1. **Introduction to python**

Python implementation began in December 1989by [**Guido van Rossum**](https://en.wikipedia.org/wiki/Guido_van_Rossum) **released in**  1991. Python we refer commonly as language is Cpython(original and most-maintained implementation of Python)。Other examples of python are **Jython, IronPython, Pypy, Stackless Python, ActiveState ActivePython, Pythonxy, Portable Python:, Anaconda Python,**

The python code you write is compiled into python bytecode, which creates file with extension *.pyc*. If compiles, again question is, why not compiled language.

*Note that this isn't compilation in the traditional sense of the word. Typically, we’d say that compilation is taking a high-level language and converting it to machine code. But it is a compilation of sorts. Compiled in to intermediate code not into machine code (Hope you got it Now).*

Back to the execution process, your bytecode, present in pyc file, created in compilation step, is then executed by appropriate virtual machines, in our case, the CPython VM (actually we call it interpreter, right?).

|  |
| --- |
| .com/ |
| **Execution of Python Code** |

1. Installing and running python program
2. **Setting up python path**

whereis python

export PATH="/usr/local/bin/python:$PATH"

1. **First Python Program**

$python

>>> name='abhay'

>>> empid=123

>>> print 'Hello ',name, 'your empid is : ', empid

Hello abhay your empid is : 123

OR

#!/bin/python

#Python Hello program

'''

This is a multiline

comment

'''

name='abhay'

empid=123

print 'Hello Mr.', name, 'your empid is: ', empid

1. **User Input & Program with them**

#!/bin/python

print "Hello World!"

name = raw\_input("Enter Name : ")

emp\_id = raw\_input("Enter Emp\_id : ")

print "Hello {} your Emp id is {}".format(name, emp\_id)

print "Hello %s your Emp id is %d" % (name, int(emp\_id))

print "Hello", name, "your Emp id is", emp\_id

**input() :**interprets and evaluates the input which means that if user enters integer,an integer will be returned ,if user enters string,string is returned.

**raw\_input():**raw\_input() takes exactly what user typed and passes it back as string .It doesn’t interprets the user input.Even an integer value of 10 is entered or a list is entered its type will be of string only.

1. **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 is a case sensitive programming language.

1. Python Keywords

|  |  |  |
| --- | --- | --- |
| 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 |

1. **Lines and Indentation**
2. **Multiline Statements**

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

>>> print "Hello....\

... welcome to python \

... programming"

Hello....welcome to python programming

>>>

1. **Multiple Statements on a Single Line**

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block.

>>> a=1;b=2;c=a+b;print c;

3

>>>

1. **Quotation in Python**

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

1. **Comments in python**

A hash sign (#) that is not inside a string literal begins a comment.

1. **Command Line Arguments**

The Python **sys** module provides access to any command-line arguments via the **sys.argv**. This serves two purposes:

 sys.argv is the list of command-line arguments.

 len(sys.argv) is the number of command-line arguments.

>>> import sys

>>> dir(sys)

['\_\_displayhook\_\_', '\_\_doc\_\_', '\_\_excepthook\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_stderr\_\_', '\_\_stdin\_\_', '\_\_stdout\_\_', '\_clear\_type\_cache', '\_current\_frames', '\_getframe', '\_mercurial', '\_multiarch', 'api\_version', 'argv', 'builtin\_module\_names', 'byteorder', 'call\_tracing', 'callstats', 'copyright', 'displayhook', 'dont\_write\_bytecode', 'exc\_clear', 'exc\_info', 'exc\_type', 'excepthook', 'exec\_prefix', 'executable', 'exit', 'flags', 'float\_info', 'float\_repr\_style', 'getcheckinterval', 'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'gettrace', 'hexversion', 'last\_traceback', 'last\_type', 'last\_value', 'long\_info', 'maxint', 'maxsize', 'maxunicode', 'meta\_path', 'modules', 'path', 'path\_hooks', 'path\_importer\_cache', 'platform', 'prefix', 'ps1', 'ps2', 'py3kwarning', 'pydebug', 'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version\_info', 'warnoptions']

>>> help(sys)

>>> help(sys.exit)

#!/bin/python

import sys

print "Number of arguments =", len(sys.argv)

print "Argument List =", str(sys.argv)

arg1 = sys.argv[0]

arg2 = sys.argv[1]

arg3 = sys.argv[2]

arg4 = sys.argv[3]

print "Name of script =", arg1

print "First argument =", arg2

print "Second argument =", arg3

print "Third argument =", arg4

1. **Variable types**

Variables are nothing but reserved memory locations to store values. This means when you create a variable, you reserve some space in memory.

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.

Python has five standard data types:

 Numbers

 String

 List

 Tuple

 Dictionary

1. Python Numbers,

|  |  |  |
| --- | --- | --- |
| **int** | **long** | **Float** |

a=11111111111111111

b=2.222222222222222

c=a+b

print c

del a

del b

>>> a=111111111111111111111111

>>> b=222222222222222222222222

>>> c=a+b

>>> print c

333333333333333333333333

>>> import sys

>>> sys.maxint

2147483647

>>> 2147483648

2147483648L

>>> 10\*\*100

10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000L

>>>

1. Python String

>>> str = "welcome to Python"

>>> print str

welcome to Python

>>> print len(str)

17

>>> print str[0]

w

>>> print str[2:7]

lcome

>>> print str[2:]

lcome to Python

>>> print str[:5]

welco

>>> print str[-3]

h

>>> print str[-3:]

hon

>>> print str[:-3]

welcome to Pyt

>>> print str \* 2

welcome to Pythonwelcome to Python

>>> print str + "Hello World"

welcome to PythonHello World

>>> print str[30]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: string index out of range

>>>

1. Python tuples

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as **read-only** lists.

#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print tuple # Prints complete list

print tuple[0] # Prints first element of the list

print tuple[1:3] # Prints elements starting from 2nd till 3rd

print tuple[2:] # Prints elements starting from 3rd element

print tinytuple \* 2 # Prints list two times

print tuple + tinytuple # Prints concatenated lists

1. Python List

#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print list # Prints complete list

print list[0] # Prints first element of the list

print list[1:3] # Prints elements starting from 2nd till 3rd

print list[2:] # Prints elements starting from 3rd element

print tinylist \* 2 # Prints list two times

print list + tinylist # Prints concatenated lists

1. Python Dictionary

Python's dictionaries are kind of hash table type.

#!/usr/bin/python

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one'] # Prints value for 'one' key

print dict[2] # Prints value for 2 key

print tinydict # Prints complete dictionary

print tinydict.keys() # Prints all the keys

print tinydict.values() # Prints all the value

print tinydict.items()

for key in sorted(d.keys()):

print ‘key: ‘, key. ‘->’ , d[key]

1. **Operators**

Python language supports the following types of operators.

 Arithmetic Operators

+,-,\*,/,%

 Comparison (Relational) Operators

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

 Assignment Operators

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

(x, y) = (1, 2)

 Logical Operators

and, or, not

 Bitwise Operators

$, | , !, ^, <<, >>

>>> a=8

>>> b=1

>>> a & b

0

>>> a | b

9

>>> a ^ b

9

>>> a >> 1

4

>>> a << 1

16

>>>

 Membership Operators

in, not in

>>> a=[1,2,3,4]

>>> 2 in a

True

>>> 5 in a

False

>>>

 Identity Operators

is, is not

The operators is and is not test for object identity: x is y is true if and only if x and y are the same object.

>>> a=[1,2,3]

>>> print a

[1, 2, 3]

>>> b=[1,2,3]

>>> print b

[1, 2, 3]

>>> a is b

False

>>> a=1

>>> b=1

>>> a is b

True

>>>

>>> a =1

>>> b=1

>>> a is b

True

>>> a=[1,2,3]

>>> b=[1,2,3]

>>>

>>>

>>> a is b

False

>>>

>>> a == b

True

DECISION MAKING

1. If

if expression:

statement(s)

>>> a=10

>>> b=5

>>> if a > b:

... print "a is bigger"

...

a is bigger

>>>

1. If Else

#!/bin/python

x = 5

if (x % 2 == 0):

print "[%d] is even" % x

else:

print "[%d] is odd" % x

#!/bin/python

import sys

x = int(sys.argv[1])

if (x % 2 == 0):

print "[%d] is even" % x

else:

print "[%d] is odd" % x

1. if else elif

#!/bin/python

import sys

num = sys.argv[1]

num = int(num)

if(num == 1):

print "ONE"

elif(num == 2):

print "TWO"

elif(num == 3):

print "THREE"

elif(num == 4):

print "FOUR"

elif(num == 5):

print "FIVE"

elif(num == 7):

print "SEVEN"

elif(num == 8):

print "EIGHT"

elif(num == 9):

print "NINE"

else:

print "Wrong Input"

Example : calc.py

LOOPS

1. while Loop

count = 0

while count < 5:

print count

count += 1

table\_no = 5

i = 1

while i <= 10:

print '%d \* %d = %d' % (table\_no, i, table\_no\*i)

i += 1

1. Infinite loop

#!/usr/bin/python

var = 1

while var == 1 : # This constructs an infinite loop

num = raw\_input("Enter a number :")

print "You entered: ", num

1. else with loop

#!/usr/bin/python

count = 0

while count < 5:

print count, " is less than 5"

count = count + 1

else:

print count, " is not less than 5"

1. for loop

for iterating\_var in sequence:

statements(s)

#!/bin/python

for i in [1, 2, 3, 4, 5]:

print i

for x in range(5):

print x

for x in range(3, 6):

print x

for x in range(3, 8, 2):

print x

str='hello world'

for char in str:

print char

items=[10, 'abc', 20, 'hello', 5.2]

for item in items:

print item

#!/usr/bin/python

for letter in 'Python': # First Example

print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # Second Example

print 'Current fruit :', fruit

Examples: Table.py, NestedTable.py , prime.py and SysInfo.py

1. Nested Loop
2. Break, continue and pass statements

#!/bin/python

count = 0

while count <= 5:

if count == 3:

break

print count

count += 1

#!/bin/python

count = 0

while count <= 5:

if count == 3:

continue

print count

count += 1

#!/bin/python

count = 0

while count <= 5:

if count == 3:

pass

print count

count += 1

1. File/IO

**Printing to the Screen using** “print”

**Reading Keyboard Input**

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are:

 raw\_input

The *raw\_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

 input

The *input([prompt])* function is equivalent to raw\_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

#!/usr/bin/python

str = input("Enter your input: ");

print "Received input is : ", str

This would produce the following result against the entered input:

Enter your input: [x\*5 for x in range(2,10,2)]

Recieved input is : [10, 20, 30, 40]

**Opening and Closing Files**

file object = open(file\_name [, access\_mode][, buffering])

**buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default (default behavior).

# Open a file

fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name

print "Closed or not : ", fo.closed

print "Opening mode : ", fo.mode

print "Softspace flag : ", fo.softspace

f = open(‘file.txt’, rU):

for line in f:

print line

f.close()

f = open(‘file.txt’, rU):

line =f.read()

print line

f.close()

1. Reading and Writing Files

Example : readFile.py , readFileusingWith.py, userList.py, filterData.py

1. Working with OS module Rename, Remove, Make Dir, Del Dir
2. **Functions**

>>> def myfunc():

... print 'This is my function'

... print 2+3

...

>>>

>>> myfunc()

This is my function

5

>>>

**Function Arguments**

You can call a function by using the following types of formal arguments:

 Required arguments

>>> def sum(x,y):

... print 'Sum = ', x + y

...

>>>

>>> sum(2, 3)

Sum = 5

>>>

 Keyword arguments

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

def printinfo( name, age ):

"This prints a passed info into this function"

print "Name: ", name;

print "Age ", age;

return;

# Now you can call printinfo function

printinfo( age=50, name="miki" );

 Default arguments

def printinfo( name, age = 35 ):

"This prints a passed info into this function"

print "Name: ", name;

print "Age ", age;

return;

# Now you can call printinfo function

printinfo( age=50, name="miki" );

printinfo( name="miki" );

 Variable-length arguments

>>> def sum(\*nums):

... sum=0

... for num in nums:

... sum = sum + num

... print 'Sum = ', sum

...

>>> sum(2,3,4,5)

Sum = 14

>>> sum(2,3)

Sum = 5

**Return statement**

>>> def sum(x,y):

... total = x + y

... return total

...

>>> sum=sum(2,3)

>>> print sum

5

>>>

**Global vs. Local variables**

1. Module
2. Class and Object Working
3. Inheritance
4. Exception Handling
5. Threading Concepts and Multithreading in Python

**Day2 Part2**

Import os

Filenames = os.listdir(path)

for filename in filenames:

path =os.path.join(path, filename)

print path

print os.path.abspath(path)

>>> import shutil

>>> shutil.copy(source,dest)

>>> help(commands.getoutput)

>>> help(commands.getstatusoutput)

>>>

>>> (status,out) = commands.getstatusoutput('ls -l')

>>> print status

0

>>> print out

total 56

drwxr-xr-x 2 apuser apuser 4096 Dec 6 10:47 Desktop

drwxr-xr-x 2 apuser apuser 4096 Aug 31 2016 Documents

drwxr-xr-x 2 apuser apuser 4096 Aug 31 2016 Downloads

-rw-r--r-- 1 apuser apuser 8980 Aug 31 2016 examples.desktop

a = [1, 2, 3, 4]

[num \* num for num in a if num > 2

[9, 16]