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 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, 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
```

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 >= 90 غير صحيح، لذا تم الانتقال إلى elif. الشرط grade >= 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".

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 > 35:
    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
```

الشرط الأول temperature > 35 غير صحيح، ولكن الشرط الثاني temperature > 25 صحيح، لذا تم تنفيذ الجملة المقابلة لهذا الشرط.

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("Eligible for a loan")
else:
    print("Not eligible for a loan")
```

في هذا المثال: الشرطان age > 18 و income > 3000كلاهما صحيح، لذا تم تنفيذ الجملة داخل if .

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. اي 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 informs 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: used to iterate over data sets.
- while loop: continues to iterate as long as the condition is true.

for loop: تستخدم للتكرار عبر مجموعات 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.

```
for i in range(1, 5):# في هذا المثال تم البدء من 1 و في المثال الاول من 0
    print(i)
```

Result:
1
2
3
4

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.

Python Progr

```
    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

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 for 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 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 are functions written 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):
    # Function body
    return parameter1 + parameter2
math = my_function(1,2)
print(math)
```

شرح الهيكل:

- **def:** هذه الكلمة المفتاحية تُستخدم لتعريف دالة جديدة.
- **my_function**: هو اسم الدالة، ويمكن أن يكون أي اسم مناسب.
- **(parameter1, parameter2)**: هذه هي المعلمات التي ستقبلها الدالة. يمكنك تحديد عدد غير محدود من المعلمات، أو تركها فارغة إذا لم تكن هناك حاجة لأي معلمات.
- **(:)**: هذا هو الفاصل الذي يدل على بداية جسم الدالة.
- return : هو الكود الذي سيتم تنفيذه عند استدعاء الدالة، ويكون مُزاحًا إلى الداخل.

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.**

جملة return تعمل على إعادة القيمة الناتجة من عملية الجمع (أو أي عملية أخرى داخل الدالة) إلى المكان الذي استدعيت فيه الدالة، بحيث يمكن تخزين هذه القيمة أو استخدامها في مكان آخر من الكود.

المكان الذي استدعيت فيه الدالة هو أي سطر أو جزء من الكود الذي يطلب تنفيذ الدالة و هنا هو (message = greet("Ali")).

Example:

```
def greet(name):
    return f"Hello, {name}"
message = greet("Ali")# هنا يتم استدعاء الدالة
print(message)
```

Result:

Hello, Ali

البرنامج يبدأ عن طريق استدعاء الدالة و تعويض القيم الحقيقية multiply(4,3) و ثم يتم بعد ذلك التنفيذ في return الذي ياخذ 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.

Example 2:

```
def multiply(a, b):
    return a * b  # إرجاع الناتج
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)
```

Examples of functions and how mathematical functions work."

A function to calculate the sum of two numbers

Example 1:

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

Example 2: A function to calculate the difference between two numbers.

```
def subtract_numbers(a, b):
    return a - b
result = subtract_numbers(15, 5)
print(result)#
```

Result: 10

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

Result:15

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 صفرًا يتم تنفيذ البرنامج و اذا كان صفرا 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

Example 4: A function to calculate the square root using the math library.

```
import math
def sqrt_number(n):
    return math.sqrt(n)
result = sqrt_number(16)
print(result)#
```

Result: 4

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 الاتية :

radians=math.**radians**(**angle in degrees)**

للتحويل من الرديان الى الدرجات نستخدم العلاق الاتية :

degree= math. **degrees**(**angle in radians**)

Python Programming

Example 3: A function to calculate the sine (degrees):

```
import math
def sine_angle(angle_radians):
    return math.sin(angle_radians)
result = sine_angle(90)
print(result)#      Result: 1.0
```

Example 5: convert from degrees to radians:

```
import math
def sine_angle(angle_degrees):
    angle_radians = math.radians(angle_degrees)# convert from degrees to radians
    return math.sin(angle_radians)
result = sine_angle(90) # Calling the function with the angle in degrees
print(result)#  Result: 1
```

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
```

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

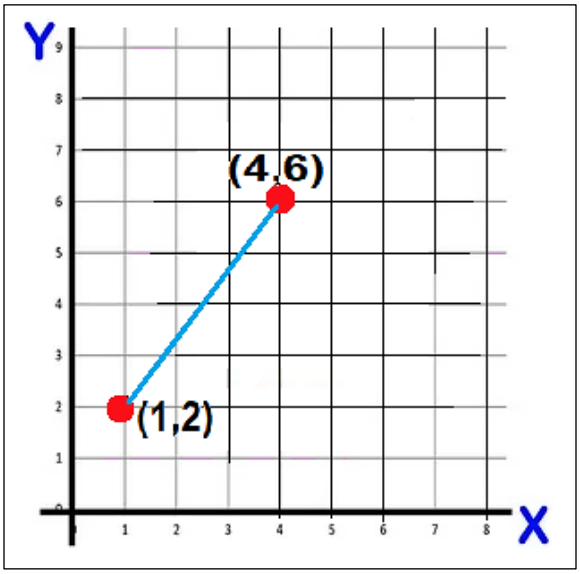
يمكنك استخدام المعادلة التالية لحساب المسافة بين نقطتين (x1,y1) و (x2,y2):

$$\sqrt{^2(y2-y1)+^2(x2-x1)}=d$$

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



Function to calculate the arithmetic mean of a set of numbers

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:

```
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:

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

Result: 7.38905609893065

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**:اي لا يمكن أن يتكرر نفس المفتاح أكثر من مرة. إذا تم تعريف نفس المفتاح مرتين، سيتم استخدام القيمة الأخيرة اي 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"
# 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"))  # print "Ahmed"
# Iterating over the dictionary
for key, value in my_dict.items():
    print(key, value) # طباعة كل مفتاح مع قيمته
```

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

```
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
```

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 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 # TypeError

3. **ValueError**: Raised when an operation or function receives an argument that has the right type but an inappropriate value.

   Example: Example: Converting a string that doesn't represent a number into an integer. num = int("abc") # ValueError

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 الاعشار كمثال يمكن تحويل العدد العشري 10.5 الى صحيح ليصبح10= (int(float.

Exception Handling Using try and except:

Example 1:

```
try:
    number = int(input("Enter a number:  "))
    print(number})
except ValueError: # If the input is not an integer, it is incorrect, and a message must be displayed below
    print("Please enter a valid number")
```

Result: "Please enter a valid number."

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.

فائدة استخدام الـ try block هي كما يلي:

- **محاولة تنفيذ الكود**: يتم استخدام try لمحاولة تنفيذ عملية قد تسبب خطأ، مثل تحويل الإدخال إلى عدد صحيح باستخدام int() .
- **إذا كانت العملية ناجحة**: إذا تم إدخال قيمة صحيحة (عدد صحيح)، سيقوم البرنامج بطباعة هذه القيمة.

بشكل عام، يستخدم try block لمعالجة الأخطاء المحتملة في البرنامج دون إيقافه، مما يسمح لك بتقديم استجابة ملائمة (مثل عرض رسالة خطأ) عند حدوث مشكلة.

```
data = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}
try:
    # Attempting to access a key that doesn't exist
    print(data['country'])
except KeyError:  # If the key is not found in the dictionary (تم وضع استثناء)
    print("Error: The key does not exist in the dictionary.")
```

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!

- انشاء ملف: يتم انشاء 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 بالتلي 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() .

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.")
```

في الكود المقابل ستظهر رسالة الخطا :

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

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

7. Object-Oriented Programming - OOP

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).

Example 1:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        return f"Hello, my name is {self.name} and I am {self.age} years old."
person = Person("Sara", 30)
print(person.greet())
```

Result: Hello, my name is Sara and I am 30 years old.

هذه القيم "Civic", "Honda", و 2021 هي بيانات تُمرر إلى الكائن person عند إنشائه، وتُخزن في خصائصه (self.brand, self.model, self.year). هذه البيانات تمثل خصائص فريدة لهذا الكائن بالذات، ويمكن الوصول إليها لاحقًا عند استدعاء الدوال الخاصة به مثل display_info().

عند تمرير هذه القيم إلى المنشئ (__init__) يتم تخزينها كخصائص خاصة بالكائن باستخدام self:

ان self هي تمثل الكائن نفسه الذي هو **car1** و يقوم بنقل البيانات الى خصائص الكائن اي يقول ان year =2020.

self : هنا يمثل الكائن الجديد الذي هو **car1**.

self.brand = brand: تعني أن خاصية **brand** للكائن **car1** ستأخذ القيمة "Toyota".

self.model = model: تعني أن خاصية **model** للكائن **car1** ستأخذ القيمة "Corolla".

self.year = year : تعني أن خاصية **year** للكائن **car1** ستأخذ القيمة 2020.

بشكل مختصر:

selfهو بمثابة **المترجم** أو **المرجع** الذي ينوب عن الكائن داخل الفئة، ويمثل الكائن نفسه عند العمل معه.

Example 2:

```
class Car:
    # (constructor)
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year
        # دالة لعرض المعلومات
    def display_info(self):
        print(f"Car: {self.brand} {self.model}, Year: {self.year}")
```

Python Programming

```
# إنشاء كائنات من الفئة Car
car1 = Car("Toyota" , "Corolla", 2020)
car2 = Car("Honda", "Civic", 2021)
# استدعاء الدالة لعرض المعلومات
car1.display_info()
car2.display_info()
```

Result:

```
Car: Toyota Corolla, Year: 2020
Car: Honda Civic, Year: 2021
```

شرح تفصيلي للكود:

class Car:

- **class:** كلمة مفتاحية في بايثون تستخدم لتعريف **فئة** (class) جديدة. الفئة هي نموذج أو قالب يمكن من خلاله إنشاء كائنات متعددة تشترك في نفس الخصائص والسلوكيات.
- **Car:** اسم الفئة. الفئة هنا تمثل "سيارة". يمكن اختيار أي اسم للفئة، ويفضل أن يكون الاسم دالاً على الكائنات التي سيتم إنشاؤها من هذه الفئة.

def __init__(self, brand, model, year):

- **def:** كلمة مفتاحية تُستخدم لتعريف دالة (function) في بايثون.
- **init:** دالة خاصة في بايثون تُسمى **constructor**(المنشئ). يتم استدعاؤها تلقائيًا عند إنشاء كائن جديد من الفئة. وظيفتها هي تهيئة الكائن وإعطاءه القيم الأولية للخصائص.
- **self**: هو مرجع يشير إلى الكائن الحالي الذي يتم إنشاؤه. يجب تمرير self كأول معامل في كل دالة داخل الفئة، لتمكين الوصول إلى خصائص ودوال الكائن.
- **brand, model, year:** هذه هي المعاملات التي يتم تمريرها إلى الدالة عند إنشاء الكائن. تمثل هذه المعاملات الخصائص الأولية للكائن: نوع السيارة(brand) ، الموديل(model) ، وسنة التصنيع.(year)

self.brand = brand

- **self.brand**:هو متغير خاص بالكائن الذي يتم إنشاؤه. باستخدام self، يمكن تخزين القيمة الممررة (brand) كخاصية للكائن، بحيث تكون مرتبطة بهذا الكائن.
- **brand:** هنا يتم تعيين القيمة الممررة كمعامل للدالة (brand) إلى خاصية الكائن self.brand. نفس العملية تتم للـ model و year.

def display_info(self):

- **def**:كلمة مفتاحية لتعريف دالة جديدة.
- **display_info:** اسم الدالة. هذه الدالة تُستخدم لعرض معلومات السيارة المخزنة في خصائص الكائن.
- **(self)** : هنا يتم تمرير self كمعامل لأن هذه الدالة ستستخدم خصائص الكائن الحالي مثل (brand و model و year).

print(f"Car: {self.brand} {self.model}, Year: {self.year}")

- **print()** دالة لطباعة النصوص على الشاشة.
- **f"Car: {self.brand} {self.model}, Year: {self.year}":** صيغة السلسلة النصية (f-string) تُستخدم لإدراج قيم المتغيرات داخل النص. هنا سيتم استبدال {self.brand}, {self.model}, و {self.year} بالقيم الفعلية الخاصة بالكائن الذي تم استدعاء الدالة عليه.

car1 = Car("Toyota", "Corolla", 2020)

- **car1:** اسم المتغير الذي سيحتوي على الكائن الجديد.
- **Car("Toyota", "Corolla", 2020):** هنا يتم إنشاء كائن جديد من الفئة **Car** باستخدام المعاملات "Toyota" كاسم العلامة التجارية، "Corolla" كاسم الموديل، و2020 كسنة التصنيع. عند إنشاء هذا الكائن، يتم استدعاء دالة المنشئ __init__ لتهيئة الخصائص الثلاثة.

car2 = Car("Honda", "Civic", 2021)

- نفس العملية السابقة تحدث هنا لإنشاء كائن آخر باسم car2 مع قيم مختلفة للخصائص "Honda" :للعلامة التجارية، "Civic" للموديل، و2021 كسنة التصنيع.

car1.display_info()

- يتم استدعاء الدالة display_info() على الكائن car1. الدالة ستقوم بطباعة معلومات السيارة (العلامة التجارية، الموديل، وسنة التصنيع) الخاصة بهذا الكائن على الشاشة.

car2.display_info()

- يتم استدعاء نفس الدالة على الكائن car2، وستطبع معلومات هذا الكائن (Honda Civic، سنة 2021).