====================

Oracle Python Training

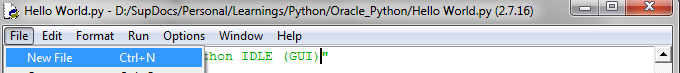
Documentation:

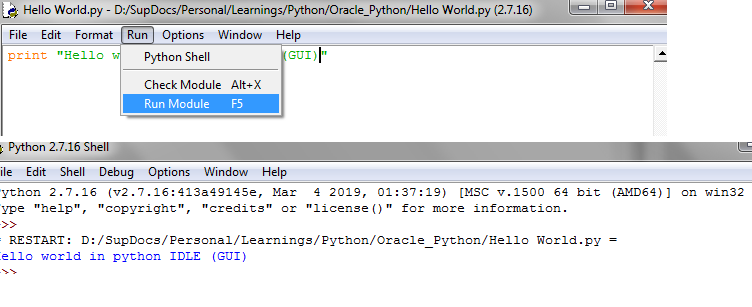
<https://docs.python.org/2/tutorial/index.html>

* Django used to make web applications using python
* On Linux and Mac python it comes inbuilt
* Downloaded Python 2.7.16
* Add python path (C:\Pyton27) to path environment variable
* On the Windows explorer we see IDLE (python GUI). This is the inbuilt IDE provided by python
* Three ways of executing Python programmer
* Command line interpreter
* IDLE (GUI)
* PyCharm
* Script from the command line

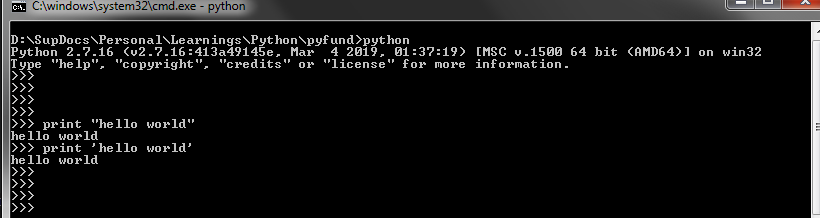
**Running through Default IDE (GUI)**

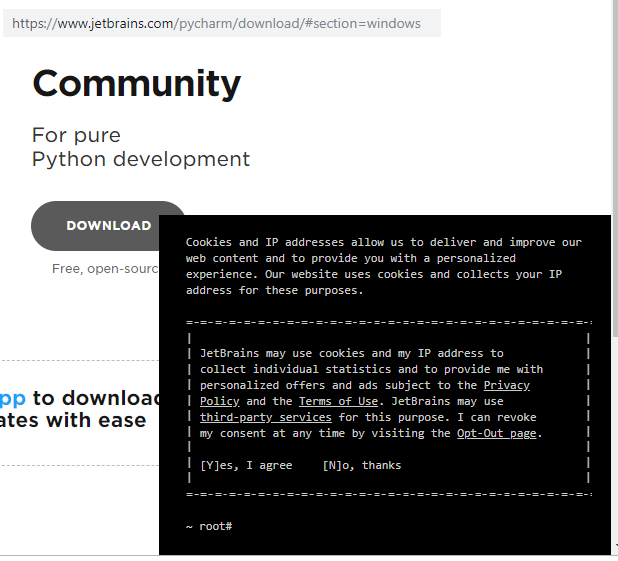


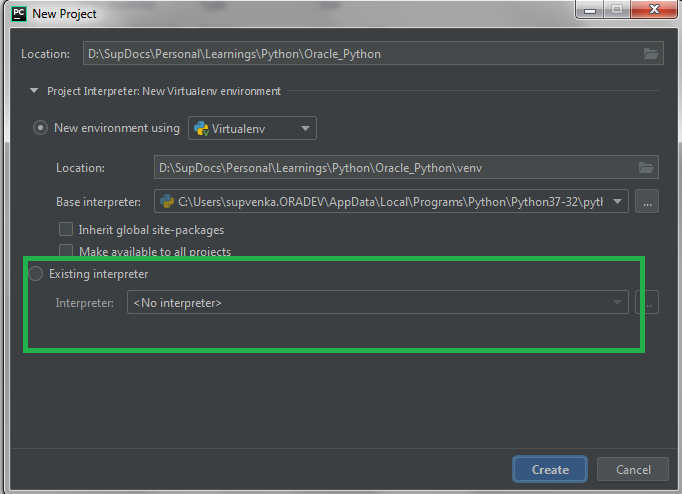


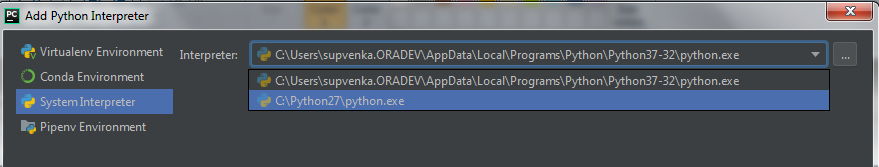


**On CMD Prompt**

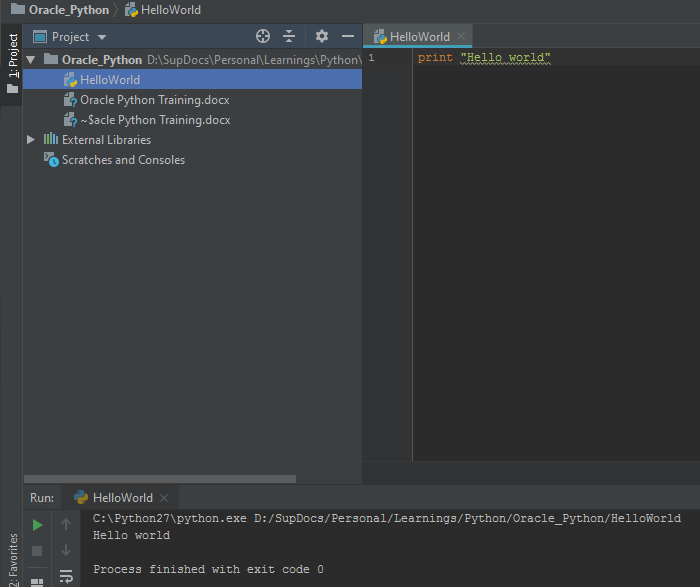
****

**Download Pycharm**

****

****

**Through pyCharm**

****

**MULTILINE**

Multiline is indentified by “ \

total = item\_one + \

“xyz” +

**QUOTES**

word =’word’

sentence = “This is a senstence”

paragraph = “”” This is a paragraph

“””

**Comment**

We need to use a # for a single line comment

# First Comment

**User input**

name=raw\_input(“\n\nEnter your name”)

print ‘Hello’, name

Multiple statements on a single line

**a = 5; b = a\* 100; c = b/50**

Ctr+J or ctr+Q for getting documentation of the method

Python has 5 DataTypes

Number: integer, long integer, Floating point, Complex number, Boolean(is also interpreted as an integer)

Str

List

Dictionary

Pass: null operation. Syntactically it should work , but we are waiting for some logic to build

We can use pass

All alpha bets are ASCII (American std code), some characs like Arabic, emojis etc need to be addressed using unicode

For Unicode prefix it with u

Ex u ‘123’

r ‘C:\newfolder\myFile.txt’

Here if we see \n can be interpreted as a new line to avoid this and to take it as te same way user has prvided we put it as raw string prefixed with r

Ex: r ‘C:\newfolder\myFile.txt’

**Interpolation operator %**

>>> print 'hello %s' % 'Priya'

hello Priya

>>> i =10

>>> j =20

>>> print '%d \* %d = %d' % (i,j,i\*j)

10 \* 20 = 200

>>> print '%03d' %i

010

>>>

Interpolation we need to take care of the variables, the datatype

The above is achieved using format method in str class

>>>

>>> print '{0} {1} {2}'.format('a','b','c')

a b c

>>> print '{} {} {}'.format('a','b','c')

a b c

>>> print '{2} {0} {1}'.format('a','b','c')

c a b

>>>

Day2 : Lists, tuple,dictionary

List: [, ,] : Square bracket comma separated.

Flexible array .

We can have a collection objects of different data type

Ex: [10,20, [‘x’,’y’,’z’]]

Access list using index data[0], data[-1]

Data = [10,20, [‘x’,’y’,’z’]]

Data[-1,0] = x

Data[-1,-1] = z

List is mutable

List is a **container** sq brackets

Tuple: Collection of various datatype

y = (‘a’,’10,23.89)

Tuple is immutable

y[0]

We cannot add, remove or update a tuple

We change list into a tuple

L = [1,3,4]

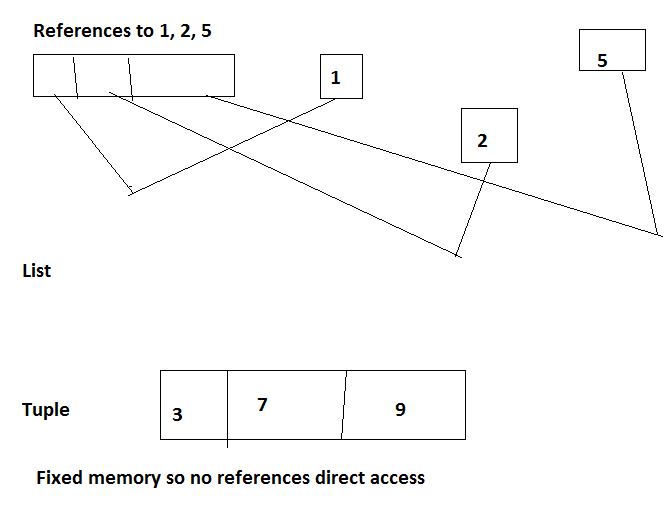
x = tuple(1,3,4]

Tuple is faster: Memory is fixed so accessing is faster in tuple

U =4,7,8

Even if we do not provide ( in the beginning and end it is consider as a tuple

Single element needs a trailing,



**Accessing elements in a list**

>>> [4,6,7][1]

6

>>> [4,6,7][-2]

6

>>> [4,6,7,8, 6][1:3]

[6, 7]

>>> [4,6,7,8, 6][:3]

[4, 6, 7]

>>>

>>>

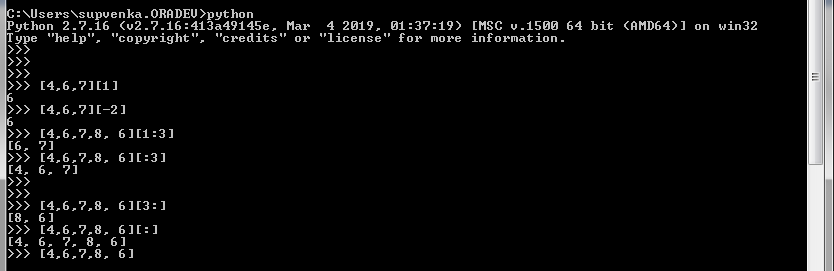
>>> [4,6,7,8, 6][3:]

[8, 6]

>>> [4,6,7,8, 6][:]

[4, 6, 7, 8, 6]

>>> [4,6,7,8, 6]



Updating list

List = [‘p’,’c’,1997,2000]

List[2] = 2001

>>>

>>> l = ['p','c',1997,2000]

>>> l[2]

1997

>>>

>>>

**Methods in List:**

**Adding/updating, insert**

Extend when you want to add another listing to the existing list

Append : add to the exiting list an element

Update/insert:

If you want to add a tuple

[4,6,8,1,2,9].append(‘10’): Adding a tuple

[4,6,8,1,2,9,’10’]

To update at a specific index

[4,6,8,1,2,9].insert(0,‘10’)

[’10’,4,6,8,1,2,9]

**Removing elements in the list**

List1=[‘p’,’c’,1997,2000]

del list[2] : 2 is the memory ref or index

remove : We need to specify the element to remove

list1.pop()

Pop removes the first element and returns the element

Pop(index=list[-1])

**Membership** operator: In , not in

**Compare Methods**

To compare two lists = cmp ()

Cmp(list1, list2)

Cmp(tuple1, tuple2)

Remove, sort, reverse: Is no there for tuple as it is immutable

**Dictionary**

Collection of key and value

Key is unique, Value is not duplicate

Key is immutable: so keys can be tuple, strings , Number but NOT list as list is immutable

Value is mutable

Dictionary does not follow a sequence protocol and hence cannot be accessed using index.

Order is not guarantee

We use {} brackets and : to associate the value to the key

Separated by a ,

Dictinary they themselves are mutable so we can update, add , remove etc

contacts = {‘biil’:’b’, ‘cool’:’c’}

Dictionary copy is shallow copy

Iterations : items, key(), values()

>>> demo={2:['a','b'], (2,4):99,'x':{1:2.5,'a':3}}

>>> demo

{'x': {'a': 3, 1: 2.5}, 2: ['a', 'b'], (2, 4): 99}

>>> demo[2]

['a', 'b']

>>> demo[(2,4)]

99

>>> demo['x']

{'a': 3, 1: 2.5}

>>> demo['x'][1]

2.5 ---------🡪 [x] is the key so it gives value :{1:2.5,'a':3}}, then the value of key 1 it gives so ans = 2.5

>>> demo[2][1]

'b'

>>>

>>> for key in demo: ---------🡪 note here we never saif demo.keys() it is the same like key in demo , no need to specifically say demo.keys()

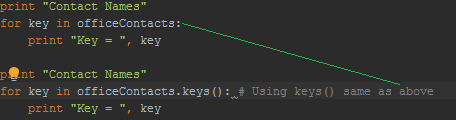
... print key

...

x

2

(2, 4)



--------

Default value when you enter a non present key we can do it

My\_dict.get(key , value= none): returns value or default value = Null

Provide a list to a dict so that the list entries become the keys and then we need to add values to the keys that is formed from the list

We use fromKeys

friends = ['abc', 'xyz']  
friendsContacts = dict.fromkeys(friends) # no values for the keys  
print "friendsContacts = ", friendsContacts  
friendsContacts = dict.fromkeys(friends, 500) # Providing a default value  
print "friendsContacts = ", friendsContacts

Output

friendsContacts = {'xyz': None, 'abc': None}

friendsContacts = {'xyz': 500, 'abc': 500}

SET: No duplicates

We can convert List to Set to remove duplicates

**User Defined functions**

**Syntax:**

**def** functionName (paramters):

“DoctString for the ”

Function Code

return (optional)

* return is return by value
* def of the function should always be before we call the function, since this is an interpreter
* If nothing is returned then it means default value returned is NONE
* If the parameter that is passed is mutable then when it is altered in the function it is pass by value
* If the parameter that is passed is **immutable** then when it is altered in the function it is pass by reference
* Function overloading is NOT available in python

*"""Functions Sample"""*def add(a,b):  
 *"Simple addition: a, b : both as int or str "* if type(a) is int and type(b) is int:  
 pass  
 elif type(a) is str and type(b) is str:  
 pass  
 else:  
 print "Invalid type"  
 return  
  
 result = a + b  
 return result  
  
  
  
c = add('hi', 'oracle') # strings are immutable : Pass by ref  
print " c = ", c  
  
d = add('hi', 3) # Invalid type  
  
#Function overloading is not available in Python  
def add(a,b,c):  
 *"Another add"* return a+b+c  
  
print " 10 + 20 = " ,add(10,20)  
print "10+20+30 = " add(10,20,30)

C:\Python27\python.exe D:/SupDocs/Personal/Learnings/Python/Oracle\_Python/funtionsSample.py

c = hioracle

Invalid type

10 + 20 =

Traceback (most recent call last):

File "D:/SupDocs/Personal/Learnings/Python/Oracle\_Python/funtionsSample.py", line 28, in <module>

print " 10 + 20 = " ,add(10,20)

* TypeError: add() takes exactly 3 arguments (2 given) --🡪 Function overloading is NOT available in python

When two funcstions have the same name but different number paramters, it always take the last /latest defined func into consideration

Process finished with exit code 1

Functions can have 4 types of arguments

* Required arguments
* Keyword arguments
* Default arguments /optional arguments
* Variable Length : \* symbol is associated to the argument which is of variable length and it must be tuple. It always placed in the end.

Required: mandatory to be supplied in the current positional order how it is defined in the function

Keyword arguments: Here if I donot want to supply in the order then I give the parameter as keyword and then value

Ex

def myfunc(name, age, ht)

myfunc(ht=6, name=’kapil’, age=20) //here I pass the same parameter name in the definition but I can call the function by passing parameter in any order since the call is identified by the parameter name

**Default arguments:** Here in the function, the parameter will have a default value that we associate it to

Ex myFUnc(name, age=50)

SO here name is required parameter and age is default/optional parameter

We need to provide the default parameter in the end in the functional definition

Ex myFunc (name, age=50, std =5)

Default arguments

**Variable Length arguments**

* Rules- placed after required args, can have only one variable arg
* # Cannot have default and variable paramters together

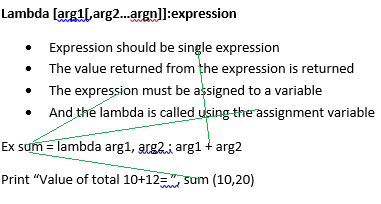
**Anonymous functions using lambda**

**Lambda [arg1[,arg2…argn]]:expression**

* Expression should be single expression
* The value returned from the expression is returned
* The expression must be assigned to a variable
* And the lambda is called using the assignment variable

Ex sum = lambda arg1, arg2 : arg1 + arg2

Print “Value of total 10+12= ”, sum (10,20)



**Sope of a variable : global , local**

To change the global variable value in the local when both have the same name

Use global keyword in the local block

**Functional tools:**

* Filter
* map
* reduce
* List comprehension

OOPS:

Class : Blueprint of all objects

Properties: data, methods

Object/instance :

In python dot (.) operator is used to access the properties/methods

Car = Vehicle()

Car.noOfTyres = 4

Car.drive()

Inheritance : Similar + extra characteristics we use inheritance

Ex Class AutomaticVehicle : Vehicle

Data members are of two types

* Class variable
* instance Variable

**Common data** : Class variable to be shared across all instances of the class like a **static variable** in java.

If the value is changed then the changed value is reflected across all instances

**Instance variable**: each instance has its own value

Init\_ is similar to a ctr in Java, it can be skipped if we want to

self is similar to **this** in Java

any variable we initialize using self is a instance variable

ex : self.name = name

All self /instance variables are set in the ctr ie \_init\_ method and accessed using self.<instance\_Variable>

class variable ie common data is Common data is accessed using the name if the class ex:Employee.empCount

Where Employee is the class name and empCOunt is the COmmonData variable

In Python all variables in a class is by default everything is public

In python **per** object we can delete or add own properties.

We can add, remove or modify attributes of classes and objects at any time.

Each instance can have their own property.

This is said as ‘ property is dynamic’ in python

Use \_\_dict\_\_ to get the all the attributes of the obj

Ex: emp1\_\_dict\_\_

{'salary': 1000, 'lastName': 'Smith', 'fullName': 'John Smith', 'bonus': 100, 'firstName': 'John'}

To check for attributes use hasattr(self,’<attribute\_name’)

Ex: hasattr(self, 'bonus')

To set new attributes use hasattr(self,’<attribute\_name’, ‘default value’)

Ex: setattr(self, 'bonus', 10)

Private Access Speciifer

To make your access specifier private use double underscore \_\_

To make your attribute private: we need to prefix the property with double underscore \_\_ ex: \_\_myaddr This is accessible only in the class

The instance variable or the class variable both can be made private using \_\_

Instance variables can be made private in the init method

Property Decorator

This is used only on instance variables

This can turn a attribute into a method. This is done using @ property for the property

We create a method using the SAME property Name with property decorator

Property decorator should be used for property/properties that is dependent on other instance variables (which are set in init function) ex: fullName is dependant on firstName and lastName

Types of Methods:

* Instance Methods
* **Class** Methods
* Static Method

**Instance Methods :**

* operates on obj . Here the parameter passed is self.

**Class Method :**

* operates on class. Here we use decorator @classmethod and pass parameter **cls**
* And access using cls.<propertyname>
* And call it using class Name

Ex

@classmethod

def fly(cls, name):

print (‘{} flies with {} wings’.format(name, cls.wings)

Employee.fly -🡪called used class Name

**Static Method :**

* Operates neither on object or class
* We use @staticmethod: These are used to manipulate outside the class variables.
* Not used on instance or class variables
* These do not have any class or instance variables defined in them

**Inheritance:**

Class SubClassName (ParentClass 1, ParentClass2):

Types of inheritance:

1 base class 1 child class

Multilevel inheritance: ClassA🡪ClassB🡪ClassC

Hierarchical: Class A has 2 child classes

Multiple inheritance : Class A and Class B and child class C is derived from both Class A and Class B

If we donot have init mehod for child then it will call init method of parent

If the Child class has init then it will not call parent’s init method and will execute the child init method only

To call the parents overriding method we use the keyword super just like Java

**Files and Directories**

* Get File Handle: We use open (filename [, mode][,buffering])
* If the file is in the same path no need to give the path name
* Open method returns a file obj, on which we perform operations
* W mode we can just write into the file, but NOT read
* + is for both reading and writing into the file
* R is for read mode
* tell() : tells the current pointer in the file
* seek(offset,0) : The file pointer goes to the index 0 is for beginning of the file, 1 is the current position and 2 is the end of the file
* Deleting and renaming a file
* os module we need to use
* mkdir: os.mkdir()
* change directory = os.chdir(<path to change>)
* ex os.chdir(“C:\myFolder)
* Current Directory = os.getDir()
* Remove directory = rm.dir()

**Modules:** Module is like a package

* We have used os, datatime
* Syntax = import module1[, module2[, module]
* FileName becomes your module name

**Support.py------------🡪 here now this becomes your module name**

def print\_func(par):

print “Hello”, par

Now I want to import **Support.py in file xyz.py**

**Using import command**

**import Support**

**Support.print\_func(“Zara”)**

* When we use the import command <Module>
* Then we give <ModuleName>.<methods> to access the methods of that class
* We use the dot operator

**Now say we want to use Support in a different folder**

Then use import relative

Import ../Support

**Second syntax form import**

**from fib import fibonacci**

When we use ‘from ‘ keyword to import then we can call the function without the **class.**funcName (not need to use className.) directly use the function name

**Custom import**

We can use **alias** in the import statement

import support **as** st

from support import print\_func as pfunc

* **Import Statement**

When we import the class Name without using **from** key word to import we should access the methods using the dot operator

import FileSample

ex: FileSample.readFromFile()

* **Importing using from keyword**

**from** FileSample import readFromFile

# no need to use the dot operator ie ClassName.method ie FileSample.readFromFile()

readFromFile() # Directly calling the method . This we can do when we use the from keyword to import the class pointing to the method

* **Importing using from keyword with as keyword ie alias**

Using alias to refer to the method name

from FileSample import writeToFile as wf #

wf()

* Using **\***

# if I want to import all the methods

from FileSample import **\*** # imports all methods

readFromFile()

writeToFile()

**Importing methods from a different package**

* When we want to import Classes /methods from a different package
* Then in myPackage (which is actually a directory) we need to create an **EMPTY** file named **\_\_init\_\_.py**
* Then we can use the import statement like this in the class outside the above package

from myPackage import test1

from myPackage import test2

* But we cannot use \* like this import myPackage **\***

from myPackage import \* // This will not work

To make \* work

In the myPackage (folder) in the \_\_init\_\_.py we write the statements

import test1

import test2

Where test1 and test2 are the classes in package myPackage (explicitly written in the **\_\_init\_\_.py file)**

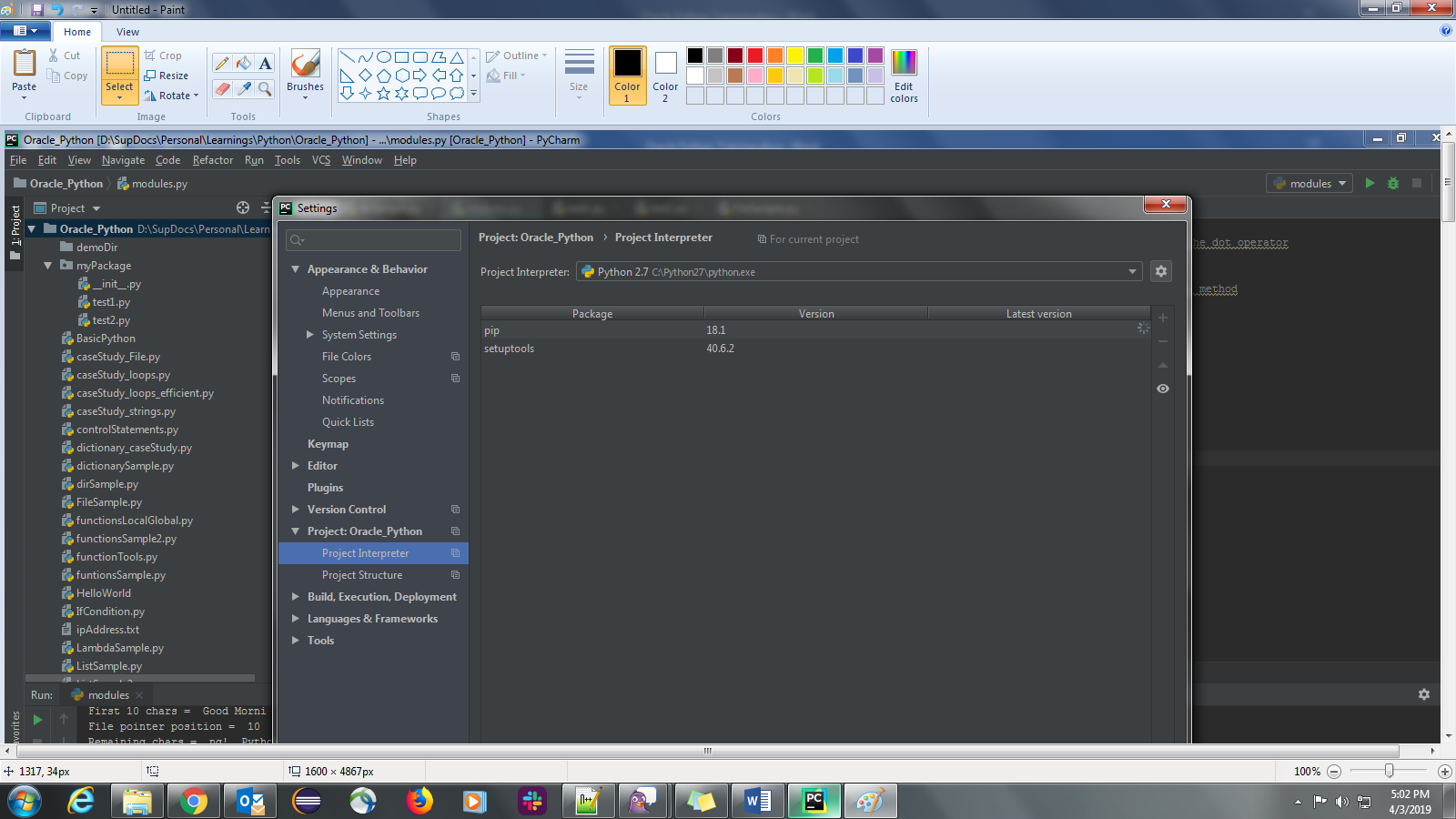
* We need to add import <className> that we want to use in the **\_\_init\_\_.py file**

import test1

import test2

ie write import statements for all the classes that you want the user to access outside this package using the \*

Only those classes listed in the\_\_init\_\_py will be available when using \* during import

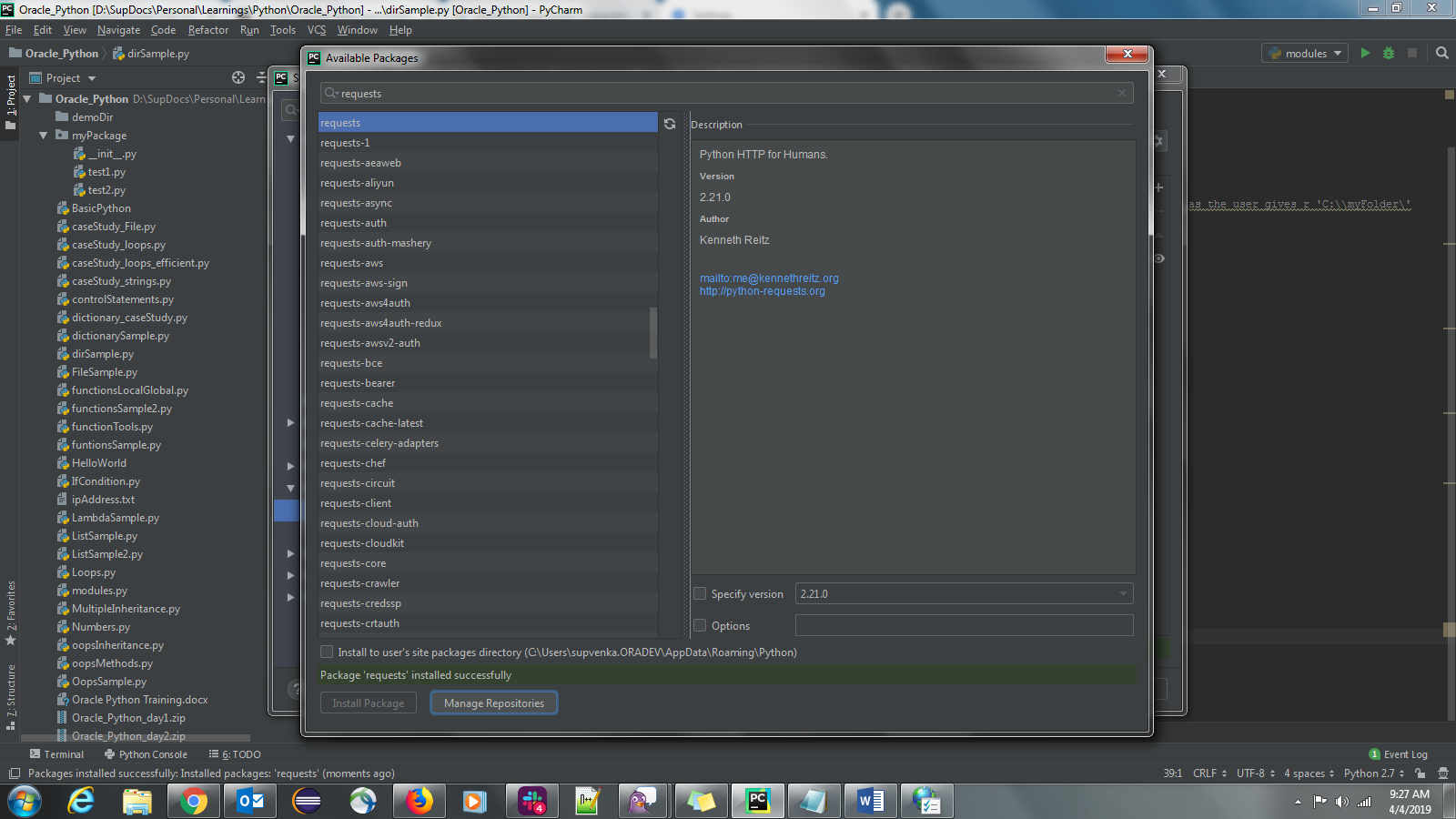


Do not use proxy : File/Settings/ search for proxy : No proxy

Click File/Settings/ <Your proj>/Project interpreter setuptools

Search for requests

Click Install package



**UserDefined Exception class**

Create a class derived from Exception class

Use the init function in your UserDefinedException class

Raise UserDefinedException

And catch UserDefinedException

**Regular Expressions**

* Import module called re
* Put the expression in a raw ex r’m\w\w’
* Assign it to a variable
* Then compile the regular expression which returns the compile object /regex object
* And exceute method (ex search) on the compiled object
* We get a match obj or NONE
* Space is also considered as a char in your search

Ex

**Step 1**

import re

reg = r’m\w\w’ ---------🡪 must be a raw input

compiledObj = re.compile(reg)

str = ‘cat mat rat sat’

result = compiledObj.search(str)

print(result.group())

**Step 2**

passing both the regular expr and the str will internally do the compiling and have the compiledObj

ex re.search(reg,str) ---------🡪 this is equivalent to

compiledObj = re.compile(reg)

str = ‘cat mat rat sat’

But Step1 is better where we have the compiled obj already and useful when you have to do many repeated searches and just not one search

If there is only one search then Step2 is ok

+, \*, ? {} etc are called **quantifiers**

* + one or more
* \* 0 or more
* ? 0 or 1 🡪 exactly one occurrence
* {m}--🡪exact number occurrence
* {m,n} -🡪 min m occurrence and max n occurrence
* {,n}-🡪 min 0 occurrence and max n occurrence
* {m}-🡪 min m occurrence and no maximum n occurrence

**NETWORKING**

Low level connection happens via sockets

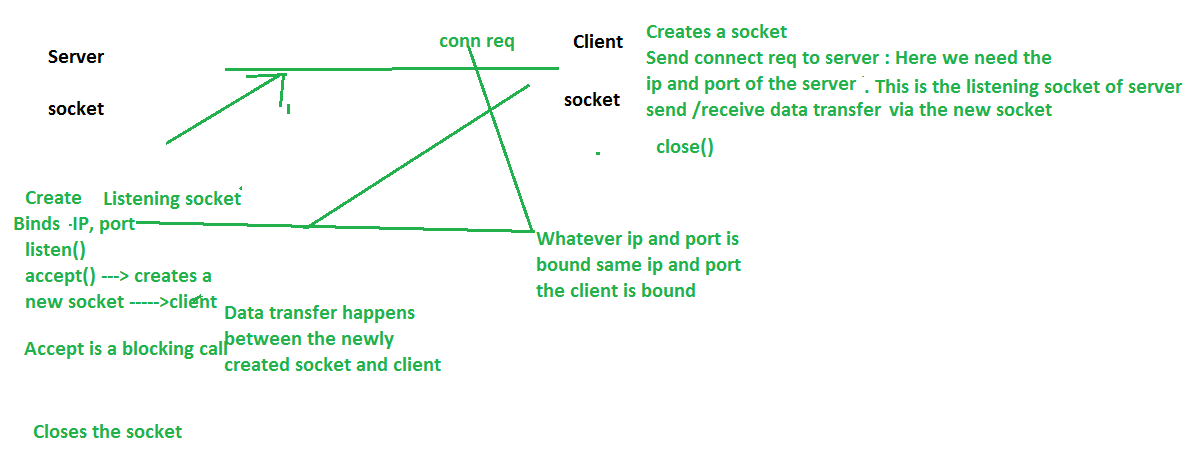
High level connection happens via http https

Module : Socket

socket func is used to create socket

default AF\_INET is the default socket

TCP socket is default which uses socket\_family AF\_INET and SOCK\_Stream type



**COMMAND Line Arguments**

Taking the arguments from the user in the command prompt and the script should be able to take /read the value provided by the user

We have

* sys module
* argparse : Recommended
* optparse : Deprecated

**Sys Module**

* Import module sys
* Retrieve the cmd line args using a list called argv ike sys.argv

Ex: python test.py arg1 arg2 arg3

sys.argv[0] = python.py -🡪 where this the script that is getting executed

sys.argv[1] = arg1

**argparse Module : Recommended to be use**

* automatic error message
* automatic -h help generation and –help is auto added to your script
* We add two type of arguments

**Optional arguments**: in the command prompt with a hypen

Ex: -a –u

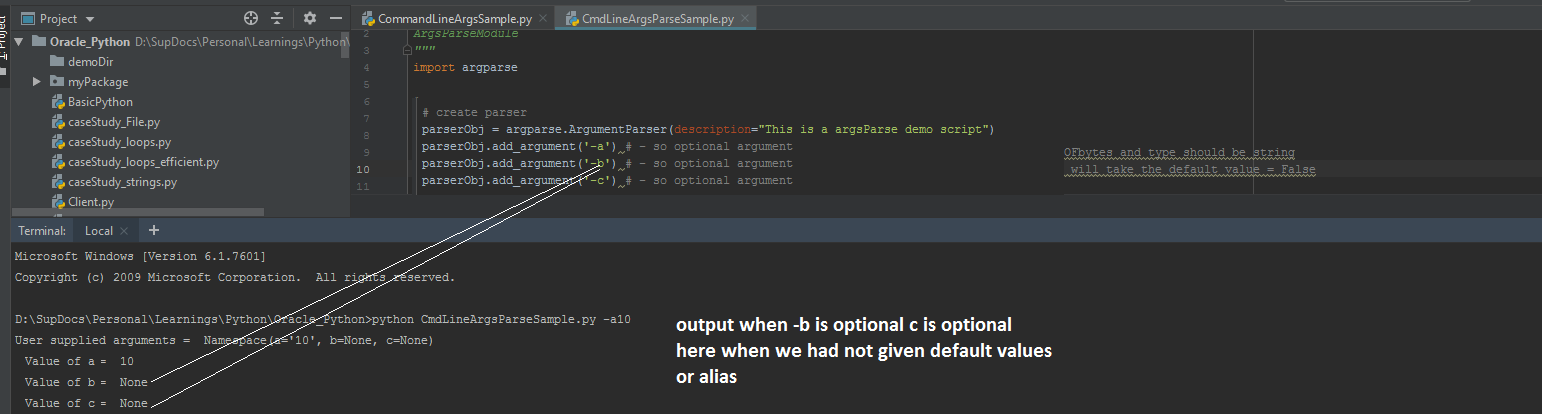
**Mandatory argument:** Note without – if we provide then it is mandatory argument. These mandatory arguments are also called positional arguments as they are based on order

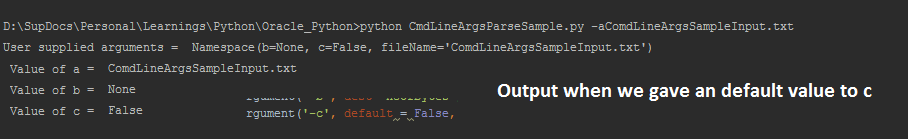
* Import argparse module
* Create an obj of type ArgumentParser
* We can use add\_arguments to add arguments to the ArgumentParser object
* We can provide default values for the arguments also
* We can create an alias for your command line value supplied

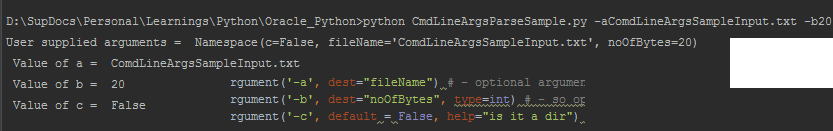
Ex (‘-c’, dest=’option2”, type=int

Here dest is the alias

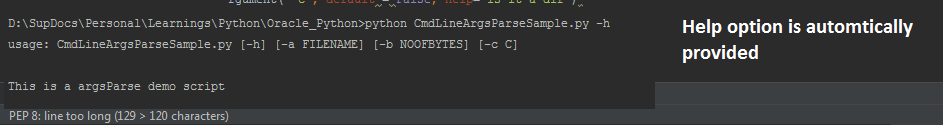
* ArgumentParser obj calling parse\_args() will retrieve the commnd line arguments

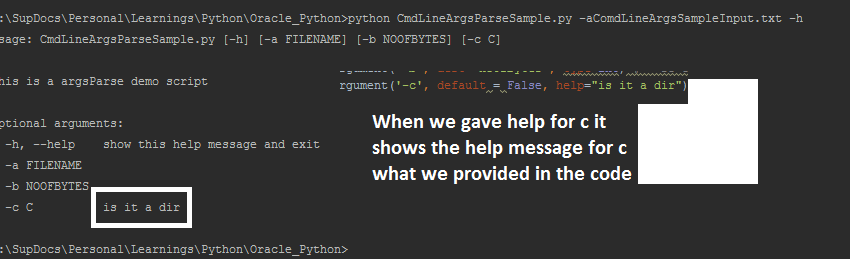


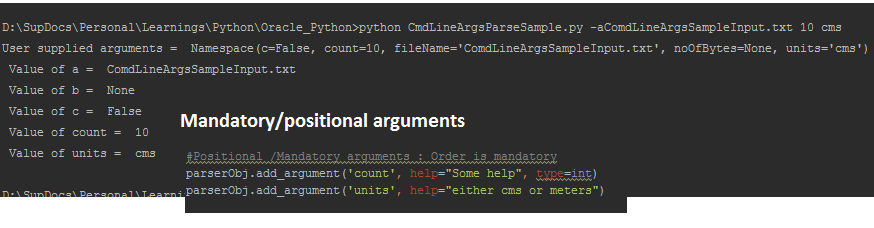


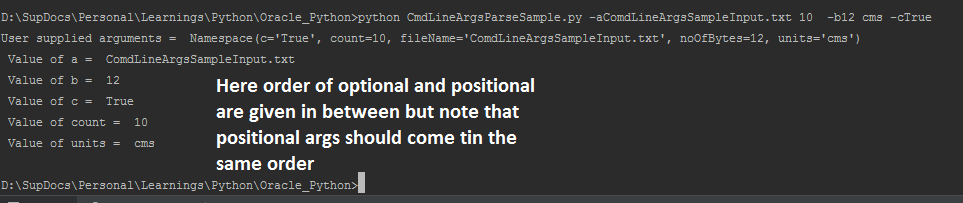




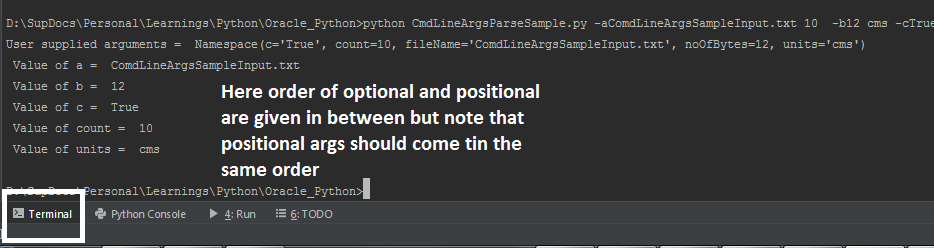








All the above should be executed on the terminal of the programm



**Subprocess**

* Execute external commands
* Create another process and execute commands and get the o/p of the command from the other process
* Any shell command can be executed using the subprocess
* call() returns the status code if the call was successful
* We can pass the command in a list of strings elements
* Or Pass the whole command as a script and provide shell=true

We need to first check what time of OS it is to execute the shell command

* check\_out() : gives the actual output of the command as a string

This does not print on the console but is assigned to a variable

And we can print the variable

* check\_call(): Executes the command raises an exception if the command fails

**PIPE**

To have an interactive process like spawn the process execute the cmd get the output then pass something from here then we use a PIPE.

* communicate(): give 2 pipes. 1st stdout and 2nd stderr()

Popen()

Ls –l | grep ie output of ls –l give it to grep

* ie stdout of one process we give it as stdin for another.
* This is achieved using Popen()

Ex output of a java program we can give it to this process using PIPE Popen()

**Debug in command line**

python –m pdb

**GUI Programming**

Graphical User interface

Inbuilt module : Tkinter

Import module Tkinter

Canvas: When we want to draw we use canvas

Frame: Other we use Frame if not Canvas

**DBCONNECTION**

* import <the db module>

ex import MSQLDb

ex import sqlite3

* connect
* getcursor
* execute method the query
* fetch result using fetChone()//One by one
* fetch all results

GoodLink:

Python documentation

* <https://docs.python-guide.org/>
* Scenario Guide for Python Applications In the above link is very good
* Django python : <https://www.djangoproject.com/> Good for your own web application
* www.python-excel.org