

Python

Introduction and Basics

Overview

What is Python

Why learn python

Where is Python used

Print() and strings

Hello world

Python datatypes

What is Python

High level Object Oriented Programming language

Created By **Guido Van Rossum** - 1989

Named after **Monty Python's flying circus**

Multi-purpose (GUI,Scripting,Web,scientific computations etc)

Interpreted

Why Python

Simple and Readable

Powerful and Scalable

English-like commands

Large Standard Library

Strongly typed and Dynamically typed

Why Python

Easy prototyping

Interactive mode enables easy debugging

Automatic garbage collection

Easy integration with c,c++,java,com,corba and activeX

Who uses Python

Google

Dropbox

IRobot

Instagram

Bittorrent

Nasa , Mozilla, Pinterest...and list goes on....

Python Basics

- Python 2.x and 3.x
- Installation
- Hello world program
- `print()` function and strings
- Variables
- Standard data types (Number, Strings, Lists, Tuples, Sets, Dictionaries)

Python Installation

<https://www.python.org/downloads/>

Download the latest version of 3.x series (currently 3.6.1)

For windows [Windows x86-64 executable installer](#)

Other platforms unzip and run the setup.py file.

Hello world!

Python 2.x

```
print "Hello World!"
```

Python 3.x

```
print ("Hello World!")
```

To Launch Interactive python shell -> Type **python** on your terminal

Store files with .py extension and execute using **python filename.py**

print() in 3.x

```
print(value1, value2.... , sep=' ',end='\n', file=sys.stdout,flush=False)
```

```
print(1,2,3,4, sep=' . ',end=' ')
```

```
>>> 1 . 2 . 3 . 4
```

```
print ('a','b','c',sep=' ,, ',end='\n\n')
```

```
a ,, b ,, c
```

Exercise 1 : print zen of python with punctuations intact.

String

```
text = "This is a string"  text = 'single quote string'
```

```
text = """This is a
```

```
multi-line string
```

```
"""
```

Exercise 2: Print **I am 6'2 tall.**

String

Exercise 2 Solution:

```
print ("I am 6'2\" tall.")
```

escape double-quote inside string

```
'I am 6\'2 tall.'
```

escape single-quote inside string

Comments

single line comment

For multi-line comment use triple quotes ''' ''' or """ """

```
text = '''
```

This is a Multi-line text not comment

```
'''
```

```
''' text = 'this is a multi-line comment' '''
```

Variables

```
name = "Rob" #string
```

```
age_years = 20 #integer
```

```
number = 20.0 #float
```

```
a,b,c = 1,2,'Rob' #multiple data types
```

```
a,b = 1,2,3 >>> error
```

```
text = 20 >>> str(text) >>> '20'
```

String methods

- `s.lower()`, `s.upper()` – returns the lowercase or uppercase version of the string
- `s.capitalize()`, `s.title()` – first letter will be capital, words start with uppercase.
- `s.strip()` – returns a string with whitespace removed from the start and end
- `s.isalpha()`/`s.isdigit()`/`s.isspace()`... – tests if all the string chars are in the various character classes
- `s.startswith('other')`, `s.endswith('other')` – tests if the string starts or ends with the given other string
- `s.find('other')` – searches for the given other string (not a regular expression) within `s`, and returns the first index where it begins or -1 if not found
- `s.replace('old', 'new')` – returns a string where all occurrences of 'old' have been replaced by 'new'
- `s.split('delim')` – returns a list of substrings separated by the given delimiter. The delimiter is not a regular expression, it's just text. `'aaa,bbb,ccc'.split(',')` -> `['aaa', 'bbb', 'ccc']`. As a convenient special case `s.split()` (with no arguments) splits on all whitespace chars.
- `s.join(list)` – opposite of `split()`, joins the elements in the given list together using the string as the delimiter. e.g. `'---'.join(['aaa', 'bbb', 'ccc'])` -> `aaa---bbb---ccc`

String Formatting

```
print >>> "we have 10 apples and 20 oranges"
```

```
name = apples
```

```
no_oran = 20
```

```
print "we have 10 %s and %d oranges" %(name,no_oran)
```

```
>>> we have 10 apples and 20 oranges
```

```
print "we have 10 { } and { } oranges".format(name,no_oran)
```

```
>>> we have 10 apples and 20 oranges
```


String concatenation

```
text = 'hello'+'world'; print text >>> 'helloworld'
```

```
'hello ' +'world' >>> 'hello world'
```

```
'give me' + 5 >>> error
```

```
'give me' + str(5) >>> give me5
```

```
'give me',5 >>> ('give me', 5) #tuple
```

```
'hello' * 3 >>> hellohellohello # string multiplication
```

String Slicing

```
fruit = 'apple'
```

```
fruit[0:3] >>> 'app' # slicing index[start,stop,step]
```

```
fruit[:-1] >>> 'appl'
```

```
fruit[-1] >>> 'e' , fruit[3] >>> 'l' , fruit[-2] >>> 'l'
```

```
fruit[0::2] >>> 'ape'
```

Numbers

```
year = 2017
```

```
year = int("2017") >>> 2017
```

```
X = 20.7 (float)
```

```
X = float('20') >>> 20.0
```

```
x,y,z = 3,6,1
```

```
max(x,y,z) >>> 6 similarly min(x,y,z) >>> 1
```

Operators

Arithmetic operator + - * / ** %

Comparison operator == != < > <= >=

Assignment operator += -= *= /= %= (eg $c+=a \Rightarrow c = c+a$)

Membership operator \Rightarrow in , not in

Identity Operator \Rightarrow is, is not #later

Lists

```
names = ['Bob', 'Alice', 'Mac', 'Sam']
```

```
names.append('alex')
```

```
names.insert(1, 'John')
```

```
print len(names)
```

```
#slicing [start:end:step]
```

```
names[0:3]
```

Lists Functions and Methods

`len(list)`

`max(list)`

`min(list)`

`list(seq)`

`list.append(obj)`

`list.count(obj)`

Lists Functions and Methods

`list.index(obj)`

`list.insert(index,obj)`

`list.pop(obj)`

`list.remove(obj)`

`list.reverse()`

`list.sort()`

Tuples

Immutable

```
tup1 = (1,2,'a','b')
```

```
tup2 = 1,2,'a','b'
```

```
tup3 = tup1+tup2
```

```
max(),min(),len(),tuple()
```

```
del tup2
```


Dictionary

```
dict = {'name' : 'Alex', 'age' : 12, 'name2' : 'bob', 'age2' : 20}
```

```
dict2 = {'name3' : 'john', 'age3' : 16}
```

```
dict.update(dict2)
```

```
dict['name3'] ⇒ 'john'
```

```
dict.keys()
```

Dictionary

`dict.items()`

`len(dict)`

`dict.values()`

`dict.get('key')`

Sets

Sets are lists with no duplicate entries.

They can be used to find intersections and differences between other sets.

Ex : a = [10,2,3,4,5,5,2,1]

```
print (set(a)) >>> {1,2,3,4,5,10} # orders
```

a.clear() - All elements will removed from a set. >>> set()

a.add(12) - A method which adds an element, which has to be immutable, to a set. >>> {1,2,3,4,5,10,12}

Set Methods

`intersection()` - elements which are contained in both sets is returned.

`union()` - the collection of all the sets.

```
x = {"a","b","c","d","e"}
```

```
y = {"c","d","e","f","g"}
```

```
>>> x.intersection(y) >>> set(['c', 'e', 'd'])
```

```
a = set(["Jake", "John", "Eric"])
```

```
b = set(["John", "Jill"])
```

```
a.union(b) >>> set(['Jill', 'Jake', 'John', 'Eric'])
```

```
x.difference(y) >>> {'a', 'b'}
```

While loop

```
condition = 1
```

```
while condition < 5:  
    print(condition)  
    condition += 1
```

#2

```
while True:  
    print('do stuff!!')    # stop by ctrl + c
```

For Loop

Iterate through something

```
example_list = [1,2,3,4,5,6]
```

```
for each in example_list:
```

```
    print (each*2)
```

```
>>> 2,4,6,8,10,12
```

For Loop

```
for i in range(5,10):
```

```
    print (i)
```

```
>>> 5,6,7,8,9
```

If Statement

```
x = 8  
y = 17
```

```
if x > y:  
    print('x is greater than y')
```

```
x = 9  
y = 19  
if x < 60:  
    print('x is less than 60')  
else:  
    print('x is not less than 60')
```


If Else Statement

```
x = 15
y = 28
if x > 60:
    print('x is greater than 60')
else:
    print('x is not less than 60')
```

```
x = 7
y = 10
if x < 60:
    print('x is less than 60')
else:
    print('x is not less than 60')
```

If Elif Else Statement

```
x = 10  
y = 20  
z = 32
```

```
if x > y:  
    print('x is greater than y')  
elif x < z:  
    print('x is less than z')  
else:  
    print('if and elif never ran...')
```

If Elif Else Statement

```
x = 5
```

```
y = 10
```

```
z = 22
```

```
if x > y:
```

```
    print('x is greater than y')
```

```
elif x > z:
```

```
    print('x is less than z')
```

```
else:
```

```
    print('if and elif never ran...')
```

Control statements for loops

Loop control statements change execution from its normal sequence.

break - It terminates the current loop and resumes execution at the next statement

```
for letter in 'Python':      # First Example
    if letter == 'h':
        break
    print 'Current Letter :', letter
```

```
o/p -> Current Letter : P
Current Letter : y
Current Letter : t
```

Control statements for loops

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.

```
for letter in 'Python':      # First Example
    if letter == 'h':
        continue
    print 'Current Letter :', letter
```

```
var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print 'Current variable value :', var
print "Good bye!"
```

o/p -> Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n

Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!

Control statements for loops

`pass` => The `pass` statement is used when a statement is required syntactically but you do not want any command or code to execute.

```
for letter in 'Python':  
    if letter == 'h':  
        pass  
        print 'This is pass block'  
    print 'Current Letter :', letter  
  
print "Good bye!"
```

```
o/p -> Current Letter : P  
Current Letter : y  
Current Letter : t  
This is pass block  
Current Letter : h  
Current Letter : o  
Current Letter : n  
Good bye!
```

in , not in

```
x = [1,3,4,8]
```

If 5 in x:

```
print ('5 in list x') >>> nothing
```

If 5 not in x:

```
Print ('5 not in list x') >>> 5 not in list x
```

Input

To take an input from user

```
>>> input('enter something: ')
```

```
>>>enter something: python
```

```
>>>'python'
```


Function

Functions are lines of code that perform a task and you use more than once. (Example adding 2 numbers).

In Python we define a function using '**def**' keyword. #define a function

```
def example():  
    print('this is our function')  
    z = 4 + 12  
    print(z)
```

#call a function

```
example()
```

Function with parameters

Idea of function parameter is to define variables dynamically within a function.

```
def sum(num1,num2):  
    result = num1 + num2  
    print('num1 is', num1 , 'and num2 is', num2)  
    print(result)
```

```
sum(5,3) >>> num1 is 5 and num2 is 3
```

```
8
```

```
sum(5,3,4) >>> error
```

```
sum(num1=5,num2=3) >>> 8    sum(4,num1=3) >>> sum() got multiple values for argument 'num2'
```

Function parameter Defaults

Python supports default argument values, that is function arguments that can either be specified by the caller or left blank to automatically receive a predefined value.

```
def simple(num1, num2=5):  
    pass
```

Defaults parameters should be the last ones listed in the function's parameters.

```
def simple(num1=5, 2):  
    pass #error
```

Function parameters *args and **kwargs

*args and **kwargs allow you to pass a variable number of arguments to a function.

When you do not know before hand that how many arguments can be passed to your function by the user so in this case you use these two keywords.

*args is used to send a **non-keyworded** variable length argument list to the function.

```
def sample(*argv):
```

```
    for arg in argv:
```

```
        print "argument printed", arg
```

```
sample('Alex','Bob','Max')
```

Output

argument printed Alex

argument printed Bob

argument printed Max

Function parameters *args and **kwargs

****kwargs** allows you to pass **keyworded** variable length of arguments to a function. You should use ****kwargs** if you want to handle **named arguments** in a function

```
def greet_me(**kwargs):  
    if kwargs is not None:  
        for key, value in kwargs.items():  
            print "%s == %s" %(key,value)
```

o/p => greet_me(name="BOB")

name == BOB

Function parameters *args and **kwargs

```
def sample(**kwargs):
```

```
    for k,v in kwargs.items():
```

```
        print (k , v)
```

```
kwargs = {'args2':3, 'args2':2, 'args1':1}
```

```
sample(**kwargs)
```

Output

```
args1 1  
args2 2
```

Files

`open()`

`file object = open(file_name [, access_mode][, buffering])`

`File_name` => The name of the file that you want to create or access.

`Access mode` => The mode in which the file has to be opened, i.e., read, write, append etc.

`Buffering` => A chunk of data is read from the raw OS filestream into a buffer until it is consumed.

If set to 0 no buffering takes place

Files

`write()` - To put data in the file we invoke the write method on the file object.

`read()` - the read method reads data from the file.

`close()` - Closing the file tells the system that we are done writing and makes the file available for reading.

`readline()` - The readline method reads all the characters up to and including the next newline character.

`readlines()` - `readlines` returns all of the remaining lines as a list of strings.

Files

Open a file

```
f = open('hello.txt','w')
```

```
f.write('hello world in a file!')
```

```
f.close()
```

#read from a file

```
f = open('hello.txt','r').read()
```

```
print (f)
```

File using with

With the "With" statement, you get better syntax and exceptions handling.

It will automatically close the file

#Open hello.txt in write mode

with open('hello.txt', 'w') as file:

```
    file.write('Hi there!')
```

#Opening a file using with

with open("hello.txt", 'r') as file:

Use file to refer to the file object

```
    data = file.read()
```

```
    # do something with data
```

File Modes

r => Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb => Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+ => Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+ => Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w => Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

File Modes

wb => Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+ => Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+ => Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a => Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

File Modes

ab => Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+ => Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+ => Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

File Positions

```
# Open a file
f = open("new.txt", "r+")
str = f.read(10);
print "Read String is : ', str

# Check current position
position = f.tell();
print "Current file position : ', position

# Reposition pointer at the beginning once again
position = f.seek(0, 0);
str = f.read(10);
print "Again read String is : ', str
# Close opened file
fo.close()
```

File Methods

`tell()` => The `tell()` method tells you the current position within the file

`seek(offset[, from])` => changes the current file position

The `offset` argument indicates the number of bytes to be moved.

If `from` is set to 0 use the beginning of the file as the reference position.

1 means use the current position as the reference position.

if it is set to 2 then the end of the file would be taken as the reference position.

```
import os
```

```
os.remove("sample.txt")
```

File Methods

next() => Is used when a file is used as an iterator, typically in a loop.

the next() method is called repeatedly.

This method returns the next input line, or raises *StopIteration* when EOF is hit.

```
f = open("new.txt", "rw+")
```

```
for index in range(5):
```

```
    line = f.next()
```

```
    print "Line No %d - %s" % (index, line)
```

```
# Close open file
```

```
f.close()
```

new.txt

Assuming file has following 5 lines

This is 1st line

This is 2nd line

This is 3rd line

This is 4th line

This is 5th line

File Methods

output

Line No 0 - This is 1st line

Line No 1 - This is 2nd line

Line No 2 - This is 3rd line

Line No 3 - This is 4th line

Line No 4 - This is 5th line

Exercise

1. Print the first and last element of the list ['jaguar', 'Honda', 'Bmw', 'Maruti'].
2. Write a Python program to find whether a given number (accept from the user) is even or odd, print out ('This is an even number' or 'This is an odd number') message respectively to the user.
3. Write a program to find the number of 2's in the given list [1,2,3,4,2,12,43,2,4,1].
4. Write a Python program to concatenate following dictionaries and create a new dictionary.
`dict1={2:25, 4:10, 6:12} dict2={2:80, 14:38, 40:12} dict3={7:100, 3:50, 8:14}`
5. Write a Python program to iterate over dictionaries using for loops {'jaguar':4, 'Honda':3, 'Bmw':5, 'Maruti':3}
6. Write a function to take name as input from the user and output Hello, name

Exercises

7. Write a Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Sample Output : fizzbuzz 1 2 fizz 4 buzz

8. Write a Python program to check whether an alphabet is a vowel or consonant taking input from the user.

9. Write a Python program to convert month name to a number of days.

List of months: January, February, March, April, May, June, July, August, September, October, November, December

Input the name of Month: February

No. of days: 28/29 days

Exercises

10. Write a program to take numbers and check if each number is divisible by 2 if it's divisible by 2 multiply that number with 5 and if its not divisible by 2 multiply that number with 6.

take numbers in function call