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’

fruits[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 ⇒ c = c+a)

Membership operator ⇒ in , not in

Identity Operator ⇒ 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 opend 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 opend 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
```

Csv file

CSV literally stands for comma separated variable, where the comma is what is known as a "delimiter."

```
import csv

with open('example.csv') as csvfile:

    readCSV = csv.reader(csvfile,delimiter=',')

    dates = []

    colors = []

    for row in readCSV:

        color = row[3]

        date = row[0]

        dates.append(date)

        colors.append(color)

        print (row[0],row[1],row[2],)

    print (dates)

    print (colors)
```

Csv file

```
whatColor = input('what color do you wish to know the date of?')

coldex = colors.index(whatColor.lower())

theDate = dates[coldex]

print ('the date of', whatColor, 'is:', theDate)
```

example.csv

```
1/2/2014,5,8,red
1/3/2014,5,2,green
1/4/2014,9,1,blue
```

Csv file

o/p

```
('1/2/2014', '5', '8', 'red')
('1/3/2014', '5', '2', 'green')
('1/4/2014', '9', '1', 'blue')
['1/2/2014', '1/3/2014', '1/4/2014']
['red', 'green', 'blue']

what color do you wish to know the date of?'GreeN'
1
1/3/2014
('the date of', 'GreeN', 'is:', '1/3/2014')
```

Json file

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language.

Writing a json object to a file

```
Import json

with open('data.json', 'w') as outfile:

    json.dump(data, outfile)
```

Json file

data is a json object :

```
data = {
    "maps": [
        {
            "id": "house",
            "iscategorical": "0"
        },
        {
            "id": "house",
            "iscategorical": "0"
        }
    ],
     "masks": {
            "id": "bangalore"
        },
        "om_points": "value",
        "parameters": {
            "id": "bangalore"
        }
    }
```

Json file

Reading a json object from a file

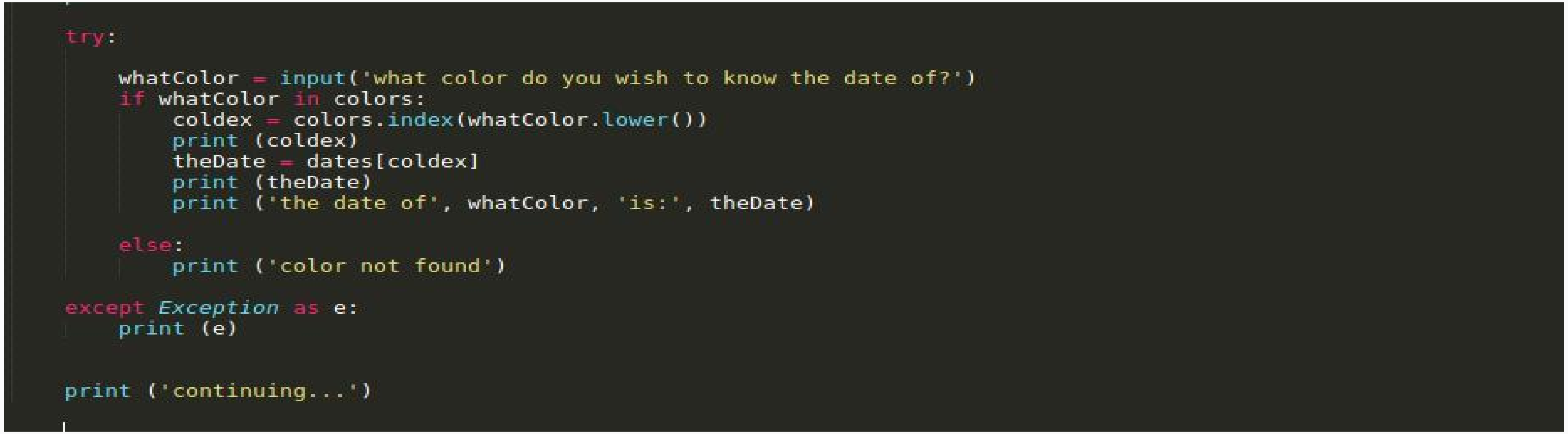
```
import json

with open('data.json') as data_file:

    data = json.load(data_file)
```

```
print (data["maps"][0]["id"])

>>> house

print (data["masks"]["id"])

>>> bangalore

print (data["om_points"])

>>> value
```

Try and except

Sometimes we want to execute an operation that might cause an exception, but we don’t want the program to stop. We can **handle the exception** using the try and except statements.

Previous example with exception handling



```
try:

    whatColor = input('what color do you wish to know the date of?')
    if whatColor in colors:
        coldex = colors.index(whatColor.lower())
        print (coldex)
        theDate = dates[coldex]
        print (theDate)
        print ('the date of', whatColor, 'is:', theDate)

    else:
        print ('color not found')

except Exception as e:
    print (e)


print ('continuing...')
```

List comprehensions

Useful way of replacing for loops

```
odds = [ x for x in range(50) if x%2 ]
```

```
odds = [ ]

for x in range(50):

    if x%2:

        odds.append(x)

print (odds)
```

Output

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49]
```

Module import syntax

when you import a module, you are basically loading that module into memory. Think of a module like a script with Python definitions and statements.

Many if not most modules are just a single python script. So, when you go to import it, you use the file name. This can help keep code clean and easy to read.

"as" statement are used in imports. This will allow you to basically rename the module to whatever you want.

```
import statistics

example_list = [5,2,5,6,1,2,6,7,2,6,3,5,5]

print(statistics.mean(example_list))
```

```
import statistics as s

print(s.mean(example_list))
```

Module import syntax

import each function within the module

```
from statistics import mean
# so here, we've imported the mean function only.
print(mean(example_list))

# and again we can do as
from statistics import mean as m
print(m(example_list))
```

```
#More functions

from statistics import mean, median
# here we imported 2 functions.
print(median(example_list))

from statistics import mean as m, median as d

print(m(example_list))
print(d(example_list))
```

Module import syntax

Import everything from statistics

```
from statistics import *

print(mean(example_list))
```

Create your own module

Modules are just Python scripts stored in your Lib or Lib/site-packages folder, or local to the script being run.

Create a file with examplemod.py

```
def ex(data):
    print(data)
```

```
import examplemod
examplemod.ex('test')
```

the examplemod.py file must be in the same directory as your running-program, or in your Python packages directory. If you are on a windows machine, this will be C:/python34/Lib/site-packages/

Decorators

Decorators are a way for us to "wrap" a function inside another function, without actually hard-coding it every time.

```
def add_wrapping(item):
    def wrapped_item():
        return 'a wrapped up box of {}'.format(str(item()))
    return wrapped_item
```

```
@add_wrapping
def gift():
    return 'a new gift!'
print(gift())
```

```
>>> a wrapped up box of a new gift!
```

Generators

A generator is simply a function which returns an object on which you can call next, such that for every call it returns some value, until it raises a StopIteration exception, signaling that all values have been generated. Such an object is called an *iterator*.using yield makes it a generator.

Example

```
def gen(n):
    yield n
    yield n+1
    yield n + 2
```

```
#call the function
g = gen(5)
next(g) >>> 5
next(g) >>> 6
next(g) >>> 7
next(g) >>> error
```

```
#next under the hood
for n in gen(5): ...
print(n)
```

Generators

python3 range() function.

```
#iterate through something 4 times

for i in range(4):    # if we had to iterate through 500000 entries

    print (i)
```

Range is a generator, so it's generating the values in order on-the-fly, it do not store in memory hence memory is managed.

Generators

Generator expression:

```
xyz = (i for i in range(50000000))
```

print (xyz) >>> generator object which has to be iterated over

Iteration can be done in three ways

```
for i in xyz:
    print (i)
```

Or convert it into list and access

```
print(list(xyz)[:5])
```

```
next(xyz)
```

Random module

random module contains a number of random number generators.

randint - #Generate integers between 1,10. The first value should be less than the second.
random.randint(1, 10)

randrange - #Generate a randomly selected element from range(start, stop, step)
random.randrange(start, stop[, step])

choice - #Generate a random value from the sequence sequence.
random.choice(['red', 'black', 'green']).

shuffle - #Shuffles the elements in list in place, so they are in a random order.

random.shuffle(list)

Enumerate function

enumerate() function adds a counter to an iterable.

for each element in cursor, a tuple is produced with (counter, element)

the for loop binds that to row_number and row

```
elements = ('apple', 'mango', 'kiwi')
```

```
for elem in elements:

    print elem

>>> apple

    mango

    kiwi
```

```
for count, elem in enumerate(elements):

 print count, elem

>>> 0 apple

    1 mango

    2 kiwi
```

Enumerate function

By default, enumerate() starts counting at 0

second integer argument can be specified to start from that number instead.

```
for count, elem in enumerate(elements, 42):
    print count, elem
```

output

```
42 apple
43 mango
44 kiwi
```

Object oriented programming in python

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

Example : House

An object is also called an instance of a class and the process of creating this object is called **instantiation**.

we define a class using the keyword class.

The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

Oops in python

Here is a simple class definition.

```
class MyNewClass:
    '''This is a docstring. I have created a new class'''
    pass
```

Attributes may be data or functions.

Special attributes begins with double underscores (__). For example, __doc__ gives us the docstring of that class

MyNewClass.__doc__   >>>   This is a docstring. I have created a new class

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

Oops in python

```
Example :

class MyClass:

    "This is my second class"

    a = 10

    def func(self):

        print('Hello')
```

```
# Output: 10

print(MyClass.a)

# Output: <function MyClass.func at
0x0000000003079BF8>

print(MyClass.func)

# Output: 'This is my second class'

print(MyClass.__doc__)
```

Oops in python

creating new object instances

```
ob = MyClass()
```

```
Ob.__doc__ >>> This is my second class
```

The first argument of the function in class must be the object (instance being created) itself. This is conventionally called self.

Constructors in Python

Class functions that begins with double underscore (__) are called special functions and have special meaning.

`__init__()` function is a special function that gets called whenever a new object of that class is instantiated.

This type of function is also called constructors.

We normally use it to initialize all the variables.

Class example

```
class ComplexNumber:

   def __init__(self,r = 0,i = 0):

      self.real = r

      self.imag = i

   def getData(self):

      print("{0}+{1}j".format(self.real,self.imag))
```

```
# Create a new ComplexNumber object

c1 = ComplexNumber(2,3)

# Call getData() function

# Output: 2+3j

c1.getData()
```

Class example

```
# Create another ComplexNumber object

# and create a new attribute 'attr'

c2 = ComplexNumber(5)

c2.attr = 10

# Output: (5, 0, 10)

print((c2.real, c2.imag, c2.attr))

print ('done here')

# but c1 object doesn't have attribute 'attr'

# AttributeError: 'ComplexNumber' object has no attribute 'attr'

c1.attr
```

```
o/p
2+3j
(5, 0, 10)
done here

Traceback (most recent call last):
  File "<stdin>", line 26, in <module>
    c1.attr
AttributeError: 'ComplexNumber' object has no
attribute 'attr'
```

Deleting Attributes and objects

attributes of an object can be created on the fly. We created a new attribute `attr` for object `c2` and we read it as well. But this did not create that attribute for object `c1`.

```
c1 = ComplexNumber(2,3)
>>> del c1.imag
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'imag'
```

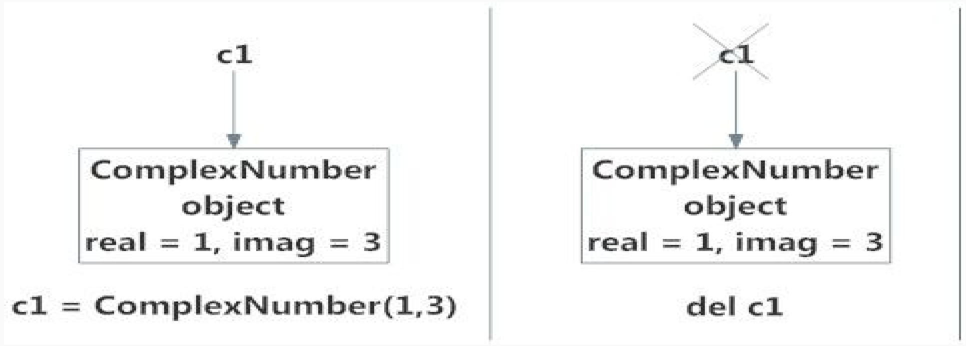
```
del ComplexNumber.getData
#delete the obj itself

c1 = ComplexNumber(1,3)
>>> del c1
```

Garbage collection

When we do `c1 = ComplexNumber(1,3)`, a new instance object is created in memory and the name `c1` binds with it.

On the command del c1, this binding is removed and the name `c1` is deleted from the corresponding namespace. The object however continues to exist in memory and if no other name is bound to it, it is later automatically destroyed.

This automatic destruction of unreferenced objects in Python is also called garbage collection.


c1
ComplexNumber object real = 1, imag = 3
c1 = ComplexNumber(1,3)
~~c1~~
ComplexNumber object real = 1, imag = 3
del c1

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