

Programming in Python is characterized by its ease and simplicity. It is considered one of the most powerful multi-purpose programming languages, used in various fields such as artificial intelligence, data analysis, and web application development.

1. Conditional Statements

if : When you want to check a **single condition** and execute specific code if the condition is true.

else : When you want to execute specific code if the condition in the **if** statement is false (when there are **two conditions**).

elif: When you have more than one condition to check, and if the condition in if statement is false, you will check other conditions using elif (when there are **more than two conditions or additional conditions**).

A- if Statement

في بعض الأحيان، يمكن استخدام if لوحدها بدون الحاجة إلى else أو elif إذا كان الشرط الوحيد الذي تريد التحقق منه.

Example 1: Write a program in Python that prints the message "The number is even" using the if statement alone.

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

```
if number % 2 == 0:
```

```
# "If the remainder of dividing the number
```

```
by 2 is equal to zero."
```

```
    print("The number is even")
```

The expression `% 2 == 0` is used to check if a number is even. Here's a breakdown:

% : the **modulus** operator (عامل باقي القسمة), which returns the remainder of the division of one number by another.

2 : the divisor, so the expression `number % 2` calculates the remainder when number is divided by 2.

== 0 : checks if this remainder is equal to zero.

This checks if the number is even by verifying if the remainder when dividing by 2 is zero. Since `number = 5` in this case, nothing will be printed because the condition is false.

Example 2: Write a program in Python that prints the message " You are an adult " using the if statement alone.

```
age = 20
```

```
if age >= 18:
```

```
    print("You are an adult")# Result: You are an adult
```

X = 0 : تعني اسناد قيمة الى متغير (تخزين قيمة 0 في X).

X == 0 : تعني مقارنة اي هل قيمة x تساوي 0؟ (تستخدم في الشروط).

B- if and else

Example 1: Write a program in Python that compares **two** conditions and prints one of them.

```
age = 18
if age >= 18:
    print("you are an adult")
else:
    print("you are minor") # Result: you are an adult
```

Example 2:

```
# Define two variables
a = 10
b = 20
# Compare the two conditions
if a > b:
    print("a is greater than b")
else:
    print("a is less than or equal to b")
```

C- if , else and elif

```
grade = 85
if grade >= 90:
    print("Excellent grade")
elif grade >= 75:
    print("Very good grade")
else:
    print("Average grade")
```

Result: Very good grade

الشرط الأول $grade \geq 90$ غير صحيح، لذا تم الانتقال إلى elif. الشرط $grade \geq 75$ صحيح، وبالتالي تم تنفيذ الجملة داخل elif.

كلمة elif هي اختصار لـ "else if"، وتستخدم في البرمجة وخاصة في لغة بايثون، لتحديد شرط إضافي بعد جملة if. كلمة elif يمكن ترجمتها إلى العربية كـ "إلا إذا" أو "وإذا لم يكن الشرط السابق صحيحًا، أو فإذا تحقق هذا الشرط".

Using multiple conditions with **elif**: Meaning using it multiple times in the same code

Example 1: Write a program in Python that compares three temperature values and prints the message "The weather is very hot" when the highest temperature among them is met.

```
temperature = 30
if temperature > 40:
    print("The weather is very hot")
elif temperature > 25:
    print("The weather is hot")
elif temperature >= 15:
    print("The weather is moderate")
else:
    print("The weather is cold") # Result: The weather is hot
```

في Python (وباقى لغات البرمجة) عندما تستخدم سلسلة شروط if / elif / else فإنها تنفذ أول شرط يتحقق فقط، ثم تتجاهل باقي الشروط و ان كانت متحققة.

Example 2:

```
score = 50
if score >= 90:
    print("Excellent")
elif score >= 80:
    print("Very Good")
elif score >= 70:
    print("Good")
elif score >= 60:
    print("Passable")
else:
    print("Fail") # Result: Fail
```

Comparison between multiple conditions using logical comparison operators.

You can use multiple conditions in an **if statement** using logical operators such as **and** and **or**.

❖ Example (using: and):

Write a program in Python that uses the logical operator "**and**" to compare two conditions, and if both are true, it prints the message "**Eligible for a loan.**"

```
age = 25 # first condition
income = 5000 # second condition

if age < 18 and income < 3000:
    print("Ineligible for a loan")
elif age > 18 and income > 3000:
    print("Eligible for a loan")
else:
    print("Ineligible for a loan")
```

عند استخدام and يجب تحقق الشرطين معًا

النقطتان (:) بعد الشرط تعني "كل الأوامر الموجودة في السطر أو الأسطر التالية، والمزاحة بمستوى (indentation)، تُنفَّذ إذا تحقق الشرط."

بدون النقطتين، بايثون لا يعرف أين يبدأ جسم الشرط، وسيعطي خطأ.
مثال: الجزء `age < 18 and income < 3000` هو الشرط بالتالي فان النقطتان تقول لبايثون "ابدأ تنفيذ الكود التالي (المُزاح بمسافة الذي هو print) إذا تحقق الشرط."

Result: Eligible for a loan (مؤهل للحصول على قرض)

❖ Example (using: or):

Write a program in Python that uses the logical operator (**or**) to compare two conditions, and if either condition is true, it prints the message "**Eligible for a loan.**"

```
age = 25
income = 5000

if age > 18 or income > 3000:
    print("Eligible for a loan")
else:
    print("Not eligible for a loan")
```

في هذه الحالة، سيكون الشرط صحيحًا إذا كان العمر أكبر من 18 أو الدخل أكبر من 3000. أي المطلوب تحقق احد الشرطين.

Result: Eligible for a loan

❖ Using if with Boolean values.

Boolean values **True** and **False** can be used directly in if statements.

Example:

Write a program in Python that uses **Boolean statements** to print "Take an umbrella" if it is raining, and print "No need for an umbrella" if the condition is not met.

```
is_raining = True# try using False
```

```
if is_raining:
```

```
    print("Take an umbrella")
```

```
else:
```

```
    print("No need for an umbrella")#
```

The program checks the value of the variable `is_raining`, if the value is `True` (meaning it is raining), the program asks you to take an umbrella. However, if the value is `False` (meaning it is not raining), it tells you that there is no need for an umbrella.

Result: Take an umbrella

2. Loops

There are two main types of loops in Python:

- **for** loop: it is used to iterate over data sets.
- **while** loop: continues to iterate as long as the condition is true.

for loop: تستخدم للتكرار عبر مجموعة بيانات .
while loop : تستمر في التكرار طالما أن الشرط صحيح.

Example 1 : (for Loop)

Write a program in Python that displays the numbers from 0 to 4 using a **for** loop.

```
for i in range(5):
```

```
    print(i)
```

Result:

```
0
1
2
3
4
```

Example 2: Write a program in Python that displays the numbers from 1 to 5 using a **for** loop.

في هذا المثال تم البدء من 1 و في المثال الاول من 0 :# `for i in range(1, 6):`

```
    print(i)
```

Result:

```
1
2
3
4
5
```

القيمة `i` تأخذ القيم من الدالة `range` واحدة تلو الأخرى.

`range(1,5)` تُنشئ تسلسلاً من الأعداد:

1, 2, 3, 4

ثم الحلقة **for** تجعل المتغير `i` يأخذ كل قيمة منها بالتسلسل.

Example 1: (while Loop)

Write a program in Python to display the numbers less than 5 using the **while** function.

```
i = 0
```

```
while i < 5:
```

- يبدأ المتغير `i` بقيمة 0.

- حلقة **while** تستمر في التكرار طالما أن قيمة `i` أقل من 5.

- في كل تكرار، يتم طباعة قيمة `i`، ثم تزداد قيمته بمقدار 1 باستخدام `i += 1`.

- عندما تصبح قيمة `i` تساوي 5، يتوقف تنفيذ الحلقة.

في حلقة **for** و كما في المثال (1) لا نحتاج إلى `i += 1` لأن الدالة `range(5)` تقوم تلقائيًا بزيادة `i` في كل دورة (من 0 إلى 4).

أما في حلقة **while** مثل كما في المثال (1) فأنت من يجب أن تزيد قيمة `i` يدويًا داخل الحلقة، وإلا ستصبح الحلقة لا نهائية (infinite loop)، لأن الشرط `i < 5` سيبقى دائمًا صحيحًا لو لم تزد `i`.

Dr.Alaa Najem

```
print(i)
i += 1
```

Result:

0
1
2
3
4

Example 2: Write a program in Python to display the numbers from 0 to 5 using the **while** loop.

```
i = 0
while i <= 5:
```

```
    print(i)
    i += 1
```

Result:

0
1
2
3
4
5

- وضع $i += 1$ بعد `print(i)` يبدأ الطباعة من 0. حيث: $i = 0$ في البداية 0 يطبع ثم يزيدها إلى 1.

- وضع $i += 1$ قبل `print(i)` يبدأ الطباعة من 1. و هنا عكس الحالة الاولى.

$i = i + 1 \rightarrow i += 1$ تعني

Example 3: calculate the squares of numbers from 1 to 4 using the **while** loop:

```
i = 1
```

```
while i <= 4:
```

```
    print(f"The square of {i} is: {i**2}")
```

```
    i += 1
```

Result:

The square of 1 is: 1
The square of 2 is: 4
The square of 3 is: 9
The square of 4 is: 16

تقوم **f-string** تلقائيًا بتحويل المتغيرات والتعبيرات الموجودة داخل الأقواس {} إلى نصوص ودمجها في السلسلة النصية. وذلك لوجود ارقام و حروف. يمكنك حذف الحرف **f** إذا كنت تطبع أرقامًا فقط.

3. Functions

Functions in Python are programming units that allow code reuse to save time and effort. They can accept inputs (parameters) and return outputs. In addition to simple functions, you can also create complex mathematical functions using Python.

There are two types of functions in Python:

1- Built-in Functions: These are the functions that come with Python by default, such as `print()` and `len()`. They can be used directly without needing to be defined.

2- User-defined Functions: These functions are defined by the user according to their needs using the keyword **def**

A function is defined using the **def** keyword followed by the **function name** and **parentheses**. If the function takes parameters, they are included within the parentheses.

The structure of the function is denoted as follows:

```
def my_function(parameter1, parameter2):
    return parameter1 + parameter2 # Function body

math = my_function(1,2)

print(math)
```

شرح هيكل الدالة:

- **def**: هذه الكلمة المفتاحية تُستخدم لتعريف دالة جديدة.
- **my_function**: هو اسم الدالة، ويمكن أن يكون أي اسم مناسب.
- **(parameter1, parameter2)**: هذه هي المعلومات التي ستقبلها الدالة. يمكنك تحديد عدد غير محدود من المعلومات، أو تركها فارغة إذا لم تكن هناك حاجة لأي معلومات.
- **(:)**: هذا هو الفاصل الذي يدل على بداية جسم الدالة.
- **return**: هو الكود الذي سيتم تنفيذه عند استدعاء الدالة، ويكون مُزاحًا إلى الداخل و هو **جسم الدالة**.
- إذا لم يتم توفير جملة **return**، فإن الدالة ستعيد **None**.
- باستخدام **return**، يمكنك استخدام الدالة عدة مرات في أجزاء مختلفة من الكود دون الحاجة لكتابة نفس المنطق مرة أخرى. **مرادف كلمة ارجاع هو ارسال او تسليم القيمة بعد حسابها.**

Dr.Alaa Najem

P

Example 1:

```
def greet(name):
    return f"Hello, {name}"

message = greet("Ali")#
print(message)
```

Result:

Hello, Ali

```
def add(a, b):
    return a + b

result = add(3, 5)# هنا يتم استقبال القيمة الراجعة
print(result)
```

return تعمل على:

□ تنفذ التعبير أو العملية الموجودة بعدها مثل (a + b)

□ تُخرج الناتج النهائي من داخل الدالة.

□ و تُعيده إلى المكان الذي استدعيت فيه الدالة الذي هو **result** ولهذا اصبحت **result=8**.
return هي الجسر الذي ينقل النتيجة من داخل الدالة إلى خارجها حيث تقع **return** داخل الدالة لأنها مزاحة الى الداخل.

Example 2:

```
def multiply(a, b):
    return a * b
```

البرنامج يبدأ عن طريق استدعاء الدالة و تعويض القيم الحقيقية multiply(4,3) و ثم يتم بعد ذلك التنفيذ في return الذي يأخذ القيم و ينفذ العملية الرياضية و اخراج النتيجة الذي و ارسالها الى مكان الخزن الذي هو الجهة التي تستدعي الدالة و هو output ليتم طباعتها.

```
a = float(input("Enter the first number: "))# Input the first number and convert it to float type."
```

```
b = float(input("Enter the second number: ")) # Input the second number and convert it to float type."
```

```
output = multiply(a, b) # استدعاء الدالة مع القيم المدخلة
print(output)
```

Enter the first number: 5.7

Enter the second number: 3.3

Result: 15

```
def multiply(a, b):
```

```
    return a * b #
```

```
a = int(input("Enter the first number: "))
```

```
b = int(input("Enter the second number: "))
```

```
output = multiply(a, b)
```

```
print(output)
```

❖ A function to calculate the sum of two numbers

Example:

```
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 10)
print(result)
```

Result:15

Create a function to calculate the difference between two numbers (**subtraction**).

```
def subtract_numbers(a, b):
```

```
    return a - b
```

```
result = subtract_numbers(15, 5)
```

```
print(result)#
```

Result: 10

❖ A function to calculate division

Example 3:

```
def divide_numbers(a, b):
```

```
    if b != 0: # Check to avoid division by zero, where != means not equal
```

```
        return a / b
```

```
    else:
```

```
        return "Division by zero is not allowed!"
```

```
result = divide_numbers(10, 2)
```

```
print(result)
```

Result: 5.0

هنا: إذا كان b يساوي قيمة ليس صفراً يتم تنفيذ البرنامج و إذا كان صفراً يتم الدخول في الجملة `else` والتي تقوم بإرجاع رسالة نصية تقول "القسمة على الصفر غير مسموحة!" لمنع القسمة على الصفر، والتي تعد عملية غير صحيحة في الرياضيات.

❖ A function to calculate the exponent (power):

Example 1:

```
def power(base, exponent): #def power(x,a)
```

```
    return base ** exponent # return x**a
```

```
result = power(2, 3)
```

```
print(result) # print(power(base, exponent))
```

Result: 8

Example 2: A function to calculate the sine (in radians, i.e., the radian angle):

```
import math
```

```
def sine_angle(angle_radians):
```

```
    return math.sin(angle_radians)
```

```
result = sine_angle(math.pi / 2)
```

```
print(result) # Result: 1.0
```

باستخدام `math.pi`، تحصل على قيمة دقيقة لـ π بدلاً من استخدام تقريب يدوي، مما يقلل من الأخطاء في العمليات الرياضية. بدلاً من تعريف قيمة π بنفسك، يمكنك ببساطة استيراد مكتبة `math`.

يمكن حذف `math` من `math.pi` لكن بشرط أن تكون قد استوردت `pi` من مكتبة `math` قبل استخدامه:

```
from math import pi
```

```
result = sine_angle(pi / 2)
```

```
print(result)
```

للتحويل من الدرجات الى الراديان نستخدم العلاق الآتية :

radians=math.radians(angle in degrees)

للتحويل من الراديان الى الدرجات نستخدم العلاق الآتية :

degree= math.degrees(angle in radians)

دالة `math.sin()` تعمل فقط بوحدة الراديان، لذلك نستخدم `math.radians()` لتحويل الزاوية من درجات إلى راديان قبل الحساب. لاحظ الأمثلة 3 و 4 و 5.

Example 3: convert from degrees to radians:

```
import math
```

```
def cos_angle(angle_degrees):
```

```
    angle_radians = math.radians(angle_degrees) #
```

Example 4: A function to calculate the sine (degrees):

```
import math
```

```
def sine_angle(angle_degrees):
```

```
    radians = math.radians(angle_degrees)
```

```
    return math.sin(radians)
```

```
result = sine_angle(90)
```

```
print(result) # Result: 1.0
```

```

return math.cos(angle_radians)
result = sine_angle(90)
print(result) # Result: 0.5

```

Example 5: convert from radians to degrees:

```

import math
r = math.asin(0.5) # تعطي الزاوية بالراديان
deg = math.degrees(r) # تحويلها إلى درجات
print(deg)

```

❖ Function to calculate the circumference of a circle:

Example:

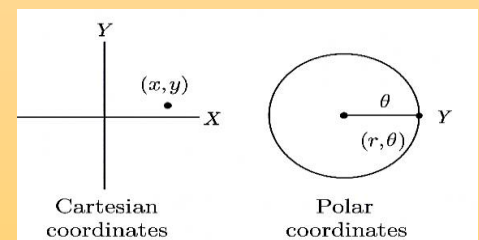
```

import math
def circle_circumference(radius):
    return 2 * math.pi * radius # circle_circumference = 2πr
result = circle_circumference(5)
print(result) # Result: 31.41592653589793

```

في الكارتيزي Cartesian system : النقطة تحدد بـ (x, y) (إحداثيات خطية على المحورين).

في القطبي polar system : النقطة تحدد بـ (r, θ) حيث r المسافة من الأصل و θ الزاوية مع المحور الأفقي.



❖ Function to calculate the distance between two points in the Cartesian plane:

Example:

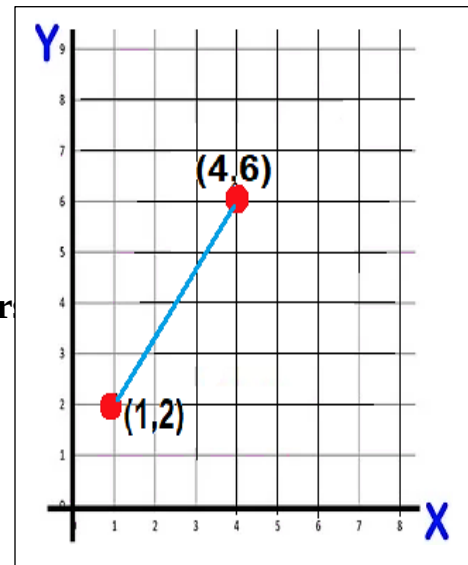
```

import math
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
result = distance(1, 2, 4, 6) # (x1, y1, x2, y2)
print(result)
Result: 5.0

```

يمكنك استخدام المعادلة التالية لحساب المسافة بين نقطتين (x_1, y_1) و (x_2, y_2) :

$$\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} = d$$



Function to calculate the arithmetic mean of a set of number

Example:

```

def average(numbers):
    return sum(numbers) / len(numbers)
result = average([1, 2, 3, 4, 5])
print(result)
Result: 3.0

```

❖ Advanced mathematical functions using the math library:

The math library in Python contains many useful mathematical functions. Here are some examples of built-in mathematical functions in this library:

Calculating the natural logarithm:

Example 1:

```
import math
def natural_log(x):
    return math.log(x)
result = natural_log(10)
print(result)
Result: 2.302585092994046
```

Calculating the value of the exponential function:

Example 2:

```
import math
def exponential(x):
    return math.exp(x)
result = exponential(2)
print(result)
```

Example 3:

```
from math import exp
print(exp(2))
```

EX 1, EX 2: تم تعريف الدالة exponential(x) ثم استدعائها، مناسب لإعادة الاستخدام اي بمجرد كتابة اسم الدالة فقط.

EX 3: استدعى exp مباشرة وطبع النتيجة، أقصر وأبسط لكنه لا يعيد استخدام الدالة.

4. Dictionaries

Dictionaries in Python are a type of data structure that store data in the form of **key-value** pairs. Dictionaries are used to store data when we need to associate each item with a unique value, and values are accessed through their keys rather than **indices**, as is the case with lists.

Example:

```
my_dict = {"name": "Ahmed", "age": 25, "city": "Cairo"}
print(my_dict["name"])
```

Result: Ahmed

الوصول إلى العناصر: يتم الوصول إلى القيم باستخدام المفاتيح.

```
print(my_dict["name"])
```

- أزواج مفتاح-قيمة: يتم تخزين كل عنصر في القاموس على شكل مفتاح مرتبط بقيمة.

- هنا، name, age, و city هي **المفاتيح**، والقيم المرتبطة بها هي "Ahmed", 25، و "Baghdad".

- **المفاتيح فريدة:** أي لا يمكن أن يتكرر نفس المفتاح أكثر من مرة. إذا تم تعريف نفس المفتاح مرتين، سيتم استخدام القيمة الأخيرة أي الثانية المكررة.

Basic operations on dictionaries

Dictionaries are changeable, and key-value pairs can be added, modified, or deleted after the dictionary is created.

```
my_dict = {"name": "Ahmed", "age": 25, "city": "Baghdad"}
# Adding a new element
my_dict["job"] = "Engineer"
```

يمكن الحصول على جميع القيم باستخدام print عدة مرات او استخدام دالة التكرار .iteration

Modifying a value

```
my_dict["age"] = 30
```

Printing the dictionary

```
print(my_dict) # {'name': 'Ahmed', 'age': 30, 'city': 'Baghdad', 'job': 'Engineer'}
```

Deleting an element

```
del my_dict["city"]
```

Accessing a value

```
print(my_dict.get("name")) # Result: "Ahmed"
```

Iterating over the dictionary

```
for key, value in my_dict.items():
```

```
    print(key, value) # طباعة كل مفتاح مع قيمته
```

```
my_dict = {"name": "Ahmed", "age": 25, "city": "Cairo"}
print(my_dict["name"]) # This will print: Ahmed
print(my_dict["age"]) # This will print: 25
print(my_dict["city"]) # This will print: Cairo
```

- تستخدم حلقة for للتكرار على جميع العناصر في القاموس my_dict (dictionary).

- الدالة items() ترجع أزواج المفاتيح والقيم (key-value pairs) في القاموس.

و ان هذه الدالة هي التي ترتب المفاتيح و القيم على شكل ازواج.

- في كل تكرار، يتم تعيين المفتاح إلى المتغير key، والقيمة المقابلة له إلى المتغير value.

5. Exceptions

In Python, exceptions are errors that occur during the execution of a program. When an error occurs, Python stops the execution and generates an error message, also known as an exception. Python provides a way to handle these exceptions (errors) so that the program can continue to run, even when an error occurs.

Common Exceptions in Python:

1. **ZeroDivisionError**: Raised when division by zero is attempted.
2. **TypeError**: Raised when an operation or function is applied to an object of inappropriate type. **Example:** `Result = "hello" + 5` # هنا نوع البيانات مختلف
3. **ValueError**: Raised when an operation or function receives an argument that has the right type but an inappropriate value. ادخال قيمة غير مناسبة للدالة.
Example: Converting a string that doesn't represent a number into an integer. `num = int("abc")`
4. **KeyError**: Raised when a dictionary key is not found.
5. **FileExistsError**: This error occurs when attempting to create a file that already exists. **See example 2 in files section.**

في بايثون لا يمكن تحويل السلسلة الحرفية string الى عدد صحيح لكن يمكن العكس. كذلك يمكن تحويل العدد العشري الى عدد صحيح بحيث يزيل الاشار كمثال يمكن تحويل العدد العشري 10.5 الى صحيح ليصبح 10. `int(float)=10`

Exception Handling Using **try** and **except**:**Example 1:****try:**number = **int**(**input**("Enter a number: "))**print**("You entered:", number)**except ValueError:** # إذا كان الإدخال ليس رقمًا صحيحًا**print**("Error: The input is not a valid integer.")number = **int**(**input**("Please enter a valid integer: "))**print**("You entered:", number)**argument:** هو المدخل الذي تدخله للدالة أو للعملية لتعمل عليه.**لماذا نقول "استثناء except"؟**

لأن البرنامج يتوقع أن الأمور تجري بشكل طبيعي، لكن إذا حدث شيء غير عادي أو خارج عن المتوقع (مثل محاولة الوصول إلى مفتاح غير موجود)، فهذا الحدث يتم التعامل معه كـ "استثناء" للقاعدة أو للمسار العادي للبرنامج، أي حالة استثنائية (غير طبيعية).

في بايثون يتوقف التنفيذ فقط إذا لم يتم التعامل مع الخطأ. لكن عند استخدام **try / except**، أنت تُخبر بايثون أنه إذا حدث خطأ، فلا تتوقف؛ بل انتقل إلى جزء **except** ونفذ ما بداخله، ثم أكمل تشغيل البرنامج.

Example 2: Write a program in Python that defines a dictionary containing information about a person (such as name, age, and city). Then, attempt to print the value associated with a key that does not exist in the dictionary. Use exception handling to ensure that the program does not crash, and when a **KeyError** occurs, the program should display a message indicating that the key does not exist in the dictionary.

```
data = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}
```

- بعد وقوع الخطأ، البرنامج يطلب من المستخدم إدخال مفتاح آخر.
- إذا كان المفتاح صحيحًا، يعرض قيمته.
- إذا كان خاطئًا مجددًا، يستخدم (**get**) لتجنب التوقف وإظهار رسالة مناسبة أي ربما تخطئ المفتاح بشكل خاطئ أكثر من مرة.
- بشكل عام، يستخدم **try** لمعالجة الأخطاء المحتملة في البرنامج دون إيقافه، مما يسمح لك بتقديم استجابة ملائمة (مثل عرض رسالة خطأ) عند حدوث مشكلة.

try:key = **input**("Enter the key you want to access: ")**print**("Value:", data[key])**except KeyError:****print**("Error: This key does not exist.")key = **input**("Please enter a valid key: ")**print**("Value:", data.get(key, "The key still does not exist"))

6. Files

"Python provides simple interfaces for reading and writing files, and there are two ways to create files: using mode 'x' and mode 'w'."

Example 1:

`with open("example.txt", "w") as file:`

`file.write("Hello, Python!")`

`with open("example.txt", "r") as file:`

`content = file.read()`

`print(content)`

Result: Hello, Python!

Example 2:

`# Using "w"`

`with open("example.txt", "w") as file:`

`file.write("This will overwrite any existing content.")`

`# Using "x"`

`try:`

`with open("example.txt", "x") as file:`

`file.write("This will only create the file if it doesn't exist.")`

`except FileExistsError:`

`print("The file 'example.txt' already exists.")`

- إنشاء ملف: يتم إنشاء الملفات بطريقتين :

`with open("example.txt", "w")`: ينشئ الملف سواء كان موجود مسبقا ام لا اذا كان الملف موجودا مسبقا فانه يمسح المحتوى القديم و يضيف الجديد.
`with open("example.txt", "x")`: ينشئ ملف فقط اذا كان غير موجود مسبقا حتى يمنع الكتابة فوق البيانات السابقة للملف الموجود مسبقا بالتالي يحمي من حذف البيانات القديمة.

- كتابة إلى ملف:

• `open("example.txt", "w")`: يفتح الملف "example.txt" في وضع الكتابة (w) ، ويقوم بكتابة النص "Hello, Python!" في الملف.

- قراءة من ملف:

• `open("example.txt", "r")`: يفتح الملف "example.txt" في وضع القراءة (r) ، ويقرأ محتواه ويعرضه باستخدام `print()`.

في الكود المقابل ستظهر رسالة الخطأ :

("The file 'example.txt' already exists.")

ذا كان الملف موجوداً مسبقاً، لأن وضع "x" يُستخدم لإنشاء ملف جديد فقط إذا كان غير موجود. لا يمكن الكتابة على الملف إذا كان موجوداً مسبقاً، وفي هذه الحالة ستظهر رسالة الخطأ بدلاً من كتابة المحتوى الجديد في الملف. إذا تم إنشاء ملف جديد غير موجود مسبقاً باستخدام وضع "x"، يمكنك الكتابة عليه.

w هو اختصار لـ "write" (الكتابة).

7. Object-Oriented Programming - OOP (البرمجة كائنية التوجيه) (البرمجة الشيئية)

Object-Oriented Programming (OOP) in Python is a method that relies on encapsulating data and functions into objects. Each object represents a model of something in the real world, having attributes (data) and behaviors (functions) like eat, move and stop, etc.

Example 1:

`class WeatherDay:`

```
def __init__(self, temperature, humidity, pressure, solar_radiation,
            rain, snow, wind_speed, dust, clouds):
```

```
    self.temperature = temperature    # درجة الحرارة
    self.humidity = humidity          # % الرطوبة
    self.pressure = pressure          # hPa الضغط
    self.solar_radiation = solar_radiation # الإشعاع الشمسي W/m²
    self.rain = rain                  # كمية الأمطار mm
    self.snow = snow                  # كمية الثلج cm
    self.wind_speed = wind_speed      # سرعة الرياح km/h
    self.dust = dust                  # % مستوى الغبار
    self.clouds = clouds               # % نسبة الغيوم
```

```
def show_info(self):
```

```
    print(f"Temperature: {self.temperature} °C")
    print(f"Humidity: {self.humidity} %")
    print(f"Pressure: {self.pressure} hPa")
    print(f"Solar Radiation: {self.solar_radiation} W/m²")
    print(f"Rain: {self.rain} mm")
    print(f"Snow: {self.snow} cm")
```

تستخدم البرمجة الكائنية لتعريف الخصائص والسلوكيات مرة واحدة في الصنف (Class) ثم إنشاء أي عدد من الكائنات (Objects) دون إعادة كتابة هذه الخصائص والسلوكيات لكل نوع لهذا الصنف.

الآلية عمل كل من `__init__` و `self`:

بداية الكائن object هو النوع من الصنف كمثال لو كان الصنف الذي نعمل عليه هو السيارات فان النوع هو احد انواع السيارات و لنفترض Toyota و اذا كان الصنف حيوانات فالنوع لنفترض انه اسد و في هذا المثال كان الصنف الطقس اليومي و النوع هو احد الايام و هو الكائن و هكذا.

تخيل أن المُفسّر يقول لـ `__init__`:

"ها هو الكيس الفارغ (الكائن)، وهذه الأشياء (القيم اي قيم الخصائص) رتبها داخل الكيس. فـ `__init__` تفتح الكيس باستخدام `self` و تملئه بالقيم .

اي ان: دالة `__init__` تأخذ القيم المدخلة عند استدعاء الصنف وتعيّن كل واحدة إلى الخاصية التي تحددها أنت (مثل `self.name = name`).

عندما نكتب في بداية الكود

```
self.temperature
self.humidity
self.pressure
```

فهذا يعني ان `self` هي النائب عن الكائن (تنوب عنه و تأخذ مكانه لغاية ان يتم تحديد او ادخال الكائن الحقيقي) و ان خصائصه هي `humidity, temperature, pressure`. فاذا افترضنا ان الكائن الحقيقي الفعلي هو `day` عليه يكون شكل الكود كالآتي:

```
day.temperature
day.humidity
day.pressure
```

يعني السطر الآتي:

```
self.temperature = temperature
```

انك تخزن قيمة درجة الحرارة (التي تقع يسار العلامة =) في الخاصية `temperature` (التي تقع على يمين العلامة =).

```

print(f"Wind Speed: {self.wind_speed} km/h")
print(f"Dust Level: {self.dust} %")
print(f"Cloud Cover: {self.clouds} %")
print(f"Weather Condition: {self.evaluate_condition()}")
print("-----")

```

دالة تقييم الحالة الجوية بناءً على البيانات

```
def evaluate_condition(self):
```

```
    # snow
```

```
    if self.snow > 2:
```

```
        return "Snowy"
```

```
    # Rain
```

```
    if self.rain > 5:
```

```
        return "Rainy "
```

```
    # Heavy dust
```

```
    if self.dust > 60:
```

```
        return "Dusty "
```

```
    # Strong winds
```

```
    if self.wind_speed > 40:
```

```
        return "Windy "
```

```
    # Overcast
```

```
    if self.clouds > 80:
```

```
        return "Overcast "
```

بدون استخدام show_info() تحتاج كل مرة تطبع فيها بيانات الطقس أن تكتب print لكل متغير كما في اسفل:

```

print(day1.temperature)
print(day1.humidity)
print(day1.pressure)
print(day1.solar_radiation)
print(day1.rainfall)
print(day1.snow_level)

```

show_info() تجعل طباعة البيانات أسهل وأسرع وأكثر ترتيباً، بدل تكرار أوامر الطباعة كل مرة.

Weather Report:

- Temperature: 30°C
- Humidity: 40%
- Pressure: 1012 hPa
- Solar Radiation: 500 W/m²
- Rainfall: 0 mm
- Snow Level: 0 cm

باختصار هي تقوم بثلاث مهام رئيسية:

1- طباعة خصائص اليوم الجوي مثل الحرارة، الرطوبة، الرياح... إلخ.

2- استدعاء دالة evaluate_condition() لإظهار الحالة الجوية النهائية تلقائياً.

3- تجميع كل هذه القيم في إخراج واحد يساعدك على قراءة حالة اليوم بسهولة أثناء الاختبار أو التحليل.

بعبارة أخرى:

هي دالة عرض (Display Function) تجعل قراءة بيانات الكائن أسهل وأسرع.

Partly cloudy

if 41 < self.clouds <= 80:

return "Partly cloudy "

High temperature

if self.temperature > 35:

return "Hot"

Cold weather

if self.temperature < 10:

return "Cold"

Moderate

return "Mild"

الصنف بشكل عام (weatherDay) هنا هو مجرد شكل يدل على الكائن الذي نريد تصنيفه و لا يؤثر على عمل الكود.

تبدأ عملية انشاء الكائن الجديد من السطر الاتي الذي يعتبر هو الكائن الاول و كل ما قبله مجرد عرض و ادخال لخصائص هذا الكائن:

day1 = WeatherDay(38, 20, 1010, 850, 0, 0, 10, 5, 10)

من المهم معرفة انه بمجرد كتابة هذا السطر فان المفسر Interpreter سيقوم بالاتي:

1. يُنشئ كائنًا جديدًا من نوع الصنف WeatherDay.
2. يجمع القيم التي كتبها داخل الأقواس 2, 23, 4 : و يمررها تلقائيًا كمعطيات إلى دالة الباني (__init__) ، بالإضافة إلى self (وهو المرجع الذي يمثل الكائن الجديد).
3. أنت لا تستدعي الدالة __init__ بنفسك — بل المفسر يُفعله نيابةً عنك كلما استدعيت الصنف لإنشاء كائن.
4. أي أن المفسر يُنفذ شيئًا شبيهًا بـ:

__init__(day, 2, 23, 4)

- الدالة show_info تسترجع هذه البيانات من خلال self.
- evaluate_condition أيضًا تعتمد على self لمعرفة البيانات الحالية وتحديد الحالة الجوية.

إذا لم يُحقّق أي شرط، فالتنفيذ سيصل تلقائيًا إلى السطر الأخير ويعيد "Mild".

إنشاء 4 أيام مختلفة بقيم مختلفة اي يتم هنا انشاء 4 كائنات #

day1 = WeatherDay(38, 20, 1010, 850, 0, 0, 10, 5, 10)

day2 = WeatherDay(12, 60, 1005, 300, 6, 0, 15, 10, 70)

day3 = WeatherDay(6, 80, 1012, 150, 0, 5, 5, 20, 90)

day4 = WeatherDay(28, 35, 1013, 700, 0, 0, 55, 30, 25)

عرض الحالات

day1.show_info()

day2.show_info()

day3.show_info()

day4.show_info()

الية العمل:

عند استدعاء الصنف لانشاء الكائن كما في الكود الاتي:

day1 = WeatherDay(38, 20, 1010, 850, 0, 0, 10, 5, 10)

يُمرّر المفسر الكائن + القيم إلى __init__ ، ويترك __init__ مهمة التهيئة .

هو مثل "المساعد الذكي" الذي:

- يحضر الكيس الفارغ (الكائن) اي يقوم بانشاء الكائن،
- ويُسلّم المكونات (38, 20, ...) للمُعَدِّ (__init__) ،
- ثم يعطيك الكيس الجاهز (day1) اي الكائن.

شرط: في self.temperature = temperature فان الاسم على يمين العلامة (=) يجب أن يكون نفس اسم الخاصية في __init__ لكي تتعرف عليه الدالة init و تعوض القيم فيه . اما الاسم على يسار علامة (=) الذي هو self.temperature فيتم اختياره كما نريد.

باختصار :

كل شيء يبدأ سطر من الاستدعاء، والمفسر ينسّق التمرير، و __init__ تُنفذ التخزين عبر self.

Example 2:

Define the Student class

class Student:

def __init__(self, name, stage, age, grade, subject):

self.name = name

self.stage = stage

self.age = age

self.grade = grade

self.subject = subject

Display student info and status

def show_info(self):

print(f"Name: {self.name}")

print(f"Stage: {self.stage}")

print(f"Age: {self.age}")

print(f"Subject: {self.subject}")

print(f"Grade: {self.grade}")

print(f"Status: {self.evaluate_status()}")

print("-----")

Evaluate student status based on grade

def evaluate_status(self):

if self.grade >= 50:

return "Pass"

else:

return "Fail"

Create 3 students (objects)

```
student1 = Student("Ahmed", "First Stage", 18, 72, "Math")
```

```
student2 = Student("Layla", "Second Stage", 19, 48, "Physics")
```

```
student3 = Student("Marwan", "Third Stage", 20, 89, "Programming")
```

Display students' info

```
student1.show_info()
```

```
student2.show_info()
```

```
student3.show_info()
```

Output:

student1

Name: Ahmed
Stage: First Stage
Age: 18
Subject: Math
Grade: 72
Status: Pass

student2

Name: Layla
Stage: Second Stage
Age: 19
Subject: Physics
Grade: 48
Status: Fail

student3

Name: Marwan
Stage: Third Stage
Age: 20
Subject: Programming
Grade: 89
Status: Pass