Introduction to Python

Class 1

Python is a high level, interpreted language for general purpose computer programming. The language has been designed to provide emphasis on code readability. Python was developed by Guido van Rossum in 1989.

Python Interpreter can be used in two modes:

1) Interactive mode: Entering the code line by line, the output of each being shown immediately.
2) Argument passing: Storing all the code in a .py file(called a python script) and then passing the name of the script to the interpreter.

Note: We will be dealing with Python3 here, as Python2 has started becoming obsolete and it's usage has reduced. So when we say python anywhere, we are referring to Python3.

Creating variables :

In python, we do not need to declare variables as such. We don't need to mention their datatypes at all. We can directly give them values, like the following example. Here, the variable a will store 3.

```
a = 3
```

However, this doesn't mean datatypes don't exist in python. The interpreter automatically recognizes the datatype. You can find out the datatype of the variable by passing it to the type() function.

```
type(5)
type(a)
type(12.75)
```

Printing to the screen :

Python has a print() function for outputting to the screen. Whatever values are passed to the function will be printed with spaces in between and a newline character in the end by default. This can be modified by changing the value of **sep** and **end** parameters while passing the values to print.

```
a = 5
print("The value of a is:", a , sep="@", end="$")
```

The above code will give the following output:

```
The value of a is:@5$
```

Also, python allows for the use of both single quotes as well as double quotes to represent strings. The only rule is that if you start with a single quote, you MUST end with a single quote as well. The same goes for double quotes.

Note: Python2 has a different version of the print() function.

Datatypes in Python:

1) **Numbers in Python:** Numbers are represented by four datatypes in python:
   - int and long - used to represent numbers without a fractional part
   - float - used to represent numbers with fractional parts
   - complex - used to represent complex numbers

   As mentioned, there is no need to specify the datatype during variable creation. The interpreter will recognize the datatype based on the value stored.

2) **Strings in Python:** Strings in python are sequences of characters enclosed within single quotes OR double quotes. In python, strings are of datatype str. There is no equivalent of C's char in python.

   The length of strings can be found by passing the variable name to the len() function. An individual character at index i can be accessed using the techniques of splicing, similar to C.

```
a = 'ISTE'
print( a[2] )
```

   The output to the above code will be 'T'.

   We can generate subsets of strings as well by using splicing. The general syntax is :

```
string_variable_name[ starting_index : ending_index +1 ]
```

```
A="HELLO EVERYONE"
print(A[1:9]
```

   Remember, the element at the index coming after the colon will NOT be included. By default, if the starting index isn't mentioned, then it begins from 0. Similarly, if the ending index isn't mentioned, it goes up to the length of the string.

   Python also allows for negative indexes. For example, index -1 refers to the last element, -2 to the second last, and so on. We can incorporate this with the splicing rules mentioned above as well.

3) **Boolean values in Python**: Python provides two Boolean values - True and False.

These are generated by conditional statements.

4) **Lists in Python:** Lists are the python equivalent of C's arrays. Except with a lot more of functionality. An empty list can be created using one of the two following methods:

```
a = [ ]
OR
a = list()
```

Elements can be added to the list by using the append() function. The elements are added to the right-most end of the list.

```
a = [ 1,2,3,4 ]
a.append( 5 )
```

The splicing rules mentioned above for string also apply to lists. Lists are mutable, unlike strings. This means the values in the lists can be modified after creation.

The most important feature of python lists is that they can store elements of different datatypes at the same time. So the list a = [1, 2.34 , 'abc' , [1,2,3,4]] is valid in python. Note the fourth element of list a is a list in itself.

Note: More on lists will be taught in upcoming sessions.

Operators in Python:

1) **Arithmetic operators:** Python supports all of the arithmetic operators as in C (+,-,*,/,%). Along with these, it has two new arithmetic operators. The first is the floor division operator (//) This will divide and apply the floor() function to it. The second one is the exponentiation operator(**). This will raise the first operand to the power of the second operand.

```
print( 5//2 )
print( 5**2)
```

This will output 2 and 25, in different lines.

2) **Relational operators:** Python supports all the relational operators as in C(<, >, ==) and brings in the new identity operator (is). This operator gives True when the two values being compared have the same id (id of a value / variable can be checked by passing it to id() function). Otherwise False .

```
a=5

b=5

print( a is b)
```

The above code prints True . But having the same value might not always mean True will be returned, as in the case of two lists having the same elements. The working is very tricky and we suggest you try it out on a bunch of values to get familiar with it.

3) **Bitwise operators and Assignment operators:** Bitwise and Assignment operators work the same as in C. There is an additional feature in python, called multiple assignments, as seen below.

```
a,b,c = 1,2,3
```

4) **Logical operators:** Logical operators are much more verbose in python. They are: and, or and not. The working is as the name says for each of those. We use these logical operators to combine conditional statements.

5) **Membership operator:** Python has something called the membership operator (in). This is used to see if an element is present in an object. For example, to check if an element is present in a list, or of a smaller string is a substring of a larger string.

```
a = [ 1,2,3,4,5 ]

b = 'abcde'

print( 2 in a )

print( 'bcd' in b )
```

The above code prints True for both. This operator can be combined with the not operator to check if an element is NOT in an object as well.

Input():

```
X=input("give input for x= ")
```

The input function takes one line from the interface and returns it to the variable. The default type of input is string but we can type cast this into other datatypes. A prompt message can be used in the input function.