Introduction to Python

Understanding upGrad Coding Console

Solving Coding Console Problems

In the course of the program, you will come across questions that will need you to write codes. These questions are spread across the program, and they would help develop your problem-solving and coding abilities. Hence, it is crucial for you to spend time answering these questions as these would help you make the most of the upGrad learning experience.

Now, before you start solving the coding questions, it is necessary that you go through the tutorial for the coding console that will be used for answering the questions on the upGrad platform.

Also, you need to know about certain specific keywords in order to attempt the coding questions. Here is a quick list of these keywords, along with their meaning:

- **Problem Stub -** A problem stub (also known as a 'language stub') is the skeleton code, which the teacher has provided for a particular language. You have to solve the coding problem by adding your code to the given skeleton code.
- **Execution Time Limit -** This refers to the time taken for the code to execute from start to finish for the given input on the server. Note: This time is independent of your internet speed. In case your code exceeds the time limit for a particular input, you will receive the following error message: 'Time limit exceeded. The execution took more than 5.00 s.'
- **Load Language Stub -** This is the action that you can find in the overflow menu. On clicking this action, you will be able to go back to the problem stub. This

action will help you replace your current solution code with the problem stub, or the skeleton code.

Demystifying Test Cases

Test cases are a tool to evaluate your code. A test case determines whether or not the solution code (code written by you) works for a particular scenario. In most questions, your code is checked against multiple test cases to check whether or not it meets the requirements specified by the teacher.

A test case has the following two major components: an input, i.e., the scenario for which your code is being tested, and an expected output, i.e., the expected results for that particular input. If the output returned by your code is the same as the expected output, given a specific input, then the test case is deemed as ‘Passed’. Otherwise, it is deemed as ‘Failed”.

Let’s consider an example of a test case:

```
Input:
2
87 98
87 89 67 56
Expected Output:
2
4
```

In this example, the test case includes the input of two arrays with lengths 2 and 4. If the output of your code for the test case matches the expected output, which is 2 and 4, then your code will pass the test case.

Primary Actions

So far, you have learnt how the coding console looks and how you can use it to write your code. But you might be wondering how you can test and submit your code using the editor.

You can perform the following three different actions using the coding console:

1. **Run Code**

- It checks the output of your code against a custom input provided under the 'input' section.
    - It can be done as many times as possible.
    - Run Code is commonly used to debug and check for various errors and mistakes made during the process of writing the solution code.
2. **Verify**
    - It verifies your code against the sample test cases.
    - It can be done as many times as possible.
3. **Submit**
    - This refers to a formal submission. It tests your code for correctness by validating it against all the test cases.
    - It can be performed only a limited number of times.

Types of Test Cases

Test cases are of the following two types:

1. **Sample Test Case**
    - It is executed when you perform the action of "Verify" and "Submit".
    - You will be able to see the Input and the Expected output at all times.
    - The primary purpose here is to assess how your code is performing against some test cases, so that you make corrective changes to the code and then submit it.
2. **Non-Sample Test Case**
    - It is executed only when you perform the action of "Submit".
    - You will be able to see the Input and the Expected output only when you exhaust all the Submissions or have not fully understood the question.
    - The primary purpose here is to check the correctness of the solution code.

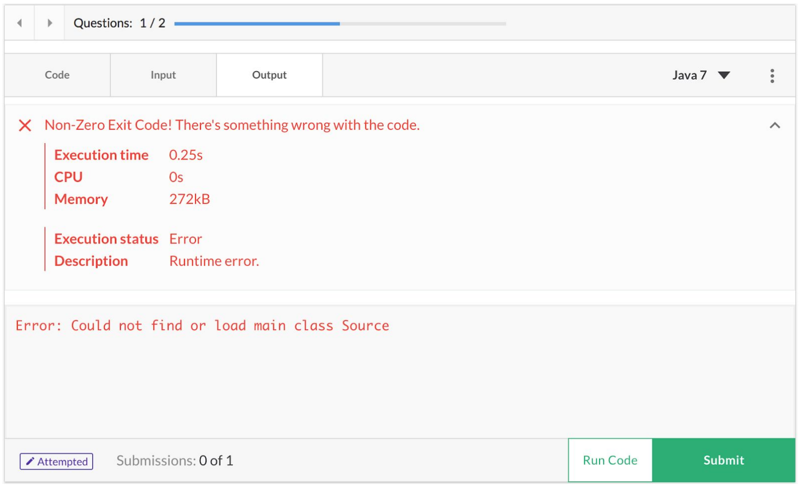
Understanding Statuses and Important Pointers

Now that you have learnt how to submit your code, let's see how we can complete this question by making further attempts. After that, we will provide you with some additional pointers that will help you use this tool effectively.

As we went through the process of solving the entire coding question, you would have come across certain statuses. So, you will now learn about some of the statuses that you would encounter while attempting a coding question:

- **Unsolved -** You have not made any changes to the problem stub.
- **Attempted -** You have made some changes to the problem stub, which have been saved.
- **Submitted -** This is a temporary state, which comes when you have clicked on 'Submit', and the test cases are being executed on the server. After the test cases are executed, the status changes. Note that this will be the final state in questions where there are no test cases.
- **Rejected -** The code has failed all of the non-sample test cases.
- **Partially Correct -** The code has failed some of the non-sample test cases.
- **Accepted -** The code has passed all of the non-sample test cases. Your solution code is correct.

Note: In Java, the name of the class needs to be 'Source'. Please make sure you do not tamper this part, as doing so would result in the following error message.



```
Non-Zero Exit Code! There's something wrong with the code.
Execution time    0.25s
CPU               0s
Memory            272kB

Execution status  Error
Description       Runtime error.

Error: Could not find or load main class Source
```

Now, let's take a look at a few important pointers before we end this tutorial:

1. We match the output exactly. Spaces, Prints, etc. are to be managed separately.
2. We strongly recommend that the code be written here and not transported to an external IDE. **Please note -** As the console automatically loads the libraries used

by Python, copy-pasting this code to Jupyter or to any other external IDE directly would require you to add the libraries manually for the code to run.

3. Spelling mistakes in output statements would result in failed test cases. Please be careful.

Basics of Python

Welcome to the module on 'Introduction to Python'. In this session, you will be introduced to the basic syntax of Python and learn the basics of programming in Python.

The first module consists of the following four sessions:

· **Basics of Python**: In this session, you will be introduced to Python programming and the environment on which you will be coding. Once you have the basic set-up ready, you will learn how to write your first program in Python and will be introduced to the different data types. Towards the end of this session, you will learn about the various arithmetic and string operations that are supported by Python.

· **Data Structures in Python**: In this session, you will first learn about the various data structures in Python, including tuples, lists, sets and dictionary. You will then learn about these data structures in detail and the various operations related to them.

· **Control Structure and Functions in Python**: In this session, you will learn about the various decision-making control structures and functions supported by Python. These structures and functions form the essence of programming as they help computers to automate repetitive tasks intelligently

· **OOP(Object Oriented Programming) in Python**: In this session, you will learn about the various object-oriented programming methodologies in Python, including classes, objects and methods.

In the first session, you will understand the reasons why Python is the language of choice for Data Science. In order to learn how to design and code programs, you need to understand the types of data that you want to work with and learn how to manipulate these data types. You will learn about the different data types and the different operations that you can perform on each data type. This will be followed by a series of practice exercises that will give you a hands-on experience of these concepts. Please go through the content multiple times until you have understood the concepts clearly before attempting the practice questions, especially if you are new to programming.

How to learn a new programming language, especially if you are new to programming?

If you are new to programming, please go through the videos multiple times to understand the concepts clearly. Programming is essentially a different way of thinking. Just like driving a vehicle, programming may feel challenging initially, but once you get the hang of it, it will almost become second nature to you.

Please do not go through the content in one go. To maximise your learning, you need to set up dedicated time daily throughout the week for learning and reviewing the content. If you do this, you will surprise yourself with how much you have learnt and retained throughout each week.

Unlike other fields, the only way to learn programming is through practice. Feel free to make as many mistakes as you want while practising different programming tasks. You will rarely get the program right the first time you write it. So, expect to make mistakes but remember to learn from them. Slowly, you will start making sense of the concepts.

History of Python and Its Importance

Python is a general-purpose programming language that was named after **Monty Python**, a British surreal comedy troupe. It resembles the English language, and hence, it is simple and easily readable. Still, why should you use Python?

Python was created by Guido van Rossum as a hobby project but soon became a general-purpose programming language. Today, it is used for building software for multiple purposes. Python is an open source language and it is very easy to build packages in Python. It provides packages for data visualisation, database management, EDA(Exploratory Data Analysis), machine learning algorithms and many more.

Python is a high-level scripting language that can be used for various text processing, system administration and internet-related tasks. It is an interpreter-based and a true object-oriented language that is available on a wide variety of platforms. It also supports multiple language paradigms.

Python is used in almost every domain. You can go through its official **website** to get an overview on this. In addition to its features, its simplicity and efficiency in requiring fewer lines of code for performing tasks has encouraged many developers across the world to take it up.

Installation Documentation

You will need various **Python packages (or libraries)** for specific purposes.

Anaconda is an open-source distribution that simplifies package management and deployment. The package management system 'Conda' manages various package versions.

We strongly recommend using Anaconda to install Python because it comes preloaded with most of the packages that you will need for performing various operations and tasks.

Advantages of using Anaconda

1. It is easy to manage and supports most of the libraries that are required for machine learning or artificial intelligence problems.

2. It comes preloaded with many libraries such as NumPy, OpenCV, SciPy, PyQt and the Spyder IDE.

You can download and install Anaconda from anaconda.org like any other normal software. You need not download Python separately; the Anaconda installer will do it for you. **Do remember to select Python 3.x while downloading Anaconda.**

Note for experienced Python programmers: If you are already using Python along with an existing package manager such as pip or easy_install, you can continue to do so. However, ensure that you are using Python 3.x.

Jupyter Notebook

You will be using the Jupyter IPython Notebook as the main environment for writing Python code throughout this programme. The key advantage of using Jupyter Notebook is that you can write both code and normal text (using the Markdown format in Jupyter) in the notebooks. These notebooks are easy to read and share, and can also be used for presenting your work to others. Here's a brief overview of the Jupyter Notebook.

The document provided below contains instructions for installing Python and the Jupyter Notebook using Anaconda.

Also, you can refer to the document in case you need help with installing Anaconda. Note that you need to install Python 3.x (the latest 3.x version available), not 2.x.

You can proceed to the next segment only after you have installed Anaconda and the Jupyter Notebook.

Introduction to Jupyter Notebook

Welcome to the introductory segment on the Jupyter Notebook. You will be using the Jupyter IPython Notebook as the main environment for writing Python code throughout this course. Therefore, it is extremely important for you to understand the various functionalities in the Jupyter Notebook so that your coding experience going forward can be smooth. The main advantage of using the Jupyter Notebook is that you can write both code and normal text (using the Markdown format in Jupyter) in the notebooks. These notebooks are easy to read and share, and can also be used for presenting your work to others. Now let's take a look at some useful details regarding Jupyter Notebook.

Headings

- `#` for titles
- `##` for main headings
- `###` for subheadings
- `####` for smaller subheadings
- `#####` for subheadings in italic

Emphasis

- `__string__` or `**string**` for bold text
- `_string_` or `*string*` for italic text

Monospace fonts

- A single back quotation mark (`` ` ``) on both sides for monospace fonts

Line breaks

- `<br>` for a line break (Sometimes, the notebook does not give you the required line break where you want it)

Indenting

- > to indent the text
- >> for further indenting it, and so on

Bullets and numbering

- A single dash (`-`) followed by two spaces to create bullet points
- A number and a dot followed by a space (`1.`) to create numbered lists

Colouring

- `<font color = blue, yellow, red, pink, green, etc.> String </font>` to give your font the colour of your choice

LaTeX equations

- `$` on both sides of the text to write LaTeX equations

Python allows you to print anything you want through the `print()` function.

It is important that you are aware of the various shortcuts while using the Jupyter Notebook.

Command mode shortcuts

- Esc: To go into command mode
- Enter: To go back to edit mode
- M: To convert a cell into a markdown cell
- Y: To convert a cell back into a code cell
- A: To insert a new cell above
- B: To insert a new cell below
- D + D: To delete a cell
- Z: To undo the last operation
- F: To find word or phrases quickly
- Shift + Up/Down: To select multiple cells
- Space: To scroll notebook downwards
- Shift + Space: To scroll notebook upwards

Edit mode shortcuts

- Shift + Enter: To execute the code in the current cell and to go to the next cell

- Alt + Enter: To execute the code in the current cell and to insert a new cell below
- Shift + Tab: To get a brief documentation of the object that you have typed in the coding cell
- Ctrl + Shift + -: To split the cell at the cursor
- Shift + M: To merge the selected cells

The link to all the keyboard shortcuts for Mac users is provided below.

- Jupyter Notebook Mac shortcuts

Data Types in Python

Now that you know how to use the Jupyter Notebook and write the basic “hello, name” program in Python, let’s learn how to declare a variable in Python. You will also learn about the different data types available in Python.

Variables are nothing but a memory location that is used for storing the values assigned to them. Some of the properties of variables are as follows:

1. You **need not** declare the data type of the variable as required in other programming languages, such as C, C++ and JAVA, as shown below.

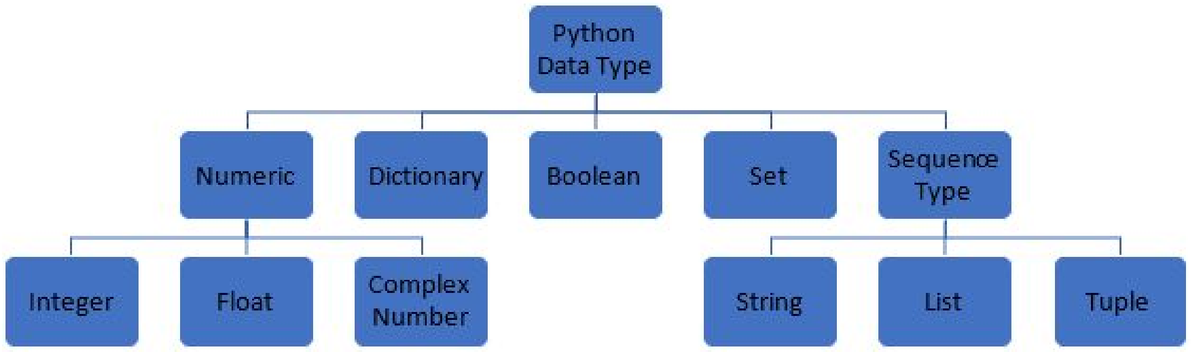
```
int c = 5
string name = 'Rahul'
```

2. The variable name (or identifier) cannot start with a number. For example, declaring 2name = 7 will throw an error.

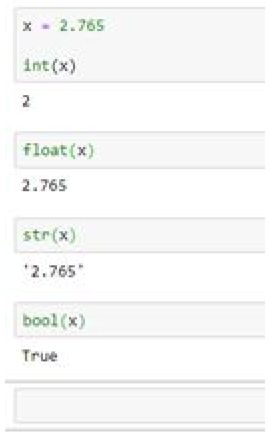
3. Python is case sensitive, or in other words, variables are case sensitive. For example, declaring name= 2 and Name = 4 will create two different variables.

You can also assign a user-defined value to a variable using the ‘**input**’ command with the appropriate prompt. For example, name = input (“Enter Your Name”).

You will learn about some of the common data types that are available in Python. You will also learn how to determine the type of a particular variable. In Python, data types can be classified into the following categories. You can check the data type of any variable using type(data).



In Python, you can change the data type of a variable through **typecasting.** In order to change the data type of a particular variable, you can simply call the following inbuilt methods in Python.



```
x = 2.765
int(x)
2
float(x)
2.765
str(x)
'2.765'
bool(x)
True
```

In the snapshot of THE code above, you are assigning a value to a variable (x) and using typecasting to convert the variable into different data types. For example, when you convert x into an integer, typecasting converts the floating-point value into an integer value.

Arithmetic Operations

Arithmetic operations are an integral part of every programming language. Some of the common Arithmetic operations are as follows:

Operations	**Results**
x + y	Sum of *x* and *y*
x - y	Difference between *x* and *y*
x * y	Product of *x* and *y*
x / y	Quotient of *x* and *y*
x // y	Floored quotient of *x* and *y*
x % y	Remainder of x / y
x ** y	*x* to the power *y*

Generally, in order to perform a mathematical operation involving one or more operators, you need to follow certain rules. Similarly, in Python, if you want to perform multiple operations in a single problem, you need to use the operator precedence rule.

Let’s understand the operator precedence rule using the example given below.

a = **4** + (**8** ** **2**) - **3** ** **2** % **1**

Here, to find the value of the variable 'a', you need to perform the following steps:

Step 1: You need to deal with the brackets as they hold the highest precedence over all the other operators in the given expression. The expression inside these brackets [(8**2)] will get executed first to return 64.

Updated expression: 4 + 64 − 3 ** 2 % 1

Step 2: You need to deal with the exponentiation operator [3**2] as it holds the next highest precedence over the other operators in the expression.

Updated expression: 4 + 64 − 9 % 1

Step 3: Now you need to deal with the remainder operator as it holds a higher precedence over subtraction and addition. This means that the value ‘9 % 1’ gets evaluated to return 0.

Updated expression: 4 + 64 − 0

Step 4: In this step, the addition operator gets executed as it holds higher precedence over subtraction.

Updated expression: 68 − 0

Step 5: In the final step, the subtraction operator gets executed.

Answer: 68

This is how the operator precedence rule plays an important part in performing arithmetic operations in Python.

String Operations

A string is perhaps one of the most important data types. Therefore, it is extremely important for you to learn how to work with strings in Python. Let’s learn about the various operations and manipulations that can be performed with a string.

Strings are a collection of characters. You can create a string constant inside a Python program by enclosing the text with single quotes (’), double quotes (") or a collection of three different types of quotes (’’’ or """).

Length

To determine the length of any string, you can use the **len()** function, as shown below.

```
string = ‘UpGrad’
len (string)
```

Here, it returns 6.

Indexing

Always remember that forward indexing starts with **0** and reverse indexing starts with **-1**. Strings are immutable, which means that they cannot be changed once they are created. Strings can be modified by slicing a part of it and concatenating these slices to the set.

```
string[0] : 'U'
string[2] : 'G'
string[-1] : 'd'
```

Immutability

You cannot change an element of a variable at a particular index. For example, string[1] = 'n' will throw an error. However, you can change the complete variable as many times as you want.

String concatenation

You can combine two strings by simply using '+', as shown below.

```
A = 'Up'
B ='Grad'
C = A + B
print (C)
```

Here, the output will be 'upGrad'.

You can concatenate an integer value to a string using **str()** function.

Slicing

In order to extract a particular part of the string, you can use the slicing functionality, as shown below.

```
string = "Have fun with examples"
string[5:8]  => 'fun'
string[:4] => 'Have'
string[14:] => 'examples'
```

Membership

You can use the following technique to find a given string or character in a string:

```
string = "Have fun with examples"
```

```
print('with' in string) => True
print('with' not in string) => False
```

Repetition

Arithmetic Multiplication operator '*' can also be used with string which replicates the string.

```
string = 'abcd'
string*4 => abcdabcdabcdabcd
```

Uppercase

You can convert an entire string into uppercase, as shown below.

```
string = 'upper case'
string.upper() => 'UPPER CASE'
```

Lowercase

Similarly, you can convert an entire string into lowercase, as shown below.

```
string = 'LOWER CASE'
string.lower() => 'lower case'
```

Strip

Strip removes all the white spaces, custom strings and characters from either side of the given string.

lstrip('x') removes multiple occurrences of x sequence from the left-hand side of the string.
rstrip('x') removes multiple occurrences of x sequence from the right-hand side of the string.
strip('x') remove multiple occurrences of x sequence from both sides of the string.

Count

This counts the number of occurrences of a particular character or string in a given string.

```
string = 'This is it'
string.count('i') => 3
```

You can also count the number of characters or string occurrences after a particular index.

string.count(‘i’,5) => 2

String format

This is used for string formatting, as shown below.

A = ’Data’
B = ’Analysis’
C = ’Pandas’
‘{0} {1} using {2}’.format(A,B,C) = > 'Data Analysis using Pandas'

Split

This splits a string function into an array, splitting the function based on the given character or string.

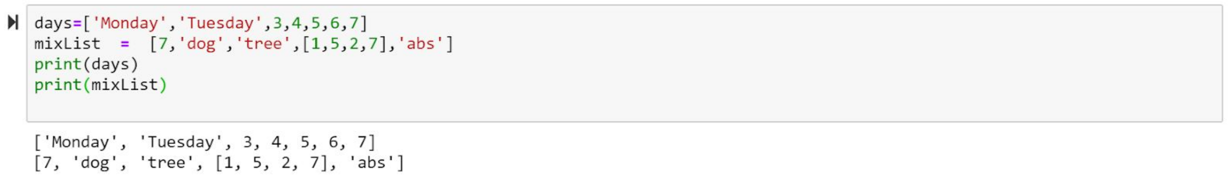
string = ‘A B C D’
string.split(‘ ‘) => [‘A’,’B’,’C’,’D’]

Help function

help(str) shows you all the available functions that can be applied on a string.

Lists

A list is a collection of values. It may contain different types of values. Lists are **mutable**, which means that elements of a list can be changed. It can be defined as follows:



```
days=['Monday','Tuesday',3,4,5,6,7]
mixList  =  [7,'dog','tree',[1,5,2,7],'abs']
print(days)
print(mixList)
```

```
['Monday', 'Tuesday', 3, 4, 5, 6, 7]
[7, 'dog', 'tree', [1, 5, 2, 7], 'abs']
```

The operations that can be performed on lists are as follows:

- Slicing a list: Split the list in parts.

```
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
sliced_nums = nums[2:7]
print(sliced_nums)
```

```
[30, 40, 50, 60, 70]
```

Similar to indexing for a string, a list starts with 0, and negative indexing means starting from the last element of the list.

- Len(): The length of a list is an inbuilt function.

```
print(len(nums))
```

```
9
```

- A list is mutable, that is, one can reassign elements in it.

```
num = [10, 20, 30, 40, 50, 60, 70, 80, 90]
num[0] = 0
print(num )
```

```
[0, 20, 30, 40, 50, 60, 70, 80, 90]
```

- Accessing elements of a list means that to loop over the list, you can use the **for** loop.

```
num = [1,2,5,6,8]
for n in num:
    print(n)
```

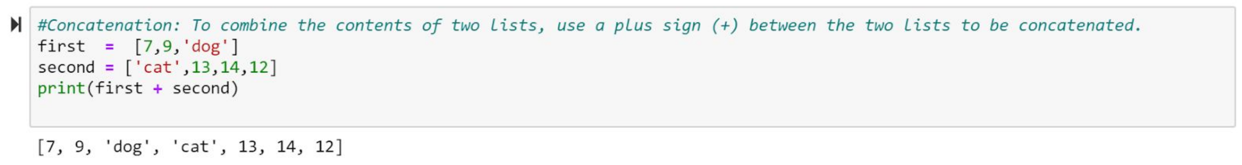
```
1
2
5
6
8
```

- Multidimensional lists: A list may have more than one dimension.

```
#Multidimensional Lists- A list may have more than one dimension.
a=[[1,2,3],[4,5,6]]
print(a)
nestlist  =  [1,2,[10,20,30,[7,9,11,[100,200,300]]],[1,7,8]]
print(nestlist)
```

```
[[1, 2, 3], [4, 5, 6]]
[1, 2, [10, 20, 30, [7, 9, 11, [100, 200, 300]]], [1, 7, 8]]
```

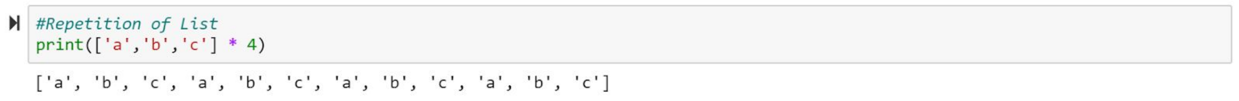
- Concatenation: A plus sign (+) should be used between two lists to concatenate them.



```
#Concatenation: To combine the contents of two lists, use a plus sign (+) between the two lists to be concatenated.
first  =  [7,9,'dog']
second = ['cat',13,14,12]
print(first + second)
```

```
[7, 9, 'dog', 'cat', 13, 14, 12]
```

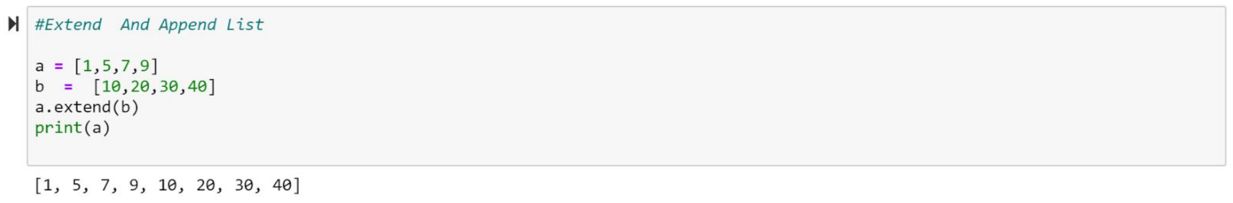
 Repetition of a list



```
#Repetition of List
print(['a','b','c'] * 4)
```

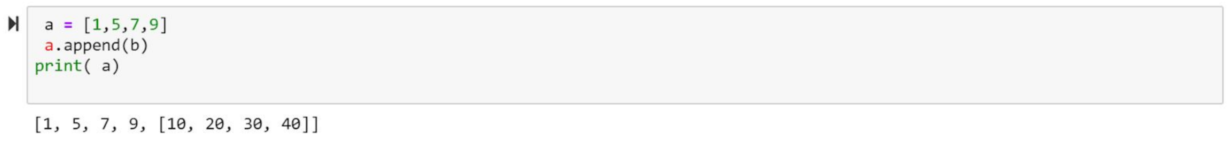
```
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

 Extend append list
The extend() function works only on lists. However, you can append() an integer to the list.



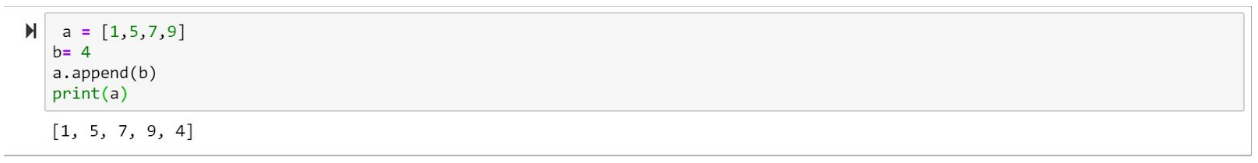
```
#Extend  And Append List

a = [1,5,7,9]
b  =  [10,20,30,40]
a.extend(b)
print(a)
```

```
[1, 5, 7, 9, 10, 20, 30, 40]
```



```
 a = [1,5,7,9]
 a.append(b)
print( a)
```

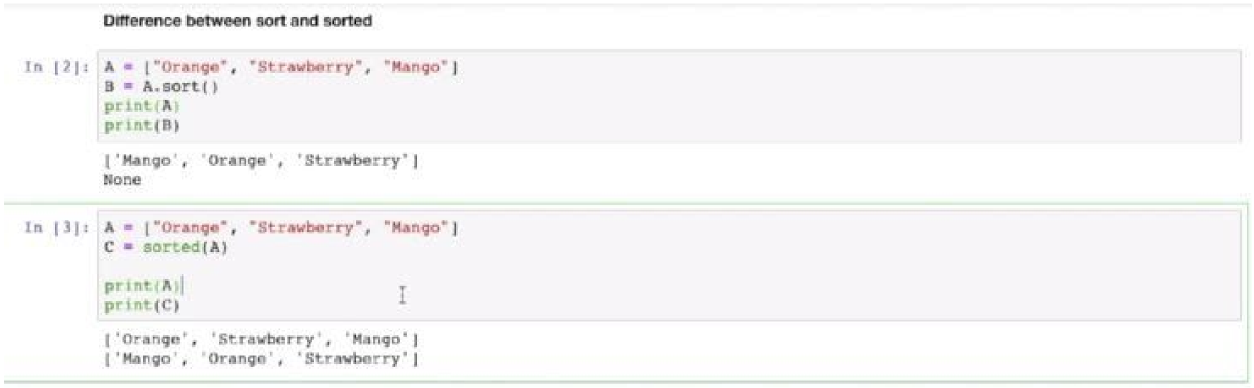
```
[1, 5, 7, 9, [10, 20, 30, 40]]
```



```
 a = [1,5,7,9]
b= 4
a.append(b)
print(a)
```

```
[1, 5, 7, 9, 4]
```

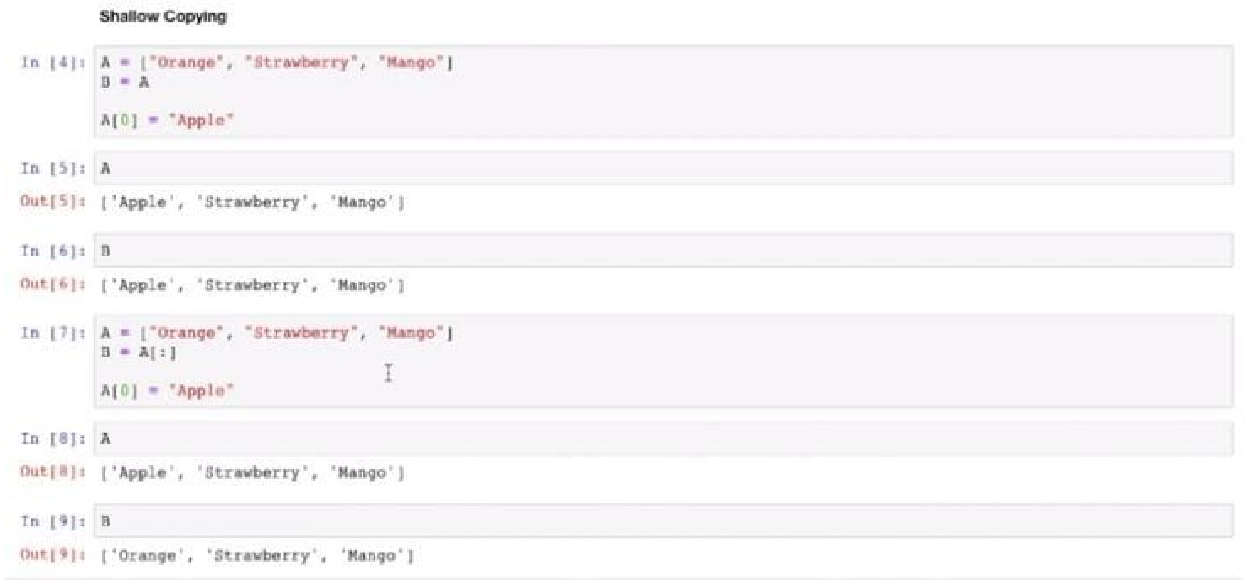
 The difference between sort and sorted is given below:


Difference between sort and sorted

```
In [2]: A = ["Orange", "Strawberry", "Mango"]
        B = A.sort()
        print(A)
        print(B)

        ['Mango', 'Orange', 'Strawberry']
        None

In [3]: A = ["Orange", "Strawberry", "Mango"]
        C = sorted(A)

        print(A)
        print(C)

        ['Orange', 'Strawberry', 'Mango']
        ['Mango', 'Orange', 'Strawberry']
```

The function ‘A.sort()’ does not return a separate sorted list to B, but the function ‘sorted(A)’ returns a separate sorted list to the variable C.

- Shallow copying
  It constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.


Shallow Copying

```
In [4]: A = ["Orange", "Strawberry", "Mango"]
        B = A

        A[0] = "Apple"

In [5]: A

Out[5]: ['Apple', 'Strawberry', 'Mango']

In [6]: B

Out[6]: ['Apple', 'Strawberry', 'Mango']

In [7]: A = ["Orange", "Strawberry", "Mango"]
        B = A[:]

        A[0] = "Apple"

In [8]: A

Out[8]: ['Apple', 'Strawberry', 'Mango']

In [9]: B

Out[9]: ['Orange', 'Strawberry', 'Mango']
```

By assigning B = A, you refer to the same list object in the memory, and changes that are made to list A will be reflected in list B as well. With A[:], you create a new object, which is assigned to B; here, any changes that are made in list A will not affect list B.

Tuples

Tuples are an ordered sequence of mixed data types. They are written as comma-separated elements within parenthesis. An important feature of a tuple is immutability, which means that the elements of a tuple **cannot** be altered. It is used to store data such as employee information, which is not allowed to be changed.

```
####Tuples
## A tuple is like a list. It differs from the list in using parentheses instead of square brackets.
```

```
fruit=('Apple','Banana','Orange')
print(fruit)
```

```
('Apple', 'Banana', 'Orange')
```

The operation that can be performed on tuples are as follows:

- Accessing and slicing a tuple: It is the same as that for a list.

```
fruit=('Apple','Banana','Orange')
print(fruit[1])
```

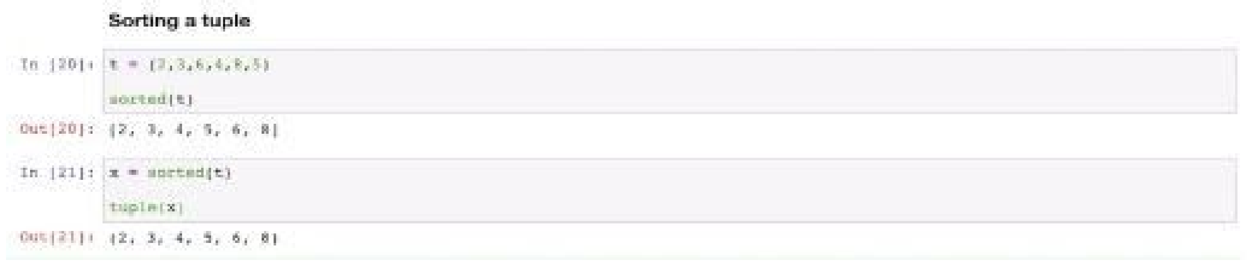
```
Banana
```

```
print(fruit[0:2])
```

```
('Apple', 'Banana')
```

Tuples are ordered sequences, which means that the order in which elements are inserted remains the same. This makes them flexible for indexing and slicing, similar to strings. In indexing of strings, each character is assigned an index; similarly, each element is assigned an **index** here.

Tuples can be sorted using the following command:



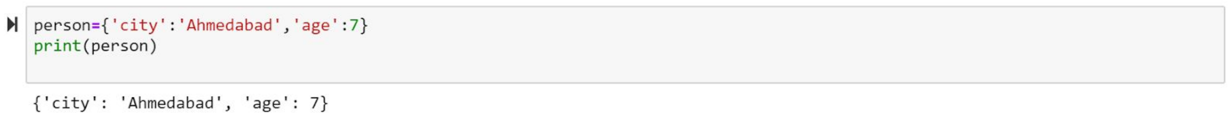
Sorting a tuple

```
In [20]: t = (2,3,6,4,8,5)

         sorted(t)
Out[20]: [2, 3, 4, 5, 6, 8]

In [21]: x = sorted(t)

         tuple(x)
Out[21]: (2, 3, 4, 5, 6, 8)
```

The Python Tuple is immutable. Once declared, you cannot change its size or elements.
fruit [2]=’Mango’
Traceback (most recent call last):
File “<pyshell#107>”, line 1, in <module>
fruit [2]=’Mango’
TypeError: ‘tuple’ object does not support item assignment

Dictionaries

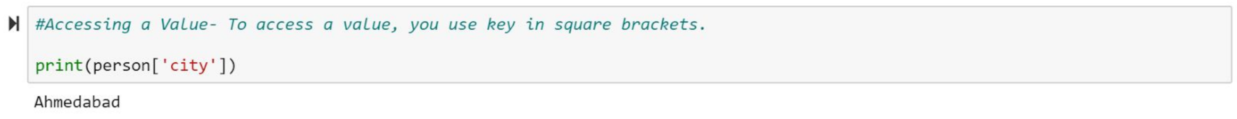
A dictionary is a key-value pair. It can be defined as follows:



```
person={'city':'Ahmedabad','age':7}
print(person)
```

```
{'city': 'Ahmedabad', 'age': 7}
```

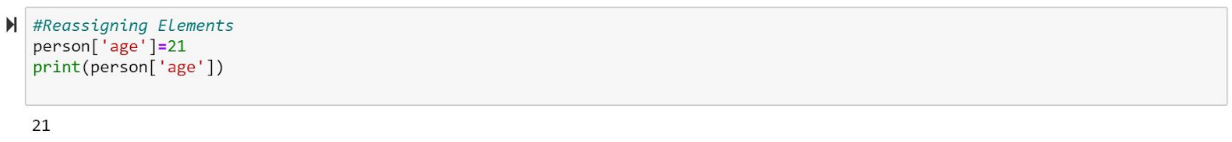
- Accessing a value: To access a value, use the key in square brackets.



```
#Accessing a Value- To access a value, you use key in square brackets.

print(person['city'])
```

```
Ahmedabad
```

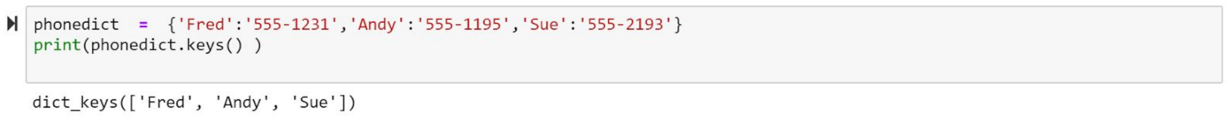
- Reassigning elements



```
#Reassigning Elements
person['age']=21
print(person['age'])
```

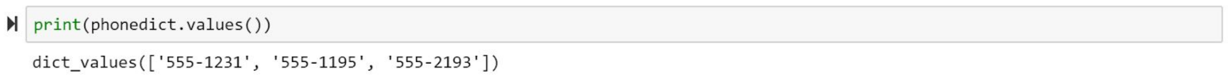
```
21
```

- Accessing all the keys and values in a list:



```
phonedict  =  {'Fred':'555-1231','Andy':'555-1195','Sue':'555-2193'}
print(phonedict.keys() )
```

```
dict_keys(['Fred', 'Andy', 'Sue'])
```



```
print(phonedict.values())
```

```
dict_values(['555-1231', '555-1195', '555-2193'])
```

Boolean

A Boolean can take true or false values.

```
a=2>1
print(a)
```

```
True
```

Sets

A set can have a list of values that are defined using {}.

```
a={1,2,3}
print(a)
```

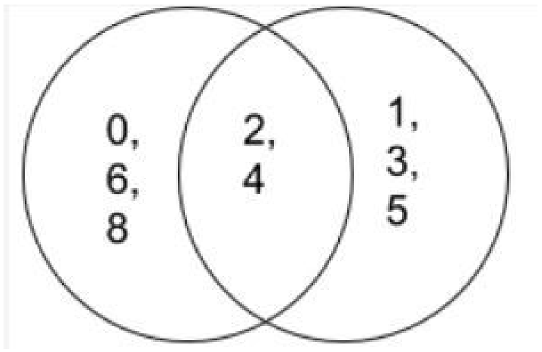
```
{1, 2, 3}
```

- The add() and remove() functions can be used for adding and removing elements, respectively. Every set element is unique without duplicates and must be immutable, that is, cannot be changed. However, a set itself is mutable.

```
#add() and remove() for adding and removing element. Set is also immutable.

a={1,2,3,4}
print(a)
a.remove(4)
print(a)
a.add(5)
print(a)
```

```
{1, 2, 3, 4}
{1, 2, 3}
{1, 2, 3, 5}
```

Suppose you have the following two sets: A = {0,2,4,6,8} and B = {1,2,3,4,5}.



- A union represents the total unique elements in both sets.

  A.union(B) → {0, 1, 2, 3, 4, 5, 6, 8}

- An intersection represents elements that are common to both sets.

  A.intersection(B) → {2, 4}

- Difference(A-B) represents elements that are present in A and not in B.

  A.difference(B) → {0, 6, 8}

- A symmetric difference represents the union of the elements A and B minus the intersection of A and B.

  A^B → {0, 6, 8, 1, 3, 5}

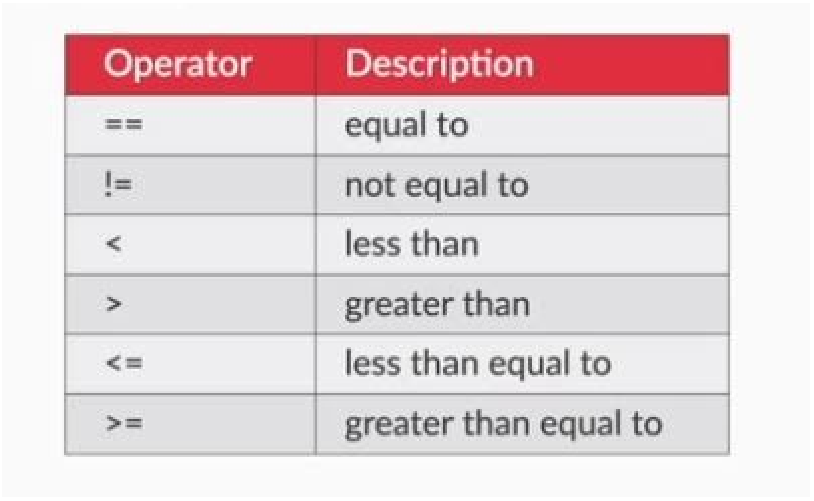
Binary Operators

The arithmetic operators are presented in the table given below:

Operation	Result	Notes	Full documentation
x + y	Sum of *x* and *y*		
x - y	Difference of *x* and *y*		
x * y	Product of *x* and *y*		
x / y	Quotient of *x* and *y*		
x // y	Floored quotient of *x* and *y*	(1)	
x % y	Remainder of x / y	(2)	
-x	*x* negated		
+x	*x* unchanged		
abs(x)	Absolute value or magnitude of *x*		abs()
int(x)	*x* converted to integer	(3)(6)	int()
float(x)	*x* converted to floating point	(4)(6)	float()
complex(re, im)	A complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	(6)	complex()
c.conjugate()	Conjugate of the complex number *c*		
divmod(x, y)	The pair (x // y, x % y)	(2)	divmod()
pow(x, y)	*x* to the power *y*	(5)	pow()
x ** y	*x* to the power *y*	(5)	

Integer division: The result value is a whole integer. The result is always rounded towards minus infinity.
1//2 is 0,
(-1)//2 is -1,
1//(-2) is -1 and
(-1)//(-2) is 0.
Python defines pow(0, 0) and 0 ** 0 to be 1, as is common for programming languages.

Relational operators:
1. A type of binary operators
2. Used to compare values
3. Return a boolean value
4. Include the following :



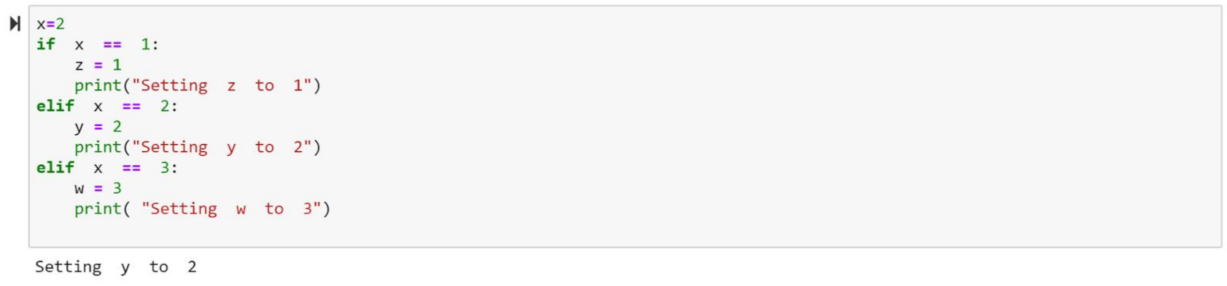
Operator	Description
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than equal to
>=	greater than equal to

If-Else Statements

The basic form of the statement can be given as follows:

```
if expression:
        statement(s)
elif expression:
         statement(s)
elif expression:
        statement(s)
. . .
else:
```

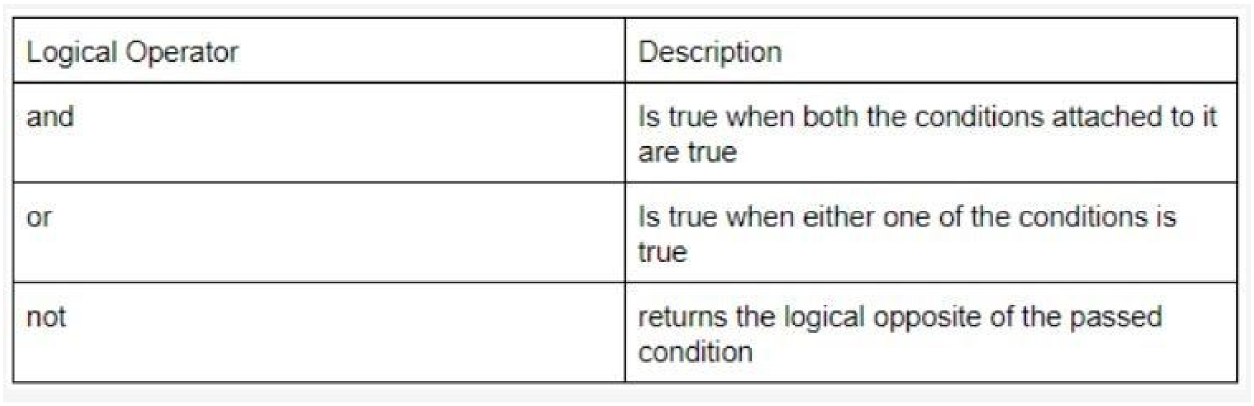
statements



```
x=2
if  x  ==  1:
    z = 1
    print("Setting  z  to  1")
elif  x  ==  2:
    y = 2
    print("Setting  y  to  2")
elif  x  ==  3:
    w = 3
    print( "Setting  w  to  3")
```

```
Setting  y  to  2
```

The description of logical operators that are used in the if-else statements is presented in the table given below:



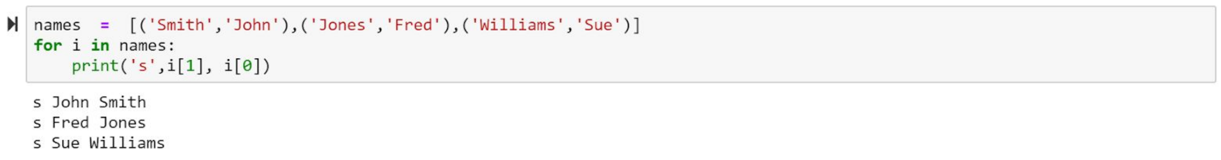
Logical Operator	Description
and	Is true when both the conditions attached to it are true
or	Is true when either one of the conditions is true
not	returns the logical opposite of the passed condition

Loops

for loops

The basic form of the for loop is as follows:

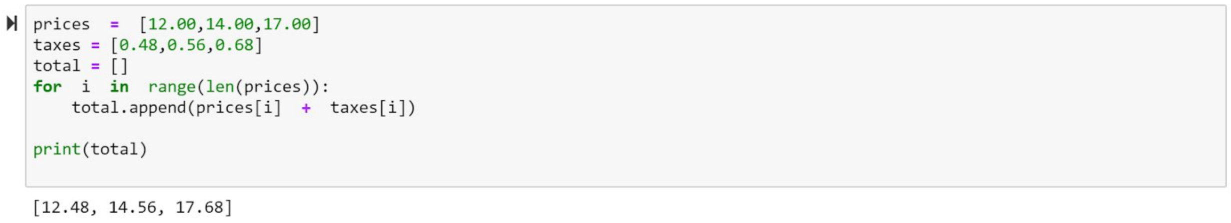
```
for var in sequence:
        statements
```



```
names  =  [('Smith','John'),('Jones','Fred'),('Williams','Sue')]
for i in names:
    print('s',i[1], i[0])
```

```
s John Smith
s Fred Jones
s Sue Williams
```

- for loops and range functions: The range function accepts one, two or three arguments. With a single integer argument, the range returns a sequence of integers from 0 to one less than the argument provided. With two arguments, the first argument is used as a starting value instead of 0, and with three arguments, the third argument is used as an increment instead of the implicit default of 1.



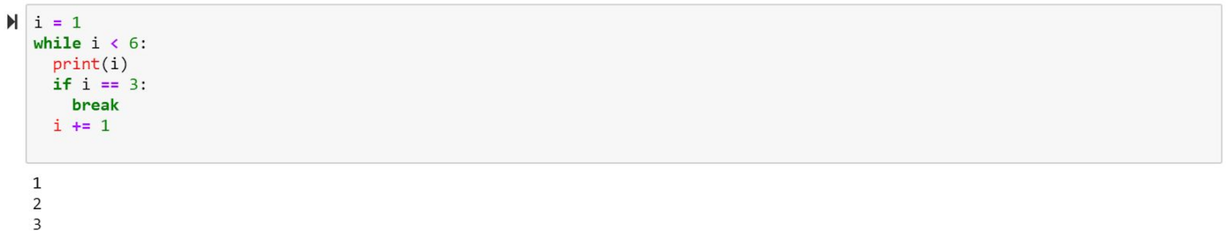
```
prices  =  [12.00,14.00,17.00]
taxes = [0.48,0.56,0.68]
total = []
for  i  in  range(len(prices)):
    total.append(prices[i]  +  taxes[i])

print(total)
```

```
[12.48, 14.56, 17.68]
```

while loops

The basic syntax of the while loop is as follows:

```
while expression:
        statements
else:
        statements
```



```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
3
```

Map, Filter and Reduce

Map

Map applies a function to all the items in an input list. The basic syntax of the map function is as follows:

```
map(function_to_apply, list_of_inputs)
```

You can pass the list elements to a function one by one and then collect the output. Take a look at the following example:

```python
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

```
[2, 4, 6, 8]
```

Filter

The filter() method filters the given sequence through a function that tests whether each element in the sequence is true or not.

```python
# a list contains both even and odd numbers.
seq = [0, 1, 2, 3, 5, 8, 13]

# result contains odd numbers of the list
result = filter(lambda x: x % 2 != 0, seq)
print(list(result))

# result contains even numbers of the list
result = filter(lambda x: x % 2 == 0, seq)
print(list(result))
```

```
[1, 3, 5, 13]
[0, 2, 8]
```

Reduce

The reduce() function accepts a function and a sequence and returns a single value that is calculated through the following process:

1. Initially, the function is called with the first two items from the sequence, and the result is returned.
2. The function is called again, with the result obtained in step 1 and the next value in the sequence. This process continues to repeat until there are items in the sequence.

```python
from functools import reduce

def do_sum(x1, x2):
    return x1 + x2

print(reduce(do_sum, [1, 2, 3, 4]))
```

```
10
```

Functions

A function is a block of organised, reusable code that is used to perform a single, related action. Functions provide better modularity for the application and a high degree of code reuse.
Users can create their own functions called user-defined functions.

 Defining and calling a function

```
def newFunction():
    print("This is Python function call")

newFunction ()
```

```
This is Python function call
```

Calling a function with an incorrect number of arguments will cause Python to throw an error.

```
def printme( str ):
  "This prints a passed string into this function"
  print str
  return;

# Now you can call the printme function
printme()
```

When the previous code is executed, it produces the following result:

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
   printme();
TypeError: printme() takes exactly 1 argument (0 given).
```

Default arguments can be set in a function as follows: If default values are provided for an argument, the number of arguments can vary. Take a look at the following example.

```
def newFunction(name, age = 35):
    print("name" ,name)
    print("Age" ,age)
newFunction("First",50)
newFunction("Second")
```

```
name First
Age 50
name Second
Age 35
```

Variable length arguments: If the number of arguments is not fixed, then you can use the variable argument syntax in the function definition. If an asterix (*) is used, it is interpreted as any number of arguments.

```
def newFunction(first, *varValue):
    print("“Arguments are  :“")
    for x in varValue:
        print(x)
newFunction(1)
newFunction(1,2,3,4,5)
```

```
“Arguments are  :“
“Arguments are  :“
2
3
4
```

Lambda

A lambda function is a small anonymous function, which can take any number of arguments. However, it can only have one expression.

```
x = lambda a : a + 10
print(x(5))
```

```
15
```

A lambda function can call user-defined functions on the given input.

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

OOP in Python

Introduction

In this session, you will understand some of the most important concepts of OOPs in the field of Data Science. An object-oriented programming method enables you to think like you would with real-life entities or objects. Here, you will learn about classes, objects and its methods along with inheritance and overriding. An object-oriented programming is a world in itself, but for now, you will be exploring only as much as required.

Class and Objects

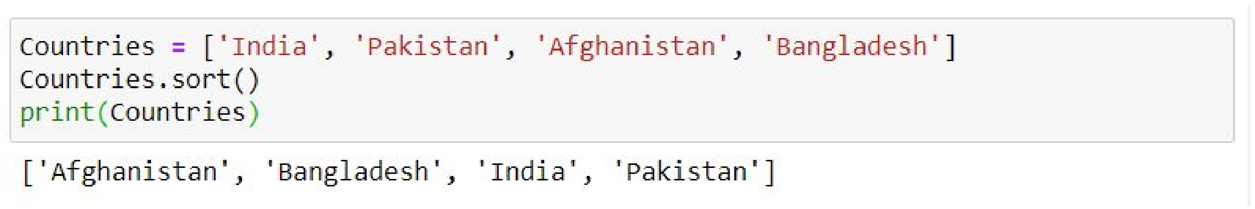
Python has many different data types such as string, int, dictionary, etc. Everything is an object in Python. Every object has the following three characteristics:

1. Type
2. Internal representation
3. Methods

An object is an instance of a particular type. For example, every time you create an integer you are creating an instance of the integer data type. For example,

    a, b, c = 2, 3, 4

Here, we are creating three instances of the integer data type or integer objects. Similarly, every time you create a list, you are creating an instance of a list or a list object.
Methods are instances of the class or data type that is provided to interact with objects. We have been using methods such as sort in a class list. Methods can be called by adding a dot **'.'** at the end of the object's name.
The code snippet for a list object 'Countries' calling sort() methods is as follows:

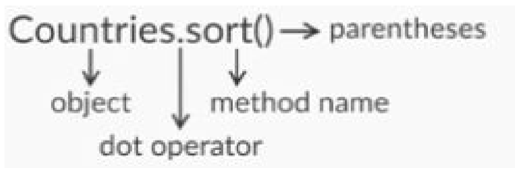


```
Countries = ['India', 'Pakistan', 'Afghanistan', 'Bangladesh']
Countries.sort()
print(Countries)
```

```
['Afghanistan', 'Bangladesh', 'India', 'Pakistan']
```

The characteristics of the above list object are as follows:

1. List object, in the code snippet given above it is 'Countries'
2. Sort method, it changes the data within the object. You can see that the structure of calling method 'sort()' for object 'Countries' is as follows:



3. The method changes the state of the object, 'Countries'

We do not know the inner workings of a class and its methods such as list, dictionary or any other pre-defined object, we just know how to use them.

You can create your own type of class with data attributes and methods. You can also create instances of this self-defined object. The **class** keyword is used for defining a class, and in the __init__ method, you can initialise the attributes that define your class. You can create the ‘Employee’ class with three attributes, age, name and employee id. In this case, name, age and employee id are the attributes that define your ‘Employee’ class, and using the **self** keyword, you can define these arguments inside the __init__ method, as shown below.

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

It is extremely important for you to understand the init method because this method is instantiated automatically when a particular class is being used; this method also determines the number of values to be passed. You can now create an object in this class by just passing the details of ‘Employee’ as arguments, as shown below.

```
E1= Employee(24,'Ravi',101)
type(E1)
```

```
__main__.Employee
```

Here, E1 = Employee(24,'Ravi',101) → This would create E1 with age = 24, name = Ravi and eid = 101. The object E1 is nothing but an instance of the class Employee. When you try to apply the type function on E1, it will return the class to which it belongs.

You can also create certain attributes that are common to all instances of the class.

So, let’s add a class variable called ‘company code’ to your ‘Employee’ class as follows:

```
class Employee :
    company_code = "EMZ"
    def __init__(self,age, name,eid):
            self.age = age
            self.name = name
            self.eid = eid
```

This would make the company code a common property among all the employees. After creating an ‘Employee’ instance, the attribute ‘company code’ and its value is automatically assigned to the employee as shown below.

```
E1 = Employee(24, 'Ravi', 101)
print(E1.company_code)
```

```
EMZ
```

You cannot simply use E1.company_code = 'XYZ’ to change the company_code. This would change the company_code of the employee E1; however, since the company code applies to all employees, you need to write the following:

```
Employee.company_code ='XYZ'
print(E1.company_code)
```

```
XYZ
```

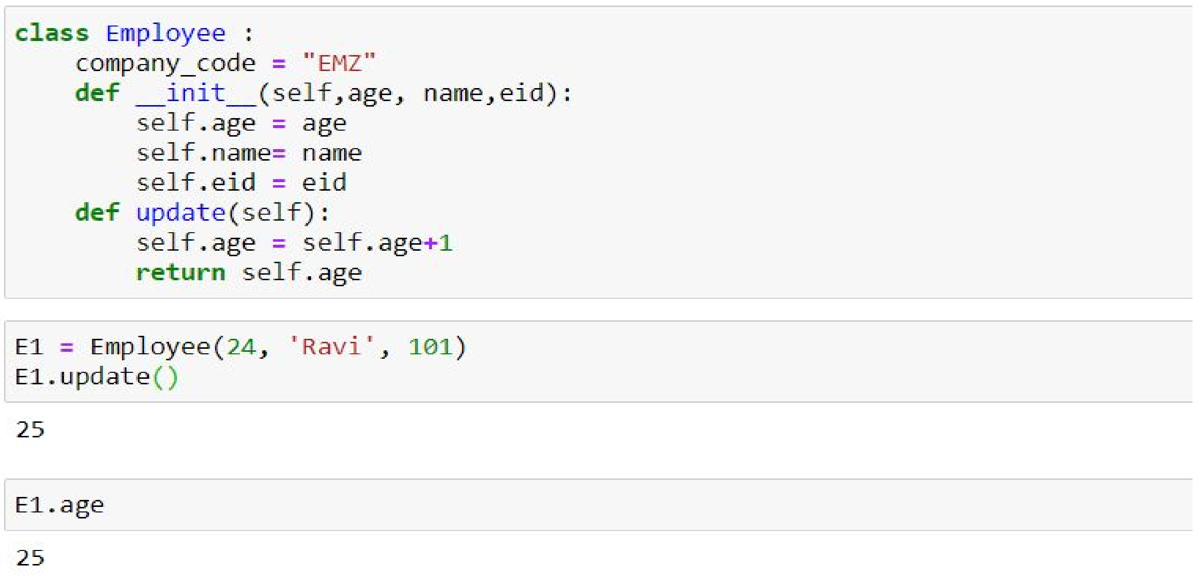
Now, each instance of an employee will have the same value for class variable, i.e., company_code.

Note: Instance variables are also called data attributes.

Methods

You used many in-built methods in the case of lists, tuples or some other data structures. Essentially, methods are functions that are responsible for implementing a certain functionality when they are used in code.

Methods are functions that interact with and change data attributes. The main characteristic of a method is that it is called by using the object name with the dot operator. To create your own method in the Employee class, you can implement an update method to increase the age of an employee by 1.



```
class Employee :
    company_code = "EMZ"
    def __init__(self,age, name,eid):
        self.age = age
        self.name= name
        self.eid = eid
    def update(self):
        self.age = self.age+1
        return self.age
```

```
E1 = Employee(24, 'Ravi', 101)
E1.update()
```

```
25
```

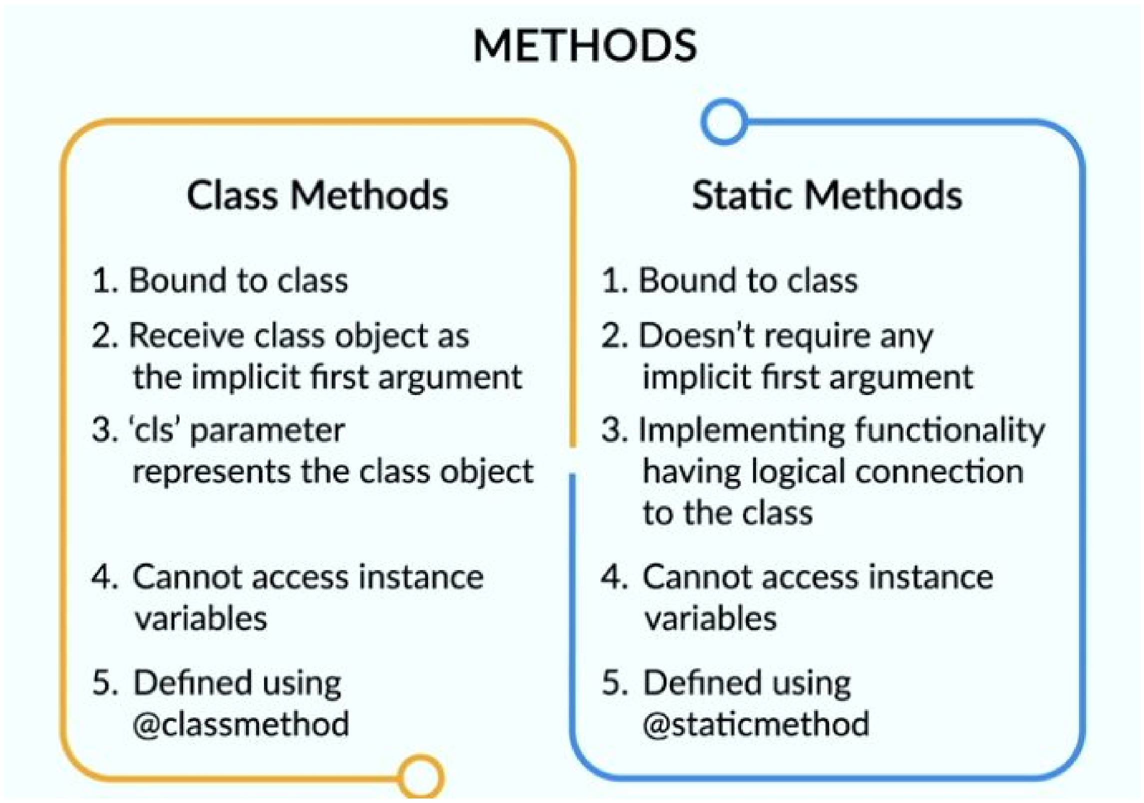
```
E1.age
```

```
25
```

In the execution, you can observe that the E1 employee object is created, and on calling the update method, it returns the updated age and also updates the age of the employee. These methods are called on instances of the class and, therefore, known as Instance methods.

You can write a similar function to update the company code as well; however, there would be a critical flaw if you did so because handling class variables should not be present within an ordinary method that can be accessed/changed by any instance object. For this, you use other methods called class methods.

The differences between Class Methods and Static Methods are summarised below.

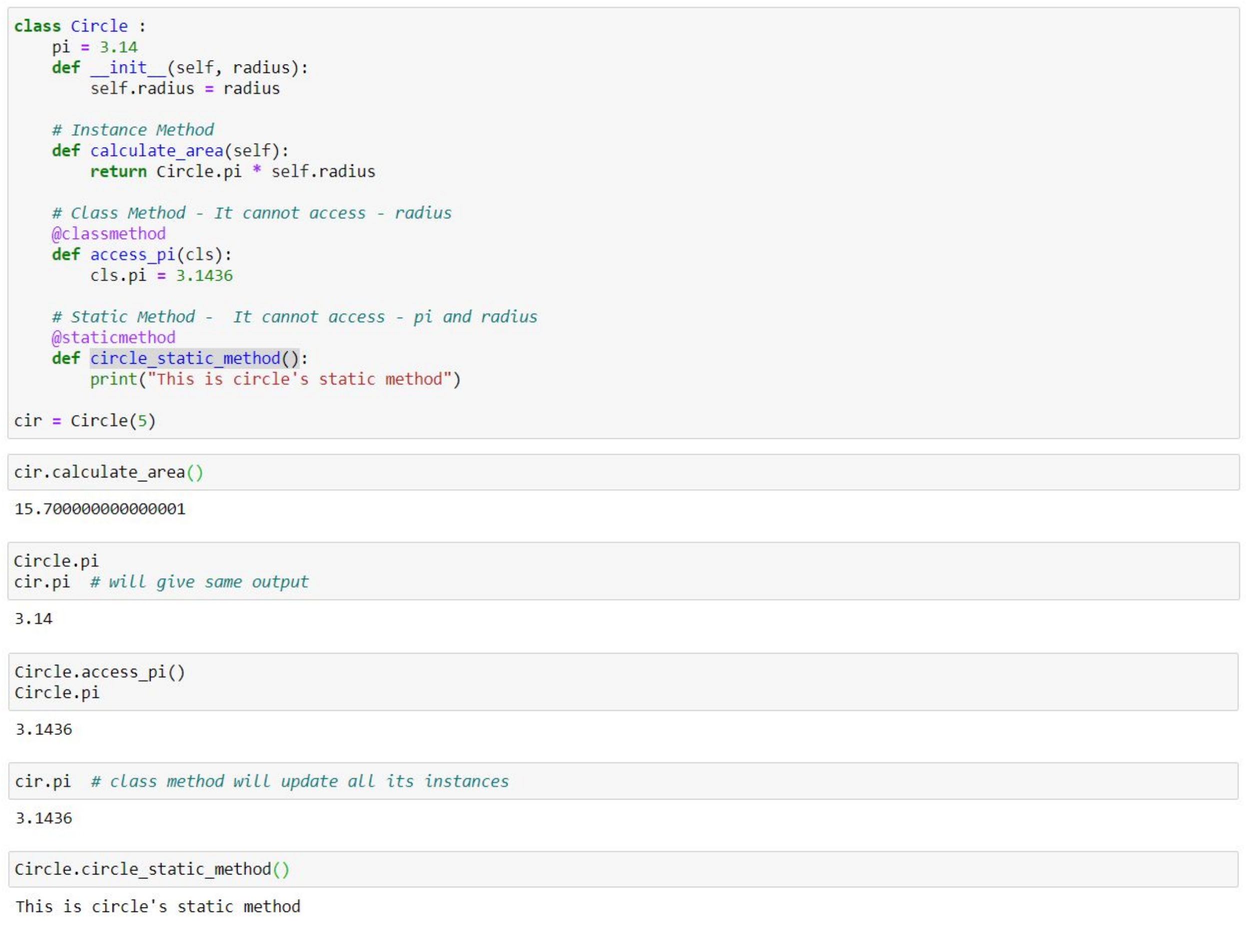

METHODS

Class Methods	Static Methods
1. Bound to class	1. Bound to class
2. Receive class object as the implicit first argument	2. Doesn't require any implicit first argument
3. 'cls' parameter represents the class object	3. Implementing functionality having logical connection to the class
4. Cannot access instance variables	4. Cannot access instance variables
5. Defined using @classmethod	5. Defined using @staticmethod

Class and statics methods are bound to a class but not to an instance of the class, and they cannot access instance variables.

A class method is defined using a class method decorator (@classmethod) and takes a class parameter (cls) that has access to the state of the class. In other words, the changes made using the class method would apply to all the instances of the class.

The code snippet given below shows 'Circle' object and different types of methods used:



```
class Circle :
    pi = 3.14
    def __init__(self, radius):
        self.radius = radius

    # Instance Method
    def calculate_area(self):
        return Circle.pi * self.radius

    # Class Method - It cannot access - radius
    @classmethod
    def access_pi(cls):
        cls.pi = 3.1436

    # Static Method -  It cannot access - pi and radius
    @staticmethod
    def circle_static_method():
        print("This is circle's static method")

cir = Circle(5)
```

```
cir.calculate_area()
```

```
15.700000000000001
```

```
Circle.pi
cir.pi  # will give same output
```

```
3.14
```

```
Circle.access_pi()
Circle.pi
```

```
3.1436
```

```
cir.pi  # class method will update all its instances
```

```
3.1436
```

```
Circle.circle_static_method()
```

```
This is circle's static method
```

Here, the class method will reflect the change in all of its instances.

The basic objective of defining a static method is to identify certain functionalities of a class and encapsulate them together inside one object or one class. The static method does not take any parameter, instance or class object. It belongs to a class only and must be called by using the class name. In order to access any attribute inside class methods, you need to call it by the class name. You cannot access instance variables or instance methods inside a static method.

So, depending on the application or requirement, you can choose the type of the method that best suits your scenario.

Method vs Functions

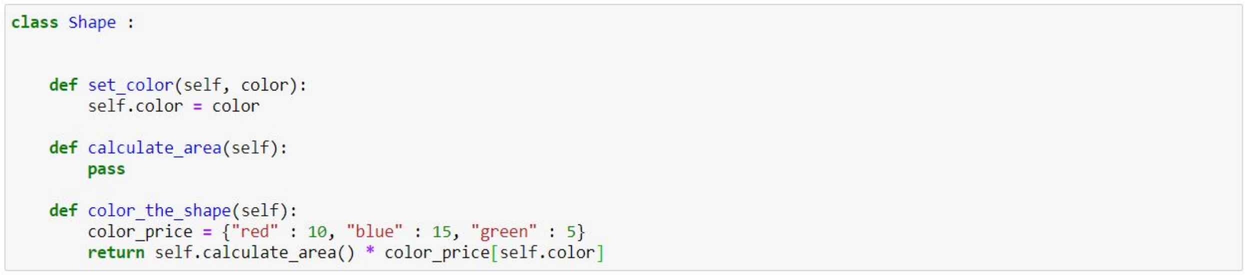
1. Python functions are called generically, whereas methods are called on an object because they can access the data within it.

2. A 'method' may alter an object’s state, but a Python 'function' usually only operates on an object first and then returns a value.

Class Inheritance and Overriding

Inheritance, as the term suggests, means ‘to receive something’. In Python, inheritance has a similar meaning; it means that when class A is defined on class B, class A has all the properties of class B. So, you need not write the code again. Inheritance is transitive in nature, which means that if class B inherits properties from class A, then all subclasses of class B will also inherit the properties like attributes and methods of class A.

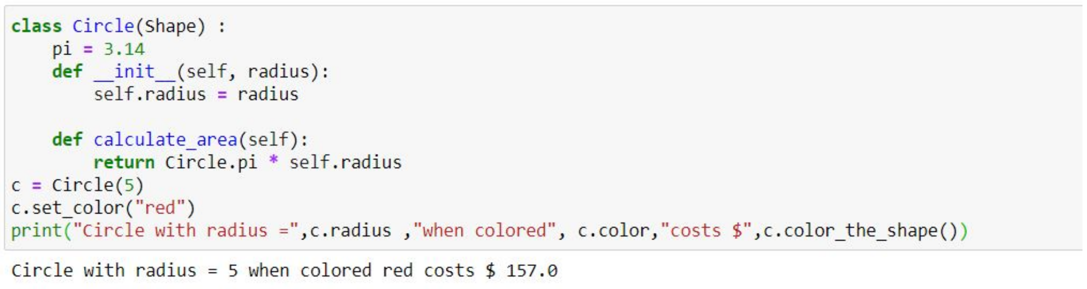
Inheritance enables code reusability. Just like a child inherits their parents' qualities, a class that inherits properties is known as a child class, and the class from which the child class inherits the properties is known as the parent class. Let's take a look at an example to understand this better.



```
class Shape :


    def set_color(self, color):
        self.color = color

    def calculate_area(self):
        pass

    def color_the_shape(self):
        color_price = {"red" : 10, "blue" : 15, "green" : 5}
        return self.calculate_area() * color_price[self.color]
```

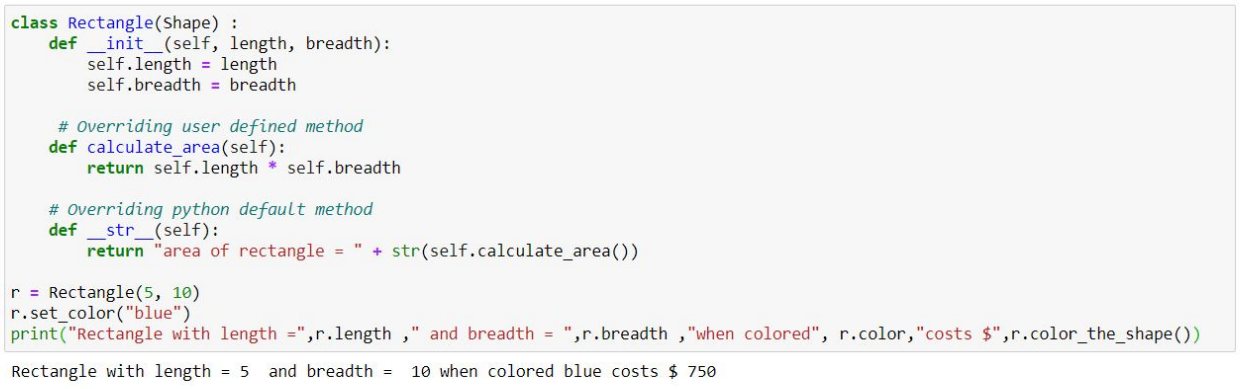
In the code given above, we have used the keyword ‘**pass**’, which means that the method is associated with a class, but it is abstract in nature. We cannot use calculate_area() for ‘shape’ because it does not have any dimensions. So, we are creating a method that will be inherited by all the subclasses of Shape.

To define inheritance or relationship, we need mention the parent class in parentheses and define the calculate_area() method as follows:


```
class Circle(Shape) :
    pi = 3.14
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return Circle.pi * self.radius
c = Circle(5)
c.set_color("red")
print("Circle with radius =",c.radius ,"when colored", c.color,"costs $",c.color_the_shape())
```

```
Circle with radius = 5 when colored red costs $ 157.0
```

This technique of specifying a more specific version of a method in a child class is called Overriding. In Python, you can enable method overriding by simply defining a method in the child class using the same name as that of the method in the parent class. When you define a method in the object, you can make the object satisfy that method call so that the implementations of its ancestors are not applied. In a child class, calculate_area() is an override method.


```
class Rectangle(Shape) :
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

     # Overriding user defined method
    def calculate_area(self):
        return self.length * self.breadth

    # Overriding python default method
    def __str__(self):
        return "area of rectangle = " + str(self.calculate_area())

r = Rectangle(5, 10)
r.set_color("blue")
print("Rectangle with length =",r.length ," and breadth = ",r.breadth ,"when colored", r.color,"costs $",r.color_the_shape())
```

```
Rectangle with length = 5  and breadth =  10 when colored blue costs $ 750
```

In the example shown above, the rectangle and circle inherit the properties of the parent class shape, and because of this parent-child relationship, you do not need to define set_colour and colour_the_shape again in the circle and rectangle classes.

One more thing to observe here is the method ‘calculate area’. Since it would be unique to different classes, it is initiated in the parent class, and in the child class, this method is defined as per the child class functionality. This is called overriding.

Summary

So, in the first session of the module, you wrote your first program in Python and learnt about the different **data types** that are supported by Python. You learnt about the various **arithmetic** and **string** operations using the python programming language .

In the second session, you understood the concepts of different **data structures** supported by Python, including **Lists**, **Tuples**, **Sets** and **Dictionary**.

In the third session, you learnt about **control structures** and **functional programming** in Python. You also learnt how **decision-making** statements and **loops** play an essential role in execution of statements.

Finally, you learnt about **classes**, **objects, methods** and various other OOP methodologies. You learnt what classes and objects are, how these two are related and how to implement methods. Next, you learnt about one of the most crucial object-oriented programming methodology called '**inheritance**', how Python supports inheritance and how you can override it using a method.

You created a class using the '**class**' keyword, a constructor and the '**self**' keyword. The 'self' keyword plays a significant key role in linking the arguments and class while creating an object to the attributes defined in the class. Next, you learnt about the following three types of method creations:

1. Instance methods: This method accepts both instance and class variables.
2. Class methods: This method is defined using **@classmethod** and accepts the state of the class as argument(cls). The class method has no access to instance variables.
3. Static methods: This method does not have access to either instance or class variables. It is defined using **@staticmethod**.

Finally, you understood the concepts of inheritance and overriding, which play an important role in code reusability. Inheritance enables users to define a class that can take all the functionalities from the parent class and allows you to add more functionalities to it. Overriding helps in updating or changing the method defined in a parent class.