HW-WEEK4-#BaharSohrabi#40113841054157

Chapter9

1. A class is a blueprint or a template for creating objects, while an object is an instance of a class that contains its own set of attributes and behaviors. a class defines a category of objects, while an object is a specific instance of that class.

2. Other names for instance variables include attributes and fields.

3.They are known also as operations.

4. The dot (.) symbol associates an object with a method invocation in object-oriented programming.

5.The key difference between a method and a function is that a method is associated with an object or class, and can access and manipulate its data, while a function is a

standalone unit of code that can be called from anywhere. a method is ultimately a function, but it is defined and exists inside an
object.

6. The “strip()” method from the string class returns a new string with no leading or trailing whitespace. It removes all spaces, tabs, and newline characters from the beginning and end of the string.

7. The function that returns the length of its string argument is “len()”.

8. The “open()” function returns a file object.

9. The second parameter of the open() function represents the mode in which the file should be opened. This can be used to specify whether the file should be opened for reading, writing, or both. The mode can also determine whether the file should be created if it doesn't exist, truncate the file if it does exist, or allow the file to be accessed in binary mode. Common modes include "r"

for reading, "w" for writing, "a" for appending, and "b" for binary mode.

10.

```
 # Open the file for writing
with open('numbers.txt', 'w') as f:
    # Write the first 100 integers to the file
    for i in range(1, 101):
        f.write('{}\n'.format(i))
```

11.

```
def sumfile(filename):
    total = 0
    with open(filename, 'r') as file:
        for line in file:
            number = int(line.strip())
            total += number
    return total
```

12.

(a) The syntactic sugar for the sub method would be the subtraction operator -. It would allow us to subtract one fraction from another using the - operator, rather than calling the sub method explicitly.

Example: f1 - f2 is equivalent to f1.sub(f2)

(b) The syntactic sugar for the eq method would be the equality operator ==. It would allow us to check if two fractions are equal using ==, rather than calling the eq method explicitly.

Example: f1 == f2 is equivalent to f1.eq(f2)

(c) The syntactic sugar for the neg method would be the negation operator -. It would allow us to negate a fraction using the - operator, rather than calling the neg method explicitly.

Example: -f1 is equivalent to f1.neg()

(d) The syntactic sugar for the gt method would be the greater than operator >. It would allow us to compare two fractions and check if one is greater than the other using >, rather than calling the gt method explicitly.

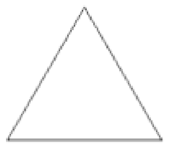
Example: f1 > f2 is equivalent to f1.gt(f2)

13. When using a Turtle object from the Turtle graphics module in Python, we first have to create an instance of the object before using its methods or attributes. This is typically done by creating a Turtle object and assigning it to a variable, such as t = turtle.Turtle(). We can then use the methods and attributes of the Turtle object by calling them using the dot notation, such as t.penup().

On the other hand, when using the free functions provided by the Turtle graphics module, we do not need to create an instance of any object. We can directly call the functions and use them to draw on the turtle canvas, such as penup().

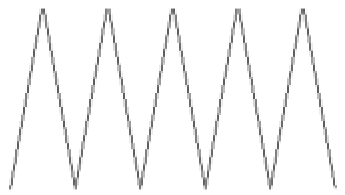
In summary, using a Turtle object allows for greater control and customization of the turtle's attributes and methods, while the free functions offer a more streamlined and simple approach to drawing on the canvas.

14.

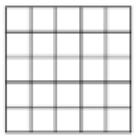


```
from turtle import *
t = Turtle();
t.pensize(5)

for i in range(3):
	t.forward(200);
	t.left(120)

t.hideturtle();
exitonclick();
```

```
from turtle import *
t = Turtle();
t.pensize(5)

t.left(36)
t.forward(300);

for i in range(4):
        t.left(144)
        t.forward(300);

t.hideturtle();
exitonclick();
```

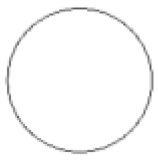


```
from turtle import *
t = Turtle();
t.pensize(5)

t.left(75);
t.forward(150);

right_bool = True;
 for i in range(9):
        if right_bool:
              t.right(150);
 else:
                t.left(150);
        t.forward(150);
        right_bool = not right_bool

t.hideturtle();
exitonclick();
```

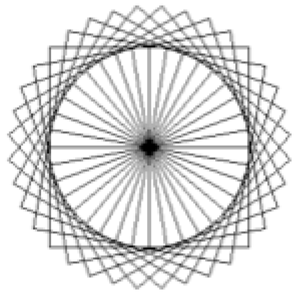


```
from turtle import *

def create_square(amount):
       for i in range(amount):
             for i in range(4):
                  t.forward(20);
                  t.right(90);
        t.penup();
        t.forward(20);
        t.pendown();

t = Turtle();
t.pensize(5)
```

```
t.left(90);
x = 0;
for i in range(5):
        t.penup();
        t.setposition(x, 0);
        t.pendown();
        create_square(5)
        x += 20;


t.hideturtle();
exitonclick();
```



```
from turtle import *

t = Turtle();
t.pensize(5)

t.circle(100);

t.hideturtle();
exitonclick();
```



```
from turtle import *

def square():
    for i in range(4):
        t.forward(150);
        t.right(90);

t = Turtle();
t.pensize(3)
t.left(90);

for i in range(36):
    square();
    t.right(10);

t.hideturtle();
exitonclick();
```

15. No, Python strings are immutable which means you cannot change a single character or symbol in a string directly.

However, you can create a new string with the desired changes by using string slicing and concatenation.

For example, to change the symbol at index 2 to 'x' in the string 'hello':

```
original_string = 'hello'
new_string = original_string[:2] + 'x' + original_string[3:]
print(new_string) # output: 'hexlo'
```

This creates a new string by concatenating the slice of the original string up to index 2, the new character 'x', and the slice of the original string from index 3 onwards.

16. We could not change its attributes, like pencolor or pensize.

17. In the context of programming, garbage refers to the computer memory that contains previously used data or code that is no longer needed by the program. This can happen when a program creates a new object or variable but doesn't delete it when it's no longer needed, causing it to take up unnecessary space in memory. Garbage can slow down a program's performance and even cause it to crash, which is why programmers use garbage collectors or other memory

management techniques to clean up and free memory used by unnecessary objects and variables.

18. Garbage collection is the process of automatically identifying and removing the unnecessary objects or memory allocations in a program's memory to free up space for new or reusable memory. In Python, garbage collection is done through a mechanism called reference counting.

In reference counting, Python maintains a count of the number of references pointing to each object in memory. When an object's reference count reaches zero, meaning it is no longer being used by any part of the program, the Python interpreter automatically deallocates the memory used by that object.

However, reference counting alone may not be sufficient for all cases, specifically when objects contain circular dependencies, where two or more objects are mutually referencing each other. To solve this problem, Python also employs a cycle-detection algorithm that identifies and collects such objects using a mark-and-sweep method.

Overall, Python's garbage collector automatically manages the memory, allowing developers to focus on writing code instead of manual memory management.

19.

(a) At the end of the code's execution, the reference count for the string object "ABC" is 1, since there is no longer any variable pointing to that object.

(b) No, at the end of the code's execution, b still references the string object "ABC", while a references the string object "XYZ". Therefore, b and a are not aliases of each other.

(c) Yes, at the end of the code's execution, both b and c reference the same string object "ABC". Therefore, they are aliases of each other.

The END >.<