PYTHON LOOP, CLASSES & FUNCTIONS

We learn about how to define classes and functions as well as different types of loops

LOOPS

There comes situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. Decision-making statements in programming languages decide the direction(Control Flow) of the flow of program execution.

if statement

```
if condition:
  # body of if statement
```

If condition is evaluated to True, the code inside the body of if is executed. If condition is evaluated to False, the code inside the body of if is skipped.

if condition:
statement1
statement2

```
    #Example
number = 10
# check if number is greater than 0
if number > 0:
  print('Number is positive.')
print('The if statement is easy')
```

#Output Number is positive.
The if statement is easy

```
number = -5 # if we supply number as negative
# check if number is greater than 0
if number > 0:
 print('Number is positive.')
print('The if statement is easy')
```

#Output *The if statement is easy This is because the value of* number *is less than 0. Hence, the condition evaluates to False. And, the body of if block is skipped*

if...else Statement

```
if condition:
  # block of code if condition is True
else:
  # block of code if condition is False

    number = 10
    if number > 0:
      print('Positive number')
    else:
      print('Negative number')
   print('This statement is always executed')
```

#Output: Positive number This statement is always executed

```
    Samplevar1 = 20
    if (Samplevar1< 15):
      print("Samplevar1  is smaller than 15")
      print("I am in if Block")
    else:
      print("Samplevar1 is greater than 15")
          print("I am in else Block")
    print("I am not in if and not in else Block")
```

#Output: Samplevar1 is greater than 15 I am in else Block I am not in if and not in else Block

Nested-if statement

```
if (condition1):
  # Executes when condition1 is true
  if (condition2):
    # Executes when condition2 is true
  # if Block is end here
# if Block is end here
```

```
Samplevar1 = 10
if (Samplevar1 == 10):
    #  First if statement
   if (Samplevar1 < 15):
      print("Samplevar1 is smaller than 15")
   # Nested - if statement
   # Will only be executed if statement above   is true
   if (Samplevar1 < 12):
      print("Samplevar1 is smaller than 12 too")
   else:
      print("Samplevar1 is greater than 15")
```

#Output: Samplevar1 is smaller than 15 Samplevar1 is smaller than 12 too

```
#Another example
number = 5
# outer if statement
if (number >= 0):
   # inner if statement
   if number == 0:
     print('Number is 0')
    # inner else statement
   else:
           print('Number is positive')
# outer else statement
else:
   print('Number is negative')
```

 # Output: Number is positive

if...elif...else Statement

```
if condition1:
   # code block 1
elif condition2:
   # code block 2
else:
   # code block 3
```

#Example

```
number = 0
if number > 0:
      print("Positive number")
elif number == 0:
         print('Zero')
else:
      print('Negative number')

print('This statement is always executed')
```

#Output : Zero This statement is always executed

For Loops in Python

For loop is used for sequential traversal i.e. it is used for iterating over an iterable like string, tuple, list, set or dictionaries **Note: In python for loops only implement the collection-based iteration.**

```
#Example
samplelanguages = ['Swift', 'Python', 'Go', 'JavaScript']
# run a loop for each item of the list
for language in samplelanguages:
   print(language)
```
#Output
Swift
Python
Go
JavaScript
```
#Example
for x in 'Python':
   print(x)
```
Output
P
y
t
h
o
n
```
#Example
d = dict()
 d['xyz'] = 123
d['abc'] = 345
for i in d:
   print("% s % d" % (i, d[i]))
```
Output:
Dictionary Iteration
xyz 123
abc 345

For Loop with a step size

This code uses a for loop in conjunction with the) function to generate a sequence of numbers starting from 0, up to (but not including) 10, and with a step size of 2. For each number in the sequence, the loop prints its value using the print() function. The output will show the numbers 0, 2, 4, 6, and 8.

```
for i in range(0, 10, 2):
   print(i)
```
Output :
0
2
4
6
8

For Loop inside a For Loop

```
for i in range(1, 4):
   for j in range(1, 4):
      print(i, j)
```
Output :
1 1
1 2
1 3
2 1
2 2
2 3
3 1

3 2
3 3

For Loop with Zip()

```
fruits = ["apple", "banana", "cherry"]
colors = ["red", "yellow", "green"]
for fruit, color in zip(fruits, colors):
   print(fruit, "is", color)
```

Output :
apple is red
banana is yellow
cherry is green

```
# Prints all letters except 'e' and 's'
for letter in 'geeksornerds':
    if letter == 'e' or letter == 's':
       continue
   print('Current Letter :', letter)
```

Output:
Current Letter : g
Current Letter : k
Current Letter : o
Current Letter : r
Current Letter : n
Current Letter : r
Current Letter : d

```
for letter in ''geeksornerds'':
 # break the loop as soon it sees 'e'or 's'
 if letter == 'e' or letter == 's':
       break
 print('Current Letter :', letter)
```

#Output:
Current Letter : g
 # An empty loop

```
for letter in “geeksornerds”:
   pass
print('Last Letter :', letter)
```

for loop with else

```
#Example
digits = [0, 1, 5]
for i in digits:
   print(i)
else:
   print("No items left.")
```

#Output:
0
1
5
No items left Note: The else block will not execute if the for loop is stopped by a break statement.

while Loop

```
while condition:
   # body of while loop
```

```
# program to display numbers from 1 to 5
# initialize the variable
samplestart = 1
sampleend = 5
# while loop from 1 to 5
while samplestart <= sampleend:
   print(sampleend)
   sampleend = sampleend + 1
#Output:
1
2
3
4
5
# program to calculate the sum of numbers
# until the user enters zero
total = 0
number = int(input('Enter a number: '))
# add numbers until number is zero
while number != 0:
   total += number    # total = total + number
   # take integer input again
   number = int(input('Enter a number: '))
print('total =', total)
#Output
Enter a number: 12
Enter a number: 4
Enter a number: -5
Enter a number: 0
total = 11
# Infinite time run example
age = 32
# the test condition is always true
while age > 18:
   print('You can vote')
#pop item from list using while
#checks if list still contains any element
Samplelista = [1,2,3,4,5]
While a:
            print(Samplelista.pop())
#Output:
5
4
3
2
1
# Prints all letters except 'e' and 's'
i = 0
samplestringa = 'geeksvsnerds'
 while i < len(samplestringa):
   if samplestringa[i] == 'e' or samplestringa[i] == 's':
      i += 1
      continue
 print('Current Letter :', samplestringa[i])
```

```
 i += 1
```
Output

Current Letter : g
Current Letter : k
Current Letter : v
Current Letter : n
Current Letter : r
Current Letter : d

```
counter = 0
while counter < 3:
   # loop ends because of break
   # the else part is not executed
   if counter == 1:
      break
   print('Inside loop')
   counter = counter + 1
else:
   print('Inside else')
#Output :
Inside loop
Inside else
#Example
samplenumber = int(input('Enter a number (-1 to quit): '))
 while samplenumber != -1:
   samplenumber = int(input('Enter a number (-1 to quit): '))
```

Different looping techniques using Python data structures are:

Using enumerate(): enumerate() is used to loop through the containers printing the index number along with the value present in that particular index.

```
#Example

for key,value  in enumerate ( ['AWS','AZURE','GCP','OPENSHIFT']):
        print(key,value)
```

#Output:
0 AWS
1 AZURE
2 GCP
3 OPENSHIFT

Using zip(): zip() is used to combine 2 similar containers(list-list or dict-dict) printing the values sequentially. The loop exists only till the smaller container ends.

```
#Example
Sampleques = [ 'favorite-fruit', 'favorite-game', 'favorite-place']
Sampleans = ['orange','motogp','sikkim']
for q,a in zip(Sampleques,Sampleans):
        print(" What is your {0} ? I like {1} .".format(q,a))
```

#Output:
What is your favorite-fruit? I like orange.
What is your favorite-game? I like motogp.
What is your favorite-place? I like sikkim.

Using items(): items() performs the similar task on dictionary as iteritems() but have certain disadvantages when compared with iteritems().

- It is very time-consuming. Calling it on large dictionaries consumes quite a lot of time.
- It takes a lot of memory. Sometimes takes double the memory when called on a dictionary.

Example

```
sampledict = {"tata": "nexon-ev", "mahindra": "xuv-400"}
print("To print the dictionary key-value: ")
for  i, j in sampledict.items():
        print(i,j)
```

Output:

To print the dictionary key-value:
tata nexon-ev
mahindra xuv-400

Another E.x.

```
Sampledict = {'Birbal:'Knowledgeable Person','Chandragupta':'Most skillfull king'}
for key, value in Sampledict.items():
        print(key, value)
```

#Output:
Birbal Knowledggeable Person
Changragupta Most Skillful king

Function

Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.Some Benefits of Using Functions Increase Code Readability ,Increase Code Reusability

Standard library functions - These are built-in functions in Python that are available to use.
User-defined functions - We can create our own functions based on our requirements.

```
def function_name(arguments):
   # function body

   return
```

def - keyword used to declare a function
function_name - any name given to the function
arguments - any value passed to function
return (optional) - returns value from a function

#Sample Example

```
def greet():
   print('Hello World!')
# call the function
greet()
print('Outside function')
```

#Output:
Hello World!
Outside function

Function return Type

```
def find_square(num):
   result = num * num
```

```
    return result
# function call
square = find_square(3)
print('Square:',square)
```
 # Output: Square: 9

Python Library Function

```
import math
# sqrt computes the square root
square_root = math.sqrt(4)
print("Square Root of 4 is",square_root)
# pow() comptes the power
power = pow(2, 3)
print("2 to the power 3 is",power)
```
#Output
Square Root of 4 is 2.0
2 to the power 3 is 8
Note:- the statement,import math ,Since sqrt() is defined inside the math module, we need to include it in our program.

Types of Python Function Arguments

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments *args and **kwargs)

#Default argument

```
def demofun(x, y = 50) :
        print("x: ",x)
        print("y: ",y)
demofun(10) # function call only x value provided
```
#Output:
x: 10
y: 50

#Keyword Argument

```
def my_function(child3,child2,child1):
        print("The youngest child is:"+ child3)
#keyword arguments
my_function(child1 = "Rohan", child2 ="Roshini",child3="Arunima")
```
#Output: The youngest child is Arunima

Python Arbitrary Keyword Arguments can pass a variable number of arguments to a function using special symbols. There are two special symbols:
***args in Python (Non-Keyword Arguments)**
****kwargs in Python (Keyword Arguments)**

```
# example of *args
def find_sum(*numbers):
   result = 0

   for num in numbers:
      result = result + num
```

```
    print("Sum = ", result)
# function call with 3 arguments
find_sum(1, 2, 3)
# function call with 2 arguments
find_sum(4, 9)
#Output:
Sum =  6
Sum =  13
# example of **kwargs
def sample_func(**kwargs) :
        for key, value in kwargs.items():
                print("%s -> %s" %(key,value))
sample_func(fruit='Apple',blend='Cold-Pressed',packaging='Tetra-Pack')
#Output:
fruit -> Apple
blend -> Cold-Pressed
packaging -> Tetra-Pack
#Last example combine both *args,**kwargs
def mysampl_func(*args, **kwargs):
        print("args :- ", args)
        print("kwargs :- ", kwargs)
mysampl_func('geeks' ,'or','nerds',company="TATA",model="Nexon",type="Hybrid")
#Output:
args: ('geeks', 'or', 'nerds')
kwargs: {'company': 'TATA', 'model': 'Nexon', 'type': 'Hybrid'}
```

Docstring

The first string after the function is called the Document string or docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

```
#Adding Docstring to the function
def my_evenodd(samplevar) :
        """Function to check if the number is even or odd"""
        if(samplevar % 2 == 0) :
                print("even")
        else :
                print("odd")
print(my_evenodd.__doc__)
my_evenodd(49)
#output:
Function to check if the number is even or odd
odd
```

lambda Function Declaration

We use the lambda keyword instead of def to create a lambda function. Here's the syntax to declare the lambda function:

lambda argument(s) : expression

Here,

argument(s) - any value passed to the lambda function

expression - expression is executed and returned

```
# example,
greet = lambda : print('Hello World')
# call the lambda
greet()
```

lambda Function with an Argument

```
# lambda that accepts one argument
greet_user = lambda name : print('Hey there,', name)
# lambda call
greet_user('Delilah')
# Output: Hey there, Delilah
```

Class

A class is a template for creating objects. A class specifies the attributes (data) and methods (functions) that a class's objects can have. Classes are defined using the `class` keyword, and objects are created using the class constructor.

```
class Bike:
  name = ""
  gear = 0
```

Here,

- Bike - the name of the class
- name/gear - variables inside the class with default values "" and 0 respectively.

To understand the meaning of classes we have to understand the built-in __init__() function.All classes have a function called __init__(), which is always executed when the class is being initiated.Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created

Another term is the self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

#Example:

```
class Person:
  def __init__(self, name, country):
    self.name = name
    self.country = country
person = Person("Rishab", "Canada")
print(person.name)   # OUTPUT "Alice"
print(person.country)    # OUTPUT "Canada"
```

```
#Example
class TrafficLight:
  '''This is a traffic light class'''
  color = 'green'

  def action(self):
    print('Go')

traffic = TrafficLight()
```

```
print("Print the docstring",traffic.__doc__)
print("What is the color",traffic.color)
print(traffic.action())
#Output
Print the docstring This is a traffic light class
What is the color green
Go
None

# Different
class TrafficLight:
    '''This is an updated traffic light class'''
    def __init__(self, color):
        self.color = color

    def action(self):
        if self.color=='red':
            print('Stop & wait')
        elif self.color=='yellow':
            print('Prepare to stop')
        elif self.color=='green':
            print('Go')
        else:
            print('Stop drinking ')

sampleobj = TrafficLight('yellow')
sampleobj.action()
#Output prepare to stop
```

Creating and Deleting Object Attributes in Python

It's possible to add a new attribute to a class object after its instantiation:

```
sampleobj = TrafficLight('green')
sampleobj2 = TrafficLight('yellow')
sampleobj.next_color = 'red'
print(sampleobj.next_color)
print(sampleobj2.next_color)
#Output
red
Traceback (most recent call last):
ERROR!
File "<string>", line 26, in <module>
AttributeError: 'TrafficLight' object has no attribute 'next_color'
```

In the code above, we created a new attribute next_color for the sample object and assigned it to "red." Since this attribute isn't in the TrafficLightclass definition, when we try to check it for another object (yellow), we get an AttributeError.

Delete Attributes and object

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
```

```
del p1.age
print(p1.age)
#output
Traceback (most recent call last):
  File "./prog.py", line 11, in <module>
AttributeError: 'Person' object has no attribute 'age'
```

```
class Person:
  def __init__(mysself, name, age):
    mysself.name = name
    mysself.age = age

  def myfunc(mysself):
    print("Hello my name is " + mysself.name)

p1 = Person("Travis", 36)
del p1
print(p1)
#output
Traceback (most recent call last):
  File "./prog.py", line 11, in <module>
NameError: name 'p1' is not defined
```

Class and Instance Variables

The difference between class and instance variables is where their values were assigned in a class definition, which also determines the scope of the variables.

Class variables: assigned in the class declaration but outside any method definition (including the class constructor). They are related to all object instances of that class.

Instance variables: assigned inside a class method or constructor definition. They are unique to each object instance of that class.

```
class TrafficLight:
    '''This is an updated traffic light class'''
    # Class variable
    traffic_light_address = 'NYC_Cranberry_Hicks'
    def __init__(self, color):
        # Instance variable assigned inside the class constructor
        self.color = color
    def action(self):
        if self.color=='red':
            # Instance variable assigned inside a class method
            self.next_color = 'yellow'
            print('Stop & wait')
        elif self.color=='yellow':
            self.next_color = 'green'
            print('Prepare to stop')
        elif self.color=='green':
            self.next_color = 'red'
            print('Go')
        else:
            self.next_color = 'Brandy'
```

```
            print('Stop drinking')
# Creating class objects
for c in ['red', 'fuchsia']:
    c = TrafficLight(c)
    print(c.traffic_light_address)
    print(c.color)
    c.action()
    print(c.next_color)
    print('\n')
#ouptut
NYC_Cranberry_Hicks
red
Stop & wait
yellow

NYC_Cranberry_Hicks
fuchsia
Stop drinking
Brandy
```

For all the class instances above, we have the same value of the traffic_light_address variable, which is a class variable. The color variable was assigned inside the class constructor, and the next_color variable was calculated inside the action() class method based on the value of color for each class instance. These two are instance variables.