1-

In Python programming, a class is a blueprint for creating objects, while an object is an instance of a class.

A class defines a set of attributes and methods that describe the behavior and properties of objects that belong to that class. It serves as a template for creating objects with similar characteristics.

On the other hand, an object is a specific instance of a class that has its own attributes and methods. When you create an object, you are creating a unique copy of the class that can have its own values for the attributes defined in the class.

Here's an example to illustrate the difference:

# Define a class

class Dog:

def init (self, name, breed):

self.name = name

self.breed = breed

def bark(self):

print("Woof!")

#Create an object of the Dog class

my\_dog = Dog("Buddy", "Golden Retriever")

# Access the attributes of the object

print(my\_dog.name)    # Output: Buddy

print(my\_dog.breed)   # Output: Golden Retriever

#Call the methods of the object

my\_dog .bark()         # Output: Woof!

In this example, Dog is a class that defines the attributes name and breed, and the method bark(). my \_ dog is an object of the Dog class that has its own values for name and breed. When we call the bark() method on my \_ dog , it prints "Woof!"

2-

In Python programming, instance variables are also commonly referred to as instance attributes or object attributes

These terms are interchangeable and refer to the variables that are defined within a class and are unique to each instance of that class. Instance variables hold data that is specific to each object and can be accessed and modified using dot notation

For example, in the following class definition, name and age are instance variables:

class Person:

def init(self, name, age):

self.name = name

self.age = age

When we create an object of this class, we can set the values of name and age for that object:

person1 = Person("Alice", 30)

person2 = Person("Bob", 25)

person1.name and person1.age are instance variables that hold the values "Alice" and 30, respectively. Similarly, person2.name and person2.age are instance variables that hold the values "Bob" and 25, respectively

3-

In Python programming, a method is a function that is associated with an object or a class. It is a set of instructions that can be called on an object to perform a specific task.

Another name for the term method in Python programming is member function. This term is commonly used in object-oriented programming (OOP) to refer to a function that is a member of a class .

Methods are defined within a class and can access the data stored in the instance variables of that class. They can also modify the state of the object or perform some action on it

Here's an example to illustrate the use of a method in Python:

class Circle:

def init(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

Create an object of the Circle class#

my\_circle = Circle(5)

Call the area() method on the object#

print(my\_circle.area())   # Output: 78.5

In this example, Circle is a class that defines the init() method to initialize the radius instance variable, and the area() method to calculate the area of the circle. We create an object of this class and call the area() method on it to calculate the area of the circle with a radius of 5 units.

4-

In Python programming, the dot (.) symbol is used to associate an object with a method invocation

When you call a method on an object, you use the dot notation to specify the object that the method belongs to. The syntax for calling a method on an object is:

Object \_name . method\_ name (arguments)

Here, object\_name is the name of the object that the method belongs to, method\_name is the name of the method that you want to call, and arguments are any values that need to be passed to the method

For example, let's say we have a class called Person that has a method called greet():

class Person:

def init (self, name):

self.name = name

def greet(self):

print("Hello, my name is", self.name)

#Create an object of the Person class

person1 = Person("Alice")

#Call the greet() method on the object

person1.greet()   # Output: Hello, my name is Alice

In this example, we create an object of the Person class and call the greet() method on it using the dot notation. The greet() method prints a message that includes the name of the person

5-

In Python programming, a method is a function that is associated with an object or a class, while a function is a standalone block of code that performs a specific task

Here are some key differences between methods and functions in Python:

1.

Associated with an object or class: A method is associated with an object or a class, while a function is not. Methods are defined within a class and can access the data stored in the instance variables of that class. They can also modify the state of the object or perform some action on it. On the other hand, functions are not associated with any object or class and can be called from anywhere in the program.

2.

Arguments: When you call a method, the first argument that is passed is always the object that the method is called on. This argument is conventionally named self. When you call a function, you can pass any number of arguments to it.

3.

Namespace: Methods have access to the namespace of the object they are associated with, while functions have their own namespace. This means that methods can access the instance variables of the object they are associated with, while functions cannot.

Here's an example to illustrate the difference between a method and a function in Python:

class Rectangle:

def init(self, length, width):

self.length = length

self.width = width

def area(self):

return self.length \* self.width

#Create an object of the Rectangle class

my\_rectangle = Rectangle(5,3)

#Call the area() method on the object

print(my\_rectangle.area())   # Output: 15

#Define a function to calculate the area of a rectangle

def calculate\_area(length, width):

return length \* width

Call the function to calculate the area of a rectangle#

print(calculate\_area(5, 3))   # Output: 15

In this example, Rectangle is a class that defines the init ( ) method to initialize the length and width instance variables, and the area() method to calculate the area of the rectangle. We create an object of this class and call the area() method on it using the dot notation .We also define a function called calculate\_area() that takes length and width as arguments and returns the area of the rectangle. We call this function to calculate the area of a rectangle with a length of 5 units and a width of 3 units

6-

In Python programming, the strip() method from the string class returns a new string with no leading or trailing whitespaces

The strip() method removes any leading (at the beginning of the string) or trailing (at the end of the string) whitespace characters such as spaces, tabs, and newlines from the string. It returns the modified string as a new string object

Here's an example to demonstrate the use of the strip() method:

#Define a string with leading and trailing whitespaces

my\_string = "  Hello World"

Call the strip() method to remove the whitespaces#

new\_string = my\_string.strip()

Print the modified string#

print(new\_string)   # Output: "Hello World"

In this example, the strip() method is called on the my \_ string object to remove the leading and trailing whitespaces. The modified string is then assigned to a new variable called new\_string. Finally, the modified string is printed to the console, which outputs "Hello, World!" with no leading or trailing whitespaces

7-

In Python programming, the built-in function len () is used to return the length of its string argument

The len () function takes a string as its argument and returns the number of characters in the string, including any whitespace characters. It can also be used to return the length of other collection types such as lists, tuples, and dictionaries.

Here's an example to demonstrate the use of the len() function:

Define a string#

my\_string = "Hello, World"

Call the len() function to get the length of the string#

string\_length = len(my\_string)

#Print the length of the string

print(string\_length)   # Output: 13

In this example, the len() function is called on the my\_string object to get the length of the string. The length of the string is then assigned to a variable called string\_length. Finally, the length of the string is printed to the console, which outputs 13

8-

In Python programming, the built-in open() function returns a file object. The file object represents a file that has been opened for reading, writing, or both. The file object provides methods and attributes that allow you to read from or write to the file, control the position of the file pointer, and perform other operations on the file. Here is an example of opening a file in Python:

file = open("example.txt", "r")

In this example, open() returns a file object that is stored in the variable file. The first argument to open() is the name of the file to be opened, and the second argument is the mode in which the file should be opened (in this case, "r" for reading)

9-

In Python programming, the second parameter of the open() function represents the mode in which the file should be opened. The mode parameter is a string that specifies the access mode and the file type

Here are the most commonly used modes:

"r": read mode (default). The file is opened for reading, and if the file does not exist, an error is raised.

"w": write mode. The file is opened for writing, and if the file exists, it is truncated to zero length. If the file does not exist, a new file is created

"a": append mode. The file is opened for writing, and if the file exists, the data is appended to the end of the file. If the file does not exist, a new file is created

"x": exclusive creation mode. The file is opened for writing, but only if the file does not already exist. If the file exists, an error is raised

"b": binary mode. The file is opened in binary mode, which is used for non-text files like images or binary data

t": text mode (default). The file is opened in text mode, which is used for text files"

You can also combine modes by using the + character. For example, "w+" opens the file for both reading and writing

10-

Here's an example program in Python that stores the first 100 integers to a text file named numbers.txt, with each number appearing on a line all by itself:

with open('numbers.txt', 'w') as file:

for i in range:(1, 101)

file.write(str(i) + '\n')

In this program, we first use the open() function to create a new file named numbers.txt in write mode ('w'). We use the with statement to ensure that the file is automatically closed when we're done with it

Next, we use a for loop to iterate over the range of integers from 1 to 100. For each integer, we convert it to a string using the str() function, and then write it to the file using the write() method of the file object. We also add a newline character ('\n') after each number to ensure that each number appears on its own line

When the loop is finished, the file is automatically closed, and the program terminates

11-

Here's an example implementation of the sumfile() function in Python that reads a collection of integers from a text file named numbers.txt, where each number appears on a line all by itself. The function accepts a single parameter, a string representing the file name, and returns the sum of the integers in the file:

def sumfile(filename):

total = 0

with open(filename, 'r') as file:

for line in file:

total += int(line.strip())

return total

In this function, we first initialize a variable total to 0 to keep track of the running total of the integers in the file.

We then use the open() function to open the file in read mode ('r'). We use the with statement to ensure that the file is automatically closed when we're done with it.

Next, we use a for loop to iterate over each line in the file. For each line, we use the strip() method to remove any leading or trailing whitespace, and then convert the resulting string to an integer using the int() function. We add the resulting integer to the total variable.

When the loop is finished, we return the final value of total, which represents the sum of all the integers in the file

.

12-

Here are the syntactic sugar methods for each of the following methods of the Fraction class in Python:

(a) sub: The sub() method is used to subtract one fraction from another. The syntactic sugar for this method is the - operator. For example, f1 - f2 is equivalent to f1.sub(f2).

(b) eq: The eq() method is used to check if two fractions are equal. The syntactic sugar for this method is the == operator. For example, f1 == f2 is equivalent to f1.eq(f2).

(c) neg: The neg() method is used to negate a fraction (i.e., change its sign). The syntactic sugar for this method is the - operator when used as a unary operator. For example, -f1 is equivalent to f1.neg().

(d) gt: The gt() method is used to check if one fraction is greater than another. The syntactic sugar for this method is the > operator. For example, f1 > f2 is equivalent to f1.gt(f2)

.

13-

In Python's Turtle graphics module, using a Turtle object is different from using the free functions in a few ways:

1. Object-Oriented Programming: When using a Turtle object, you are using an instance of a class that has been created from the Turtle class. This means that you are using object-oriented programming (OOP) to control the turtle. With free functions, you are not using OOP and are simply calling functions that are defined within the Turtle graphics module.

2. Method Invocation: When using a Turtle object, you use dot notation to call the methods of the object. For example, t.penup() calls the penup() method of the t object. With free functions, you simply call the function by its name, such as penup().

3. Object State: When using a Turtle object, you can modify the state of the object, such as its position and orientation, by calling its methods. For example, t.forward(100) moves the turtle object t forward by 100 units. With free functions, you cannot modify the state of the turtle object directly.

4. Multiple Turtles: When using a Turtle object, you can create multiple turtles by creating multiple instances of the Turtle class. Each turtle object can have its own state and can be controlled independently. With free functions, you can only use one turtle at a time.

In summary, using a Turtle object allows you to use OOP to control the turtle, call its methods using dot notation, modify its state, and create multiple turtles. Using free functions is simpler and more straightforward, but does not allow for the same level of control as using a Turtle object

.

14-

در فایل های جدا نوشتم

15-

No, Python strings are immutable, which means that once a string is created, it cannot be modified. However, you can create a new string that is a modified version of the original string.

To change a symbol in a string object, you need to create a new string with the modified symbol(s) using string slicing or string concatenation. Here's an example where we replace the symbol at index 3 with a new symbol 'z':

string = 'hello world'

new \_ string = string[:3] + 'z' + string[4:]

print(new \_ string)  # Output: helzo world

In this example, we create a new string new \_ string by slicing the original string string into two parts: the substring before the symbol we want to replace (string [:3]) and the substring after the symbol we want to replace (string [4:]). We then concatenate the two substrings with the new symbol 'z' in between to create the modified string new \_ string

16-

If a turtle .Turtle object were immutable in Python programming, it would not be possible to modify the properties or state of the object once it has been created.

This would severely limit the usefulness of the turtle module for creating complex drawings, animations, and simulations. Modifying the properties and state of a Turtle object is essential for creating complex and varied shapes, moving the turtle to different locations on the screen, changing the pen color and size, and creating animations and simulations that involve multiple objects with different behaviors.

For example, if the position or orientation of a Turtle object could not be changed, it would be impossible to draw complex shapes or move the turtle to different locations on the screen. Similarly, if the pen color or size could not be changed, it would be difficult to create colorful and varied drawings.

In addition, if the Turtle object were immutable, it would not be possible to create multiple turtles with different properties or states. This would limit the ability to create complex animations or simulations that involve multiple objects with different behaviors.

Overall, the ability to modify the properties and state of a turtle.Turtle object is essential for creating complex and dynamic drawings and animations using the turtle module in Python programming.

17-

In Python programming, garbage refers to objects that are no longer needed or used by a program, but are still taking up memory space. These objects are referred to as "garbage" because they are no longer useful or needed, and are simply taking up space in memory.

Python has a garbage collector that automatically frees up memory by identifying and removing objects that are no longer being used by the program. The garbage collector runs periodically in the background, and frees up memory by deleting objects that are no longer needed.

In some cases, it may be necessary to manually free up memory by deleting objects that are no longer needed using the del statement. However, in most cases, the Python garbage collector is able to automatically manage memory and remove garbage objects, making memory management easier and more efficient for programmers

18-

Garbage collection is the automatic process of freeing up memory space that is no longer being used by a program. In Python, the garbage collector works by tracking the references to objects in the program. When an object is no longer being referenced by any part of the program, it is considered garbage and is eligible for collection by the garbage collector

The garbage collector periodically runs in the background and identifies the objects that are no longer being used by the program. It then frees up the memory space used by those objects, making it available for future use by the program

Python's garbage collector uses a technique called reference counting to keep track of the references to objects. Each object in Python has a reference count, which is the number of references to that object. When the reference count of an object drops to zero, the object is considered garbage and is eligible for collection

In addition to reference counting, Python's garbage collector also uses a technique called cycle detection to identify and collect cyclic references. A cyclic reference occurs when two or more objects reference each other in a loop, forming a cycle. In such cases, the reference count of each object in the cycle is greater than zero, even though the program no longer needs the objects. The garbage collector detects such cycles and breaks them by setting the reference count of one or more objects in the cycle to zere

Overall, Python's garbage collector is a useful feature that helps programmers manage memory usage in their programs and avoid memory leaks

19-

(a)

At the end of this code's execution, the reference count fo.r the string object "ABC" will be 1. When the code assigns a new value "XYZ" to the variable a, it creates a new string object in memory with a reference count of 1. The old string object "ABC" is still referenced by the variables b and c, but not by a, so its reference count drops to 1

(b)

No, at the end of this code's execution, b is not an alias of a. When the code assigns a new value "XYZ" to the variable a, it creates a new string object in memory with a reference count of 1. The variable b still references the old string object "ABC", so a and b are no longer aliases of each other

(c)

Yes, at the end of this code's execution, b is still an alias of c. When the code assigns the value of a to b and the value of b to c, it creates aliases of the same string object "ABC". When a is later assigned a new value "XYZ", it creates a new string object in memory, but it does not affect the aliases b and c. So b and c are still aliases of the same string object "ABC"