدالة **Function Body** :

- تحتوي على التعليمات البرمجية التي يتم تنفيذها عند استدعاء الدالة.
- `print()` دالة مدمجة في Python لطباعة النصوص.
- `"Hello, Kuwait!"` النص الذي سيتم طباعته.

3. `greet()`

○ استدعاء الدالة **Function Calling**

- عند استدعاء الدالة `greet`، يتم تنفيذ التعليمات الموجودة في جسم الدالة (طباعة النص).
- في حالة الاستدعاء الدالة `greet` لأكثر من مرة.

## استدعاء الدالة أكثر من مرة

### Code

```
1 def greet(): # تعريف الدالة
2 print("Hello, Kuwait!") # تعليمات داخل الدالة، تعرض رسالة ترحيبية عند استدعائها
3
4 greet() # استدعاء الدالة لتنفيذ ما بداخلها
5
6
7 greet()
8 greet()
9 greet()
```

### Output

```
Hello, Kuwait!
Hello, Kuwait!
Hello, Kuwait!
Hello, Kuwait!
```

## مثال 2:

### إنشاء دالة (مع معاملات)

كتابة دالة باسم `add_numbers` تنفذ الآتي:

- تأخذ رقمين كمدخلات (معاملين) `(num1, num2)`
- تطبع ناتج جمع الرقمين.

يتم استدعاء الدالة باستخدام القيم 10 و 30

#### Code

```
1 def add_numbers(num1, num2): # (num1, num2) حيث معاملين تعريف الدالة add_numbers
2
3 # طباعة الناتج
4 print(num1+num2)
5
6 # استدعاء الدالة بقيم 10 و 30
7 add_numbers(num1: 10, num2: 30)
8
```

#### Output

40



**التفسير:** برنامج يمثل إضافة عنصر إلى عربة التسوق ( shopping cart ) من نوع List ، وعرض محتوياتها.

## Code

```

1 # تعريف دالة لإضافة عنصر إلى سلة التسوق
2 def add_item(shopping_cart, item):
3 """
4 تضيف عنصراً جديداً إلى سلة التسوق وتعرض رسالة تأكيد وتعيد القائمة بعد التحديث.
5 """
6 shopping_cart.append(item) #إضافة العنصر إلى القائمة
7 print(f"This item: {item} added to shopping cart") # طباعة رسالة تأكيد
8 return shopping_cart # إعادة القائمة بعد التعديل ←
9
10 # تعريف دالة لعرض محتويات سلة التسوق
11 def view_cart(shopping_cart):
12 """
13
14 تعرض جميع العناصر الموجودة في سلة التسوق.
15 إذا كانت السلة فارغة، يتم عرض رسالة توضح ذلك
16 """
17 if not shopping_cart: #التحقق إذا كانت السلة فارغة
18 print("Your shopping cart is empty.") # رسالة عند عدم وجود عناصر
19 else:
20 print("Shopping cart contains:") # رسالة توضح أن السلة تحتوي على عناصر
21 for item in shopping_cart: #عرض كل عنصر في السلة
22 print(f"- {item}")
23
24 # إنشاء سلة تسوق فارغة
25 shopping_cart = []
26
27 # إضافة عناصر باستخدام الدالة وإرجاع القائمة المحدثة كل مرة
28 shopping_cart = add_item(shopping_cart, "Apple") #إضافة تفاحة
29 shopping_cart = add_item(shopping_cart, "Banana") #إضافة موزة
30 shopping_cart = add_item(shopping_cart, "Bread") #إضافة خبز
31
32 # عرض محتويات السلة
33 view_cart(shopping_cart)

```

## فكرة البرنامج:

البرنامج يمثل سلة تسوق إلكترونية ويمكن للمستخدم:

1. إضافة عناصر إلى السلة.
2. عرض العناصر الموجودة في السلة. إذا لم تكن هناك عناصر، سيعرض البرنامج رسالة تُوضح أن السلة فارغة.

### Code

```
1 # تعريف دالة لإضافة عنصر إلى سلة التسوق
2 def add_item(shopping_cart, item):
3 """
4 تضيف عنصرًا جديدًا إلى سلة التسوق وتعرض رسالة تأكيد.
5 """
6 shopping_cart.append(item) # إضافة العنصر إلى القائمة
7 print(f'This item: {item} added to shopping cart') # طباعة رسالة تأكيد
8
```

## شرح المثال:

- تعريف الدالة **add\_item**.
  - إضافة عنصرًا جديدًا إلى سلة التسوق.
  - تُطبع رسالة تعلم المستخدم أن العنصر تمت إضافته.
- المعاملات: **(Parameters)**
  - **shopping\_cart**: القائمة التي تمثل سلة التسوق.
  - **item**: العنصر الذي سيتم إضافته.
  - **append()** تستخدم لإضافة العنصر إلى القائمة ويتبعها رسالة تأكيد باستخدام **print()**.
- تعريف الدالة الثانية **view\_cart**.
  - عرض محتويات سلة التسوق.
- المعاملات: **(Parameters)**
  - **shopping\_cart**: قائمة تحتوي على العناصر المضافة.
- **if not shopping\_cart**: للتحقق إذا كانت سلة التسوق فارغة.
  - إذا كانت السلة فارغة، يتم طباعة رسالة توضح "Your shopping cart is empty".
  - إذا كانت تحتوي على عناصر، يتم عرض قائمة بجميع العناصر.

### Code

```
print('Shopping cart contains:')
for item in shopping_cart:
 print(f'- {item}')
```

○ إنشاء سلة التسوق : `shopping_cart = []`

○ من خلال تعريفه كمتغير فارغ باسم `shopping_cart` ليكون سلة تسوق لإضافة العناصر فيها.

○ إضافة عناصر للسلة باستدعاء الدالة `add_item` ثلاث مرات لإضافة العناصر:

- `"Apple"`
- `"Banana"`
- `"Bread"`

#### Code

```
add_item(shopping_cart, "Apple")

add_item(shopping_cart, "Banana")

add_item(shopping_cart, "Bread")
```

عرض محتويات سلة التسوق باستدعاء الدالة `view_cart`.

مثال 4:

دالة ترجع قيمة.

إنشاء دالة لجمع عددين وإرجاع الناتج باستخدام `Return Expression`.

#### Code

```
1 # تعريف دالة لجمع عددين
2 def add_numbers(a, b):
3
4 return a + b # ترجع ناتج جمع العددين
5
6 # استدعاء الدالة add_numbers بالقيمتين 5 و 3
7 result = add_numbers(a: 5, b: 3) # تخزين ناتج الجمع في المتغير result
8
9 # طباعة الناتج باستخدام دالة print
10 print(result) # عرض الناتج: 8
11
```

#### Output

8

## فكرة البرنامج:

- البرنامج يعرف دالة `add_numbers` لجمع عددين.
- يستدعي الدالة ويضيف القيم إليها ويتم تخزين الناتج في متغير ثم طباعته.

## شرح المثال:

|                                                                                |                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| تعريف الدالة:                                                                  | اسم الدالة <code>add_numbers</code><br>المعاملات (Parameters) التي تستقبل القيم عند استدعاء الدالة<br><code>return a + b</code> تجمع القيمتين                                                   |
| استدعاء الدالة وتخزين الناتج:                                                  | <code>result = add_numbers(3, 5)</code><br>في المتغير <code>result</code> يتم تخزين الناتج الذي تُعيده الدالة <code>add_numbers(5, 3)</code><br>القيمتين: <code>a = 5</code> <code>b = 3</code> |
| <code>result</code> لطباعة القيمة المخزنة في المتغير <code>print()</code> دالة |                                                                                                                                                                                                 |

## التأكيد على :

- **الدالة (Function):** تحتوي على مجموعة من التعليمات - وتعمل على إعادة استخدام التعليمات البرمجية.
- **المعاملات (Parameters):** متغيرات داخل الدالة تستقبل القيم عند الاستدعاء.
- **الإرجاع (Return):** تُعيد قيمة أو نتيجة معينة من الدالة إلى مكان استدعائها.

## ملاحظة:

- يمكنك تعريف دوال تحتوي على معامل واحد أو أكثر.
- تستخدم `return` لإرجاع نتيجة معالجة الدالة عند استدعائها.

## مثال 5:

إنشاء دالة (بمعاملات غير محددة )، والإرجاع **return** بأكثر من متغير.  
**التفسير:** برنامج ينشئ دالة تحسب المجموع الكلي والمتوسط لدرجات الطلاب المدخلة كمعاملات.

### Code

```
1 # student_result تعريف الدالة
2 def student_result(*degrees):
3
4 total_sum = 0 # متغير لتخزين المجموع الكلي
5 number_degrees = 0 # متغير لتتبع عدد الدرجات المدخلة
6
7 # حلقة لحساب المجموع الكلي وعدد الدرجات
8 for degree in degrees:
9 number_degrees += 1 # زيادة عدد الدرجات بمقدار 1
10 total_sum += degree # إضافة الدرجة إلى المجموع الكلي
11
12 # إرجاع المتوسط والمجموع الكلي
13 return (total_sum / number_degrees), total_sum
14
15 # استدعاء الدالة student_result مع تمرير درجات الطلاب
16 student_degrees, total_sum = student_result(*degrees: 80, 85, 99, 70, 88)
17 # طباعة المجموع الكلي للدرجات
18 print(f"Total student degree: {total_sum}")
19 # طباعة متوسط الدرجات
20 print(f"Average student degree: {student_degrees}")
21
```

### Output

```
Total student degree: 422
Average student degree: 84.4
```

### فكرة البرنامج:

- إنشاء دالة **student\_result** تأخذ عددًا غير محدد من درجات الطلاب كمداخلات.
- تقوم الدالة بحساب:
  1. المجموع الكلي للدرجات.
  2. متوسط الدرجات.
- **Return** تستخدم لإرجاع كل من المتوسط والمجموع الكلي كقيمتين.

## تعريف الدالة:

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <b>*degrees</b> <ul style="list-style-type: none"> <li>○ النجمة (*) تستخدم لجعل الدالة تستقبل عدد غير محدود من المعاملات.</li> <li>○ يتم تخزين جميع الدرجات المدخلة في قائمة باسم <b>degrees</b></li> </ul> </li> <li>• <b>total_sum = 0</b> <ul style="list-style-type: none"> <li>○ متغير لتخزين المجموع الكلي للدرجات.</li> </ul> </li> <li>• <b>number_degrees = 0</b> <ul style="list-style-type: none"> <li>○ متغير لتتبع عدد الدرجات المدخلة.</li> </ul> </li> <li>• <b>الحلقة (for)</b> <ul style="list-style-type: none"> <li>○ المرور على كل درجة داخل قائمة degrees</li> <li>○ تزيد الدرجات بمقدار 1 في كل دورة باستخدام <b>number_degrees += 1</b></li> <li>○ تضيف كل درجة إلى المجموع باستخدام <b>total_sum += degree</b></li> </ul> </li> <li>• <b>return (total_sum / number_degrees), total_sum</b> <ul style="list-style-type: none"> <li>○ تعيد الدالة قيمتين: <ul style="list-style-type: none"> <li>— متوسط الدرجات.</li> <li>— المجموع الكلي للدرجات.</li> </ul> </li> </ul> </li> </ul> | <p>تعريف الدالة student_result :</p>                 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <p>student_result لاستدعاء الدالة وتخزين النتائج</p> |
| <ul style="list-style-type: none"> <li>• <b>print(f"Total student degree: {total_sum}")</b> <ul style="list-style-type: none"> <li>■ طباعة المجموع الكلي للدرجات.</li> </ul> </li> <li>• <b>print(f"Average student degree: {student_degrees}")</b> <ul style="list-style-type: none"> <li>■ طباعة متوسط الدرجات.</li> </ul> </li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | <p>طباعة النتائج:</p>                                |

تطوير البرنامج: إدخال معامل الصف.  
ترتيب إدخال المعاملات.

## إثرائني تابع مثال 5:

### Code

Reader

```
1 # تعريف الدالة student_result
2 def student_result(grade, *degrees):
3
4 total_degrees = sum(degrees) # حساب مجموع الدرجات
5 average_degrees = total_degrees / len(degrees) # حساب متوسط الدرجات
6 return average_degrees, total_degrees, grade # إرجاع النتائج الثلاث
7
8 # استدعاء الدالة student_result وتمثيل معامل الصف والدرجات
9 average_degrees, total_degrees, grade = student_result(grade: 12, *degrees: 80, 85, 99, 70, 88)
10
11 # طباعة معامل الصف
12 # طباعة مجموع الدرجات
13 print(f"Student Grade: {grade}\n Total Student Degrees: {total_degrees}")
14
15 # طباعة متوسط الدرجات
16 print(f"Average Student Degrees: {average_degrees}")
17
```

### Output

```
Student Grade: 12
Total Student Degrees: 422
Average Student Degrees: 84.4
```

استكمالا من المثال السابق:

نجد أن المعاملات في هذا المثال

- grade معامل الصف.

- \*degrees قائمة غير محدودة من الدرجات.

وأن الدالة

- `total_degrees = sum(degrees)` تجمع الدرجات المدخلة باستخدام الدالة المدمجة `sum`

- `average_degrees = total_degrees / len(degrees)` إيجاد المتوسط بقسمة المجموع الناتج على عدد الدرجات باستخدام `len` من خلال حساب عدد عناصر القائمة (list).

- `return average_degrees, total_degrees, grade` تعيد الدالة القيم الثلاث: متوسط الدرجات، المجموع الكلي، و معامل الصف.

- `return average_degrees, total_degrees, grade` تعيد الدالة القيم الثلاث: متوسط الدرجات، المجموع الكلي، و معامل الصف.

ادرس التعليمات البرمجية التالية، وحدد وظيفة الجزء المحدد منها:

إثرائي تابع  
مثال(5) تطوير  
البرنامج:

#### Code

```
1 # تعريف الدالة student_result
2 def student_result(*degrees):
3 # حساب مجموع الدرجات
4 total_degrees = sum(degrees)
5 # حساب أعلى درجة ممكنة (عدد المواد × 100)
6 max_total = len(degrees) * 100
7 # طباعة مجموع الدرجات من أصل الدرجة الكلية
8 print(f"Sum of student's degrees: {total_degrees} from {max_total} degrees")
9 # حساب نسبة الدرجات
10 rate = (total_degrees / max_total) * 100
11 # طباعة النسبة المئوية للدرجات (مئلتين عشريتين)
12 print(f"Rate of student's degrees: {rate:.2f} %")
13 # تحديد حالة النجاح أو الفشل بناءً على النسبة
14 print("Congratulations, You succeeded") if rate ≥ 50 else print("Sorry! You failed")
15 # استدعاء الدالة مع درجات الطالب
16 student_result(*degrees: 100, 80, 90, 95, 50.5)
```

#### Output

```
↑ C:\Users\HP\Documents\projectfunction\.venv\Scripts\
↓ Sum of student's degrees: 415.5 from 500 degrees
Rate of student's degrees: 83.10 %
Congratulations, You succeeded
Process finished with exit code 0
```

#### شرح المثال للمتعلمين :

- حساب المجموع : (`total_degrees`) يتم جمع الدرجات باستخدام دالة `sum`.
- حساب الحد الأقصى : (`max_total`) يتم حساب أعلى درجة بضرب عدد المواد في 100.
- طباعة المجموع : يتم عرض المجموع الكلي للدرجات مقارنة بالحد الأقصى.
- حساب النسبة المئوية : يتم تقسيم مجموع الدرجات على الحد الأقصى وضرب الناتج في 100 للحصول على النسبة.
- طباعة النسبة : يتم تنسيق النسبة بحيث تظهر مع منزلتين عشريتين.
- `{rate:.2f}` المقصود `f"{value:.nf}"`.
- `rate` المتغير الذي يحتوي على النسبة المئوية للدرجات.
- `f` التنسيق خاص بالأعداد العشرية (`float`).
- `(.2f)` : عرض القيمة العشرية مع منزلتين عشريتين الناتج 83.10 %.
- `(.1f)` : عرض القيمة العشرية مع منزلة عشرية الناتج 83.1 %.
- ملاحظة : (`.0f`) : لإزالة الكسور العشرية (عدد صحيح) الناتج 83 %.



- تحديد النجاح أو الفشل يتم التحقق من أن النسبة أكبر من أو تساوي 50 لتحديد حالة النجاح، وإظهار الرسالة .

## الحل المطلوب في المثال:

- طباعة النسبة المئوية للدرجات (منزلتين عشريتين)

```
print(f"Rate of student's degrees: {rate:.2f} %")
```

- تحديد حالة النجاح أو الفشل بناءً على النسبة

```
print ("Congratulations, You succeeded") if rate >= 50 else print ("Sorry! You failed")
```

القيم الافتراضية لمعاملات الدوال default parameter .

## مثال 6:

**التفسير:** برنامج ينشئ دالة تتكون من معامليين أحدهما افتراضي (في حال عدم تمرير قيمة له؛ يتم استخدام القيمة الافتراضية للمعامل).

### Code

```
1 # تعريف دالة لإظهار اسم الطالب ودرجته
2 # يتم استخدام قيمة افتراضية للدرجة إذا لم يتم تمريرها
3 def student_grade(name, grade="Excellent"):
4 # طباعة اسم الطالب والدرجة
5 print(f"Hi {name} your grade is {grade}")
6
7 # استدعاء الدالة مع تمرير اسم الطالب فقط
8 # "Excellent" هي القيمة الافتراضية للدرجة وهي
9 student_grade("Ali")
10 # الناتج: Hi Ali your grade is Excellent
11
12 # استدعاء الدالة مع تمرير اسم الطالب والدرجة
13 # هنا يتم استبدال القيمة الافتراضية للدرجة بالقيمة الممررة وهي "Good"
14 student_grade(name="Ali", grade="Good")
15 # الناتج: Hi Ali your grade is Good
16
```

### Output

```
C:\Users\HP\Documents\projectfunction\
Hi Ali your grade is Excellent
Hi Ali your grade is Good
Process finished with exit code 0
```

**ملاحظة:** المعاملات الافتراضية لا تسبق في ترتيب كتابتها المعاملات التي تتطلب قيم argument.

def student\_grade(name, grade="Excellent") :

شرح مثال 6:

تعريف الدالة ( student\_grad ):

يجب كتابة ( المعاملات الإلزامية name - بدون قيم افتراضية ) أولاً  
ثم كتابة المعاملات grade ( ذات القيم الافتراضية Excellent ) بعدها.

الشرح:

1. المعاملات الإلزامية تأتي أولاً:
  - المعامل name في هذا المثال هو معامل إلزامي، يعني أنه يجب تمرير قيمة له عند استدعاء الدالة لاحقاً
  - إذا لم يتم تمرير قيمة لهذا المعامل، سيحدث خطأ.
2. المعاملات الافتراضية تأتي بعد الإلزامية:
  - المعامل grade هو معامل يحتوي على قيمة افتراضية وهي "Excellent".
  - إذا لم يتم تمرير قيمة لهذا المعامل أثناء استدعاء الدالة، سنستخدم القيمة الافتراضية.

دوال: Lambda

- هي دوال مجهولة الاسم (Anonymous Functions) تكتب باستخدام الكلمة المفتاحية lambda.
- تسمى هذه الدوال بالدوال المجهولة، وهي دوال بسيطة تكتب من سطر واحد.
- لا يتم الإعلان عنها بالطريقة القياسية باستخدام الكلمة المحجوزة (def).
- يمكن أن تأخذ أي عدد من (Arguments) ولكنها تُرجع قيمة واحدة فقط في شكل تعبير قصير.
- والصيغة العامة لإنشاء دالة (Lambda) في بايثون تكون: lambda arguments : expression

إنشاء دالة lambda لجمع عددين.

مثال 7:

Cod

```
1 # إنشاء دالة Lambda لجمع عددين
2 add = lambda x, y: x + y # الدالة تأخذ معاملي (x, y) وتعيد ناتج جمعها
3
4 # استدعاء الدالة المجهولة Lambda وتمرير العددين 5 و 10 وطباعة النتيجة
5 print(add(x: 5, y: 10))
6
```

Output

15

إنشاء دالة **Lambda** لحساب الأس التربيعي لعدد.

مثال 8:

#### Code

```
1 # إنشاء دالة Lambda لحساب تربيع عدد
2 exponention = lambda x: x ** 2 # الدالة تأخذ معاملاً واحداً x وتعيد x مرفوعاً للأس 2
3
4 # استدعاء الدالة المجهولة وطباعة الناتج
5 print(exponention(5)) # يتم تمرير العدد 5 إلى الدالة
6
```

#### Output

25

إنشاء دالة (**Lambda**) لتحديد نوع العدد (فردياً أم زوجياً).

مثال 9:

#### Code

```
1 # إنشاء دالة Lambda لتحديد إذا كان العدد زوجياً أو فردياً
2 even_odd = lambda x: print("even") if x % 2 == 0 else print("odd")
3
4 # استدعاء الدالة مع أخذ المدخل من المستخدم
5 even_odd(float(input("Enter a number: ")))
6
```

#### Output

Enter a number: 9  
odd

### نطاق المتغيرات Scope

مفهوم يحدد مكان وكيفية الوصول إلى المتغيرات داخل البرنامج.

**المتغير المحلي Local Variable:** المتغيرات التي تُعرّف داخل دالة من التعليمات تكون محلية، مما يعني أنها يمكن الوصول إليها فقط داخل تلك الدالة.

## مثال 10:

مجال المتغير المحلي في الدالة: (Local Variable).

التفسير: عدم إمكانية استخدام المتغير المحلي `local_var` خارج نطاق الدالة وظهور رسالة الخطأ (المتغير غير معرف)

### Code

```
1 def my_function():
2 local_var = 10 # تعريف متغير محلي
3 print (local_var)
4
5 my_function()
6 print(local_var) # خطأ: المتغير غير معرف خارج الدالة
```

### Output

```
C:\Users\HP\Documents\projectfunction\.venv\Scripts\python.exe C:\Users\HP\Documents\projectfunction\
10
Traceback (most recent call last):
 File "C:\Users\HP\Documents\projectfunction\.venv\Scripts\ex10.py", line 6, in <module>
 print(local_var) # خطأ: المتغير غير معرف خارج الدالة
 ^^^^^^^^^^^^^
NameError: name 'local_var' is not defined

Process finished with exit code 1
```

## شرح المثال:

### داخل الدالة : my\_function

- يتم تعريف المتغير `local_var` داخل الدالة `my_function()`.
- المتغير هنا يُعتبر محلياً (Local Variable) حيث تعريفه داخل الدالة فقط.
- عند استدعاء `my_function()`، يتم تنفيذ التعليمة البرمجية داخل الدالة، وطباعة قيمة `local_var` التي هي 10

### خارج الدالة:

- عند محاولة طباعة `local_var` الموجودة في السطر (7) خارج الدالة `my_function`.
- يظهر خطأ لأن المتغير `local_var` لا يمكن الوصول إليه لأنه خارج الدالة.

## مثال 11:

مجال المتغير العام في الدالة: (Global Variable).

التفسير: يتم تعريف المتغير العام `global_var` حيث يمكن استخدامه خارج نطاق الدالة.

### Code

```
1 def my_function():
2 local_var = 10 # تعريف متغير محلي داخل الدالة
3 print(local_var) # طباعة قيمة المتغير المحلي
4
5 global global_var # الإشارة إلى أن المتغير global_var هو متغير عام
6 global_var = 500 # تعديل أو إنشاء المتغير العام داخل الدالة
7
8
9 my_function()
10
11 # طباعة قيمة المتغير العام بعد تعديلها داخل الدالة
12 print(f"Global Variable is: {global_var}")
13
```

### Output

```
C:\Users\HP\Documents\projectfunction\
10
Global Variable is: 500

Process finished with exit code 0
```

### شرح المثال:

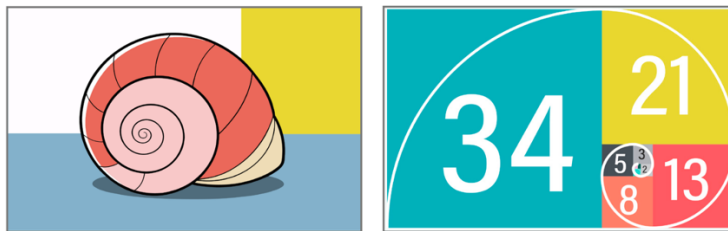
- تعريف الدالة `my_function`: تضمنت تعليمات لتعريف متغير محلي وآخر عام.
- `local_var` هو متغير محلي تم تعريفه داخل الدالة يمكن الوصول إلى `local_var` فقط داخل نطاق الدالة.
- `global_var`: تم إنشاؤه كمتغير عام باستخدام الكلمة المفتاحية `global` وتعيين قيمته 500.

## مثال 12:

## متتالية فيبوناتشي والنسبة الذهبية:

النسبة الذهبية، والمعروفة أيضًا باسم الرقم الذهبي أو النسبة الذهبية أو النسبة الإلهية، هي نسبة بين رقمين تساوي تقريبًا 1.618. وعادةً ما تُكتب بالحرف اليوناني  $\Phi$ ، وهي مرتبطة ارتباطًا وثيقًا بمتتالية فيبوناتشي. وهي سلسلة من الأرقام حيث يضاف كل رقم إلى الأخير قبله. أرقام فيبوناتشي هي 0، 1، 1، 2، 3، 5، 8، 13، 21، وهكذا مع نسبة كل رقم والرقم السابق يقترب تدريجيًا من 1.618، أو فاي ( $\Phi$ ).

البرنامج التالي يوضح كيفية الحصول على سلسلة فيبوناتشي مع حساب النسبة الذهبية، قدم حلول برمجية بديلة للحصول على سلسلة فيبوناتشي وحساب النسبة الذهبية.



أمله على النسبة الذهبية في الحياة

### Code

```
1 def fibonacci(number_fibonacci): # تعريف دالة لتوليد متتالية فيبوناتشي وحساب النسبة الذهبية
2 number_1 = 0 # الرقم الأول في متتالية فيبوناتشي
3 number_2 = 1 # الرقم الثاني في متتالية فيبوناتشي
4 sum_number = 1 # الرقم الثالث في المتتالية (مجموع الرقمين الأولين)
5
6 if number_fibonacci > 3: # التحقق من أن عدد العناصر أكبر من 3
7 print(number_1, number_2, sum_number, end=" ") # طباعة الأرقام الثلاثة الأولى في المتتالية
8
9 # توليد الأرقام الباقية باستخدام حلقة
10 for i in range(3, number_fibonacci): # بدءًا من الرقم الرابع وحتى العدد المطلوب
11 # تحديث قيم الأرقام بناءً على قواعد متتالية فيبوناتشي
12 number_1, number_2 = number_2, sum_number # تبديل القيم
13 sum_number = number_1 + number_2 # حساب الرقم التالي
14 print(sum_number, end=" ") # طباعة الرقم الجديد
15
16 # حساب النسبة الذهبية
17 golden_ratio = sum_number / number_2 # قسمة الرقم الأكبر على الرقم الذي يسبقه
18 print(f"\nThe Golden Ratio is: {golden_ratio:.2f}") # طباعة النسبة الذهبية
19 else:
20 print("Enter number more than 3") # طباعة رسالة إذا كان العدد المدخل أقل من 3
21
```

### Output

```
C:\Users\HP\Documents\projectfunction\.venv\Scripts\python.exe
Enter number of Fibonacci Sequence: 6
0 1 1 2 3 5
The Golden Ratio is: 1.67
Process finished with exit code 0
```

## شرح المثال:

- كيفية توليد متتالية فيبوناتشي:
- إذا أردنا توليد أول 6 أرقام من المتتالية: 0 ، 1 ، 1 ، 2 ، 3 ، 5
- الرقم الأول 0
- الرقم الثاني 1
- الرقم الثالث  $1 = 1 + 0$
- الرقم الرابع  $2 = 1 + 1$
- الرقم الخامس  $3 = 2 + 1$
- الرقم السادس  $5 = 3 + 2$
- الرقم السابع  $8 = 5 + 3$
- □ تخصيص القيمة 0 للمتغير n
- التأكيد متتالية فيبوناتشي تبدأ دائماً بـ 0 و 1
- وضع شرط للمستخدم ان يجب عدد العناصر المطلوبة في السلسلة أكبر من 3
- إذا كان العدد أقل من أو يساوي 3، يتم طباعة رسالة للمستخدم

```
print("Enter number more than 3")
```

النسبة الذهبية تُحسب باستخدام آخر رقمي وهي تقريباً تساوي :

$$1,67 \approx \frac{5}{3} = \frac{\text{الأخير الرقم}}{\text{يسبقه الذي الرقم}} = \text{Golden Ratio}$$

```
golden_ratio = sum_number / number_2
```

```
print(f"\nThe Golden Ratio is: {golden_ratio:.2f}")
```

- ```
print(number_1, number_2, sum_number, end=" ")
```
- طباعة أول ثلاثة أرقام مباشرة 0 ، 1 ، 1
- استخدام `end=" "` لطباعة الأرقام في نفس السطر مع مسافات بينها.

مثال 13:

برنامج آلة حاسبة بسيطة
باستخدام الدوال.

توظيف الدوال - لإجراء العمليات الحسابية بين عددين مدخلين من قبل المستخدم.

Code

```
1  # تعريف دالة الجمع
2  def addition(a, b):
3      return a + b # تُرجع ناتج جمع العددين
4  # تعريف دالة الطرح
5  def subtraction(a, b):
6      return a - b # تُرجع ناتج طرح العدد الثاني من الأول
7  # تعريف دالة الضرب
8  def multiplication(a, b):
9      return a * b # تُرجع ناتج ضرب العددين
10 # تعريف دالة القسمة
11 def division(a, b):
12     if b == 0: # التحقق من أن القاسم ليس صفرًا
13         return "Sorry, Cannot divide by zero!!" # رسالة خطأ إذا كان القاسم صفرًا
14     return a / b # تُرجع ناتج القسمة
15 # تعريف دالة الحاسبة الرئيسية
16 def simple_calculator():
17     # طلب المدخلات من المستخدم
18     number1 = float(input("Enter first number: ")) # إدخال العدد الأول
19     number2 = float(input("Enter second number: ")) # إدخال العدد الثاني
20     # طلب العملية الحسابية المطلوبة
21     user_input = input("Enter your choice (+, -, *, /): ") # إدخال نوع العملية
22     # تنفيذ العملية بناءً على اختيار المستخدم
23     if user_input == "+":
24         result = addition(number1, number2) # استدعاء دالة الجمع
25     elif user_input == "-":
26         result = subtraction(number1, number2) # استدعاء دالة الطرح
27     elif user_input == "*":
28         result = multiplication(number1, number2) # استدعاء دالة الضرب
29     elif user_input == "/":
30         result = division(number1, number2) # استدعاء دالة القسمة
31     else:
32         # رسالة إذا كانت العملية المدخلة غير صحيحة
33         print("Invalid input.")
34         exit() # إنهاء البرنامج
35     # طباعة النتيجة النهائية
36     print(result)
37     # استدعاء دالة الحاسبة الرئيسية لتشغيل البرنامج
38     simple_calculator()
```

Output

```
C:\Users\HP\Documents\projectfunction\
Enter first number: 43
Enter second number: 5
Enter your choice :(+, -, *, /): /
8.6
Process finished with exit code 0
```

شرح المثال:

- التأكيد على تعريف الدوال الحسابية تعريف أربع دوال، كل دالة مسؤولة عن عملية حسابية واحدة.
- دالة الحاسبة الرئيسية: (Simple Calculator)

- def simple_calculator:()

- جمع المدخلات من المستخدم، تحديد العملية المطلوبة، وتنفيذ العملية باستخدام الدوال المعرفة مسبقاً.

- باستخدام تعليمات الشرط: if-elif-else

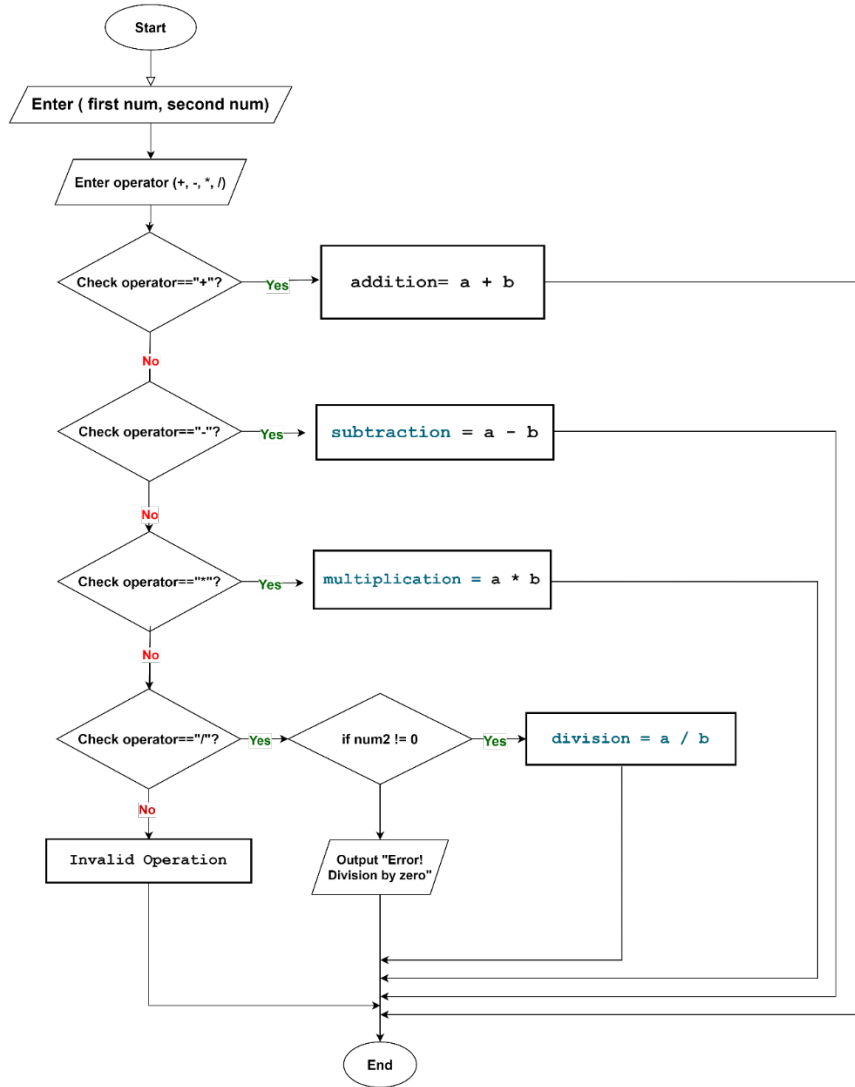
▪ التحقق من رمز العملية المدخل.

▪ استدعاء الدالة المناسبة بناءً على العملية.

▪ إذا كان الإدخال غير صحيح، يتم طباعة رسالة خطأ والخروج من البرنامج.

- التأكد من ظهور رسالة تنبيه في حال القسمة على صفر.

خريطة التدفق



تطوير البرنامج بحيث يعمل بشكل مستمر.
إيقاف تشغيل البرنامج بطريقة صحيحة (الخروج من التكرار).
التأكد من إدخال المستخدم لقيم عددية فقط.

إثرائني: تطوير البرنامج
في مثال 13:

Code

Reader Mode

```
1  # تعريف دالة الجمع
2  def addition(a, b):
3      return a + b
4  # تعريف دالة الطرح
5  def subtraction(a, b):
6      return a - b
7  # تعريف دالة الضرب
8  def multiplication(a, b):
9      return a * b
10 # تعريف دالة القسمة
11 def division(a, b):
12     if b == 0: # التحقق من أن القاسم ليس صفرًا
13         return "Sorry, Cannot divide by zero!!"
14     return a / b
15 # تعريف دالة الحاسبة الرئيسية
16 def simple_calculator():
17     while True: # حلقة تعمل بشكل مستمر
18         try:
19             # إدخال العدد الأول
20             user_input1 = input("Enter first number (or type 'exit' to quit): ")
21             if user_input1.lower() == "exit": # التحقق من الخروج
22                 print("Exiting the calculator. Goodbye!")
23                 break
24             number1 = float(user_input1) # تحويل الإدخال إلى عدد
25             # إدخال العدد الثاني
26             user_input2 = input("Enter second number (or type 'exit' to quit): ")
27             if user_input2.lower() == "exit": # التحقق من الخروج
28                 print("Exiting the calculator. Goodbye!")
29                 break
30             number2 = float(user_input2) # تحويل الإدخال إلى عدد
31             # إدخال العملية الحسابية
32             operation = input("Enter an operation (+, -, *, /) or type 'exit' to quit: ")
33             if operation.lower() == "exit": # التحقق من الخروج
34                 print("Exiting the calculator. Goodbye!")
35                 break
36             # تنفيذ العملية بناءً على اختيار المستخدم
37             if operation == "+":
38                 result = addition(number1, number2)
39             elif operation == "-":
40                 result = subtraction(number1, number2)
41             elif operation == "*":
42                 result = multiplication(number1, number2)
43             elif operation == "/":
44                 result = division(number1, number2)
45             else:
46                 print("Invalid operation. Please try again.")
47                 continue
48             # طباعة النتيجة النهائية
49             print(f"Result: {result}")
50         except ValueError:
51             print("Invalid input! Please enter numeric values only.")
52     # استدعاء دالة الحاسبة الرئيسية لتشغيل البرنامج
53     simple_calculator()
```

| البرنامج الأول | البرنامج الثاني |
|--|--|
| <p>1. أدخل قيمة العدد مباشرة</p> <pre> number1 = float(input("Enter first number: ")) number2 = float(input("Enter second number: ")) </pre> | <p>1. إضافة خيار الخروج أثناء إدخال القيمة العدد وذلك بإضافة شرط</p> <pre> user_input1 = input("Enter first number (or type 'exit' to quit): ") if user_input1.lower() == "exit:": print("Exiting the calculator. Goodbye!") break number1 = float(user_input1) ===== user_input2 = input("Enter second number (or type 'exit' to quit): ") if user_input2.lower() == "exit:": print("Exiting the calculator. Goodbye!") break number2 = float(user_input2) </pre> |
| <p>2. لا يعمل في حلقة، يتم إنهاء التنفيذ مباشرة بعد إتمام عملية واحدة أو إدخال خاطئ، يحتاج المستخدم لإعادة تشغيل البرنامج يدوياً.</p> | <p>2. حلقة مستمرة باستخدام while True، الهدف إجراء عمليات عدة دون إعادة تشغيل البرنامج.</p> |
| <p>3. لا يوجد معالجة للأخطاء في حالة الخطأ بإدخال قيمة غير عددية، سيؤدي إلى تعطل البرنامج.</p> | <p>3. معالجة للأخطاء باستخدام try/except:</p> <pre> Try: مقاطع برمجية except ValueError: print("Invalid input! Please enter numeric values only.") </pre> |
| <p>مزايا البرنامج : بسيط وسريع – غير مرن – لا يعالج الأخطاء.</p> | <p>مزايا البرنامج : التنظيم – مرن – معالجة الأخطاء.</p> |

Code

```

1  # قائمة المواقع المشبوهة
2  suspicious_domains = ["hacker.com", "fake.com", "phishingsite.com"]
3  # دالة لعرض قائمة النطاقات المشبوهة
4  def view_suspicious(suspicious_domains):
5      print(suspicious_domains) # طباعة النطاقات المشبوهة
6  # دالة لإضافة نطاق جديد إلى قائمة المواقع المشبوهة
7  def add_suspicious(suspicious_domains, email):
8      domain = email.split("@", 1)[-1] # استخراج النطاق (domain) من البريد الإلكتروني
9      if domain not in suspicious_domains: # التحقق إذا كان النطاق غير موجود في القائمة
10         suspicious_domains.append(domain) # إضافة النطاق إلى القائمة
11         print(f"{domain} added") # إعلام المستخدم بأنه تمت الإضافة
12     else:
13         print(f"{domain} already found") # إعلام المستخدم بأن النطاق موجود مسبقاً
14 # (Main Loop) الحلقة الرئيسية
15 while True:
16     # عرض الخيارات للمستخدم
17     print("\n[1] Detect & Add Email\n[2] Print Domain List\n[3] Exit")
18     choice = input("Enter Your Choice: ") # طلب إدخال الخيار من المستخدم
19     # في حال اختيار المستخدم الخيار الأول
20     if choice == "1":
21         email = input("Enter Email: ") # طلب إدخال الخيار من المستخدم
22         if "@" in email: # التحقق من صحة البريد (يحتوي على @)
23             add_suspicious(suspicious_domains, email) # إضافة النطاق إلى القائمة إذا كان البريد صحيحاً
24         else:
25             print("Invalid email. Please include '@'.") # رسالة خطأ إذا كان البريد غير صالح
26     # في حال اختيار المستخدم الخيار الثاني
27     elif choice == "2":
28         view_suspicious(suspicious_domains) # عرض قائمة النطاقات المشبوهة
29     # في حال اختيار المستخدم الخيار الثالث (الخروج)
30     elif choice == "3":
31         break # إنهاء البرنامج
32     # في حال إدخال خيار غير صحيح
33     else:
34         print("Wrong choice.") # رسالة خطأ إذا كان الخيار غير صحيح

```

الغرض من البرنامج:

a. إدارة قائمة تحتوي على النطاقات المشبوهة (Suspicious Domains).
السماح للمستخدم بالقيام بما يلي:

1. التحقق من نطاق البريد الإلكتروني المدخل لمعرفة إذا كان مشبوهاً.
2. إضافة النطاقات الجديدة المستخرجة من عناوين البريد الإلكتروني إلى القائمة إذا لم تكن موجودة مسبقاً.
3. عرض قائمة النطاقات المشبوهة.
4. الخروج من البرنامج بشكل سهل.

b. التعليمات البرمجية المناسبة لتنفيذ المهام:
إذا تم إدخال بريد إلكتروني غير صحيح.

```
email = input("Enter Email: ")
if "@" in email:
    add_suspicious(suspicious_domains, email)
else:
    print("Invalid email. Please include '@'.")
```

إضافة بريد إلكتروني مدخل من المستخدم إلى قائمة البريد الإلكتروني

```
def add_suspicious(suspicious_domains, email):
    domain = email.split
    if domain not in suspicious_domains :
        suspicious_domains.append(domain)
        print(f"{domain} added")
    else:
        print(f"{domain} already found")
```

c. ناتج تنفيذ التعليمات البرمجية التالية:

```
add_suspicious(suspicious_domains, email):
```

يتم التحقق من النطاق المستخرج من البريد الإلكتروني

```
domain = email.split("@")[-1]
```

استخراج النطاق (domain) من البريد الإلكتروني

```
if domain not in suspicious_domains:
```

التحقق إذا كان النطاق غير موجود في قائمة

```
suspicious_domains.append(domain)
```

يتم إضافته إلى القائمة

```
print(f"{domain} added")
```

يتم طباعة رسالة: "{domain} added" :

```
else:
```

```
print(f"{domain} already found")
```

إعلام المستخدم بأن النطاق موجود مسبقًا

```
def view_suspicious(suspicious_domains)
    print(suspicious_domains)
```

وظيفة الدالة طباعة جميع النطاقات الموجودة حاليًا في قائمة suspicious_domains

المكتبات Libraries

تتميز لغة البرمجة Python بتوفر العديد من المكتبات في مختلف المجالات؛ منها تحليل البيانات Data Analysis، الذكاء الاصطناعي Artificial Intelligence، تعلم الآلة Machine Learning، التعلم العميق Deep Learning، الأمن السيبراني Cybersecurity، وتطوير الألعاب Gaming Development.

المكتبات في بايثون (Python Libraries):

المكتبة عبارة عن ملف يتضمن مجموعة من التعليمات البرمجية والوظائف المعرفة، كالدوال functions، والفئات classes والمتغيرات variables، والتي تُسهل في حل مشكلات معينة، أو تنفيذ عمليات محددة، والتي يمكن استدعاؤها واستخدامها في البرامج لتسهيل تنفيذ العديد من المهام.

أهمية استخدام المكتبات:

- **إعادة استخدام المقاطع البرمجية:** المكتبات توفر مقاطع برمجية جاهزة للاستخدام بدلاً من إعادة كتابتها بشكل متكرر.
- **توفير الوقت والجهد:** من خلال استدعاء الدوال الموجودة في المكتبات بدلاً من بناء دوال من الصفر.
- **سهولة الصيانة والتحديث:** المكتبات تتيح استخدام حلول مجربة، ومتقدمة مع القدرة على تحديثها بسهولة عندما يتم إصدار تحديثات جديدة.

أشهر مكتبات لغة بايثون:

- **مكتبة NumPy:** تتعامل مع الدوال الخاصة بالعمليات الحسابية، الجبر الخطي، والمصفوفات.
- **مكتبة Pandas:** استكشاف وتحليل ومعالجة البيانات.
- **مكتبة Matplotlib:** تمثيل البيانات Data Visualization، وتصميم الأشكال البيانية Graph.
- **مكتبة OpenCV:** تعمل في مجال الرؤية الحاسوبية Computer Vision، وتحليل الصور ومقاطع الفيديو.
- **مكتبة TensorFlow:** مكتبة رائدة في مجالات تعلم الآلة والتعلم العميق.
- **مكتبة PyTorch:** مكتبة شهيرة في مجال التعلم العميق ومعالجة اللغات الطبيعية والرؤية الحاسوبية.

أقسام المكتبات:

المكتبات الخاصة

Customize Libraries

3

مكتبات يتم تطويرها من قبل المبرمج (إنشاء ملف Python) تتضمن مجموعة من الدوال، functions، الفئات classes، المتغيرات variables، لتلبية احتياجات محددة بالمشروع.

المكتبات الخارجية

Third-Party Libraries

2

مكتبات يتم تطويرها من قبل مجتمع مطوري لغة Python، الشركات والمؤسسات، الجامعات والمراكز البحثية، المنظمات غير الربحية، والمبرمجين، ويجب تثبيتها للتعامل معها على سبيل المثال مكتبة NumPy للعمليات الحسابية وتحليل البيانات، مكتبة Pandas لتحليل ومعالجة البيانات، ومكتبة TensorFlow لتعلم الآلة والذكاء الاصطناعي.

المكتبات القياسية

Standard Libraries

1

مكتبات مدمجة مع لغة Python، تغطي العديد من الوظائف الأساسية مثل التعامل مع الملفات، العمليات الحسابية، معالجة النصوص، والتعامل مع البيانات؛ ومنها مكتبة math لإجراء العمليات الحسابية، ومكتبة datetime للتعامل مع الوقت والتاريخ، ومكتبة random لتوليد القيم العشوائية.

التعامل مع المكتبات الخارجية Third-Party Libraries:

تثبيت المكتبات الخارجية Third-Party Libraries:

الطريقة الأولى:

مثال: تثبيت مكتبة numpy باستخدام الأمر pip install:

1- فتح المشروع في PyCharm



2- من الشريط أسفل نافذة البرنامج PyCharm اختيار تبويب Terminal

```
pip install (library name)
```

3- يتم كتابة الأمر

Terminal screen

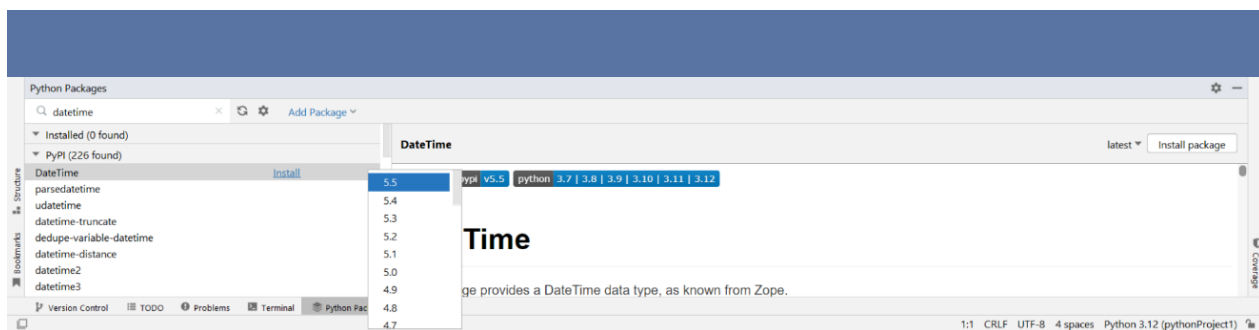
```
Terminal: Local x + v
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS C:\Users\HP\Documents\pythonProjectnInstall> pip install NumPy
Collecting NumPy
  Obtaining dependency information for NumPy from https://files.pythonhosted.org/packages/a6/84/fa11dad3404b7634aaab50733581ce11e5350383311ea7a7010f464c0170/numpy-2.1.3-cp312-cp312-win_amd64.whl.metadata
  Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl.metadata (60 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.8/60.8 kB 404.8 kB/s eta 0:00:00
  Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl (12.6 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.6/12.6 MB 25.2 MB/s eta 0:00:00
Installing collected packages: NumPy
Successfully installed NumPy-2.1.3

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\HP\Documents\pythonProjectnInstall>
```

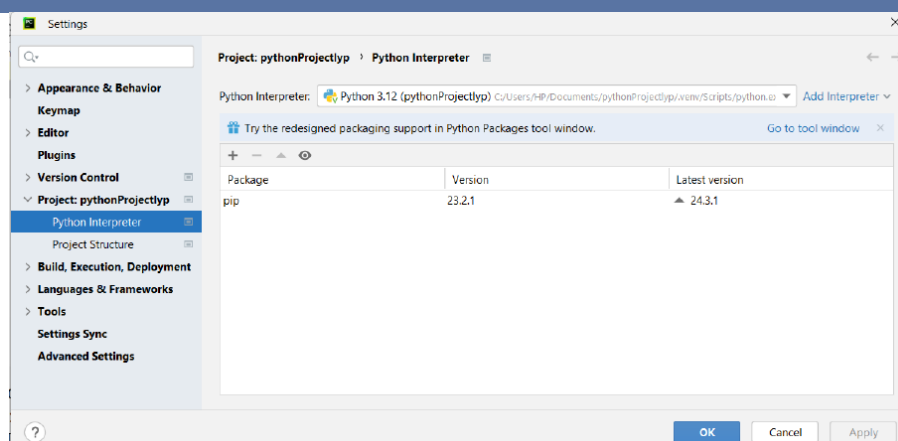
الطريقة الثانية:

- 1- فتح المشروع في PyCharm .
- 2- من الشريط أسفل نافذة PyCharm اختيار تبويب Python Package
- 3- البحث عن المكتبة المناسبة في خانة البحث.
- 4- بعد ظهور نتيجة البحث يتم تحديد اسم المكتبة واختيار الإصدار المناسب ثم تثبيتها .



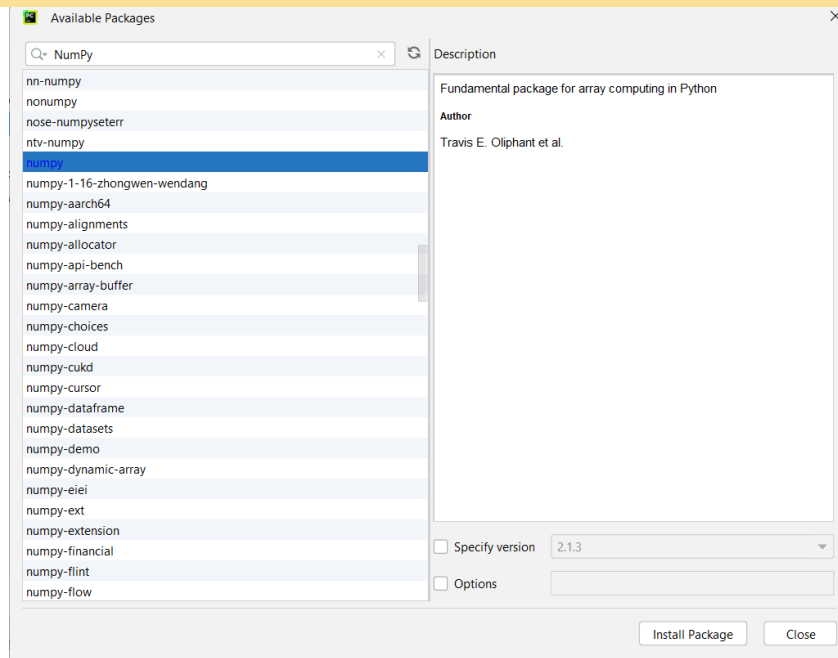
الطريقة الثالثة:

- 1- اختيار الأمر Settings من قائمة File .
- 2- اختيار أمر Project (اسم المشروع) من القائمة الجانبية، ثم تحديد Python Interpreter.

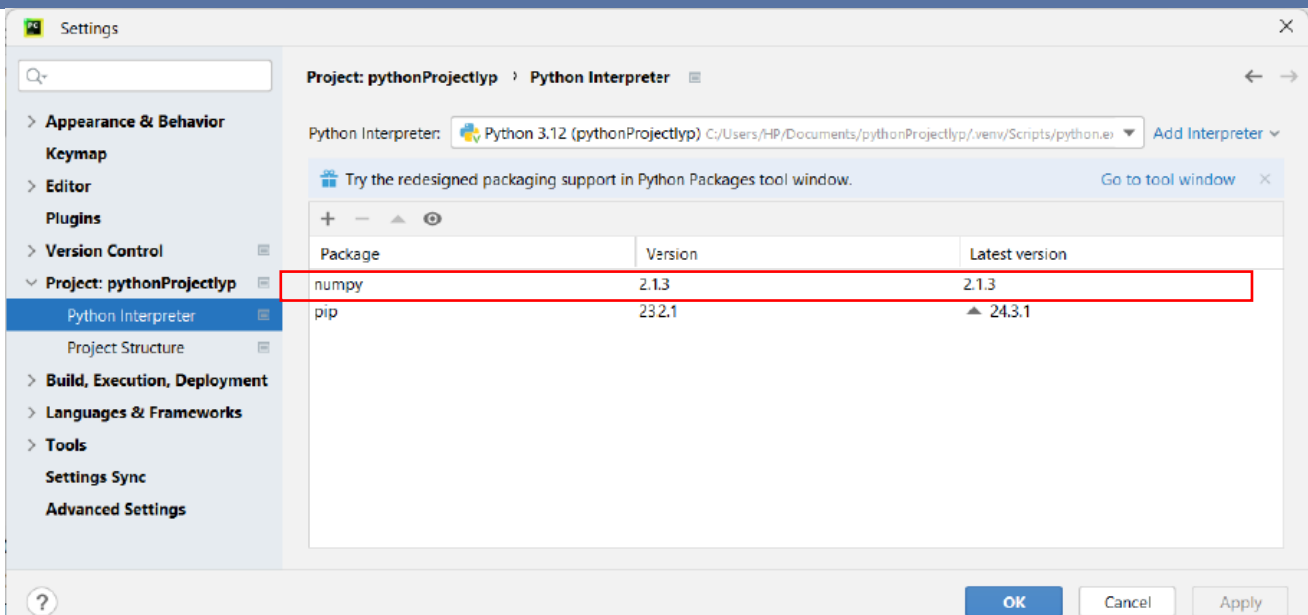


3. تحديد زر (+) في الجانب الأيمن لفتح نافذة Available Packages

Available Packages



4. كتابة اسم المكتبة "NumPy" في شريط البحث وستظهر في قائمة النتائج.
5. بمجرد العثور على المكتبة المطلوبة، يتم الضغط على **Install Package**.



إلغاء تثبيت المكتبات الخارجية :Uninstall Third-Party Libraries

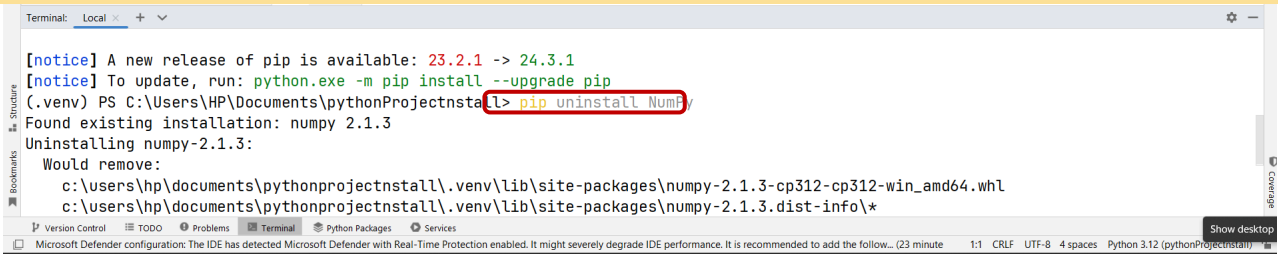
الطريقة الأولى:

1. من الشريط أسفل نافذة البرنامج **PyCharm** اختيار تبويب **Terminal**.

2. كتابة أمر إلغاء التثبيت الأمر **pip uninstall** متبوعاً باسم المكتبة التي تريد إلغاء تثبيتها

`pip uninstall (library name)`

مثال :



```
Terminal: Local x + v
[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\HP\Documents\pythonProjectnsta> pip uninstall NumPy
Found existing installation: numpy 2.1.3
Uninstalling numpy-2.1.3:
  Would remove:
    c:\users\hp\documents\pythonprojectnsta\venv\lib\site-packages\numpy-2.1.3-cp312-cp312-win_amd64.whl
    c:\users\hp\documents\pythonprojectnsta\venv\lib\site-packages\numpy-2.1.3.dist-info\*
  Proceed (Y/n)? y
Successfully uninstalled numpy-2.1.3
(.venv) PS C:\Users\HP\Documents\pythonProjectnsta>
```

تأكيد إلغاء التثبيت اضغط **y** ثم **Enter** للتأكيد

الطريقة الثانية:

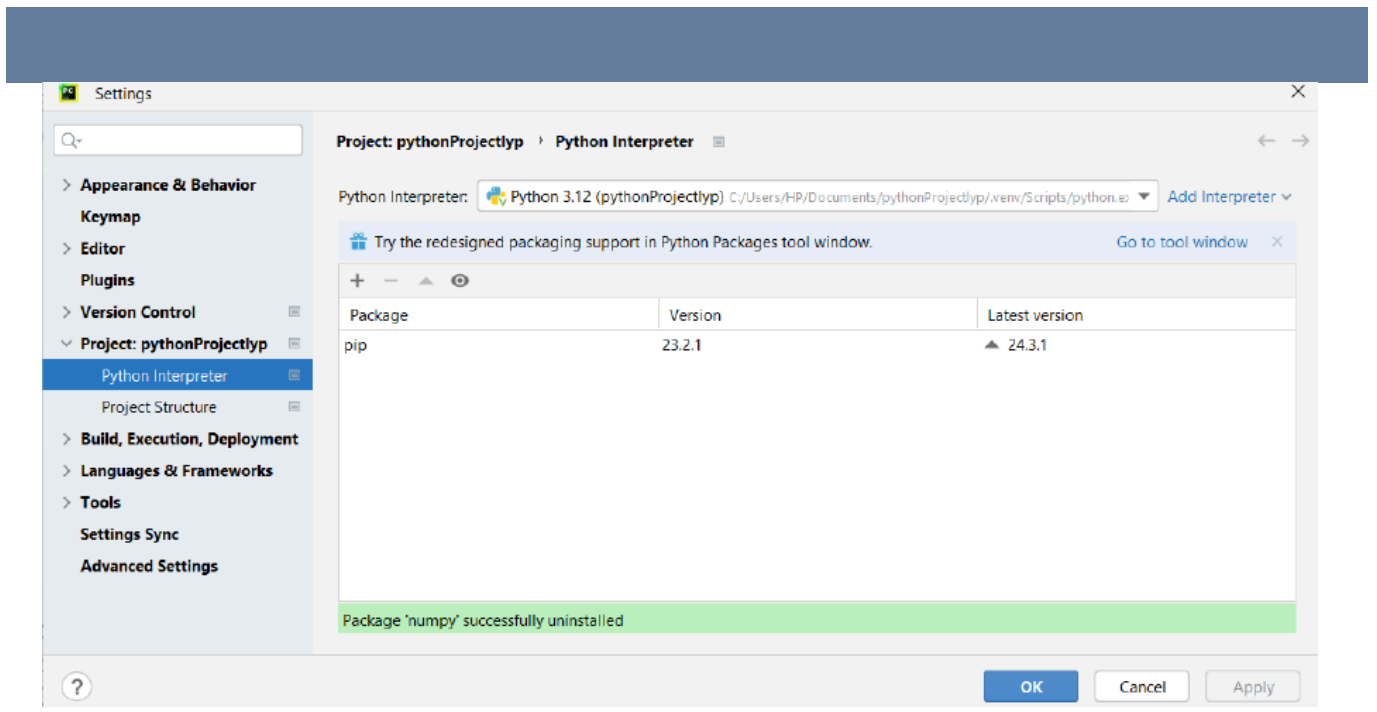
1- اختيار الأمر **Settings** من قائمة **File**

2- اختيار **Project** (اسم المشروع) من القائمة الجانبية، ثم تحديد **Python Interpreter**

3- تظهر قائمة بجميع المكتبات المثبتة في المشروع يتم البحث عن المكتبة التي تريد إلغاء تثبيتها (مثل NumPy)

4- تحديد المكتبة، ثم الضغط على زر **(-) Uninstall** بجانب اسم المكتبة

5- تظهر رسالة تأكيد إلغاء التثبيت لإزالة المكتبة من المشروع.



استيراد المكتبات الخارجية Import Libraries:

بعد تثبيت المكتبة باستخدام `install` لتفعيل المكتبة واستخدام الدوال والوظائف التي تحتوي عليها نحتاج إلى استيرادها عن طريق التعليمة برمجية.

`import library name`

أولاً : استيراد مكتبة كاملة

استيراد مكتبة **datetime** المخصصة للتعامل مع التاريخ والوقت. في المثال التالي تم استيراد دالة التاريخ والوقت باستخدام الأمر `import`. ثم أدرجنا التعليمات البرمجية التي تعرض كيفية الوصول إلى مكونات التاريخ (اليوم، الشهر، السنة) باستخدام مكتبة **datetime**، مما يسهل التعامل مع التواريخ في البرامج.

مثال 1 :

ملاحظة : يجب ادراج اسم المكتبة أولاً متبوعاً باسم الدالة .

Code

```
1 import datetime
2
3 print(datetime.date.today()) # طباعة التاريخ الكامل
4 print(datetime.date.today().day) # طباعة اليوم
5 print(datetime.date.today().month) # طباعة الشهر
6 print(datetime.date.today().year) # طباعة السنة
7
```

Output

```
↑ C:\Users\HP\Documents\pythonProject
↓ 2024-11-10
⌵ 10
⌵ 11
⌵ 2024
🗑 Process finished with exit code 0
```

عرض الوقت والتاريخ الحالي ○

Code

```
1 import datetime
2
3 # الحصول على التاريخ والوقت الحالي
4 current_datetime = datetime.datetime.now()
5
6 print("التاريخ والوقت الحالي:", current_datetime)
7
```

Output

```
▶ ↑ C:\Users\HP\Documents\pythonProject\pythonProject\Scripts
⌵ ↓ 11:33:03.347206 10-11-2024 التاريخ والوقت الحالي:
⌵
```

استيراد مكتبة random .

مكتبة **random** من المكتبات المدمجة في **Python** ، وتساعد على العمل مع الأعداد العشوائية لتوليد عدد عشوائي ضمن مجال محدد، سنتناول بعض دوالها بالأمثلة التوضيحية :

مثال 2 : استخدم مكتبة random لتوليد عدد صحيح عشوائي بين 0 , 1

random

توليد رقم عشوائي بين 0 و 1

```
import random
random_num=random.random()
print (random_num)
```

Output

0. 7784885907298166

randint

الدالة random.randint هي دالة مخصصة لتوليد أعداد صحيحة بشكل عشوائي ضمن مجال محدد. وهي جزء من مكتبة random .

ملاحظة: عند استدعاء المكتبة كاملة باستخدام الأمر import، يجب كتابة اسم المكتبة random في كل مرة نستخدم فيها أحد دوالها متبوعاً بنقطة ثم اسم الدالة المطلوبة مثل (randam.randint).

مثال 3: برنامج "رمي النرد" افتراضياً والحصول على رقم بين 1 و 6

Code

```
1 import random
2
3 # توليد رقم عشوائي بين 1 و 6 (يمثل رمي النرد)
4 nrd = random.randint(a: 1, b: 6)
5 print("نتيجة رمي النرد:", nrd)
6
```

Output

```
C:\Users\HP\Documents\pythonProject\yp\
نتيجة رمي النرد: 5
Process finished with exit code 0
```

مثال 4 : توليد رقم عشوائي بين 1 و 100

Code

```
1 import random
2
3 # توليد عدد عشوائي بين 1 و 100
4 random_number = random.randint(a: 1, b: 100)
5
6 print("العدد العشوائي هو:", random_number)
7
```

Output

```
C:\Users\HP\Documents\pythonProject\yp\
73 العدد العشوائي هو:
Process finished with exit code 0
```

مثال 5 : توليد رقم عشوائي بين 1 و 10

Code

```
1 import random
2
3 random_num = random.randint(a: 1, b: 10) # توليد عدد عشوائي بين 1 و 10
4 print(f"The random integer is = {random_num}") # طباعة العدد العشوائي
5
```

Output

```
C:\Users\HP\Documents\pythonProject\yp\
The random integer is = 9
Process finished with exit code 0
```

شرح المثال :

في البرنامج يتم استيراد مكتبة random: بكتابة التعليمة البرمجية import random:

- توليد عدد عشوائي باستخدام randint :
 - نستخدم الدالة randint(1, 10) من مكتبة random لتوليد عدد صحيح عشوائي بين 1 و 10.
 - يتم تخزين العدد العشوائي الناتج في المتغير random_num.
 - طباعة العدد العشوائي باستخدام f-string:

```
print(f"The random integer is = {random_num}")
```

f-string لإدراج قيمة random_num داخل النص

ملاحظة:

- **f-string** هي طريقة سهلة ومباشرة لإدراج القيم المتغيرة داخل النصوص ،
- لإنشاء **f-string**، إدراج الحرف **f** قبل علامات التنصيص **" "** أو **' '** داخل النص ثم يمكنك وضع القيم المتغيرة بين **{ }** لتظهر كجزء من النص.
- عند استدعاء مكتبة كاملة في Python باستخدام الأمر **import**، يجب كتابة اسم المكتبة متبوعاً بنقطة. قبل استخدام أي دالة من دوالها **تكتب للتأكد أنها موجودة في تلك المكتبة.**

ثانياً : استيراد عنصر محدد (التعليمات البرمجية الدوال functions ، والفئات classes ، والمتغيرات variables) من

مكتبة محددة

From library name import part of library

يمكن استيراد عنصر محدد بدلاً من استيراد المكتبة كاملة مما يجعل التعليمة البرمجية أبسط وذات كفاءة حيث تستورد فقط ما يتم احتياجه من المكتبة.

استيراد الدالة `asctime` فقط، من مكتبة `time` ، والتعامل معها.

مثال 1 :

Code

```
1 # استيراد دالة asctime فقط من مكتبة time
2 from time import asctime
3
4 # طباعة التاريخ والوقت الحالي باستخدام الدالة asctime()
5 print(f"The current date and time is = {asctime()}")
```

Output

```
C:\Users\HP\Documents\pythonProject\python.exe
The current date and time is = Mon Nov 11 08:30:20 2024
Process finished with exit code 0
```

نستورد الدالة `asctime` فقط من مكتبة `time`، ولا نستورد المكتبة كاملة.

مما يسمح لنا استخدام `asctime()` مباشرة دون الحاجة لكتابة `time.asctime()`، مما يجعل الكود أكثر اختصارًا.

مثال 2 :

استدعاء أكثر من دالة من نفس المكتبة.

استدعاء دالتين محددتين (`randint` و `randrange`) من مكتبة `random` لتوليد أعداد صحيحة عشوائية.

Code

```
1 # استيراد الدالتين randint و randrange فقط من مكتبة random
2 from random import randint, randrange
3
4 random_num1 = randint(a: 1, b: 10) # توليد عدد صحيح عشوائي بين 1 و 10
5 random_num2 = randrange(10) # توليد عدد صحيح عشوائي من 0 إلى 9
6
7 print("العدد العشوائي بين 1 و 10 هو:", random_num1)
8 print("العدد العشوائي بين 0 و 9 هو:", random_num2)
```

Output

```
C:\Users\HP\Documents\pythonProject\python.exe
العدد العشوائي بين 1 و 10 هو: 4
العدد العشوائي بين 0 و 9 هو: 2
Process finished with exit code 0
```

لم نستورد المكتبة كاملة، بل استوردنا الدالتين اللتين نحتاجهما فقط، مما يجعل التعليمات البرمجية أكثر وضوحًا وكفاءة.

الفرق بين الدالة randint و randrange

| الدالة (end) randrange | الدالة (start, end) randint |
|--|---|
| random_num2 = randrange(10)
randrange(10) ستُعيد عددًا عشوائيًا بين 0 والقيمة داخل القوس 9 (يشمل 0 ويستثني 10)
النتيجة قد تكون أي رقم من الأرقام التالية: 0، 1، 2، ...، 9. | random_num1 = randint(1, 10)
تُعيد عددًا صحيحًا عشوائيًا بين القيمتين start و end ، (يشمل العددين 1 و 10)
النتيجة قد تكون أي رقم من الأرقام التالية: 1، 2، 3، ...، 10. |

مثال 3 : استيراد أكثر من دالة من مكتبات متعددة

Code

```

1 from dateutil.relativedelta import relativedelta
2 from datetime import datetime
3 today_date = datetime.today() # تاريخ اليوم
4 birth_date = datetime(2000, 12, 17) # تاريخ الميلاد
5 # كائن delta من النوع relativedelta، والذي يمثل الفرق بين التاريخين
6 delta = relativedelta(today_date, birth_date) # حساب العمر (الفرق بين تاريخ الميلاد وتاريخ اليوم)
7 print(f"Age: {delta.years} years, {delta.months} months, {delta.days} days")
    
```

ملاحظة:

- استيراد الدالة relativedelta من مكتبة dateutil.relativedelta: حساب الفرق بين تاريخين بدقة.
- استيراد الدالة datetime.today من مكتبة datetime: الحصول على التاريخ الحالي.
- تأكد من تثبيت مكتبة dateutil (في Terminal) باستخدام الأمر pip install python-dateutil.

ثالثا : استيراد المكتبة والتعامل معها باسم مختصر

Import library name as shortcut name

يمكن التعبير عن اسم المكتبة بشكل مختصر لسهولة التعامل معها

مثال 1: التعبير عن اسم المكتبة `datetime` باسم مختصر `dt`.

Code

```
1 import datetime as dt
2
3 print(dt.date.today()) # طباعة التاريخ الكامل
4 print(dt.date.today().day) # طباعة اليوم
5 print(dt.date.today().month) # طباعة الشهر
6 print(dt.date.today().year) # طباعة السنة
7
```

رابعا : استيراد جميع الكائنات (مثل الدوال، المتغيرات، الفئات) من مكتبة معينة.

مثال 1: استيراد جميع الكائنات (مثل الدوال، الفئات، التعاون) من المكتبة `datetime`:

يمثل `*` رمز **Asterisk** إمكانية التعامل مع جميع محتويات المكتبة، دون الحاجة إلى كتابة اسم المكتبة قبلها.

From **library name** import *

Code

```
1 from datetime import * # استيراد جميع العناصر من مكتبة datetime
2
3 print(date.today()) # طباعة التاريخ الكامل بصيغة YYYY-MM-DD
4 print(date.today().day) # طباعة اليوم فقط كعدد صحيح
5 print(date.today().month) # طباعة الشهر فقط كعدد صحيح
6 print(date.today().year) # طباعة السنة فقط كعدد صحيح
7
```

Output

```
C:\Users\HP\Documents\pythonProject\yp\
2024-11-12
12
11
2024
Process finished with exit code 0
```

تم استيراد `datetime` باستخدام الرمز `(*)` ثم استخدمنا `date.today()` للحصول على تاريخ مع تفاصيله (اليوم، الشهر، السنة) وطباعة كل منها بشكل منفصل.

مثال 2 : `from random import *`

Code

```
1 # استيراد جميع العناصر من مكتبة random
2 from random import *
3
4 # استخدام الدالة randint لتوليد عدد صحيح عشوائي بين 1 و 10 مباشرة
5 random_num1 = randint(a: 1, b: 10)
6 print("العدد العشوائي بين 1 و 10 هو:", random_num1)
7
8 # توليد عدد صحيح عشوائي من 0 إلى 9
9 random_num2 = randrange(10)
10 print("العدد العشوائي بين 0 و 9 هو:", random_num2)
11
12 # اختيار حرف عشوائي من السلسلة النصية
13 random_char = choice("python")
14 print("الحرف العشوائي المختار هو:", random_char)
15
16 # استخدام الدالة random لتوليد عدد عشوائي بين 0 و 1
17 random_num2 = random()
18 print("العدد العشري العشوائي بين 0 و 1 هو:", random_num2)
```

Output

```
C:\Users\HP\Documents\pythonProjectLyp\.venv\Scripts\python.exe
العدد العشوائي بين 1 و 10 هو: 10
العدد العشوائي بين 0 و 9 هو: 6
الحرف العشوائي المختار هو: t
العدد العشري العشوائي بين 0 و 1 هو: 0.8845971067438608
Process finished with exit code 0
```

1. استيراد جميع العناصر من مكتبة **random**:
`from random import *`

▪ القصد استيراد جميع العناصر (الدوال والمتغيرات والفئات) من مكتبة **random**.

2. توليد عدد صحيح عشوائي بين 1 و 10 باستخدام **randint**:

○ `random_num1 = randint(1, 10)`

▪ الدالة **randint(start, end)** تعرض عددًا صحيحًا عشوائيًا شامل القيمتين.

▪ كل مرة تُشغّل فيها البرنامج، نحصل على عدد مختلف.

3. توليد عدد صحيح عشوائي بين 0 و 9 باستخدام **randrange**:

○ `random_num2 = randrange(10)`

▪ الدالة **randrange(end)** تعرض عددًا صحيحًا عشوائيًا ولا يشمل العدد الأخير المذكور (**end**)

▪ وهي دالة تفيد في عرض أعداد ضمن نطاق يبدأ من 0.

4. اختيار حرف عشوائي من السلسلة النصية - **choice** باستخدام **sequence**

- `random_char = choice("python"):`
 - الدالة `choice(sequence)` تختار عنصرًا عشوائيًا من `.sequence`.
 - في هذا المثال، السلسلة النصية `"python"` تتكون من الأحرف `p, y, t, h, o, n`.
 - `choice("python")` يختار حرفًا عشوائيًا من هذه السلسلة النصية.
- 5. توليد عدد عشري عشوائي بين 0 و 1 باستخدام `random`:
- `random_num3 = random():`
 - الدالة `random ()` تعرض عددًا عشويًا عشوائيًا (`float`) بين 0 و 1 (يشمل 0 ولكنه لا يشمل 1).

توثيق المكتبات

توثيق المكتبات هو دليل تفصيلي يوضح كيفية استخدام محتويات المكتبة بشكل صحيح، بما فيها (الدوال، المتغيرات، والفئات). يساعد التوثيق المطورين على فهم وظائف المكتبة وطرق استخدامها مما يسهل عليهم العمل مع مكتبات **Python** المختلفة.

كيفية توثيق المكتبات:

1. استخدام أداة **Terminal**
 - الحصول على ملف توثيقي لأي مكتبة عن طريق استخدام الأمر (`pydoc`) متبوعًا باسم المكتبة.
 - مثال: للحصول على توثيق مكتبة `random`.
- اكتب الأمر التالي في: **Terminal** (`pydoc random`)

```
Terminal: Local x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

(.venv) PS C:\Users\HP\Documents\pythonProject\yp> pydoc random
```

- سيظهر التوثيق الكامل لمكتبة `random` ، مع شرح لكل دالة ووظيفتها .

```
Terminal: Local x + v
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.12/library/random.html

    The following documentation is automatically generated from the Python
    source files. It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations. When in doubt, consult the module reference at the
    location listed above.

-- More --
```

يمكنك الضغط على مفتاح **Ctrl** والضغط على اسم المكتبة باستخدام الزر الأيسر للفأرة

أو الضغط **Enter** بجانب **-- More --** في أداة **terminal**

Terminal: Local x + v

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION

bytes

uniform bytes (values between 0 and 255)

integers

uniform within range

sequences

pick random element
pick random sample
pick weighted random sample
generate random permutation

distributions on the real line:

uniform
triangular

distributions on the real line:

uniform
triangular
normal (Gaussian)
lognormal
negative exponential
gamma
beta
pareto
Weibull

distributions on the circle (angles 0 to 2pi)

circular uniform
von Mises

discrete distributions

binomial

General notes on the underlying Mersenne Twister core generator:

- * The period is $2^{19937}-1$.
- * It is one of the most extensively tested generators in existence.

مثال 1:

توليد كلمة مرور قوية Strong Password

توليد كلمة مرور مركبة من الأرقام، الأحرف، والرموز.

Code

```
1 # مشروع توليد كلمة مرور قوية
2 # استيراد مكتبة random لتوليد عناصر عشوائية
3 import random
4
5 # قوائم تحتوي على الأحرف، الأرقام، والرموز التي سيتم استخدامها في كلمة المرور
6 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p']
7 numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
8 symbols = ['-', '@', '&', '*', '-']
9
10 # طباعة رسالة ترحيبية للمستخدم
11 print("Welcome to the PyPassword Generator!")
12
13 # استقبال عدد الأحرف والرموز والأرقام المطلوبة من المستخدم
14 gen_letters = int(input("How many letters would you like in your password?: "))
15 gen_symbols = int(input("How many symbols would you like?: "))
16 gen_numbers = int(input("How many numbers would you like?: "))
17
18 #تهيئة متغير كلمة المرور ليكون فارغاً، وسنقوم بإضافة الأحرف العشوائية إليه لاحقاً
19 password = ""
20
21 # إضافة الأحرف العشوائية إلى كلمة المرور بناءً على العدد المطلوب
22 for char in range(1, gen_letters + 1):
23     password += random.choice(letters) # اختيار حرف عشوائي من قائمة الأحرف وإضافته إلى كلمة المرور
24
25 # إضافة الرموز العشوائية إلى كلمة المرور بناءً على العدد المطلوب
26 for char in range(1, gen_symbols + 1):
27     password += random.choice(symbols) # اختيار رمز عشوائي من قائمة الرموز وإضافته إلى كلمة المرور
28
29 # إضافة الأرقام العشوائية إلى كلمة المرور بناءً على العدد المطلوب
30 for char in range(1, gen_numbers + 1):
31     password += random.choice(numbers) # اختيار رقم عشوائي من قائمة الأرقام وإضافته إلى كلمة المرور
32
33 # طباعة كلمة المرور النهائية للمستخدم
34 print("Password is:", password)
35
```

Output

```
Welcome to the PyPassword Generator!
How many letters would you like in your password?: 5
How many symbols would you like?: 5
How many numbers would you like?: 5
Password is: ewhLH@@_--63089
```

ملاحظات:

- التأكيد على استيراد مكتبة random، وكونها توفر دوالاً لتوليد عناصر عشوائية مثل **choice()**.
- القوائم تحتوي على عناصر سيتم اختيارها عشوائياً لإنشاء كلمة المرور
 - (**letters**) تحتوي أحرف.
 - (**numbers**) تحتوي أرقام.
 - (**symbols**) تحتوي رموز.

البرنامج يطلب من المستخدم تحديد عدد كل من الأحرف والرموز والأرقام التي تكون كلمة المرور.

input() يستخدم للحصول على المدخلات. و **int()** تحولها إلى عدد صحيح.

password = "" تهيئة متغير .

- حلقة لإضافة (**letters**) أحرف - (**numbers**) أرقام - (**symbols**) رموز (العشوائية بناءً على العدد الذي تم تحديده من المستخدم
- يختار حرفاً ورمزاً ورقماً عشوائياً من قوائم الثلاث ويضيفه إلى **password**.

Code

```
for char in range(1, gen_letters + 1):
    password += random.choice(letters)

for char in range(1, gen_symbols + 1):
    password += random.choice(symbols)

for char in range(1, gen_numbers + 1):
    password += random.choice(numbers)
```

Output

```
C:\Users\HP\Documents\pythonProject\python.exe
Welcome to the PyPassword Generator!
How many letters would you like in your password?: 3
How many symbols would you like?: 2
How many numbers would you like?: 2
Password is: kmn@@63

Process finished with exit code 0
```

طور البرنامج ليتم توليد كلمة مرور مركبة لا يقل عدد عناصرها عن 8، ولا يزيد عن 12.

Code

```
1 # مشروع توليد كلمة مرور قوية
2 # استيراد مكتبة random لتوليد عناصر عشوائية
3 import random
4
5 # قوائم تحتوي على الأحرف، الأرقام، والرموز التي سيتم استخدامها في كلمة المرور
6 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p']
7 numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
8 symbols = ['-', '@', '&', '*', '-']
9
10 # طباعة رسالة ترحيبية للمستخدم
11 print("Welcome to the PyPassword Generator!")
12
13 # تكرار إدخال عدد الأحرف والرموز والأرقام حتى يكون العدد الكلي بين 8 و 12
14 while True:
15     # استقبال عدد الأحرف، الرموز، والأرقام من المستخدم
16     gen_letters = int(input("How many letters would you like in your password?: "))
17     gen_symbols = int(input("How many symbols would you like?: "))
18     gen_numbers = int(input("How many numbers would you like?: "))
19
20     # حساب العدد الكلي للعناصر
21     total_characters = gen_letters + gen_symbols + gen_numbers
22
23     # التحقق من أن العدد الكلي بين 8 و 12
24     if 8 ≤ total_characters ≤ 12:
25         break # إذا كان العدد الكلي مناسباً، نخرج من الحلقة
26     else:
27         print("The total number of characters must be between 8 and 12. Please try again.")
28
29 #تهيئة متغير كلمة المرور ليكون فارغاً، وسنقوم بإضافة الأحرف العشوائية إليه لاحقاً
30 password = ""
31
32 # إضافة الأحرف العشوائية إلى كلمة المرور بناءً على العدد المطلوب
33 for char in range(gen_letters):
34     password += random.choice(letters) # اختيار حرف عشوائي من قائمة الأحرف وإضافته إلى كلمة المرور
35
36 # إضافة الرموز العشوائية إلى كلمة المرور بناءً على العدد المطلوب
37 for char in range(gen_symbols):
38     password += random.choice(symbols) # اختيار رمز عشوائي من قائمة الرموز وإضافته إلى كلمة المرور
39
40 # إضافة الأرقام العشوائية إلى كلمة المرور بناءً على العدد المطلوب
41 for char in range(gen_numbers):
42     password += random.choice(numbers) # اختيار رقم عشوائي من قائمة الأرقام وإضافته إلى كلمة المرور
43
44 # طباعة كلمة المرور النهائية للمستخدم
45 print("Password is:", password)
46
```

أضفنا حلقة while لتكرار المدخلات عدد عناصر (أحرف + رموز + أرقام) يتراوح بين 8 و 12.

إذا كان العدد الكلي للعناصر خارج النطاق (أقل من 8 أو أكثر من 12) يظهر رسالة للمستخدم ويطلب إعادة الإدخال.

حساب العدد الكلي للعناصر

total_characters = gen_letters + gen_symbols + gen_numbers

استخدام دالة shuffle لتطوير البرنامج :

إضافة خطوة عشوائية لتحسين أمان كلمة المرور باستخدام دالة shuffle الموجودة في مكتبة random لإعادة ترتيب العناصر داخل القائمة بشكل عشوائي (وتستخدم فقط للنص)

أهمية shuffle

- في حالة ادخال (الحروف، الرموز، والأرقام) إلى كلمة المرور بالترتيب، تكون الكلمة الناتجة منظمة (مثل الأحرف أولاً، ثم الرموز، ثم الأرقام)، وهذا يقلل من قوة كلمة المرور.
- عند استخدام shuffle، يمكننا إعادة ترتيب العناصر بشكل عشوائي لضمان أن الكلمة النهائية تكون أكثر تعقيداً وأماناً.

لإضافة هذه الخطوة إلى البرنامج، نتبع الخطوات التالية :

1. نجمع الحروف، الرموز، والأرقام في قائمة واحدة.
2. نستخدم دالة shuffle لإعادة ترتيب عناصر القائمة عشوائياً.
3. نحول القائمة العشوائية إلى نص باستخدام join للحصول على كلمة المرور النهائية.

Code

```
1 # استيراد مكتبة random لتوليد عناصر عشوائية
2 import random
3 # قائمة تحتوي على الأحرف والأرقام والرموز التي سيتم استخدامها في كلمة المرور
4 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
5            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
6 numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
7 symbols = ['*', '&', '#', '@', '!', '-', '_', '+']
8 # طباعة رسالة للمستخدم
9 print("Welcome to the PyPassword Generator!")
10 # تكرار إدخال عدد الأحرف والرموز والأرقام حتى تكون القيمة بين 8 و 12
11 while True:
12     # استقبل عدد الأحرف والرموز والأرقام من المستخدم
13     gen_letters = int(input("How many letters would you like in your password?: "))
14     gen_symbols = int(input("How many symbols would you like?: "))
15     gen_numbers = int(input("How many numbers would you like?: "))
16
17     # حساب العدد الكلي للعناصر
18     total_characters = gen_letters + gen_symbols + gen_numbers
19     # إذا كان العدد الكلي بين 8 و 12 يتم الخروج من الحلقة
20     if 8 ≤ total_characters ≤ 12:
21         break
22     else:
23         print("The total number of characters must be between 8 and 12. Please try again.")
24 # قائمة لتجميع الحروف، الرموز، والأرقام
25 password_list = []
26 # إضافة حروف عشوائية إلى القائمة
27 for char in range(gen_letters):
28     password_list.append(random.choice(letters)) # اختيار حرف عشوائي من قائمة الأحرف
29 # إضافة رموز عشوائية إلى القائمة
30 for char in range(gen_symbols):
31     password_list.append(random.choice(symbols)) # اختيار رمز عشوائي من قائمة الرموز
32 # إضافة أرقام عشوائية إلى القائمة
33 for char in range(gen_numbers):
34     password_list.append(random.choice(numbers)) # اختيار رقم عشوائي من قائمة الأرقام
35 # خلط العناصر في القائمة باستخدام shuffle
36 random.shuffle(password_list) # إعادة ترتيب العناصر عشوائياً
37 # تحويل القائمة إلى نص للحصول على كلمة المرور النهائية
38 password = ''.join(password_list)
39 # طباعة كلمة المرور النهائية للمستخدم
40 print("Password is:", password)
```

مثال 2: احسب عمرك

- توظيف المكتبات لحساب العمر (بالأيام، الأشهر، والسنوات) من تاريخ الميلاد الذي يُدخله المستخدم.
- معرفة يوم الميلاد (مثل الإثنين، الثلاثاء) بناءً على التاريخ المدخل.
- استخدام مكتبة `dateutil` لتحسين عملية حساب الفرق بين التواريخ بدقة.

مكتبة `datetime` : للتعامل مع التواريخ وإنشاء تاريخ الميلاد.
مكتبة `dateutil.relativedelta` : لحساب الفرق بين تاريخين بدقة (سنوات، أشهر، أيام).

1. استيراد المكتبات

- استخدام `datetime` لإنشاء التواريخ والحصول على تاريخ اليوم.
- استخدام `relativedelta` لحساب الفرق بين التواريخ.

2. إدخال تاريخ الميلاد

- طلب إدخال اليوم، الشهر، والسنة من المستخدم.
- تحويل الإدخالات إلى أعداد صحيحة باستخدام `int`.

3. التحقق من صحة تاريخ الميلاد

- يتم استخدام كتلة `try/except` لمحاولة إنشاء تاريخ الميلاد.
- إذا كان الإدخال غير صحيح (مثل يوم 31 في شهر 2)، يتم عرض رسالة خطأ والخروج من البرنامج.

4. حساب الفرق بين تاريخ الميلاد واليوم الحالي

- الحصول على تاريخ اليوم باستخدام `datetime.date.today`.
- حساب الفرق باستخدام `relativedelta` للحصول على عدد السنوات، الأشهر، والأيام.

5. عرض النتائج

عرض العمر بالتفصيل: عدد السنوات، الأشهر، والأيام.

عرض يوم الميلاد:

- باستخدام `strftime('%A')` للحصول على اسم اليوم.

6. مجال التحسين النتائج

- عرض اسم الشهر باستخدام `%B` (مثل January أو February).
- عرض القرن باستخدام `%C` (مثل 19 للقرن العشرين).

إضافة :

- المكتبة تتعامل تلقائياً مع السنوات الكبيسة (مثل 29 فبراير).

Code

```
1 import datetime
2 from dateutil.relativedelta import relativedelta
3 # إدخال تاريخ الميلاد
4 user_day = int(input("Enter your birthday, Day: "))
5 user_month = int(input("Enter your birthday, Month: "))
6 user_year = int(input("Enter your birthday, Year (4 Digits): "))
7 # التحقق من صحة المدخلات
8 try:
9     birth_date = datetime.date(user_year, user_month, user_day)
10 except:
11     print("Error: Invalid date entered. Please check the day, month, and year.")
12     exit()
13 # الحصول على تاريخ اليوم
14 today_date = datetime.date.today()
15 # حساب الفرق بين تاريخ اليوم وتاريخ الميلاد
16 delta = relativedelta(today_date, birth_date)
17 # عرض العمر بالتفصيل
18 print(f"Your age is: {delta.years} years, {delta.months} months, and {delta.days} days.")
19 # عرض يوم الميلاد
20 print(f"You were born on a {birth_date.strftime('%A')}.")
21 # عرض اسم الشهر والسنة (القرن)
22 print(f"Month of birth: {birth_date.strftime('%B')}")
23 print(f"Century: {birth_date.strftime('%C')}")
```

مثال 3: برنامج لعبة "اختيار رقم عشوائي"

البرنامج يختار رقم عشوائي بين 1 و 20، ويطلب من المستخدم محاولة تخمين الرقم. بعد كل تخمين، يعرض البرنامج تلميحات للمستخدم حول ما إذا كان الرقم الذي أدخله أكبر أو أصغر من الرقم الصحيح. في حالة إذا نجح المستخدم في تخمين الرقم، يظهر البرنامج رسالة تهنئة وعدد المحاولات التي قام بها.

خوارزمية البرنامج

1. استيراد مكتبة **random** لتوليد الأرقام العشوائية بين 1 و 20 ، (1, 20) وتخزينه في متغير **target_number**.
2. متغير (**attempts = 0**) لتتبع عدد محاولات المستخدم.
3. طباعة رسالة توضح للمستخدم أن هناك رقمًا بين 1 و 20، وعليه تخمينه.
4. بدء حلقة التخمين تستمر حتى يُدخل المستخدم الرقم الصحيح.
5. استقبال تخمين من المستخدم باستخدام **input**، وتحويله إلى عدد صحيح.
6. زيادة المتغير **attempts** بمقدار 1 لتسجيل عدد المحاولات.

التحقق إذا كان التخمين

- أقل من الرقم العشوائي: (target_number) - طباعة رسالة "Too low! Try a higher number."
 - إذا كان التخمين أكبر من الرقم العشوائي: طباعة رسالة "Too high! Try a lower number."
 - إذا كان التخمين مساويًا للرقم العشوائي:
 - طباعة رسالة تهنئة تحتوي على الرقم الصحيح وعدد المحاولات.
 - إنهاء الحلقة باستخدام **break**.
- ينتهي البرنامج عند تخمين المستخدم للرقم الصحيح.

Code

```
1 import random # استيراد مكتبة random لتوليد رقم عشوائي
2
3 # توليد رقم عشوائي بين 1 و 20
4 target_number = random.randint(a: 1, b: 20)
5
6 # تهيئة عدد المحاولات
7 attempts = 0
8
9 # رسالة ترحيبية
10 print("Welcome to the Guessing Game!")
11 print("I have chosen a number between 1 and 20. Can you guess it?")
12
13 # حلقة التخمين
14 while True:
15     # استقبال تخمين المستخدم وزيادة عدد المحاولات
16     user_guess = int(input("Enter your guess: "))
17     attempts += 1
18
19     # التحقق من التخمين وتقديم تلميحات
20     if user_guess < target_number:
21         print("Too low! Try a higher number.")
22     elif user_guess > target_number:
23         print("Too high! Try a lower number.")
24     else:
25         # إذا كان التخمين صحيحًا
26         print(f"Congratulations! You guessed the number {target_number} correctly in {attempts} attempts.")
27         break # الخروج من الحلقة عند التخمين الصحيح
28
```

Output

```
C:\Users\HP\Documents\pythonProject\python.exe C:\Use
Welcome to the Guessing Game!
I have chosen a number between 1 and 20. Can you guess it?
Enter your guess: 5
Too low! Try a higher number.
Enter your guess: 7
Too low! Try a higher number.
Enter your guess: 17
Congratulations! You guessed the number 17 correctly in 3 attempts.

Process finished with exit code 0
```

البرنامج يستخدم مكتبة **random** لتوليد أسئلة عشوائية من قائمة أسئلة ويعرضها للمستخدم، ثم يستقبل الأجوبة منه ويحسب النتيجة النهائية بناءً على الإجابات الصحيحة. البرنامج يتعامل مع قائمة من الأسئلة والإجابات ويرتب الأجوبة لكل سؤال، ثم يعطي المستخدم النتيجة في النهاية.

مثال 4:
بنك الأسئلة

الخوارزمية

1. استيراد مكتبة **random** لتوليد أرقام عشوائية.
2. استيراد مكتبة **datetime** لعرض التاريخ الحالي.
3. تعريف قائمة **questions** التي تحتوي على الأسئلة.
4. تعريف قائمة **answers** التي تحتوي على الإجابات الصحيحة، يجب أن تتطابق كل عنصر في القائمة مع السؤال في نفس فهرس **questions**.
5. المتغير **score** ليبدأ من الصفر، وذلك حتى يتم تسجيل عدد الإجابات الصحيحة.
6. عرض التاريخ والوقت الحالي للمستخدم باستخدام **datetime.datetime.today()**.

حلقة الأسئلة العشوائية:

تكرار الخطوات التالية طالما أن عدد الأسئلة المتبقية في قائمة **questions** أكبر من 2:

تحديد فهرس عشوائي:

مستخدمة التعليمة التالية: **random.randrange(len(questions))** لتوليد فهرس عشوائي لاختيار سؤال عشوائي من قائمة **questions** وإجابته من قائمة **answers**.
تخزين الفهرس العشوائي في المتغير **random_index**.

عرض السؤال للمستخدم:

عرض جميع الخيارات في قائمة **answers** لمستخدم، مع ترقيم كل خيار ليختار الإجابة الصحيحة بسهولة.
استقبال إجابة المستخدم الذي يمثل إجابته، ثم تحويل المدخل إلى عدد صحيح **int()**.

التحقق من صحة الإجابة:

إذا كانت إجابة المستخدم صحيحة

في حالة المساواة `(user_answer - 1) == random_index`

زيادة **score** بمقدار 1 يتم عرض رسالة "Correct answer" للمستخدم.

إذا كانت الإجابة خاطئة، يتم عرض رسالة "Wrong answer" للمستخدم.

7. حذف السؤال والإجابة المقابلة له من القائمتين **questions** و **answers** باستخدام `pop()`، لضمان عدم تكرار السؤال في الدورة القادمة.

8. بعد انتهاء الأسئلة، عرض النتيجة النهائية للمستخدم بطباعة عدد الإجابات الصحيحة في المتغير **score**.

Code

```
1 # برنامج بنك الأسئلة
2 # هذا البرنامج يستخدم مكتبة random لاختيار أسئلة عشوائية من قائمة أسئلة، ثم يعرضها للمستخدم، يستقبل الإجابات، ويحسب النتيجة النهائية
3
4 # استيراد المكتبات اللازمة
5 import random # مكتبة random لتوليد أرقام عشوائية
6 import datetime # مكتبة datetime لعرض التاريخ الحالي
7 # قائمة الأسئلة، تحتوي على أسماء بعض الدوال في Python
8 questions = ["str()", "int()", "input()", "print()", "float()", "random()", "exit()"]
9 # قائمة الإجابات، كل عنصر يمثل معنى الدالة في نفس فهرس السؤال المقابل في قائمة
10 answers = ["Convert value to string", "Convert value to integer", "Enter value from user",
11 "Display values on screen", "Convert value to a decimal number",
12 "generate random numbers", "Exit program"]
13 #تهيئة درجة المستخدم لتكون صفر في البداية
14 score = 0
15 # عرض التاريخ الحالي للمستخدم في بداية البرنامج
16 print(f'Today Date: {datetime.datetime.today()}')
17
18 # حلقة لتكرار الأسئلة العشوائية حتى يتبقى سؤالان فقط في قائمة الأسئلة
19 while len(questions) > 2:
20     # اختيار سؤال عشوائي من القائمة باستخدام فهرس عشوائي
21     random_index = random.randrange(len(questions))
22     user_question = questions[random_index] # تخزين السؤال المختار
23     # عرض السؤال المختار للمستخدم
24     print(f"Question: What is the meaning of: {questions[random_index]} function")
25     # عرض الخيارات المتاحة للمستخدم للإجابة عليها
26     for i in range(len(answers)):
27         print(f"{i+1}- {answers[i]}") # طباعة كل خيار مع رقم تسلسلي
28     # استقبال إجابة المستخدم وتحويلها إلى عدد صحيح
29     user_answer = int(input("Choose the correct answer: "))
30     # التحقق مما إذا كانت الإجابة صحيحة
31     if (user_answer - 1) == random_index:
32         score += 1 # زيادة درجة المستخدم بمقدار 1 إذا كانت الإجابة صحيحة
33         print(f"Correct answer\n{'=' * 50}") # طباعة رسالة توضح أن الإجابة صحيحة
34     else:
35         print(f"Wrong answer\n{'=' * 50}") # طباعة رسالة توضح أن الإجابة خاطئة
36     # إزالة السؤال والإجابة المقابلة له من القوائم لضمان عدم تكرار السؤال
37     questions.pop(random_index)
38     answers.pop(random_index)
39     # عرض النتيجة النهائية للمستخدم بعد انتهاء الأسئلة
40     print(f"Your score is: {score}")
41
42
```

Output

```
1-2w x
C:\Users\HP\Documents\pythonProjectLyp\.venv\Scripts\python.exe C:
Today Date: 2024-11-12 16:44:31.202774
Question: What is the meaning of: str() function
1- Convert value to string
2- Convert value to integer
3- Enter value from user
4- Display values on screen
5- Convert value to a decimal number
6- generate random numbers
7- Exit program
Choose the correct answer: 1
Correct answer
=====
Question: What is the meaning of: random() function
1- Convert value to integer
2- Enter value from user
3- Display values on screen
4- Convert value to a decimal number
5- generate random numbers
6- Exit program
Choose the correct answer: 5
Correct answer
=====
Question: What is the meaning of: exit() function
1- Convert value to integer
2- Enter value from user
3- Display values on screen
4- Convert value to a decimal number
5- Exit program
Choose the correct answer: 4
Wrong answer
=====
Question: What is the meaning of: float() function
1- Convert value to integer
2- Enter value from user
3- Display values on screen
4- Convert value to a decimal number
Choose the correct answer: 4
Correct answer
=====
Question: What is the meaning of: int() function
1- Convert value to integer
2- Enter value from user
3- Display values on screen
Choose the correct answer: 1
Correct answer
=====
Your score is: 4

Process finished with exit code 0
|
```

مثال 5: لعبة أسئلة

لعبة أسئلة تفاعلية يتم فيها طرح مجموعة من الأسئلة العشوائية على المستخدم، ويتم حساب النقاط بناءً على الإجابات الصحيحة يمكن للمستخدم الحصول على تغذية راجعة بعد كل إجابة لمعرفة ما إذا كانت صحيحة أو خاطئة، وفي نهاية اللعبة يتم عرض النتيجة النهائية.

Code

```
1 import random # استيراد مكتبة random لاختيار الأسئلة عشوائيًا
2
3 # قائمة تحتوي على الأسئلة والأجوبة الصحيحة
4 questions = [
5     {"question": "What is the capital of France?", "options": ["Paris", "London", "Rome",
6     "Berlin"], "answer": 1},
7     {"question": "What is 5 + 7?", "options": ["10", "11", "12", "13"], "answer": 3},
8     {"question": "Which planet is known as the Red Planet?", "options": ["Earth", "Mars",
9     "Jupiter", "Saturn"],
10     "answer": 2},
11     {"question": "What is the largest ocean on Earth?", "options": ["Atlantic", "Indian",
12     "Arctic", "Pacific"],
13     "answer": 4},
14     {"question": "Who wrote 'To Be, or Not to Be'?", "options": ["Shakespeare", "Dickens",
15     "Hemingway", "Austen"],
16     "answer": 1}
17 ]
18 # تهيئة درجة المستخدم
19 score = 0
20
21 # رسالة ترحيبية
22 print("Welcome to the Quiz Game!")
23 print("Answer the following questions by choosing the correct option number.")
24
25 # اختيار 3 أسئلة عشوائية من القائمة
26 quiz_questions = random.sample(questions, k=3)
27
28 # طرح الأسئلة المختارة على المستخدم
29 for question_data in quiz_questions:
30     print("\n" + question_data["question"]) # عرض السؤال
31     for i, option in enumerate(question_data["options"], 1): # عرض الخيارات
32         print(f"{i}. {option}")
33
34     # استقبال إجابة المستخدم
35     user_answer = int(input("Enter the option number of your answer: "))
36
37     # التحقق من صحة الإجابة
38     if user_answer == question_data["answer"]:
39         print("Correct!")
40         score += 1 # زيادة النقاط إذا كانت الإجابة صحيحة
41     else:
42         print("Wrong! The correct answer was:", question_data["options"][question_data["answer"] - 1])
43
44 # عرض النتيجة النهائية
45 print(f"\nYour final score is: {score} out of {len(quiz_questions)}")
46
```

Output

```

Welcome to the Quiz Game!
Answer the following questions by choosing the correct option number.

What is the capital of France?
1. Paris
2. London
3. Rome
4. Berlin
Enter the option number of your answer: 1
Correct!

Which planet is known as the Red Planet?
1. Earth
2. Mars
3. Jupiter
4. Saturn
Enter the option number of your answer: 2
Correct!

What is the largest ocean on Earth?
1. Atlantic
2. Indian
3. Arctic
4. Pacific
Enter the option number of your answer: 1
Wrong! The correct answer was: Pacific

Your final score is: 2 out of 3
```


مثال يشرح كيفية استخدام مكتبة (datetime) مع بعض الأدوات الإضافية من مكتبات أخرى مثل (time.asctime) و (dateutil.relativedelta) و (random) لحساب العمر ومعرفة موعد عيد الميلاد القادم وعرض بعض العبارات الممتعة للمستخدم.

مثال 6:

Code

```
from datetime import datetime, date
from dateutil.relativedelta import relativedelta # pip install python-dateutil
import time
import random

# (1) --- الحصول على تاريخ الميلاد من المستخدم وتحويله إلى كائن تاريخ ---
dob_input = input("Enter your date of birth (YYYY-MM-DD): ").strip()
dob = datetime.strptime(dob_input, "%Y-%m-%d").date() # تحويل النص إلى كائن تاريخ

# (2) --- إظهار السنة والشهر واليوم باستخدام year, month, day ---
print(f"\nYou were born on: {dob} (year={dob.year}, month={dob.month}, day={dob.day})")

# (3) --- عرض الوقت الحالي باستخدام asctime() ---
now = datetime.now()
print("Right now (via time.asctime):", time.asctime(now.timetuple()))

# (4) --- (سنوات، أشهر، أيام) relativedelta حساب العمر بدقة باستخدام ---
today = date.today()
age_rd = relativedelta(today, dob)
print(f"\nExact age: {age_rd.years} years, {age_rd.months} months, {age_rd.days} days")

# (5) --- حساب تاريخ عيد الميلاد القادم والمدة المتبقية ---
this_year_bday = date(today.year, dob.month, dob.day)
next_bday = this_year_bday if this_year_bday >= today else date(today.year + 1, dob.month, dob.day)

until_next = relativedelta(next_bday, today)
days_until_next = (next_bday - today).days

print(f"Your next birthday: {next_bday} "
      f"({until_next.months} months, {until_next.days} days from today; {days_until_next} total days)")
```

(6) --- اختيار رسالة عشوائية باستخدام مكتبة random ---

```
messages = [  
    "Cake time incoming! 🍰🎂",  
    "Another lap around the sun! ☀️🌈",  
    "Leveling up soon—get ready! 🚀🎉",  
    "Make a wish and blow the candles! 🕯️🎁",  
    "Party planning mode: ON. 🎵🎶"  
]  
print("Random hype message:", random.choice(messages))
```

(7) --- اختيار وقت عشوائي لتذكيرك في يوم عيد ميلادك ---

```
rand_hour = random.randint(8, 21) # بين 8:00 و 21:59  
rand_min = random.choice([0, 15, 30, 45])  
suggested_dt = datetime(next_bday.year, next_bday.month, next_bday.day, rand_hour,  
rand_min)  
print(f"Suggested reminder time on your next birthday: {suggested_dt} (local time)")
```

(8) --- الحصول على الوقت الحالي بالتنسيق المطلوب ---

```
current_time = time.strftime("%H:%M:%S")  
print("الوقت الحالي هو:", current_time) # طباعة الوقت على الشاشة
```

(9) --- (الحصول على الوقت الحالي بالتنسيق المطلوب) ساعة ودقيقة فقط ---

```
current_time = time.strftime("%H:%M")  
print("الوقت الحالي بالساعة و الدقيقة هو:", current_time)
```

(10) --- لطباعة التاريخ الحالي بالتنسيق المطلوب (اليوم والشهر والسنة) ---

```
print(time.strftime("%Y-%m-%d"))
```

(11) --- لطباعة التاريخ الحالي بالتنسيق المطلوب (الشهر والسنة) ---

```
print(time.strftime("%Y-%m"))
```

مثال يشرح جميع أنواع الدوال الأساسية في بايثون، مع تغطية مفاهيم مهمة مثل المعاملات الافتراضية، المعاملات غير المحدودة، واستخدام المتغيرات العامة (global variables)، و lambda.

Code

```
# --- متغير عام (global variable) يمكن الوصول له من أي دالة ---
global_counter = 0

# --- دالة بدون معاملات ولا قيمة إرجاع (1) ---
def greet():
    # دالة تقوم بالطباعة فقط
    print("Hello from a no-parameter, no-return function!")

# --- بدون قيمة إرجاع (parameter) دالة بمعامل (2) ---
def say_hello(name):
    # تطبع الاسم الممرر إليها
    print(f"Hello, {name}!")

# --- دالة بمعامل وقيمة إرجاع (3) ---
def add_numbers(a, b):
    # ترجع حاصل جمع الرقمين
    return a + b

# --- دالة بدون معاملات لكن تعيد قيمة (4) ---
def get_today():
    # تعيد النص "Today is a great day!"
    return "Today is a great day!"

# --- لقبول عدد غير محدد من القيم *args دالة تستخدم (5) ---
def show_scores(*scores):
    # تطبع جميع القيم المرسلة مهما كان عددها
    print("Scores are:", scores)

# --- (if اختصار) one-line if مثال على (6) ---
def check_even(n):
    # "Odd" وإلا اطبع "Even" إذا كان الرقم زوجي اطبع
    print("Even") if n % 2 == 0 else print("Odd")

# --- دالة بمعامل افتراضي (default parameter) (7) ---
def greet_with_default(name="Stranger"):
    # إذا لم يرسل المستخدم اسم، يستخدم القيمة الافتراضية
    print(f"Hello, {name}!")

# --- دالة Lambda (8) ---
square = lambda x: x * x # تعيد مربع الرقم x
```

```

# --- 9) مثال على استخدام المتغير العام داخل دالة ---
def increment_global():
    # نخبر بايثون أننا سنعدل المتغير العام
    global global_counter
    global_counter += 1
    print(f"Global counter is now: {global_counter}")

```

----- تنفيذ الدوال لاختبارها -----

```

greet() # دالة بدون معاملات ولا إرجاع (1)
say_hello("Alice") # دالة بمعامل (2)
print("3 + 5 =", add_numbers(3, 5)) # دالة ترجع قيمة (3)
print(get_today()) # دالة تعيد نص (4)
show_scores(10, 20, 30, 40) # دالة تستخدم *args (5)
check_even(7) # one-line if مثال على (6)
greet_with_default() # بدون تمرير اسم ← يستخدم القيمة الافتراضية (7)
greet_with_default("Bob") # تمرير اسم ← يستخدم الاسم المدخل ← (8)
print("Square of 6 =", square(6)) # استخدام lambda (8)
increment_global() # تعديل المتغير العام (9)
increment_global() # استدعاء مرة أخرى لزيادة القيمة أكثر (9)

```

أمثلة على بعض المكتبات:

مكتبة NumPy :

اختصار لـ (Numerical Python) هي مكتبة متخصصة في العمليات الحسابية والرياضية المتقدمة في لغة بايثون. تُعتبر الأساس لمعظم مكتبات تحليل البيانات والذكاء الاصطناعي مثل **Pandas** و **Matplotlib** و **TensorFlow**. وتُستخدم للتعامل مع المصفوفات **Arrays** بسرعة ودقة في بايثون.

أهم مميزات مكتبة Numpy:

1. إنشاء ومعالجة المصفوفات (Arrays).
2. إجراء عمليات رياضية سريعة على مجموعات كبيرة من البيانات.
3. دعم المعاملات الرياضية المتقدمة (مثل الجذر، اللوغاريتم، الجيب..).
4. إمكانية الدمج مع مكتبات أخرى مثل **Pandas** و **Matplotlib**.
5. استهلاك أقل للذاكرة وسرعة عالية جدًا مقارنة بالقوائم العادية.

أمثلة عملية على استخدام NumPy :

• إنشاء مصفوفة (Array)

Code

```
1 import numpy as np
2
3 arr = np.array([1, 2, 3, 4, 5])
4 print(arr)
```

Output

```
[1 2 3 4 5]
```

• إنشاء مصفوفة ثنائية الأبعاد (2D Array)

Code

```
1 import numpy as np
2
3 arr2 = np.array([[1, 2, 3], [4, 5, 6]])
4 print(arr2)
```

Output

```
[[1 2 3]
 [4 5 6]]
```

- إجراء عمليات رياضية

Code

```
1 import numpy as np
2
3 arr = np.array([10, 20, 30, 40])
4
5 print("المجموع:", np.sum(arr))
6 print("المتوسط:", np.mean(arr))
7 print("أكبر رقم:", np.max(arr))
8 print("أصغر رقم:", np.min(arr))
```

Output

المجموع: 100
المتوسط: 25.0
أكبر رقم: 40
أصغر رقم: 10

- العمليات بين مصفوفتين

Code

```
1 import numpy as np
2
3 a = np.array([1, 2, 3])
4 b = np.array([4, 5, 6])
5
6 print("الجمع:", a + b)
7 print("الضرب:", a * b)
```

Output

الجمع: [9 7 5]
الضرب: [18 10 4]

- دالة prod() لحساب ناتج ضرب العناصر.

Code

```
1 import numpy as np
2
3 arr = np.array([2, 3, 4])
4 print("ناتج الضرب:", np.prod(arr))
```

Output

ناتج الضرب: 24

تُستخدم لإنشاء الرسوم البيانية والمخططات التوضيحية لعرض البيانات بطريقة مرئية وجذابة. تساعدنا في تحليل البيانات وفهمها بسهولة من خلال التمثيل البصري.

استيراد المكتبة في البرنامج:

عادةً نستورد الجزء الرئيسي منها وهو وحدة **pyplot** التي تُستخدم للرسم:

```
import matplotlib.pyplot as plt
```

أهم استخدامات matplotlib:

1. رسم الرسوم البيانية الخطية (Line Plot)
2. رسم الأعمدة (Bar Chart)
3. رسم المخططات الدائرية (Pie Chart)
4. رسم المخططات النقطية (Scatter Plot)
5. التحكم في الألوان والعناوين والمحاور لجعل الشكل أكثر احترافية

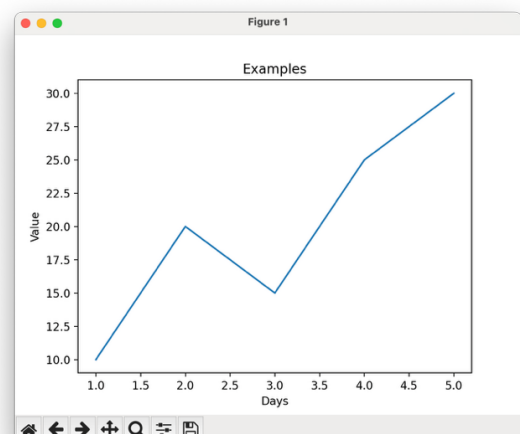
أمثلة على استخدام المكتبة:

رسم بياني خطي بسيط (الناتج: يظهر خط يمثل العلاقة بين الأيام والقيمة).

Code

```
1 import matplotlib.pyplot as plt
2
3 # القيم
4 x = [1, 2, 3, 4, 5]
5 y = [10, 20, 15, 25, 30]
6
7 # رسم خط
8 plt.plot(x, y)
9
10 # إضافة عنوان ومحاور
11 plt.title("Examples")
12 plt.xlabel("Days")
13 plt.ylabel("Value")
14
15 # عرض الرسم
16 plt.show()
```

Output

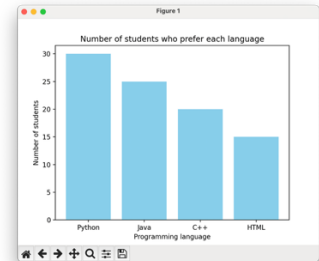


رسم مخطط أعمدة (Bar Chart) (الناتج: أعمدة ملونة تمثل عدد الطلاب لكل لغة برمجة).

Code

```
1 import matplotlib.pyplot as plt
2
3 Programs = ["Python", "Java", "C++", "HTML"]
4 Students = [30, 25, 20, 15]
5
6 plt.bar(Programs, Students, color="skyblue")
7 plt.title("Number of students who prefer each language")
8 plt.xlabel("Programming language")
9 plt.ylabel("Number of students")
10 plt.show()
```

Output

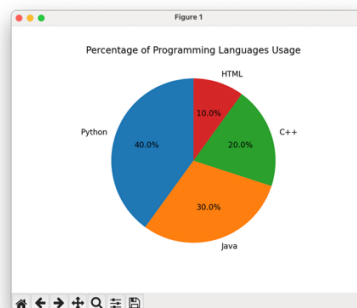


• رسم مخطط دائري (Pie Chart) (الناتج: مخطط دائري يُظهر النسب المئوية لاستخدام اللغات).

Code

```
1 import matplotlib.pyplot as plt
2
3 languages = ["Python", "Java", "C++", "HTML"]
4 percentages = [40, 30, 20, 10]
5
6 plt.pie(percentages, labels=languages, autopct="%1.1f%%", startangle=90)
7 plt.title("Percentage of Programming Languages Usage")
8 plt.show()
```

Output



من أشهر مكتبات تحليل البيانات في لغة بايثون. تُستخدم لتنظيم البيانات، وتحليلها بسهولة وفعالية، خاصة عند التعامل مع جداول.

كلمة **pandas** مأخوذة من:

”*Panel Data*” أي البيانات اللوحية أو الجدولية، وهي نوع من البيانات المستخدمة كثيرًا في الإحصاء وتحليل البيانات.

عادةً نستوردها بالاختصار الشائع:

```
import pandas as pd
```

الاختصار **pd** يُستخدم لتسهيل كتابة الأوامر داخل الكود.

أهم استخداماتها:

1. إنشاء جداول بيانات (**DataFrames**) وقراءتها من ملفات CSV أو Excel.
2. عرض البيانات وفرزها وتصفية الصفوف والأعمدة.
3. تحليل البيانات وإجراء عمليات رياضية وإحصائية.
4. معالجة القيم المفقودة وتنظيف البيانات.
5. دمج وربط أكثر من جدول بيانات بسهولة.

أمثلة على استخدامها:

- إنشاء جدول بيانات (**DataFrame**)

Code

```
1 import pandas as pd
2 # إنشاء جدول بسيط
3 data = {
4     "name": ["علي", "عبد الله", "بدر", "محمد"],
5     "age": [15, 16, 14, 15],
6     "degree": [90, 95, 88, 66]
7 }
8
9 df = pd.DataFrame(data)
10 print(df)
```

Output

| | name | age | degree |
|---|----------|-----|--------|
| 0 | علي | 15 | 90 |
| 1 | عبد الله | 16 | 95 |
| 2 | بدر | 14 | 88 |
| 3 | محمد | 15 | 66 |

Code

```

1 import pandas as pd
2 # إنشاء جدول بسيط
3 data = {
4     "name": ["محمد", "بدر", "عبد الله", "علي"],
5     "age": [15, 16, 14, 15],
6     "degree": [90, 95, 88, 66]
7 }
8 df = pd.DataFrame(data)
9 print(df)
10 import pandas as pd
11 # إنشاء جدول بسيط
12 data = {
13     "name": ["محمد", "بدر", "عبد الله", "علي"],
14     "age": [15, 16, 14, 15],
15     "degree": [90, 95, 88, 66]
16 }
17 df = pd.DataFrame(data)
18 # أقل درجة بين الطلاب
19 print("أقل درجة:", df["degree"].min())
20 # أكبر عمر بين الطلاب
21 print("عمر أكبر طالب:", df["age"].max())
22 # متوسط درجات الطلاب
23 print("متوسط درجات الطلاب هو:", df["degree"].mean())
24 print()
25 # أسماء الطلبة الذين تبلغ أعمارهم أكثر من 14 سنة
26 print("أسماء الطلبة الذين تبلغ أعمارهم أكثر من 14 سنة:")
27 for name in df[df["age"] > 14]["name"]:
28     print(name)
29 print()
30 # إيجاد أعلى درجة
31 max_degree = df["degree"].max()
32 # استخراج اسم الطالب صاحب أعلى درجة
33 student_name = df[df["degree"] == max_degree]["name"].values
34 # طباعة الجملة "أعلى درجة هي ... وهي للطالب ..."
35 print(f"أعلى درجة هي {max_degree} وهي للطالب {student_name}.")

```

Output

```

أقل درجة: 66
عمر أكبر طالب: 16
متوسط درجات الطلاب هو: 84.75

أسماء الطلبة الذين تبلغ أعمارهم أكثر من 14 سنة:
علي
عبد الله
محمد

أعلى درجة هي 95 وهي للطالب عبد الله.

```