# **Introduction to Python**

Python was developed by Guido Van Rossum at the National Research Institute for Mathematics and Computer Science in Netherlands during 1985-1990. Python is derived from many other languages, including ABC, Modula-3,C,C++,Algol-68,SmallTalk,Unix shell and other scripting languages. Rossum was inspired by Monty Python's Flying Circus, a BBC comedy series and he wanted the name of his new language to be short, unique and mysterious. Hence he named it python. It is general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python source code is available under the GNU General Public License(GPL) and is now maintained by a core development team at the National Research Institute.

**Features of Python**

1. Simple and easy-to-learn
2. Interpreted and interactive
3. Object-oriented
4. Portable
5. Scalable
6. Extendable
7. Dynamic
8. GUI Programming and Database
9. Broad Standard Library

**Using Interactive Interpreter**

You can start python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window. Get into command line of python. For Unix/Linux, you can get into interactive mode by typing $python or python%. For windows/DOS it is c:>python.

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

$ python

python 2.7.10 (default, sep 27 2015, 18:11:38)

[GCI 5.1.1 20150422 (Red Hat 5.1.1-1)] on linux2

Type "help", "copyright", "credits" or "license" for more

information1

>>>

Type the following text at python prompt and press enter:

>>>print("programming in python")

The result will be as given below

programming in python

**1 - Variable and Assignments**

**a) Scripts From Command Line**

This method invokes the interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. a python script can be executed at command line by invoking the interpreter on your application, as follows.

for Unix/Linux $python script.py or python% script.py

for windows/DOS c:>python script.py

**Identifiers**

A python identifier is a name used to identify a variable, function, class, module or any other object. Python is case sensitive and hence uppercase and lowercase letters are considered distinct. The following are rules for naming an identifier in python.

1. Identifier can be combination of letters in lowercase(a to z) or uppercase(A to Z) or digit(0-9) or an underscore(\_). For example total and Total is different.
2. Reserved keywords cannot be used as an identifier.
3. Identifiers cannot begin with a digit.
4. Special symbols like !,@,#, etc cannot be used in an identifier. For example sum@ is an invalid identifier.
5. Identifier can be of any length.

**Reserved Keywords**

These are keywords reserved by the programming language and prevent the user or the programmer from using it as it as an identifier in a program. There are 33 keywords in python 3.3. This number may vary with different versions. To retrieve the keywords in python the following code can be given at the prompt.

>>> import keyword

>>> print(keyword.kwlist)

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

**Variables**

Variables are reserved memory locations to store values. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign(=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in that variable.

**Example program**

a = 100

b = 1000

print(a)

print(b)

**Output**

100

1000

**Comments in Python**

Comments are very important while writing a program. It describes what the source code has done. Comments are for programmers for better understanding of a program. In python , we use the hash(#) symbol to start writing a comment. All the character after the #and up to the end of the physical line are part of the comment.

**Example program:**

>>>#this is demo of comment

>>>print("hello")

For multiline comments use triple quotes, either ''' or """. the statements within the start and end of triple quotes are considered as comment.

**Example program:** Consider sample python script to produce sum of two variables

“””

within this quotes all are considered as comments

This program will print sum of two variable

a is entered first

b is entered next

c will produce sum of a and b

“””

a = input(“enter a value”)

b = input(“enter b value”)

c = a + b

print “sum of a and b is “, c

**Indentation in Python**

Most of the programming languages like c, c++ use braces{} to define a block of code. Python uses indentation. A code block starts with indentation and ends with the first unindented line. The amount of indentation can be decided by the programmer, but it ,must be consistent throughout the block. Generally four whitespaces are used for indentation.

**Example program**

if True:

print("correct") # 4 space gap given at begining

else:

print("Wrong") # same 4 space gap given

**Input, Output and Import Functions**

**Displaying the Output**

The function used to print output on a screen is print statement where you can pass zero or more expression separated by commas. The print function converts the expression you pass into a string and writes the result to standard output.

**Example program**

>>>a=2

>>>print("the value of a is:",a)

the value of a is 2

**Reading the Input**

Python 2.7 provides two built-in functions to read from standard input(keyboard). These functions are raw\_input and input.

input is used to get integer or float user inputs

raw\_input is used to get string or int or float user inputs

**Example program**

**1)** a = input("enter your age:")

print("your age is: ",a)

**output**

enter your name: 20

your age is: 20

**2)** a, b, c = input(“enter a, b and c values, each separated by comma character: \n”)

print “a+b+c is: ”, (a+b+c)

**output:**

enter a, b and c values, each separated by comma character:

10,20,30

a+b+c is: 60

3) name = raw\_input(“enter your name:”)

print “entered name is: %s” %name

**output:**

enter your name: python

entered name is: python

**Import function**

When the program grows bigger or when there are segments of code that is frequently used, it can be stored in different modules. A module is a file containing python definition and statements. Python modules have a file name and end with expression .py. Definition inside a module can be imported to another module or the interactive interpreter in python. We use the import keyword to do this .

**Example program**

>>>import math

>>>math.pi

3.141592653589793

**Operators in Python**

**1) Arithmetic operators:**

**+, -, \*,/,//,%,\*\*,-**

+,-,\*,/ works as expected from their algebra class notations.

// is called floor division and it truncates the floating point value while preserving the data type.

% is called modulus and gives reminder of division. It works right to left.(?)

\*\* is exponential.

**2) Conditional operators**

**<,>,==,<=,>=,!=**

These operators result in Boolean True or False value. They are good for using in conditions

**3) Membership operators**

in, not in simply checks if the value is in a particular sequence or range and return True and False accordingly.

**Variable as Storage unit.**

Since variables hold reference to data for manipulation. Popular database operations like Insert, Update, Delete should also be possible on variable.

To something like insert or assign or define:

**Syntax is :** *variable = value*

*E.g. >>>x=5*

To update a variable value:

**Syntax is :** variable = new \_value

E.g. >>>x=10

After new value assignment, the variables are pointer is pointed to new value. Since one variable can point to only one location at a time. The value old value is garbage collected.

To delete a variable:

**Syntax is :** *del variablename*

*E.g. >>>del x*

**Unique property of the = (assignment) operator**

It right to left hence , in variable = value , value is considered first and the identification is value based.

**Variable Nomenclature**

1. Variable name must start with [\_a-z, A-Z] then followed by [\_a-z, A-Z, 0-9]

2. Variable names are case sensitive

3. Keywords cannot be used as Python Variable names

>>>import keyword

>>>keyword.kwlist

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

Examples of valid and invalid variable name.

\_a=3 #valid

Apple = “apple” #valid

0time = 0 #in-invalid

**Variable and Data Types**

number = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

x = [1,2,”users”] # A List

y = (“username”, “password”) # A Tuple

dict ={'name':'john','code':6734,'dept':'sales'} # A dictionary

**Python Variable properties**

**1. Implicitly defined**

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

**2. Strongly typed**

Python is strongly typed as the interpreter keeps track of all variables types.Compiler here cannot can tell which type a variable refers to.

**Type() function**

type(object) -> the object's type

>>>x=”name”

>>>type(x)

<type 'str'>

**Multiple assignment**

Ex: >>>*a=b=c=1*

*>>>a,b,c= 1,2,”name*

**Python default variable ‘\_’**

‘\_’ is a variable auto defined by Python and it stores last calculation. It is like ANS button on calculator.

>>> 20 + 30

50

>>> 50 + \_

*100*

## **The str(), int(), and float() Functions**

## If you want to concatenate an integer such as 29 with a string to pass to print(), you’ll need to get the value '29', which is the string form of 29. The str() function can be passed an integer value and will evaluate to a string value version of it, as follows:

>>> **str(29)**

'29'

>>> **print('I am ' + str(29) + ' years old.')**

I am 29 years old.

Because str(29) evaluates to '29', the expression 'I am ' + str(29) + ' years old.' evaluates to 'I am ' + '29' + ' years old.', which in turn evaluates to 'I am 29 years old.'. This is the value that is passed to the print() function.

The str(), int(), and float() functions will evaluate to the string, integer, and floating-point forms of the value you pass, respectively. Try converting some values in the interactive shell with these functions, and watch what happens.

>>> **str(0)**

'0'

>>> **str(-3.14)**

'-3.14'

>>> **int('42')**

42

>>> **int('-99')**

-99

>>> **int(1.25)**

1

>>> **int(1.99)**

1

>>> **float('3.14')**

3.149

>>> **float(10)**

10.0

The previous examples call the str(), int(), and float() functions and pass them values of the other data types to obtain a string, integer, or floating-point form of those values.

The str() function is handy when you have an integer or float that you want to concatenate to a string. The int() function is also helpful if you have a number as a string value that you want to use in some mathematics. For example, the input() function always returns a string, even if the user enters a number. Enter **spam = input()** into the interactive shell and enter **101** when it waits for your text.

>>> **spam = input()**

101

>>> **spam**

'101'

The value stored inside spam isn’t the integer 101 but the string '101'. If you want to do math using the value in spam, use the int() function to get the integer form of spam and then store this as the new value in spam.

>>> **spam = int(spam)**

>>> **spam**

101

Now you should be able to treat the spam variable as an integer instead of a string.

>>> **spam \* 10 / 5**

202.0

Note that if you pass a value to int() that it cannot evaluate as an integer, Python will display an error message.

>>> **int('99.99')**

Traceback (most recent call last):

File "<pyshell#18>", line 1, in <module>

int('99.99')

ValueError: invalid literal for int() with base 10: '99.99'

>>> **int('twelve')**

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

int('twelve')

ValueError: invalid literal for int() with base 10: 'twelve'

The int() function is also useful if you need to round a floating-point number down. If you want to round a floating-point number up, just add 1 to it afterward.

>>> **int(7.7)**

7

>>> **int(7.7) + 1**

8

In your program, you used the int() and str() functions in the last three lines to get a value of the appropriate data type for the code.

print('What is your age?') # ask for their age

myAge = input()

print('You will be ' + str(int(myAge) + 1) + ' in a year.')

The myAge variable contains the value returned from input(). Because the input() function always returns a string (even if the user typed in a number), you can use the int(myAge) code to return an integer value of the string in myAge. This integer value is then added to 1 in the expressionint(myAge) + 1.

The result of this addition is passed to the str() function: str(int(myAge) + 1). The string value returned is then concatenated with the strings 'You will be ' and ' in a year.' to evaluate to one large string value. This large string is finally passed to print() to be displayed on the screen.

## **The len() Function**

## You can pass the len() function a string value (or a variable containing a string), and the function evaluates to the integer value of the number of characters in that string.

print('The length of your name is:')

print(len(myName))

Enter the following into the interactive shell to try this:

>>> **len('hello')**

5

>>> **len('My very energetic monster just scarfed nachos.')**

46

>>> **len('')**

0

**id() function:**

This is used to determine the id of a variable. It is value based and not variable name based.

>>>x=5

>>>id(x)

140566819039352

>>>y=6

>>>hex(id(y))

'0x163c080'

>>>z=5

>>>id(z)

'0x164d6d0'

**Membership Operators**

Python's membership operators test for membership in a sequence, such as strings, lists, and tuples. There are two membership operators supported by python which are described in the following table.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| in | Evaluates to true if the variable on either side of the operator point to the same object and false otherwise |
| not in | Evaluates to true if it does not finds a variable in the specifies sequence and false otherwise. |

**Identity Operator**

Identity operators compare the memory locations of two objects. There are two identity operators as shown in the following table

|  |  |
| --- | --- |
| **Operator** | **Description** |
| is | Evaluates to true if the variables on either side of the operator point to the same objects and false otherwise |
| is not | Evaluates to false if the variables on either side of the operator point to the same objects and true otherwise |

**Questions**

1) Assign the value 10 to a, and value 20 to b. Assign the value of a + b to variable c.

2) Which of the following will produce the right result

a) 2 + 5 produces '25'

b) '2' + 5 produces '25'

c) '2' + '5' produces '25'

d) 2 + '5' produces '25'

3) Compute the area and perimeter of a circle with radius = 3

4) Declare a string literal by enclosing the literal using single, double or triple quotes. Triple quote allows the literal to span multiple lines.

>>> spam = 'mail'

>>> contents = """This is Line 1

... and this is Line 2

... and this is Line 3

... """

>>> spam

'mail'

>>> contents

'This is Line 1\nand this is Line 2\nand this is Line 3\n'

>>> 'How're you?' *# Syntax error for same quote*

**SyntaxError**: invalid syntax

>>> "How're you?" *# Using different quote*

"How're you?"

>>> 'How**\'**re you?' *# Escape quote with \*

"How're you?"

>>> foobar

'"No, thanks, Mom," I said, "I don**'**t know how long it will take."'

5) Certain mathematical operations such as addition and multiplication can be used on the String data type. Study the examples given below to see how it works

>>> 5 + 6 *# Adding numbers*

11

>>> '5' + '6' *# Adding strings*

'56'

>>> 5 \* 6 *# Multiplying numbers*

30

>>> '5' \* 6 *# Multiplying string with number*

*'555555'*

6) Assign strings 'Hello' and 'World!' to single variable a

**Sol:**

a = "Hello"+"World!"

# repeating string 5 times

b = "HelloWorld!"\*5

7)You can make use of the len(x) function to find out the number of characters in a string.

>>> greetings = "Hello World"

>>> len(greetings) *# get the length of string*

11

>>> greetings[0] *# get the 1st character*

'H'

8) What is the value of x?

>> x = 1 == 1 and 1 != 0 or 1 > 0

9)Extract each word from variable 'greetings' and assign to

# variables 'first', 'middle' and 'last'.

greetings = "How are you"

first = greetings[ :3]

middle = greetings[ 4:7]

last = greetings[ 8:]

**Flow Control**

**Decision Making**

Decision making is required when we want to execute a code only if a certain condition is satisfied. The if...elif....else statement is used in python for decision making.

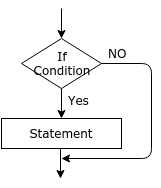
**if statement:**

**syntax**

if test expression:

statement(s)

**The following figure shows the flowchart of the if statement.**

****

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is true. If the text expression is false the statement(s) is not executed. In python, the body of the if statement is indicated by the indentation. Body starts with an indentation and ends with the first unindented line. Python interprets non-zero values as true. None and 0 are interpreted as false.

**Example program**

num = input("enter a number:")

if num == 0:

print(zero")

print("this is always printed")

**output1**

enter the number: 0

Zero

this is always printed

**output2**

enter the number:1

**this is always printed**

**if....else statement:**

**syntax**

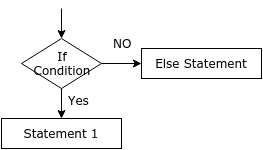
if test expression:

body of if

else:

body of else

The following figure shows the flowchart of if....else. The if...else statement evaluates the test expression and will execute body of if only when test condition is true. If the condition is false, body of else is executed. indentation is used to separate the blocks.



**Example program**

num = input("enter a number:")

if num>=0:

print("positive integer")

else:

print("negative integer")

**output1**

enter a number: 5

positive integer

**output2**

enter a number: -2

negative integer

**if...elif...else statement:**

**syntax**

if test expression:

body of if

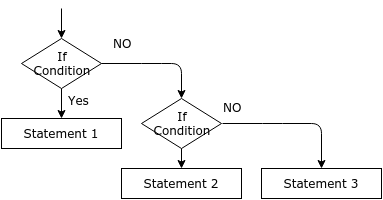
elif test expression:

body of elif

else:

body of else

The elif is short for else if. It allows us to check for multiple conditions. If the condition for if is false, it checks the condition of the next elif block and so on. if all the conditions are false, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. A if block can have only one else block. But it can have multiple elif blocks . Following fig shows the flow chart for if...elif...else statement.



**Example program**

num = input("enter a number:")

if num>0:

print("positive integer")

elif num == 0:

print("zero")

else:

print("negative integer")

**output1**

enter a number:5

positive integer

**output2**

enter a number: 0

zero

**output3**

enter a number:-7

negative number

**Nested if statement**

We can have if...elif...else statements inside another if....elif...else statement. This is called nesting in computer programming. indentation is the only way to identify the level of nesting.

**Example program**

num = input("enter a number:")

if num>=0:

if num == 0:

print("zero")

else:

print("positive integer")

else:

print("negative integer")

**output1**

enter a number:5

positive integer

**output2**

enter a number: 0

zero

**output3**

enter a number:-7

negative number

**Loops**

Generally, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on. There will be situations when we need to execute a block of code several number of times. Python provides various control structures that allow for repeated execution. A loop statement allows us to execute a statement or group of statements multiple times.

**for loop**

The for loop in python is used to iterate over a sequence or other objects that can be iterated. Iterating over a sequence is called traversal. below figure shows the flow chart of for loop.

**syntax**

for item in sequence:

body of for

here item is a variable that takes the value of the item inside the sequence of each iteration. The sequence can be list, tuple, string, set etc. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

**Example program**

num = [2,3,4,5]

sum = 0

for item in num:

sum = sum+item

print("the sum is",sum)

**output**

the sum is 14

**range()function**

We can use range() function in for loop to iterate through a sequence of number. It can be combined with the len() function to iterate through a sequence using indexing. len() function is used to find the length of a string or number of elements in a list, tuple, set etc.

**Example program**

flowers = ['rose','lotus','lily']

for i in range(len(flowers)):

print("current flower: ",flowers[i])

**output**

current flower: rose

current flower: lotus

current flower: lily

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 . We can also define the start ,stop and step\_size as range(start, stop, step\_size). The default value of step\_size is 1, if not provided. This function does not store the values in memory. It keeps track of the start, stop, step\_size and generate the next number.

**Example program**

for num in range(2,10,2):print("number = ",num)

**output**

number = 2

number = 4

number = 6

number = 8

**while loop**

The while loop in python is used to iterate over a block of code as long as the test expression is true. We generally use this loop when we don't know the number of times to iterate in advance. Below figure shows the flow chart of a while loop.

**syntax**

while test\_expression:

body of while

In while loop, test expression is checked first. the body of the loop is entered only if the test condition is true. After one iteration the test expression is checked again. This process is continued until the test expression evaluate to false. In python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line shows the end.

**Example program**

n = input("enter the limit:")

sum = 0

i=1

while(i<=n):

sum= sum+i

i = i+1

print("the sum of first",n,"natural numbers is",sum)

**output**

enter the limit:5

the sum of first 5 natural numbers is 15

**Nested loops**

Sometimes we need to place a loop inside another loop. This is called nested loop. We can have nested loops for both while and for.

syntax for nested for loop

for iterating\_variable in sequence:

for iterating\_variable in sequence:

statment(s)

statemnet(s)

syntax for nested while loop

while expression:

while expression:

statment(s)

statemnet(s)

**Example program**

import math

n = input("enter a limit:")

for i in range(1,n):

k=int(math.sqrt(n))

for j in range(2,K+1):

if i%j==0:break

else: print(i)\

**output**

enter the limit:12

1

2

3

5

7

11

The limit entered by the user is stored in n. To find whether a number is prime, the logic used si to divide that number from 2 to square root of that number. If the remainder of this division is zero at any time, that number is skipped and moved to next number. A complete division shows that the number is not prime. If the remainder is not zero at any time, it shows that it is a prime number.

The outer for loop starts from 1 to the input entered by the user. initially i=1. A variable k is used to store the half of the number. The inner for loop is used to find whether the number is completely divisible by any number between 2 and half of that number(k). If it is completely divisible by any number between 2 and k, the number is not a prime, else the number is considered prime. The same steps are repeated until the limit entered by the user is reached.

**Control Statements**

Control statements change the execution from normal sequence. Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. the break statement and continue statements are used in these cases. Python supports following three control statements.

1. break

2. continue

3. pass

**break statement**

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If it is nested loop, break will terminate the innermost loop. it can be used for both for and while loops. following figure shows the flow chart of break statement

**Example program**

for i in range(2,10,2):

if (i==6): break

print(i)

print("end of program")

**output**

2

4

end of program

**Continue statement**

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continue on with the next iteration. Continue returns the control to the beginning of the loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue Statement can be used in both while and for loop. Following figure shows the flow chart of continue statement.

**Example program**

for letter in 'abcd':

if(letter == 'c'):continue

print(letter)

**Output**

a

b

d

**pass statement**

In python programming, pass is a null statement. The difference between a comment and pass statement in python is that, while the interpreter ignores a comment entirely, pass is not ignored. But nothing happens when it is executed. It results in no operation.

It is used as a placeholder. Suppose we have a loop or a function that is not implemented yet, but want to implement it in the future. The function or loop cannot have an empty body. The interpreter will not allow this. So, we use the pass statement to construct a body that does nothing.

**Example**

for val in sequence:

pass

Questions:

1. Write a program to find GCD of 2 numbers
2. Write a program to find the factorial of a number
3. Write a program to generate fibonacci series of N terms
4. Write a program to count the number of vowels.
5. Write a program to find the sum of all items in a list.
6. write a program that prints the number from 1 to 20. But for multiples of three print "Fizz" instead of the number and for the multiples print "Buzz". For numbers which are multiple of both three and five print "FizzBuzz".
7. Write a python program which accept the user's name and print them in reverse order with a space between them.
8. Write a python program to count a number in a given list.
9. Write a python program to check whether a specified value is contained in a group of values.

Write a python program to find numbers between 100 and 400(both included)where each digit of a number is an even number. The numbers obtained should be printed in a comma-separated sequence.