Module 1

Introduction to everything

Section

Python programming: control flow, functions & generators

Python control flow

Following are the key details of flow control statements:

- Flow control statements can decide which Python instructions to execute under which conditions.
- Flow control statements depend on certain yes or no type operations, which help determine the flow of any program.

Flow control elements often start with a part called condition, which is followed by a block of code called clause.

Conditions

- Conditions always evaluate down to a Boolean value True or False.
- Conditions are the base for control statements to take a decision about the next step.

Blocks of code

Lines of code grouped together are called blocks. In Python, indentation is used to specify the start and end of a block.

If statement

1. Following are the key details of control flow – if:
2. It is the most common type of flow control statement.
3. An ‘if statement’ could be read as, “If this condition is true, execute the code in the clause.” In Python, an if statement consists of the following:
    - The if keywords
    - A condition (that is, an expression that evaluates to True or False)
    - A colon

4. Starting on the next line, an indented block of code (called the if clause)
5. All flow control statements end with a colon and are followed by a new block of code, the clause.

Else statements:

Following are the key details of else statements:

- An ‘if’ clause can optionally be followed by an else statement.
- The else clause is executed only when the ‘if’ statement’s condition is False.
- An else statement always consists of the following:
  - The else keyword
  - A colon
  - Starting on the next line, an indented block of code (called the else clause)
- An else statement doesn’t have a condition.

elif statements:

Following are the key details of ‘elif’ statements:

- The elif statement is an “else if” statement that always follows an ‘if’ or another elif statement.
- It provides another condition that is checked only if any of the previous conditions were False.
- In code, an elif statement always consists of the following:
  - The elif keyword
  - A condition (that is, an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the elif clause)

Example Code:

```
apple_color=input("Enter color of apple: ") #Taking Input from User

if apple_color=="red" or apple_color=="green":
   print("Eat it")

elif apple_color=="brown":
   print("It's rotten, Do not eat")

else:
   print("Unique Apple")
```

Output:

```
Enter color of apple: red
Eat it
```

Loops – ‘FOR loop’ and ‘RANGE () function’

Python has two loops: for and while.

A for loop is used for iterating over a sequence (that is, either a list, a tuple, a dictionary, a set, or a string). A for statement always consists of following-

- The for keyword
- A variable name
- The in keyword
- A call to the range() method with up to three integers passed to it
- A colon
- Starting on the next line, an indented block of code (called the for clause)

Example Code

```
for letter in 'Python':     # traversal of a string sequence
   print ('Current Letter:', letter)
print("\n")

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:       # traversal of List sequence
   print ('Current fruit:', fruit)

for i in range(5,10): #traversing from 5 to 9
    print(i,end="")

print ("\nGood bye!")
```

Output

```
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n

Current fruit: banana
Current fruit: apple
Current fruit: mango
56789
Good bye!
```

While loops

A block of code can be made to execute over and over again with a while statement.

The code in a while clause will be executed as long as the while statement’s condition is True. In code, a while statement always consists of the following:

1. The while keyword
2. A condition (that is, an expression that evaluates to True or False)
3. A colon
4. Starting on the next line, an indented block of code (called the while clause)

Example Code

```
count = 0
while (count < 9):

  print ('The count is:', count)
  count = count + 1

print ("Good bye!")
```

Output

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Python functions, generators, and lambda

Functions

- The functions are named blocks of code, **designated to do a specific job**, they are like **mini-programs within a program.**
- They are used to define a set of actions and those actions can be executed in the script wherever needed by calling the function.
- They can optionally take inputs called parameters and can output a value.
- By having the conditional statements and loop statements inside the function, and by coupling them with the input arguments, any kind of functionality can be created and this can be quite handy in reusability of code.
- They can write any text to the stream or it can return the text. Both of them are different ways of getting the functions output.

Example Code:

```
# Function definition is here
def printme( str ):

   "This prints a passed string into this function"
   print (str)
   return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

Output:

```
This is first call to the user defined function!
Again second call to the same function
```

Generators

A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function. Generator objects are used either by calling the **next method on the generator object** or using the generator object in a **“for in”** loop.

Example Code:

```
# A generator function that yields "apple" for first time,
# "banana" second time and "orange" third time
def simpleGeneratorFun():
    yield "apple"
    yield "banana"
    yield "orange"

# Driver code to check above generator function
for value in simpleGeneratorFun():
    print(value)
```

Output:

```
apple
banana
orange
```

In [ ]:

Lambda function

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression. It works similar to regular function but can be defined without a name. We use lambda functions when **we require a nameless function for a short period of time**.

Example Code:

```
f=lambda x: x+" is fun"
f("python")
```

Output:

```
'python is fun'
```

Summary

- Functions are the primary and the most important methods of code organization and reuse in Python.
- Python generators help in memory and compute efficient processing data and performing computations.
- Map and filter functions can be used to perform data transformation and filtering on iterable objects.
- Lambda functions are anonymous functions – a pythonic way of defining functions in one line.