CHAPTER 1

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Why to Learn Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of Python Programming –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Hello World using Python.

Just to give you a little excitement about Python, I'm going to give you a small conventional Python Hello World program.

```
print ("Hello, Python!");
```

Features of Python

As mentioned before, Python is one of the most widely used language over the web. I'm going to list few of them here:

Easy-to-learn – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read – Python code is more clearly defined and visible to the eyes.

Easy-to-maintain – Python's source code is fairly easy-to-maintain.

A broad standard library – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases – Python provides interfaces to all major commercial databases.

GUI Programming – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable – Python provides a better structure and support for large programs than shell scripting.

First Python Program

Let us execute programs in different modes of programming.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt –

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file –

```
print "Hello, Python!"
```

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```

Python Identifiers

A Python 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, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers –

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is 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

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
  print "True"
else:
  print "False"
```

However, the following block generates an error –

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks –

Python Variables

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive; for example, my name, and MyName is not the same.
- Examples of valid identifiers: a123, _n, n_9, etc.

- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Object References

It is necessary to understand how the Python interpreter works when we declare a variable. The process of treating variables is somewhat different from many other programming languages.

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class. Consider the following example.

```
1. print("John")
```

Output:

```
John
```

The Python object creates an integer object and displays it to the console. In the above print statement, we have created a string object. Let's check the type of it using the Python built-in **type()** function.

```
1. type("John")
```

Output:

```
<class 'str'>
```

In Python, variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

Let's understand the following example

```
1. a = 50
```

In the above image, the variable **a** refers to an integer object.

Suppose we assign the integer value 50 to a new variable b.

```
a = 50
b = a
```

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

```
a = 50
b =100
```

Python manages memory efficiently if we assign the same variable to two different values.

Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guaranteed that no two objects will have the same identifier. The built-in **id()** function, is used to identify the object identifier. Consider the following example.

1. a = 50
2. b = a
3. print(id(a))
4. print(id(b))
5. # Reassigned variable a
6. a = 500
7. print(id(a))

Output:

```
140734982691168
140734982691168
2822056960944
```

We assigned the **b = a, a** and **b** both point to the same object. When we checked by the **id()** function it returned the same number. We reassign **a** to 500; then it referred to the new object identifier.

Variable Names

We have already discussed how to declare the valid variable. Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

1. name = "Devansh"
2. age = 20
3. marks = 80.50
4. print(name)
5. print(age)
6. print(marks)

Output:

```
Devansh
20
80.5
```

Consider the following valid variables name.

1. name = "A"
2. Name = "B"
3. naMe = "C"
4. NAME = "D"
5. n_a_m_e = "E"
6. _name = "F"
7. name_ = "G"
8. _name_ = "H"
9. na56me = "I"
10. print(name,Name,naMe,NAME,n_a_m_e, NAME, n_a_m_e, _name, name_,_name, na56me)

Output:

```
A B C D E D E F G F I
```

In the above example, we have declared a few valid variable names such as name, _name_ , etc. But it is not recommended because when we try to read code, it may create confusion. The variable name should be descriptive to make code more readable.

The multi-word keywords can be created by the following method.

- **Camel Case -** In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.
- **Pascal Case -** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
- **Snake Case -** In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

We can apply multiple assignments in two ways, either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Consider the following example.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
```

1. print(x)
2. print(y)

3. print(z)

Output:

```
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

1. a,b,c=5,10,15
2. print a
3. print b
4. print c

Output:

```
5
10
15
```

The values will be assigned in the order in which variables appear.

Numbers in Python

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Number objects are created when you assign a value to them. For example −

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the **del** statement. The syntax of the del statement is −

```
del var1[,var2[,var3[....,varN]]]]
```

You can delete a single object or multiple objects by using the **del** statement. For example −

```
del var
del var_a, var_b
```

Python supports four different numerical types −

- **int (signed integers)** − They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
- **long (long integers)** − Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- **float (floating point real values)** − Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x 10^2 = 250).

- **complex (complex numbers)** – are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are not used much in Python programming.

Examples

Here are some examples of numbers

int	**long**	**float**	**complex**
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEL	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating point numbers denoted by a + bj, where a is the real part and b is the imaginary part of the complex number.

Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

- Type **int(x)** to convert x to a plain integer.
- Type **long(x)** to convert x to a long integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Mathematical Functions

Python includes following functions that perform mathematical calculations.

Sr.No.	**Function & Returns (description)**
1	abs(x) The absolute value of x: the (positive) distance between x and zero.
2	ceil(x) The ceiling of x: the smallest integer not less than x

3	cmp(x, y) -1 if x < y, 0 if x == y, or 1 if x > y
4	exp(x) The exponential of x: e^x
5	fabs(x) The absolute value of x.
6	floor(x) The floor of x: the largest integer not greater than x
7	log(x) The natural logarithm of x, for x> 0
8	log10(x) The base-10 logarithm of x for x> 0.
9	max(x1, x2,...) The largest of its arguments: the value closest to positive infinity
10	min(x1, x2,...) The smallest of its arguments: the value closest to negative infinity
11	modf(x) The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
12	pow(x, y) The value of x**y.
13	round(x [,n]) **x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
14	sqrt(x) The square root of x for x > 0

Random Number Functions

Random numbers are used for games, simulations, testing, security, and privacy applications. Python includes following functions that are commonly used.

Sr.No.	Function & Description
1	choice(seq) A random item from a list, tuple, or string.
2	randrange ([start,] stop [,step]) A randomly selected element from range(start, stop, step)
3	random()

	A random float r, such that 0 is less than or equal to r and r is less than 1
4	seed([x]) Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
5	shuffle(lst) Randomizes the items of a list in place. Returns None.
6	uniform(x, y) A random float r, such that x is less than or equal to r and r is less than y

Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a specific programming language. Python provides a variety of operators, which are described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0

**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to the second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.

**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Logical Operators

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both sides do not point to the same object.

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description

**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

The syntax of the if-statement is given below.

```
1. if expression:
2.     statement
```

Example 1

```
1. num = int(input("enter the number?"))
2. if num%2 == 0:
3.     print("Number is even")
```

Output:

```
enter the number?10
Number is even
```

Example 2 : Program to print the largest of the three numbers.

```
1. a = int(input("Enter a? "));
2. b = int(input("Enter b? "));
3. c = int(input("Enter c? "));
4. if a>b and a>c:
5.     print("a is largest");
6. if b>a and b>c:
7.     print("b is largest");
8. if c>a and c>b:
9.     print("c is largest");
```

Output:

```
Enter a? 100
Enter b? 120
Enter c? 130
c is largest
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

The syntax of the if-else statement is given below.

```
1. if condition:
2.     #block of statements
3. else:
4.     #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
1. age = int (input("Enter your age? "))
2. if age>=18:
3.     print("You are eligible to vote !!");
4. else:
5.     print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90
You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
1. num = int(input("enter the number?"))
2. if num%2 == 0:
3.     print("Number is even...")
4. else:
5.     print("Number is odd...")
```

Output:

```
enter the number?10
Number is even
```

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements
```

```
elif expression 2:
   # block of statements
elif expression 3:
   # block of statements
else:
   # block of statements
```

Example 1

```
1. number = int(input("Enter the number?"))
2. if number==10:
3.    print("number is equals to 10")
4. elif number==50:
5.    print("number is equal to 50");
6. elif number==100:
7.    print("number is equal to 100");
8. else:
9.    print("number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
number is not equal to 10, 50 or 100
```

Example 2

```
1. marks = int(input("Enter the marks? "))
2. f marks > 85 and marks <= 100:
3.    print("Congrats ! you scored grade A ...")
4. lif marks > 60 and marks <= 85:
5.    print("You scored grade B + ...")
6. lif marks > 40 and marks <= 60:
7.    print("You scored grade B ...")
8. lif (marks > 30 and marks <= 40):
9.    print("You scored grade C ...")
10. lse:
11.    print("Sorry you are fail ?")
```

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.

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).

Python for loop

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)
```

For loop Using Sequence

Example-1: Iterating string using for loop

```
str = "Python"
for i in str:
    print(i)
```

Output:

```
P
y
t
h
o
n
```

Example- 2: Program to print the table of the given number .

```
list = [1,2,3,4,5,6,7,8,9,10]
n = 5
```

```
3. for i in list:
4.     c = n*i
5.     print(c)
```

Output:

```
5
10
15
20
25
30
35
40
45
50s
```

Example-4: Program to print the sum of the given list.

```
1. list = [10,30,23,43,65,12]
2. sum = 0
3. for i in list:
4.     sum = sum+i
5. print("The sum is:",sum)
```

Output:

```
The sum is: 183
```

For loop Using range() function

The range() function

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:

- range(start,stop,step size)
  - The start represents the beginning of the iteration.
  - The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
  - The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Consider the following examples:

Example-1: Program to print numbers in sequence.

```
1. for i in range(10):
```

```
2.  print(i,end = ' ')
```

Output:

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

Example - 2: Program to print table of given number.

```
1. n = int(input("Enter the number "))
2. for i in range(1,11):
3.     c = n*i
4.     print(n,"*",i,"=",c)
```

Output:

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

Example-3: Program to print even number using step size in range().

```
1. n = int(input("Enter the number "))
2. for i in range(2,n,2):
3.     print(i)
```

Output:

```
Enter the number 20
2
4
6
8
10
12
14
16
18
```

We can also use the **range()** function with sequence of numbers. The **len()** function is combined with range() function which iterate through a sequence using indexing. Consider the following example.

```
1. list = ['Peter','Joseph','Ricky','Devansh']
2. for i in range(len(list)):
3.     print("Hello",list[i])
```

Output:

```
Hello Peter
Hello Joseph
Hello Ricky
Hello Devansh
```

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

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

Output:

```
0
1
2
3
4
for loop completely exhausted, since there is no break.
```

The for loop completely exhausted, since there is no break.

Example 2

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

In the above example, the loop is broken due to the 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 the break statement...came out of the loop. We will learn more about the break statement in next tutorial.

Python While loop

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

```
1. while expression:
2.     statements
```

Here, the statements can be a single statement or a group of statements. The expression should be any valid Python expression resulting in true or false. The true is any non-zero value and false is 0.

Loop Control Statements

We can change the normal sequence of **while** loop's execution using the loop control statement. When the while loop's execution is completed, all automatic objects defined in that scope are demolished. Python offers the following control statement to use within the while loop.

1. Continue Statement - When the continue statement is encountered, the control transfer to the beginning of the loop. Let's understand the following example.

Example:

```
1. # prints all letters except 'a' and 't'
2. i = 0
3. str1 = 'javatpoint'
4.
5. while i < len(str1):
6.     if str1[i] == 'a' or str1[i] == 't':
7.         i += 1
8.         continue
9.     print('Current Letter :', a[i])
10.    i += 1
```

Output:

```
Current Letter : j
Current Letter : v
Current Letter : p
```

```
Current Letter : o
Current Letter : i
Current Letter : n
```

2. Break Statement - When the break statement is encountered, it brings control out of the loop.

Example:

```
# The control transfer is transfered
# when break statement soon it sees t
i = 0
str1 = 'javatpoint'

while i < len(str1):
    if str1[i] == 't':
        i += 1
        break
    print('Current Letter :', str1[i])
    i += 1
```

Output:

```
Current Letter : j
Current Letter : a
Current Letter : v
Current Letter : a
```

3. Pass Statement - The pass statement is used to declare the empty loop. It is also used to define empty class, function, and control statement. Let's understand the following example.

Example -

```
# An empty loop
str1 = 'javatpoint'
i = 0

while i < len(str1):
    i += 1
    pass
print('Value of i :', i)
```

Output:

```
Value of i : 10
```

Example-1: Program to print 1 to 10 using while loop

```
1. i=1
2. #The while loop will iterate until condition becomes false.
3. While(i<=10):
4.     print(i)
5.     i=i+1
```

Output:

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

Example -2: Program to print table of given numbers.

```
1. i=1
2. number=0
3. b=9
4. number = int(input("Enter the number:"))
5. while i<=10:
6.     print("%d X %d = %d \n"%(number,i,number*i))
7.     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 is given in the while loop never becomes false, then the while loop will never terminate, and it turns into the **infinite while loop.**

Any **non-zero** value in the while loop indicates an **always-true** condition, whereas zero 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

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

Output:

```
Hi! we are inside the infinite while loop
Hi! we are inside the infinite while loop
```

Example 2

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

Output:

```
Enter the number:10
Entered value is 10
Enter the number:10
Entered value is 10
Enter the number:10
Entered value is 10
Infinite time
```

Using else with while loop

Python allows us to use the else statement 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. The else statement is optional to use with the while loop. Consider the following example.

Example 1

```
1. i=1
```

```
2. while(i<=5):
3.     print(i)
4.     i=i+1
5. else:
6.     print("The while loop exhausted")
```

Example 2

```
1. i=1
2. while(i<=5):
3.     print(i)
4.     i=i+1
5.     if(i==3):
6.         break
7. else:
8.     print("The while loop exhausted")
```

Output:

```
1
2
```

In the above code, when the break statement encountered, then while loop stopped its execution and skipped the else statement.

Example-3 Program to print Fibonacci numbers to given limit

```
1. terms = int(input("Enter the terms "))
2. # first two intial terms
3. a = 0
4. b = 1
5. count = 0
6.
7. # check if the number of terms is Zero or negative
8. if (terms <= 0):
9.     print("Please enter a valid integer")
10. elif (terms == 1):
11.     print("Fibonacci sequence upto",limit,":")
12.     print(a)
13. else:
14.     print("Fibonacci sequence:")
15.     while (count < terms) :
16.         print(a, end = ' ')
```

```
17.     c = a + b
18.     # updateing values
19.     a = b
20.     b = c
21.
22.  count += 1
```

Output:

```
Enter the terms 10
Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34
```

*****