HW_THIRD WEEK_FARVARDIN#Hooman_Ileayi#

1. What is the difference between a *class* and an *object*?

A class is a blueprint or template for creating objects, while an object is an instance of a class. In other words, a class defines the properties and behaviors that an object of that class can have, while an object is a specific instance of that class with its own unique values for those properties.

2. What are some other names for the term *instance variable*?

Instance variables are also commonly referred to as member variables or attributes.

3. What is another name for the term *method*?

A method is also commonly referred to as a function or a subroutine.

4. What symbol associates an object with a method invocation?

The dot (.) symbol is used to associate an object with a method invocation.

5. How does a method differ from a function?

In object-oriented programming, a method is a function that is associated with an object and can access and modify the object's data. A function, on the other hand, is a standalone block of code that performs a specific task and may or may not take inputs and return outputs. In other words, a method is a type of function that is associated with an object, while a function is a more general term that can refer to any standalone block of code.

6. What method from the string class returns a new string with no leading or trailing whitespace?

The `strip()` method from the string class returns a new string with no leading or trailing whitespace.

7. What function returns the length of its string argument?

The `len()` function returns the length of its string argument.

8. What type of object does the open function return?

The `open()` function returns a file object.

9. What does the second parameter of the open function represent?

The second parameter of the `open()` function represents the mode in which the file should be opened. It specifies whether the file should be opened for reading, writing, or both, and whether the file should be treated as a text file or a binary file.

10. Write a program that stores the first 100 integers to a text file named numbers.txt. Each number should appear on a line all by itself.

Here's an example Python program that stores the first 100 integers to a text file named :``numbers.txt

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

:(١٠١ ,١) for i in range

file. Write(str(i) + '¥n')
```

In this program, we use the `open()` function to open a new file named `numbers.txt` in write mode. We then use a `for` loop to iterate over the numbers 1 through 100, and for each number, we write it to the file as a string followed by a newline character (`¥n`). Finally, we close the file using the `with` statement, which ensures that the file is properly closed even if an error occurs.

11. Complete the following function that reads a collection of integers from a text file named numbers.txt. Each number in the file appears on a line all by itself. The function accepts a single parameter, a string text file name. The function returns the sum of the integers in the file.

```
def sumfile(filename):
# Add your code here . . .
```

Here's an example Python function that reads a collection of integers from a text file :named `numbers.txt` and returns their sum

```
: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 use the `open ()` function to open the file specified by the `filename` parameter in read mode. We then use a `for` loop to iterate over each line in the file, and for each line, we convert it to an integer using the `int()` function and add it to a running total. Finally, we return the total sum of all the integers in the file. Note that we use the `strip()` method to remove any leading or trailing whitespace from each line before converting it to an integer.**

**Here are the syntactic sugar notations for each of the following methods of the `Fraction` :class**

**`-` :`__(a) `__sub**

**The `__sub__` method is used to subtract one fraction from another. The `-` operator is the .syntactic sugar for this method**

**`==` :`__(b) `__eq**

**The `__eq__` method is used to test whether two fractions are equal. The `==` operator is .the syntactic sugar for this method**

**`-` :`__(c) `__neg**

**The `__neg__` method is used to negate a fraction (i.e., change its sign). The `-` operator is .the syntactic sugar for this method**

**`<` :`__(d) `__gt**

**The `__gt__` method is used to test whether one fraction is greater than another. The `>` operator is the syntactic sugar for this method.**

13. How is using a Turtle object from Python's Turtle graphics module different from using the free functions; for example, t.penup() versus penup()?

**Using a Turtle object from Python's Turtle graphics module allows you to create and manipulate multiple turtles, each with its own set of properties and methods. This is**
```

different from using free functions like `penup()`, which operate on a default turtle that is .automatically created when the Turtle graphics module is imported

When using a Turtle object, you first need to create a new turtle using the `Turtle()` :function, like this

```
import turtle

# Create a new turtle
t = turtle.Turtle()
```

You can then use the various methods and properties of the `t` object to control the turtle's behavior. For example, you can move the turtle forward by calling the `forward()` :method

```
# Move the turtle forward 100 units
t.forward(100)
```

In contrast, when using free functions like `penup()`, you don't need to create a new turtle object, because these functions operate on the default turtle that is automatically created when the Turtle graphics module is imported. For example, you can lift the pen up using :the `penup()` function like this

```
import turtle
```

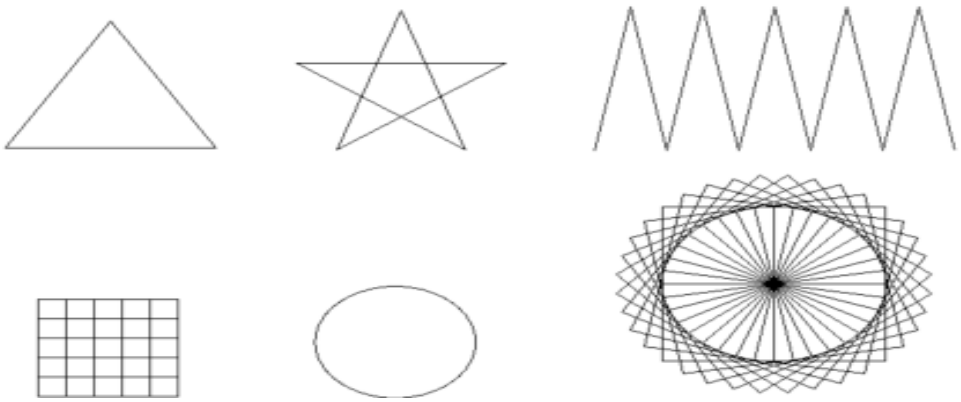
Lift the pen up #

()turtle.penup

```

**Overall, using a Turtle object allows for more fine-grained control over multiple turtles, while using free functions provides a simpler interface for controlling a single default turtle.**

14. For each of the drawings below write a program that draws the shape using a Turtle object from Python's Turtle graphics module.



**import turtle**

**Create a turtle object #**

**()t = turtle.Turtle**

**Draw a circle with radius 50 #**

**(٥٠)t.circle**

**Keep the window open until closed manually #**

**()turtle.done**

**import turtle**
```

```
Create a turtle object #
()t = turtle.Turtle

Draw a triangle with side length 100 #
:(٣)for i in range
(١٠٠)t.forward
(١٢٠)t.left

Keep the window open until closed manually #
()turtle.done

import turtle

Create a turtle object #
()t = turtle.Turtle

Draw a star with outer radius 100 and inner radius 50 #
:(٥)for i in range
(١٠٠)t.forward
(١٤٤)t.right
(٥٠)t.forward
(١٤٤)t.right

Keep the window open until closed manually #
()turtle.done
```

15. Does Python permit a programmer to change one symbol in a string object? If so, how?

No, Python does not permit a programmer to change one symbol in a string object directly. This is because strings in Python are immutable, which means that their values cannot be changed after they are created. However, you can create a new string that is based on the original string with the desired change using string slicing and .concatenation

For example, suppose you have a string `s` that you want to modify by changing the :second character to `x`. You could do this as follows

```
's = 'hello

[:٢]s = s[0] + 'x' + s

'print(s)  # Output: 'hexlo
```

In this example, we first retrieve the first character of the string using `s[0]`, then concatenate it with the new character `'x'` and the rest of the original string starting from the third character, which we retrieve using `s[2:]`. We then assign the resulting string to `s`, effectively creating a new string with the desired change.

16. What would be the consequences if a turtle.Turtle object were immutable?

If a `turtle.Turtle` object were immutable, it would not be possible to change any of its properties or attributes after it was created. This would severely limit the usefulness of the `turtle.Turtle` object, as one of its primary purposes is to allow for the creation of .interactive graphics that respond to user input

For example, if a `turtle.Turtle` object were immutable, it would not be possible to change its position on the screen, its direction of movement, or the color of its pen or fill. This

would make it difficult or impossible to create complex graphics or animations using the .Turtle graphics module

In addition, many of the methods and functions in the Turtle graphics module rely on the ability to modify the properties of a `turtle.Turtle` object, so if the object were immutable, many of these methods and functions would need to be re-implemented in a different way. This would likely result in a less efficient or less intuitive API for working with Turtle graphics.

17. In the context of programming, what is *garbage*?

In the context of programming, garbage refers to memory that has been allocated to a program but is no longer being used or referenced by the program. This can occur when a program creates objects or variables that are no longer needed, but fails to properly .release the memory associated with them

Garbage can accumulate over time and consume valuable system resources, which can lead to performance issues or even crashes if the program runs out of memory. To prevent this, many programming languages and environments include garbage collection .mechanisms that automatically identify and reclaim unused memory

Garbage collection can be performed using various algorithms, such as mark-and-sweep or reference counting, which track the usage of memory and identify objects that are no longer needed. Once these objects are identified, the garbage collector frees their associated memory so that it can be reused by the program.

18. What is *garbage collection*, and how does it work in Python?

Garbage collection is a process by which a programming language automatically identifies and frees memory that is no longer being used or referenced by a program. This helps to prevent memory leaks and other performance issues that can occur when a program .consumes too much memory

In Python, garbage collection is performed automatically by the Python interpreter using a technique called reference counting. Reference counting works by keeping track of the

number of references to each object in memory. When an object is created, its reference count is set to 1. Whenever a new reference to the object is created, such as by assigning it to a variable, the reference count is incremented. When a reference to the object is deleted, such as by reassigning the variable or when the variable goes out of scope, the .reference count is decremented

When the reference count of an object reaches 0, it means that there are no longer any references to the object in the program. At this point, the object is considered garbage and its memory can be freed. Python's garbage collector periodically scans the memory to .identify and collect garbage objects, freeing up their memory for reuse

In addition to reference counting, Python also includes a more sophisticated garbage collection mechanism called cyclic garbage collection, which is used to detect and collect objects that have circular references (i.e., objects that reference each other in a cycle). Cyclic garbage collection works by periodically tracing through all objects in memory and identifying cycles of objects that are no longer reachable from the program. Once these cycles are identified, the garbage collector can safely free their memory.

19. Consider the following code:
a = "ABC" b = a c = b a = "XYZ"
(a) At the end of this code's execution what is the reference count for the string object "ABC"? (b) At the end of this code's execution is b an alias of a?
(c) At the end of this code's execution is b an alias of c?

(a) At the end of this code's execution, the reference count for the string object "ABC" is .0, since there are no variables pointing to this object anymore

(b) No, at the end of this code's execution, b is not an alias of a. Initially, both a and b point to the same string object "ABC", but later a is reassigned to a new string object .""XYZ", while b remains pointing to the old object "ABC

(c) Yes, at the end of this code's execution, b is still an alias of c. Both b and c point to the same string object "ABC" throughout the execution of this code.