CBSE: Class: XII Computer Science

UNIT-1 CHAPTER-1: Basic Python Programming (Revision of Class XI)

What is Python

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.

Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.

Python makes the development and debugging *fast* because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

Difference between Python 2 vs. Python 3

In most of the programming languages, whenever a new version releases, it supports the features and syntax of the existing version of the language, therefore, it is easier for the projects to switch in the newer version. However, in the case of Python, the two versions Python 2 and Python 3 are very much different from each other.

A list of differences between Python 2 and Python 3 are given below:

1. Python 2 uses **print** as a statement and used as print "something" to print some string on the console. On the other hand, Python 3 uses **print** as a function and used as print("something") to print something on the console.
2. Python 2 uses the function raw_input() to accept the user's input. It returns the string representing the value, which is typed by the user. To convert it into the integer, we need to use the int() function in Python. On the other hand, Python 3 uses input() function which automatically interpreted the type of input entered by the user. However, we can cast this value to any type by using primitive functions (int(), str(), etc.).
3. In Python 2, the implicit string type is ASCII, whereas, in Python 3, the implicit string type is Unicode.
4. Python 3 doesn't contain the xrange() function of Python 2. The xrange() is the variant of range() function which returns a xrange object that works similar to Java iterator. The range() returns a list for example the function range(0,3) contains 0, 1, 2.

5. There is also a small change made in Exception handling in Python 3. It defines a keyword **as** which is necessary to be used.

Features of Python

Python provides lots of features that are listed below.

1) **Easy to Learn and Use:** Python is easy to learn and use. It is developer-friendly and high level programming language.

2) **Expressive Language**: Python language is more expressive means that it is more understandable and readable.

3) **Interpreted Language: Python** is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) **Cross-platform Language:** Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5) **Free and Open Source:** Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

6) **Object-Oriented Language :**Python supports object oriented language and concepts of classes and objects come into existence.

7) **Extensible**: It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8) **Large Standard Library :**Python has a large and broad library and prvides rich set of module and functions for rapid application development.

9) **GUI Programming Support :** Graphical user interfaces can be developed using Python.

10) **Integrated:** It can be easily integrated with languages like C, C++, JAVA etc.

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages:
  - ABC language.
  - Modula-3

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.

A list of python versions with its released date is given below.

Python Version	Released Date
Python 1.6	September 5, 2000

Python 2.7	July 3, 2010
Python 3.6	December 23, 2016
Python 3.7.6	December 23, 2018
Python 3.8.1	December 18, 2018
Python 3.8.2	February 24, 2020
Python 3.7.7	March 10, 2020

Python Execution Mode

- **Interactive Mode:** Interactive mode, as the name suggests, allows us to interact with OS.
- **Script Mode:** In script mode, we type Python program in a file and then use interpreter to execute the content of the file.

Python Fundamentals

- **Indentation:** Blocks of code are denoted by line indentation, which is rigidly enforced.
- **Comments:** A hash sign (#) that is not inside a string literal begins a single line comment. We can use triple quoted string for giving multiple-line comments.
- **Variables:** A variable in Python is defined through assignment. There is no concept of declaring a variable outside of that assignment. Value of variable can be manipulated during program run.
- **Dynamic Typing:** In Python, while the value that a variable points to has a type, the variable itself has no strict type in its definition. Data type of the variable is decided at the time of value assignment. If value is a number the variable would be interger/float and if the value is string then variable would be string.

  For example :

  num=44 # here num would be consider as integer variable

  num=”We are human” # here num would be consider as string variable
- **Static Typing:** In Static typing, a data type is attached with a variable when it is defined first and it is fixed.
- **Multiple Assignment:** Python allows you to assign a single value to several variables simultaneously.

  For example: a = b = c = 1

  a, b, c = 1, 2, “john”
- **Token :** The smallest individual unit in a program is known as a Token or a lexical unit.
- **Identifiers :** An identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Value and value are two different identifiers in Python.

Here are following identifiers naming convention for Python:

- Class names start with an uppercase letter and all other identifiers with a lowercase letter.
- Starting an identifier with a single leading underscore indicates by convention that the identifier is meant to be private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

- **Reserved Words(Keywords) :** The following list shows the reserved words in Python

Python Keyword List

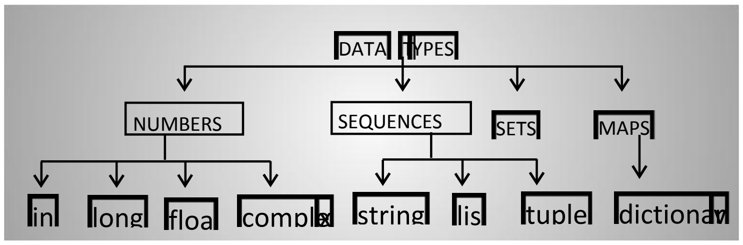
Keywords in Python				
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

The above keywords may get altered in different versions of Python. Some extra might get added or some might be removed.
You can always get the list of keywords in your current version by typing the following in the prompt.

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

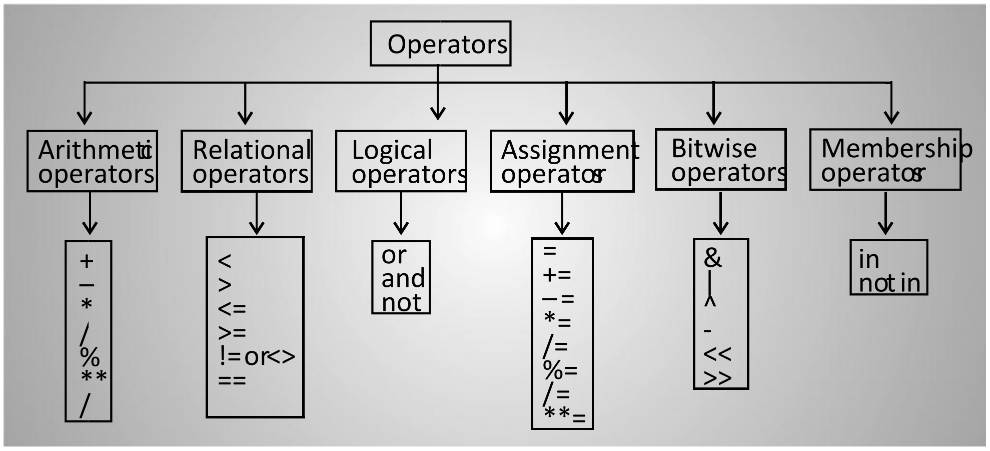
These reserved words may not be used as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

- **Literal/ Values:** Literals (Often referred to as constant value) are data items that have a fixed value. Python allows several kind of literals. String literals, Numeric literals, Boolean literals, special literal None, literal Collections
- **Data Types:** Data type is a set of values and the allowable operations on those values. Python has a great set of useful data types. Python's data types are built in the core of the language. They are easy to use and straight forward.



- **Numbers** can be either integers or floating point numbers.
- A **sequence** is an ordered collection of items, indexed by integers starting from 0. Sequences can be grouped into strings, tuples and lists.
- **Strings** are lines of text that can contain any character. They can be declared with single or double quotes.
- **Lists** are used to group other data. They are similar to arrays.
- **Tuple** consists of a number of values separated by commas.
- A **set** is an unordered collection with no **duplicate** items.
- A **dictionary** is an unordered set of key value pairs where the keys are unique.
- **An Expression** in python is a valid combination of operators, literals and variables.
- **Operator :** Operators are special symbols which perform some computation. Operators and operands form an expression and when combined together then give result. Python operators can be classified as given below :

Python operator types



-

Python Loops:

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.

Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

Example of Loops

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

```
for iterating_var in sequence:
    statement(s)
```

Example

```
i=1
n=int(input("Enter the number up to which you want to print the natural numbers?"))
for i in range(0,10):
    print(i,end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Python for loop example : printing the table of the given number

```python
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

Output:

```
Enter a number:10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Nested for loop in python

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax of the nested for loop in python is given below.

```python
for iterating_var1 in sequence:
    for iterating_var2 in sequence:
        #block of statements
#Other statements
```

Example 1

```python
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
    print()
    for j in range(0,i+1):
        print("*",end="")
```

Output:

```
Enter the number of rows you want to print?5
*
**
***
****
*****
```

Using else statement with for loop

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example 1

```
for i in range(0,5):
    print(i)
else:print("for loop completely exhausted, since there is no break.");
```

In the above example, for loop is executed completely since there is no break statement in the loop. The control comes out of the loop and hence the else block is executed.

Output:

```
0
1
2
3
4
```

for loop completely exhausted, since there is no break.

Example 2

```
for i in range(0,5):
    print(i)
    break;
else:print("for loop is exhausted");
print("The loop is broken due to break statement...came out of loop")
```

In the above example, the loop is broken due to break statement therefore the else statement will not be executed. The statement present immediate next to else block will be executed.

Output:

```
0
```

The loop is broken due to break statement...came out of loop

Python while loop

The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

The syntax is given below.

```
while expression:
    statements
```

Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.

Example 1

```
i=1;
while i<=10:
    print(i);
    i=i+1;
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example 2

```
i=1
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:
    print("%d X %d = %d \n"%(number,i,number*i));
    i = i+1;
```

Output:

```
Enter the number?10
```

```
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Infinite while loop

If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.

Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

Example 1

```
while (1):
    print("Hi! we are inside the infinite while loop");
```

Output:

```
Hi! we are inside the infinite while loop
(infinite times)
```

Example 2

```
var = 1
while var != 2:
    i = int(input("Enter the number?"))
    print ("Entered value is %d"%(i))
```

Output:

```
Enter the number?102
Entered value is 102
Enter the number?102
Entered value is 102
Enter the number?103
Entered value is 103
```

```
Enter the number?103
(infinite loop)
```

Using else with Python while loop

Python enables us to use the while loop with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

Output:

```
1
2
3
4
5
The while loop exhausted
```

Example 2

```
i=1;
while i<=5:
    print(i)
    i=i+1;
    if(i==3):
        break;
else:print("The while loop exhausted");
```

Output:

```
1
2
```

Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

```
#loop statements
break;
```

Example 1

```
list =[1,2,3,4]
count = 1;
for i in list:
    if i == 4:
        print("item matched")
        count = count + 1;
        break
print("found at",count,"location");
```

Output:

```
item matched
found at 2 location
```

Example 2

```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

Output:

```
p
y
t
h
```

Example 3: break statement with while loop

```
i = 0;
while 1:
    print(i," ",end=""),
    i=i+1;
    if i == 10:
        break;
print("came out of while loop");
```

Output:

```
0  1  2  3  4  5  6  7  8  9  came out of while loop
```

Example 3

```
n=2
while 1:
    i=1;
    while i<=10:
        print("%d X %d = %d\n"%(n,i,n*i));
        i = i+1;
    choice = int(input("Do you want to continue printing the table, press 0 for no?"))
    if choice == 0:
        break;
    n=n+1
```

Output:

```
2 X 1 = 2

2 X 2 = 4

2 X 3 = 6

2 X 4 = 8

2 X 5 = 10

2 X 6 = 12

2 X 7 = 14

2 X 8 = 16

2 X 9 = 18

2 X 10 = 20

Do you want to continue printing the table, press 0 for no?1

3 X 1 = 3

3 X 2 = 6

3 X 3 = 9
```

```
3 X 4 = 12

3 X 5 = 15

3 X 6 = 18

3 X 7 = 21

3 X 8 = 24

3 X 9 = 27

3 X 10 = 30

Do you want to continue printing the table, press 0 for no?0
```

Python continue Statement

The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

```
#loop statements
continue;
#the code to be skipped
```

Example 1

```
i = 0;
while i!=10:
    print("%d"%i);
    continue;
    i=i+1;
```

Output:

```
infinite loop
```

Example 2

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
    if i==5:
        continue;
    print("%d"%i);
```

Output:

```
1
2
3
4
6
7
8
9
10
```

Pass Statement

The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.

For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.

Pass is also used where the code will be written somewhere but not yet written in the program file.

The syntax of the pass statement is given below.

Example

```
list = [1,2,3,4,5]
flag = 0
for i in list:
    print("Current element:",i,end=" ");
    if i==3:
        pass;
        print("\nWe are inside pass block\n");
        flag = 1;
    if flag==1:
        print("\nCame out of pass\n");
        flag=0;
```

Output:

```
Current element: 1 Current element: 2 Current element: 3
We are inside pass block


Came out of pass

Current element: 4 Current element: 5
```

Python Pass

In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

Python Pass Syntax

```
pass
```

Python Pass Example

```
for i in [1,2,3,4,5]:
    if i==3:
        pass
        print "Pass when value is",i
    print i,
```

Output:

```
>>>
1 2 Pass when value is 3
3 4 5
```