**Python Standard Datatypes:**

1. **Numbers b) String c) List d) Tuple e) Dictionary**
2. var1 = 1
3. var2 = 10

**Strings :**

#!/usr/bin/python3

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str \* 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

**Lists :**

#!/usr/bin/python3

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

**Tuples :**

#!/usr/bin/python3

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

tinytuple = (123, 'john')

print (tuple) # Prints complete tuple

print (tuple[0]) # Prints first element of the tuple

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

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

print (tinytuple \* 2) # Prints tuple two times

print (tuple + tinytuple) # Prints concatenated tuple

**Dictionary :**

#!/usr/bin/python3

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 values

**Data Conversion :**

**a)**

# Python code to demonstrate Type conversion

# using int(), float()

# initializing string

s = "10010"

# printing string converting to int base 2

c = int(s,2)

print ("After converting to integer base 2 : ", end="")

print (c)

# printing string converting to float

e = float(s)

print ("After converting to float : ", end="")

print (e)

**b) Convert string to tuple , list & set**

# Python code to demonstrate Type conversion

# using  tuple(), set(), list()

# initializing string

s = 'geeks'

# printing string converting to tuple

c = tuple(s)

print ("After converting string to tuple : ",end="")

print (c)

# printing string converting to set

c = set(s)

print ("After converting string to set : ",end="")

print (c)

# printing string converting to list

c = list(s)

print ("After converting string to list : ",end="")

print (c)

**c )**

# Python code to demonstrate Type conversion

# using  dict(), complex(), str()

# initializing integers

a = 1

b = 2

# initializing tuple

tup = (('a', 1) ,('f', 2), ('g', 3))

# printing integer converting to complex number

c = complex(1,2)

print ("After converting integer to complex number : ",end="")

print (c)

# printing integer converting to string

c = str(a)

print ("After converting integer to string : ",end="")

print (c)

# printing tuple converting to expression dictionary

c = dict(tup)

print ("After converting tuple to dictionary : ",end="")

print (c)

**Getting Calender for Month:**

#!/usr/bin/python3

import calendar

cal = calendar.month(2016, 2)

print ("Here is the calendar:")

print (cal)

**Getting Formatted Time :**

#!/usr/bin/python3

import time

localtime = time.asctime( time.localtime(time.time()) )

print ("Local current time :", localtime)

**Calling Function :**

#!/usr/bin/python3

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print (str)

return

# Now you can call printme function

printme("This is first call to the user defined function!")

printme("Again second call to the same function")

**Pass by Reference vs Value**

**Ex1 :**

#!/usr/bin/python3

# Function definition is here

def changeme( mylist ):

"This changes a passed list into this function"

print ("Values inside the function before change: ", mylist)

mylist[2]=50

print ("Values inside the function after change: ", mylist)

return

# Now you can call changeme function

mylist = [10,20,30]

changeme( mylist )

print ("Values outside the function: ", mylist)

**Ex2 :**

#!/usr/bin/python3

# Function definition is here

def changeme( mylist ):

"This changes a passed list into this function"

mylist = [1,2,3,4] # This would assi new reference in mylist

print ("Values inside the function: ", mylist)

return

# Now you can call changeme function

mylist = [10,20,30]

changeme( mylist )

print ("Values outside the function: ", mylist)

**Function Arguments :**

* Required arguments:

#!/usr/bin/python3

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print (str)

return

# Now you can call printme function

printme()

* Keyword arguments :

Ex1 :

#!/usr/bin/python3

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print (str)

return

# Now you can call printme function

printme( str = "My string")

Ex2 :

#!/usr/bin/python3

# Function definition is here

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 :

#!/usr/bin/python3

# Function definition is here

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

#!/usr/bin/python3

# Function definition is here

def printinfo( arg1, \*vartuple ):

"This prints a variable passed arguments"

print ("Output is: ")

print (arg1)

for var in vartuple:

print (var)

return

# Now you can call printinfo function

printinfo( 10 )

printinfo( 70, 60, 50 )

**Anonymous Functions:**

**Ex :** #!/usr/bin/python3

# Function definition is here

sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function

print ("Value of total : ", sum( 10, 20 ))

print ("Value of total : ", sum( 20, 20 ))

**Global & Local Variables:**

#!/usr/bin/python3

total = 0 # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

# Add both the parameters and return them."

total = arg1 + arg2; # Here total is local variable.

print ("Inside the function local total : ", total)

return total

# Now you can call sum function

sum( 10, 20 )

print ("Outside the function global total : ", total )

**Python3 Opening & Closing File:**

#!/usr/bin/python3

# Open a file

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

fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend file

fo.close()

**Exception Handling :**

#!/usr/bin/python3

try:

fh = open("testfile", "w")

fh.write("This is my test file for exception handling!!")

except IOError:

print ("Error: can\'t find file or read data")

else:

print ("Written content in the file successfully")

fh.close()

**Python Classes & Objects :**

#!/usr/bin/python3

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print ("Total Employee %d" % Employee.empCount)

def displayEmployee(self):

print ("Name : ", self.name, ", Salary: ", self.salary)

#This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

#This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

print ("Total Employee %d" % Employee.empCount)

**Built-In Class Attributes:**

* **\_\_dict\_\_** − Dictionary containing the class's namespace.
* **\_\_doc\_\_** − Class documentation string or none, if undefined.
* **\_\_name\_\_** − Class name.
* **\_\_module\_\_** − Module name in which the class is defined. This attribute is "\_\_main\_\_" in interactive mode.
* **\_\_bases\_\_** − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

#!/usr/bin/python3

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print ("Total Employee %d" % Employee.empCount)

def displayEmployee(self):

print ("Name : ", self.name, ", Salary: ", self.salary)

emp1 = Employee("Zara", 2000)

emp2 = Employee("Manni", 5000)

print ("Employee.\_\_doc\_\_:", Employee.\_\_doc\_\_)

print ("Employee.\_\_name\_\_:", Employee.\_\_name\_\_)

print ("Employee.\_\_module\_\_:", Employee.\_\_module\_\_)

print ("Employee.\_\_bases\_\_:", Employee.\_\_bases\_\_)

print ("Employee.\_\_dict\_\_:", Employee.\_\_dict\_\_ )

**Garbage Collection:**

#!/usr/bin/python3

class Point:

def \_\_init\_\_( self, x=0, y=0):

self.x = x

self.y = y

def \_\_del\_\_(self):

class\_name = self.\_\_class\_\_.\_\_name\_\_

print (class\_name, "destroyed")

pt1 = Point()

pt2 = pt1

pt3 = pt1

print (id(pt1), id(pt2), id(pt3)); # prints the ids of the obejcts

del pt1

del pt2

del pt3

**Class Inheritance :**

#!/usr/bin/python3

class Parent: # define parent class

parentAttr = 100

def \_\_init\_\_(self):

print ("Calling parent constructor")

def parentMethod(self):

print ('Calling parent method')

def setAttr(self, attr):

Parent.parentAttr = attr

def getAttr(self):

print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class

def \_\_init\_\_(self):

print ("Calling child constructor")

def childMethod(self):

print ('Calling child method')

c = Child() # instance of child

c.childMethod() # child calls its method

c.parentMethod() # calls parent's method

c.setAttr(200) # again call parent's method

c.getAttr() # again call parent's method

**Overriding Function :**

**Ex1 :**

#!/usr/bin/python3

class Parent: # define parent class

def myMethod(self):

print ('Calling parent method')

class Child(Parent): # define child class

def myMethod(self):

print ('Calling child method')

c = Child() # instance of child

c.myMethod() # child calls overridden method

**Ex2 :**

class Rectangle():

def \_\_init\_\_(self,length,breadth):

self.length = length

self.breadth = breadth

def getArea(self):

print self.length\*self.breadth," is area of rectangle"

class Square(Rectangle):

def \_\_init\_\_(self,side):

self.side = side

Rectangle.\_\_init\_\_(self,side,side)

def getArea(self):

print self.side\*self.side," is area of square"

s = Square(4)

r = Rectangle(2,4)

s.getArea()

r.getArea()

**Function Overloading:**

*#!/usr/bin/env python*

**class** Human:

**def** sayHello(self, name=None):

**if** name **is** **not** None:

**print** 'Hello ' + name

**else**:

**print** 'Hello '

*# Create instance*

obj = Human()

*# Call the method*

obj.sayHello()

*# Call the method with a parameter*

obj.sayHello('Guido')

**Example of class inheritance and method overriding in Python**

#!/usr/bin/env python

class man(object):

# name of the man

name = ""

def \_\_init\_\_(self, P\_name):

""" Class constructor """

self.name = P\_name

print("Here comes " + self.name)

def talk(self, P\_message):

print(self.name + " says: '" + P\_message + "'")

def walk(self):

""" This let an instance of a man to walk """

print(self.name + " walks")

# This class inherits from Man class

# A superman has all the powers of a man (A.K.A. Methods and Properties in our case ;-)

class superman(man):

# Name of his secret identity

secret\_identity = ""

def \_\_init\_\_(self, P\_name, P\_secret\_identity):

""" Class constructor that overrides its parent class constructor"""

# Invokes the class constructor of the parent class #

super(superman, self).\_\_init\_\_(P\_name)

# Now let's add a secret identity

self.secret\_identity = P\_secret\_identity

print("...but his secret identity is '" + self.secret\_identity + "' and he's a super-hero!")

def walk(self, P\_super\_speed = False):

# Overrides the normal walk, because a superman can walk at a normal

# pace or run at the speed of light!

if (not P\_super\_speed): super(superman, self).walk()

else: print(self.secret\_identity + " run at the speed of light")

def fly(self):

""" This let an instance of a superman to fly """

# No man can do this!

print(self.secret\_identity + " fly up in the sky")

def x\_ray(self):

""" This let an instance of a superman to use his x-ray vision """

# No man can do this!

print(self.secret\_identity + " uses his x-ray vision")

# Declare some instances of man and superman

lois = man("Lois Lane")

jimmy = man("Jimmy Olsen")

clark = superman("Clark Kent", "Superman")

# Let's puth them into action!

print("\n--> Let's see what a man can do:\n")

jimmy.walk()

lois.talk("Oh no, we're in danger!")

print("\n--> Let's see what a superman can do:\n")

clark.walk()

clark.talk("This is a job for SUPERMAN!")

clark.walk(True)

clark.fly()

clark.x\_ray()

**Data Hiding :** An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then will not be directly visible to outsiders.

**Ex :**

#!/usr/bin/python3

class JustCounter:

\_\_secretCount = 0

def count(self):

self.\_\_secretCount += 1

print (self.\_\_secretCount)

counter = JustCounter()

counter.count()

counter.count()

print (counter.\_\_secretCount)

**Regular Expressions :**

### **Basic patterns that match single chars**

|  |  |
| --- | --- |
| **S.No.** | **Expression & Matches** |
| 1 | **a, X, 9, <**  ordinary characters just match themselves exactly. |
| 2 | **. (a period)**  matches any single character except newline '\n' |
| 3 | **\w**  matches a "word" character: a letter or digit or underbar [a-zA-Z0-9\_]. |
| 4 | **\W**  matches any non-word character. |
| 5 | **\b**  boundary between word and non-word |
| 6 | **\s**  matches a single whitespace character -- space, newline, return, tab |
| 7 | **\S**  matches any non-whitespace character. |
| 8 | **\t, \n, \r**  tab, newline, return |
| 9 | **\d**  decimal digit [0-9] |
| 10 | **^**  matches start of the string |
| 11 | **$**  match the end of the string |
| 12 | **\**  inhibit the "specialness" of a character. |

### **Compilation flags**

Compilation flags let you modify some aspects of how regular expressions work. Flags are available in the re module under two names, a long name such as **IGNORECASE** and a short, one-letter form such as I.

|  |  |
| --- | --- |
| **S.No.** | **Flag & Meaning** |
| 1 | **ASCII, A**  Makes several escapes like \w, \b, \s and \d match only on ASCII characters with the respective property. |
| 2 | **DOTALL, S**  Make, match any character, including newlines |
| 3 | **IGNORECASE, I**  Do case-insensitive matches |
| 4 | **LOCALE, L**  Do a locale-aware match |
| 5 | **MULTILINE, M**  Multi-line matching, affecting ^ and $ |
| 6 | **VERBOSE, X (for ‘extended’)**  Enable verbose REs, which can be organized more cleanly and understandably |

## **The match Function**

This function attempts to match RE *pattern* to *string* with optional *flags*.

Here is the syntax for this function −

re.match(pattern, string, flags = 0)

**Ex1:**

#!/usr/bin/python3

import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.\*) are (.\*?) .\*', line, re.M|re.I)

if matchObj:

print ("matchObj.group() : ", matchObj.group())

print ("matchObj.group(1) : ", matchObj.group(1))

print ("matchObj.group(2) : ", matchObj.group(2))

else:

print ("No match!!")

**Ex2 : Matching Vs Searching:**

#!/usr/bin/python3

import re

line = "Cats are smarter than dogs";

matchObj = re.match( r'dogs', line, re.M|re.I)

if matchObj:

print ("match --> matchObj.group() : ", matchObj.group())

else:

print ("No match!!")

searchObj = re.search( r'dogs', line, re.M|re.I)

if searchObj:

print ("search --> searchObj.group() : ", searchObj.group())

else:

print ("Nothing found!!")

**Ex3: Search & Replace**

#!/usr/bin/python3

import re

phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments

num = re.sub(r'#.\*$', "", phone)

print ("Phone Num : ", num)

# Remove anything other than digits

num = re.sub(r'\D', "", phone)

print ("Phone Num : ", num)

## **Regular Expression Modifiers: Option Flags**

|  |
| --- |
| **re.I**  Performs case-insensitive matching. |
| 2 | **re.L**  Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B). |
| 3 | **re.M**  Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string). |
| 4 | **re.S**  Makes a period (dot) match any character, including a newline. |
| 5 | **re.U**  Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B. |
| 6 | **re.X**  Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker. |

**Client – Server (Python Networking)**

**Ex : Server**

#!/usr/bin/python3 # This is server.py file

import socket

# create a socket object

serversocket = socket.socket(

socket.AF\_INET, socket.SOCK\_STREAM)

# get local machine name

host = socket.gethostname()

port = 9999

# bind to the port

serversocket.bind((host, port))

# queue up to 5 requests

serversocket.listen(5)

while True:

# establish a connection

clientsocket,addr = serversocket.accept()

print("Got a connection from %s" % str(addr))

msg = 'Thank you for connecting'+ "\r\n"

clientsocket.send(msg.encode('ascii'))

clientsocket.close()

**Ex :** #!/usr/bin/python3 # This is client.py file

import socket

# create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# get local machine name

host = socket.gethostname()

port = 9999

# connection to hostname on the port.

s.connect((host, port))

# Receive no more than 1024 bytes

msg = s.recv(1024)

s.close()

print (msg.decode('ascii'))

**Sending Email Using SMTP:**

**Ex:**

# Python code to illustrate Sending mail from

# your Gmail account

import smtplib

# creates SMTP session

s = smtplib.SMTP('smtp.gmail.com', 587)

# start TLS for security

s.starttls()

# Authentication

s.login("kumar.naveen2989@gmail.com", "9837378186")

# message to be sent

message = "Testing Python"

# sending the mail

s.sendmail("kumar.naveen2989@gmail.com", "kumar.naveen2989@gmail.com", message)

# terminating the session

s.quit()

**Prettty Print : Data Pretty Printer in Python**

**Ex:**

# A python code without pprint

import requests

def geocode(address):

    url = "<https://maps.googleapis.com/maps/api/geocode/json>"

    resp = requests.get(url, params = {'address': address})

    return resp.json()

# calling geocode function

data = geocode('India gate')

# printing json response

print(data)

**Ex2 :**

# A python code with pprint

import requests

from pprint import pprint

def geocode(address):

    url = "<https://maps.googleapis.com/maps/api/geocode/json>"

    resp = requests.get(url, params = {'address': address})

    return resp.json()

# calling geocode function

data = geocode('India gate')

# pretty-printing json response

pprint(data)

**Extracting Data from PDF :**

**Ex:**

# importing required modules

import PyPDF2

# creating a pdf file object

pdfFileObj = open('example.pdf', 'rb')

# creating a pdf reader object

pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

# printing number of pages in pdf file

print(pdfReader.numPages)

# creating a page object

pageObj = pdfReader.getPage(0)

# extracting text from page

print(pageObj.extractText())

# closing the pdf file object

pdfFileObj.close()

# Downloading files from web using Python

# imported the requests library

import requests

image\_url = "<https://www.python.org/static/community_logos/python-logo-master-v3-TM.png>"

# URL of the image to be downloaded is defined as image\_url

r = requests.get(image\_url) # create HTTP response object

# send a HTTP request to the server and save

# the HTTP response in a response object called r

with open("python\_logo.png",'wb') as f:

    # Saving received content as a png file in

    # binary format

    # write the contents of the response (r.content)

    # to a new file in binary mode.

    f.write(r.content)

**Download large files:**

import requests

file\_url = "<http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/ch1-2.pdf>"

r = requests.get(file\_url, stream = True)

with open("python.pdf","wb") as pdf:

    for chunk in r.iter\_content(chunk\_size=1024):

         # writing one chunk at a time to pdf file

         if chunk:

             pdf.write(chunk)

**Downloading Videos:**

**Install bs4**

**Install** html5lib

Install requests

Ex:

import requests

from bs4 import BeautifulSoup

'''

URL of the archive web-page which provides link to

all video lectures. It would have been tiring to

download each video manually.

In this example, we first crawl the webpage to extract

all the links and then download videos.

'''

# specify the URL of the archive here

archive\_url = "http://www-personal.umich.edu/~csev/books/py4inf/media/"

def get\_video\_links():

# create response object

r = requests.get(archive\_url)

# create beautiful-soup object

soup = BeautifulSoup(r.content,'html5lib')

# find all links on web-page

links = soup.findAll('a')

# filter the link sending with .mp4

video\_links = [archive\_url + link['href'] for link in links if link['href'].endswith('mp4')]

return video\_links

def download\_video\_series(video\_links):

for link in video\_links:

'''iterate through all links in video\_links

and download them one by one'''

# obtain filename by splitting url and getting

# last string

file\_name = link.split('/')[-1]

print ("Downloading file:%s"%file\_name)

# create response object

r = requests.get(link, stream = True)

# download started

with open(file\_name, 'wb') as f:

for chunk in r.iter\_content(chunk\_size = 1024\*1024):

if chunk:

f.write(chunk)

print ("%s downloaded!\n"%file\_name)

print ("All videos downloaded!")

return

if \_\_name\_\_ == "\_\_main\_\_":

# getting all video links

video\_links = get\_video\_links()

# download all videos

download\_video\_series(video\_links)

# Implementing Web Scraping in Python with BeautifulSoup

pip install requests

pip install html5lib

pip install bs4

Ex : **Accessing the HTML content from webpage**

|  |
| --- |
| import requests  URL = "<https://www.geeksforgeeks.org/data-structures/>"  r = requests.get(URL)  print(r.content) |

**MultiThreading :**

#!/usr/bin/python3

import \_thread

import time

# Define a function for the thread

def print\_time( threadName, delay):

count = 0

while count < 5:

time.sleep(delay)

count += 1

print ("%s: %s" % ( threadName, time.ctime(time.time()) ))

# Create two threads as follows

try:

\_thread.start\_new\_thread( print\_time, ("Thread-1", 2, ) )

\_thread.start\_new\_thread( print\_time, ("Thread-2", 4, ) )

except:

print ("Error: unable to start thread")

while 1:

pass

## **Creating Thread Using Threading Module**

#!/usr/bin/python3

import threading

import time

exitFlag = 0

class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("Starting " + self.name)

print\_time(self.name, self.counter, 5)

print ("Exiting " + self.name)

def print\_time(threadName, delay, counter):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print ("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

# Create new threads

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# Start new Threads

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print ("Exiting Main Thread")

## **Synchronizing Threads :**

#!/usr/bin/python3

import threading

import time

class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("Starting " + self.name)

# Get lock to synchronize threads

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# Free lock to release next thread

threadLock.release()

def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print ("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

threadLock = threading.Lock()

threads = []

# Create new threads

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# Start new Threads

thread1.start()

thread2.start()

# Add threads to thread list

threads.append(thread1)

threads.append(thread2)

# Wait for all threads to complete

for t in threads:

t.join()

print ("Exiting Main Thread")

**Installing MySQL on Python :**

pip install pymysql

pip install mysqlclient

1)Database Connection:

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.

data = cursor.fetchone()

print ("Database version : %s " % data)

# disconnect from server

db.close()

2) Creating Database Table :

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Drop table if it already exist using execute() method.

cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement

sql = """CREATE TABLE EMPLOYEE (

FIRST\_NAME CHAR(20) NOT NULL,

LAST\_NAME CHAR(20),

AGE INT,

SEX CHAR(1),

INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server

db.close()

3) INSERT Operation :

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = """INSERT INTO EMPLOYEE(FIRST\_NAME,

LAST\_NAME, AGE, SEX, INCOME)

VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

4) The above example can be written as follows to create SQL queries dynamically −

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = "INSERT INTO EMPLOYEE(FIRST\_NAME, \

LAST\_NAME, AGE, SEX, INCOME) \

VALUES ('%s', '%s', '%d', '%c', '%d' )" % \

('Mac', 'Mohan', 20, 'M', 2000)

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

5) READ Operation

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = "SELECT \* FROM EMPLOYEE \

WHERE INCOME > '%d'" % (1000)

try:

# Execute the SQL command

cursor.execute(sql)

# Fetch all the rows in a list of lists.

results = cursor.fetchall()

for row in results:

fname = row[0]

lname = row[1]

age = row[2]

sex = row[3]

income = row[4]

# Now print fetched result

print ("fname = %s,lname = %s,age = %d,sex = %s,income = %d" % \

(fname, lname, age, sex, income ))

except:

print ("Error: unable to fetch data")

# disconnect from server

db.close()

6) Update Operation :

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to UPDATE required records

sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

7) DELETE OPERATION :

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to DELETE required records

sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

8) Performing Transactions :

#!/usr/bin/python3

import pymysql

# Open database connection

db = pymysql.connect("localhost","root","hr","mysql" )

# prepare a cursor object using cursor() method

cursor = db.cursor()

# Prepare SQL query to DELETE required records

sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

9) CONSUMING REST APIS :

|  |
| --- |
| import requests |
|  |  |
|  | response = requests.get('https://api.github.com/users/jaimegildesagredo/repos') |
|  |  |
|  | assert response.status\_code == 200 |
|  |  |
|  | for repo in response.json(): |
|  | print '[{}] {}'.format(repo['language'], repo['name']) |

Creating NEW REPOSITORY USING POST request:

import json

import requests

response = requests.post('https://api.github.com/user/repos',

data=json.dumps({'name': 'foo'}), auth=('kumar.naveen2989@gmail.com', 'naveen.1989'))

assert response.status\_code == 201

Testing : <https://api.github.com/users/Naveen2989/repos>

10 ) More Examples : POST Request

import requests

def consumePOSTRequestSync():

 params = {'test1':'param1','test2':'param2'}

 url = '<http://httpbin.org/post>'

 headers = {"Accept": "application/json"}

 # call post service with headers and params

 response = requests.post(url,headers= headers,data = params)

 print "code:"+ str(response.status\_code)

 print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

 print "headers:"+ str(response.headers)

 print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

 print "content:"+ str(response.text)

# call

consumePOSTRequestSync()

GET Request:

import requests

def consumeGETRequestSync():

 params = {'test1':'param1','test2':'param2'}

 url = '<http://httpbin.org/get>'

 headers = {"Accept": "application/json"}

 # call get service with headers and params

 response = requests.get(url, headers = headers,data = params)

 print "code:"+ str(response.status\_code)

 print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

 print "headers:"+ str(response.headers)

 print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

 print "content:"+ str(response.text)

consumeGETRequestSync()

11) Parsing XML :

11 A)

<**collection shelf = "New Arrivals"**>  
<**movie title = "Enemy Behind"**>  
 <**type**>War, Thriller</**type**>  
 <**format**>DVD</**format**>  
 <**year**>2003</**year**>  
 <**rating**>PG</**rating**>  
 <**stars**>10</**stars**>  
 <**description**>Talk about a US-Japan war</**description**>  
</**movie**>  
<**movie title = "Transformers"**>  
 <**type**>Anime, Science Fiction</**type**>  
 <**format**>DVD</**format**>  
 <**year**>1989</**year**>  
 <**rating**>R</**rating**>  
 <**stars**>8</**stars**>  
 <**description**>A schientific fiction</**description**>  
</**movie**>  
 <**movie title = "Trigun"**>  
 <**type**>Anime, Action</**type**>  
 <**format**>DVD</**format**>  
 <**episodes**>4</**episodes**>  
 <**rating**>PG</**rating**>  
 <**stars**>10</**stars**>  
 <**description**>Vash the Stampede!</**description**>  
</**movie**>  
<**movie title = "Ishtar"**>  
 <**type**>Comedy</**type**>  
 <**format**>VHS</**format**>  
 <**rating**>PG</**rating**>  
 <**stars**>2</**stars**>  
 <**description**>Viewable boredom</**description**>  
</**movie**>  
</**collection**>

11 B) : Using SAX DOM Parser :

*#!/usr/bin/python3***from** xml.dom.minidom **import** parse  
**import** xml.dom.minidom  
  
*# Open XML document using minidom parser*DOMTree = xml.dom.minidom.parse(**"movies.xml"**)  
collection = DOMTree.documentElement  
**if** collection.hasAttribute(**"shelf"**):  
 print (**"Root element : %s"** % collection.getAttribute(**"shelf"**))  
  
*# Get all the movies in the collection*movies = collection.getElementsByTagName(**"movie"**)  
  
*# Print detail of each movie.***for** movie **in** movies:  
 print (**"\*\*\*\*\*Movie\*\*\*\*\*"**)  
 **if** movie.hasAttribute(**"title"**):  
 print (**"Title: %s"** % movie.getAttribute(**"title"**))  
  
 type = movie.getElementsByTagName(**'type'**)[0]  
 print (**"Type: %s"** % type.childNodes[0].data)  
 format = movie.getElementsByTagName(**'format'**)[0]  
 print (**"Format: %s"** % format.childNodes[0].data)  
 rating = movie.getElementsByTagName(**'rating'**)[0]  
 print (**"Rating: %s"** % rating.childNodes[0].data)  
 description = movie.getElementsByTagName(**'description'**)[0]  
 print (**"Description: %s"** % description.childNodes[0].data)

11 C) Parsing Using SAX Parser:

*#!/usr/bin/python3***import** xml.sax  
  
  
**class** MovieHandler(xml.sax.ContentHandler):  
 **def** \_\_init\_\_(self):  
 self.CurrentData = **""** self.type = **""** self.format = **""** self.year = **""** self.rating = **""** self.stars = **""** self.description = **""** *# Call when an element starts* **def** startElement(self, tag, attributes):  
 self.CurrentData = tag  
 **if** tag == **"movie"**:  
 print(**"\*\*\*\*\*Movie\*\*\*\*\*"**)  
 title = attributes[**"title"**]  
 print(**"Title:"**, title)  
  
 *# Call when an elements ends* **def** endElement(self, tag):  
 **if** self.CurrentData == **"type"**:  
 print(**"Type:"**, self.type)  
 **elif** self.CurrentData == **"format"**:  
 print(**"Format:"**, self.format)  
 **elif** self.CurrentData == **"year"**:  
 print(**"Year:"**, self.year)  
 **elif** self.CurrentData == **"rating"**:  
 print(**"Rating:"**, self.rating)  
 **elif** self.CurrentData == **"stars"**:  
 print(**"Stars:"**, self.stars)  
 **elif** self.CurrentData == **"description"**:  
 print(**"Description:"**, self.description)  
 self.CurrentData = **""** *# Call when a character is read* **def** characters(self, content):  
 **if** self.CurrentData == **"type"**:  
 self.type = content  
 **elif** self.CurrentData == **"format"**:  
 self.format = content  
 **elif** self.CurrentData == **"year"**:  
 self.year = content  
 **elif** self.CurrentData == **"rating"**:  
 self.rating = content  
 **elif** self.CurrentData == **"stars"**:  
 self.stars = content  
 **elif** self.CurrentData == **"description"**:  
 self.description = content  
  
  
**if** (\_\_name\_\_ == **"\_\_main\_\_"**):  
 *# create an XMLReader* parser = xml.sax.make\_parser()  
 *# turn off namepsaces* parser.setFeature(xml.sax.handler.feature\_namespaces, 0)  
  
 *# override the default ContextHandler* Handler = MovieHandler()  
 parser.setContentHandler(Handler)  
  
 parser.parse(**"movies.xml"**)

12) : Parsing JSON :

12 A) : Input JSON

{  
 **"maps"**: [  
 {  
 **"id"**: **"blabla"**,  
 **"iscategorical"**: **"0"** },  
 {  
 **"id"**: **"blabla"**,  
 **"iscategorical"**: **"0"** }  
 ],  
 **"masks"**: {  
 **"id"**: **"valore"** },  
 **"om\_points"**: **"value"**,  
 **"parameters"**: {  
 **"id"**: **"valore"** }  
}

12 B)

**import** json  
**from** pprint **import** pprint  
  
data = json.load(open(**'test.json'**))  
print(data)  
pprint(data)

12 C) JSON to dictionary object :

**import** json  
**from** pprint **import** pprint  
  
**with** open(**'data.json'**, **'r'**) **as** f:  
 distros\_dict = json.load(f)  
  
**for** distro **in** distros\_dict:  
 print(distro[**'Name'**])

13) Python Lambda :

13 A) def squareof(x):

return x\*x

p = squareof(5)

print(p)

Corresponding Lambda :

f = lambda x: x\*x

p = f(5)

print(p)

13 B) : Another Example :

x = lambda a, b : a if a>b else b

y = x(14,5)

print(y)

13 C) Python Lambda in Filter function :

weekdays = ['sun', 'mon', 'tues', 'wed', 'thurs' 'fri']

days = filter(lambda day: day if len(day)==3 else '', weekdays)

for d in days:

print(d)

13 D) Python Lambda function in map()

We can also use lambda in map(). map is a function that takes two argument one is function, another is a list. The following code is an example which finds the remainder of all the number by 5.

numbers = [ 74, 85, 14, 23, 56, 31,44 ]

remainders = map(lambda num: num%5, numbers)

for i in remainders:

print(i)