Python workshop experiments (1st Part)

"Basics of Python3, Structured & Functional programming"

Notice: These python shell experiments are done by me, to provide you with what we have done yet in python workshop(in 5 days), & to clarify the doubts that you may have, there may be errors somewhere in this document, but i am committed to correct those, just notify me, ask doubts, I am happy to help!

Request: It took me several hours to prepare this document, So I request you to read this as far as you can, it will serve as more than the detailed summary of at least 150 pages of any good python book, i am doing this for goodwill, now rest depends on you!

Future Scope: For those who are really interested in computer programming, this document is a must read to understand the very basics of python, & if you will go through this, you will be comfortable with our next classes on Object-Oriented & Little Advanced Programming in python3.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> begin='Hi, these are the complimentary shell interaction examples'
>>> type(begin)
<class 'str'>
>>> 'There are a lot of in-built data types in python, which makes language powerful
& easy'
'There are a lot of in-built data types in python, which makes language powerful &
easy'
>>> des='look, above the string is printed due to READ-EVALUATE-EXECUTE loop of
interpreter'
>>> print(des)#to print a stored variable i need to use print,  also it is always used
from script(source code)
look, above the string is printed due to READ-EVALUATE-EXECUTE loop of interpreter
>>> more='''Python can be integrated with C/C++/Java, it is multiparadigm by nature,
which means it is Structured, Object-Oriented, Functional & Scripted'''#This is a
multiline string
>>> len(more)
140
>>> #len is a built-in function for all iterables(data types that can be traversed)
>>> """it has dynamic data types management, dynamic type checking & garbage
collection"""
'it has dynamic data types management, dynamic type checking & garbage collection'
>>> #there are various ways to launch a python program, also you can import one python
program into another like, header file in C/C++, using import statement, for example
let us use math library of python
>>> import math
>>> math.sqrt(25)
5.0
>>> pi
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    pi
NameError: name 'pi' is not defined
>>> math.pi
```

```
3.141592653589793
>>> #it means when you import a module, every function or variable from it must be called by using a . (dot operator),it is a better way to avoid name collisions from two modules, having members with same name, like, we have complex number in python, & it's own math module named as cmath, let us use it
>>> import cmath
>>> cmath.sqrt(2+5j)
(1.921609326467597+1.3009928530039094j)
>>> math.sqrt(2+5j)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    math.sqrt(2+5j)
TypeError: can't convert complex to float
>>> #sqrt of complex numbers do exist but it has own module for it's operations
>>> #now let us use tha 'as' keyword for renaming modules
>>> import math as chi#this is completely an absurd, but i am just showing how to use it
>>> chi.exp
<built-in function exp>
>>> chi.e
2.718281828459045
>>> #which means you can rename longer modules to shorter one, another use of IMPORT statement is to selectively import members from a module, let me show that part too
>>> from amth import pi
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    from amth import pi
ImportError: No module named 'amth'
>>> #Sorry, i mis-spelled math
>>> from math import pi
>>> pi
3.141592653589793
>>> e
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    e
NameError: name 'e' is not defined
>>> #coz, i had just imported pi
>>> #to import everything, wedo this
>>> from math import *
>>> #now everything in math has been imported
>>> pi
3.141592653589793
>>> e
2.718281828459045
>>> sqrt(5)
2.23606797749979
>>> #but, there could be name collision if i import sqrt from cmath too
>>> from cmath import sqrt
>>> sqrt(5)
(2.23606797749979+0j)
>>> #so the sqrt of math was overwritten & can't be accessed now, so import carefully, or use . operator convention with module name
>>> #your python program can run independently or can be imported into another program
>>> #when interpreter enters your program, it just evaluates each experssion in given source code
```

```
>>> #python uses indented block structure, to mantain the readability of the code,
& better code interpretation by user
>>> print('Hello')#it is that simple in python
Hello
>>> 'Hello'#it will print it with quotes
'Hello'
>>> #let us first go with iterations in python
>>> #the WHILE statement
>>> #it is similar to other programming languages
>>> a=0
>>> while a<5:
    print(a)
    a+=1


0
1
2
3
4
>>> #this extra ENTER key is needed in INTERACTIVE mode, script(source code from file),
does need this
>>> #:(colon) character is used in python to create a indented block
>>> #the FOR statement
>>> for i in range(1,9,3):#range is a lazy-iterator in python3, while return list in
python 2
    print(i)


1
4
7
>>> #it could also be used to extract member from an iterable object, like this
>>> a=['saffron','white','green']#tricolor of INDIAN flag
>>> for i in a:
    print(i)
    i='black'


saffron
white
green
>>> i
'black'
>>> a
['saffron', 'white', 'green']
>>> # a is yet unchanged, it is similar to for-each loop of JAVA
>>> #if you want to change it, go this way
>>> for i in range(len(a)):
    print(a[i])
    a[i]='white'


saffron
white
green
```

```
>>> a
['white', 'white', 'white']
>>> #now we'll proceed to CONDITIONALS (if, elif, else)
>>> a='life'
>>> if a==life:
    print('unchanged')
else:print('mutated')

Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    if a==life:
NameError: name 'life' is not defined
>>> if a=='life':
    print('unchanged')
else:print('mutated')

unchanged
>>> a='live'
>>> if a=='life':
    print('unchanged')
else:print('mutated')

mutated
>>> # think you got that why the error appeared above, if not, read next line
>>> #life there is not surrounded by '(quotes), so it is taken as a identifier(name of data type), wich is yet undeclared
>>> #;(semicolon) is optional in python, but it has its uses too
>>> a=[1,2,3]
>>> for i in range(len(a)):print(a[i]);a[i]=0

1
2
3
>>> a
[0, 0, 0]
>>> #it can still be used to have multiple statements in same line, but i strongly recommende, not to use this
>>> #it will not leave you the practise of C/C++/Java, & the credit goes to piyush here, who showed me this during workshop
>>> #all python files have .py extension & compiled files have .pyc extension(they are similar to .java & .class files), in that context .py is file written by programmer, & .pyc is the file converted into byte-code by python, it can be also executed, but can't be understood by user, NOTICE: .pyc files are implementation dependent & may change with pyton version & environment, but they execute faster than .py files
>>>                         #Data Types of python
>>> #these are: int,float,str(string),tuple,list,set,frozenset,dict(dictionary)
>>> #variables in python don't have type, they get the type of the type of object tehy are assigned to
>>> type(name)#this variable is not yet assigned to any object
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    type(name)#this variable is not yet assigned to any object
NameError: name 'name' is not defined
>>> #The declaration happens automatically when you assign a value to a variable.
>>> #Variables can change type, simply by assigning them a new value of a different type.
```

```
>>> a=4
>>> type(a)
<class 'int'>
>>> a=4.0
>>> type(a)
<class 'float'>
>>> #Python allows you to assign a single value to several variables simultaneously.
>>> a=b=c=5
>>> a
5
>>> b
5
>>> #also multiple objects can be assigned to multiple names
>>> a,b,c=1,2,3
>>> a
1
>>> b
2
>>> #those who are actually present in the worskhop, will get the above is a tuple
assignment
>>> #& can be done this way also
>>> d=1,2,3
>>> e,f,g=d
>>> e,f
(1, 2)
>>> #NUMBERS: int, float & complex(not used that much used in programming)
>>> #numbers are immutable, taht means they can't be changed(their value can't be
changed)
>>> #i know it seems to be horrible for novice programmers, but whenever you did a=a+1
>>> #'a' is asiigned a new memory, it doesn't waste that much memory
>>> #let me show you this
>>> a='108'
>>> id(a)
788060925208
>>> a=a+1
Traceback (most recent call last):
  File "<pyshell#129>", line 1, in <module>
    a=a+1
TypeError: Can't convert 'int' object to str implicitly
>>> #my fault, i took it a string
>>> a=108
>>> id(a)
1894518576
>>> a=a+1
>>> id(a)
1894518608
>>> #see 'a' changes it's memory location
>>> #this is teh reason why ++ and -- are not given in python, coz they are just useless,
if they do exist too
>>> #integer specific now
>>> a=6
>>> a.bit_length()
3
>>> oct(15)
'0o17'
>>> hex(15)
```

```
'0xf'
>>> bin(15)
'0b1111'
>>> #they all just return a string
>>> a=int(bin(17))
Traceback (most recent call last):
  File "<pyshell#144>", line 1, in <module>
    a=int(bin(17))
ValueError: invalid literal for int() with base 10: '0b10001'
>>> #it shows that int() can convert only limited literals to integer
>>> int(4)#int-to-int
4
>>> int(4.75)#float-to-int
4
>>> int('4.8')
Traceback (most recent call last):
  File "<pyshell#148>", line 1, in <module>
    int('4.8')
ValueError: invalid literal for int() with base 10: '4.8'
>>> int('4')
4
>>> #so these conversion functions are not taht powerful, but still useful for us
>>> a.real
6
>>> #we'll see teh real. function in context of complex numbers below
>>> #complex specific now
>>> a=5.65+2.5j
>>> type(a)
<class 'complex'>
>>> a.real
5.65
>>> a.imag
2.5
>>> a.
SyntaxError: invalid syntax
>>> a.conjugate
<built-in method conjugate of complex object at 0x000000B77C188070>
>>> a.conjugate()
(5.65-2.5j)
>>> # imag & real are two data members of complex class, so they are printed, but
conjugate is a method(function), so it needs it's parentheses
>>> #float specific now
>>> a=45.75
>>> a.as_integer_ratio
<built-in method as_integer_ratio of float object at 0x000000B7791EA630>
>>> a.as_integer_ratio()
(183, 4)
>>> #it could be of great help
>>> a=10/3
>>> a
3.3333333333333335
>>> a.as_integer_ratio()
(7505999378950827, 2251799813685248)
>>> # find out why this happens, instead of giving (10, 3)??, an IMPORTANT question
>>> a.conjugate()
3.3333333333333335
```

```
>>> a.hex()
'0x1.aaaaaaaaaaaabp+1'
>>> #this could annoy you, so skip it
>>> a.is_integer()
False
>>> 5.0.is_integer()
True
>>> #COMMON NUMBER FUNCTIONS
>>> abs(-5)
5
>>> abs(-5.5)
5.5
>>> abs(3+4j)
5.0
>>> #above is the modulus of complex number sqrt(a**2+b**2)
>>> exp(1)
2.718281828459045
>>> exp(2)#e**x
7.38905609893065
>>> log(10)
2.302585092994046
>>> #natural logarithm function
>>> log(2.718281828)
0.9999999998311266
>>> #above value passed is given in our calculators
>>> log(exp(125))
125.0
>>> pow(2,5)#similar to 2**5
32.0
>>> 2**5
32
>>> pow(3+4j,2)
Traceback (most recent call last):
  File "<pyshell#190>", line 1, in <module>
    pow(3+4j,2)
TypeError: can't convert complex to float
>>> #pow just takes float or int as an argument, return a float always
>>> # i suggest you to open the math module of python & the number of in bulit functions
available there
>>> math.factorial(5)
120
>>> math.gcd(21,14)
7
>>> math.radians(90)
1.5707963267948966
>>> math.degrees(3.141592654)
180.00000002350313
>>> #this was from calculator too(100-MS)
>>> math.gamma(0.5)
1.7724538509055159
>>> #this is sqrt(pi), i think very less people remembee this from 1st year
>>> sqrt(pi)
(1.7724538509055159+0j)
>>> #actually the values are getting complex coz i used
>>> #from cmath import *            ,later
>>> from math import *
```

```
>>> sqrt(pi)
1.7724538509055159
>>> #got it
>>> #ternary operator like a=b==0?1:2 of C, do exist in python, but in more smart way
>>> #a = value1 if condition else value2
>>> b=6
>>> a=7 if b==6 else 8
>>> a
7
>>> b+=1
>>> c=7 if b==6 else 8
>>> c
8
>>> #this example is self stated
>>> #this way is much convenient & easy to read
>>>
>>> #now we'll discuss TUPLES, LISTS & STRINGS
>>> #both have some similarities, some differences
>>>
>>>                                    #TUPLES
>>> #tuples are immutable, pair of elements
>>> #it can't grow in size, nor it's element be changed
>>> #remember (1,2)==1,2
>>> (1,2)==1,2
(False, 2)
>>> #this is because, == has higher precedence than ,
>>> 1,2
(1, 2)
>>> #i mean by this, actually
>>> a=(1,2)
>>> a.count(1)
1
>>> a[0]
1
>>> a.index(2)
1
>>> #count returns the number of occurence
>>> #a[0] is index based accesss
>>> #index finds out the index of element
>>> a.index(3)
Traceback (most recent call last):
  File "<pyshell#235>", line 1, in <module>
    a.index(3)
ValueError: tuple.index(x): x not in tuple
>>> #raises ValueError if x not in tuple
>>> len(a)
2
>>> a=a+(3,)
>>> a
(1, 2, 3)
>>> #it is a new completely new tuple, previous one is ready to be collected by Garbage
Collector
>>> b=a
>>> #b point to same location as a did
>>> a+=(4,)
>>> a
```

```
(1, 2, 3, 4)
>>> b
(1, 2, 3)
>>> #now Garbage collector won't pick any of these two, coz they are assigned to
references
>>> a=1,2,3
>>> a
(1, 2, 3)
>>> a[1]=2
Traceback (most recent call last):
  File "<pyshell#249>", line 1, in <module>
    a[1]=2
TypeError: 'tuple' object does not support item assignment
>>> a[1]==2
True
>>> #as tuples are immutable, tehy can't be changes
>>> #let us do an experiment
>>> a=[1,2]
>>> b=a,3
>>> b
([1, 2], 3)
>>> a.append(4)
>>> b
([1, 2, 4], 3)
>>> #Yes, i changes the tuple, actually i didn't
>>> #Explaination: when i created b=a,3
>>> #i created the first member of tuples is the referenec of list [1,2]
>>> #so changing the list, changes the value inside tuple
>>> #but, all such modification can be made from that refernce only
>>> b[0].append(5)
>>> b
([1, 2, 4, 5], 3)
>>> a
[1, 2, 4, 5]
>>> #reverse can be also done
>>>
>>>                   #LISTS
>>> #tuples discussed above are like constant array, actually they are, they are
contigously allcoated into memory by python, hence they are somewhat faster than lists
>>> #lists are different, they are mutable, can grow or shrink, thousand of
possibilities for modification do exist
>>> a=[1,2,3,4,5,6]
>>> len(a)
6
>>> type(a)
<class 'list'>
>>> a.append(7.5)#this will append an element to end
>>> a
[1, 2, 3, 4, 5, 6, 7.5]
>>> a.append((8))
>>> a
[1, 2, 3, 4, 5, 6, 7.5, 8]
>>> a.append((9,))
>>> a
[1, 2, 3, 4, 5, 6, 7.5, 8, (9,)]
>>> #last one is a tuple
```

```
>>> a.extend([10,11,12])
>>> a
[1, 2, 3, 4, 5, 6, 7.5, 8, (9,), 10, 11, 12]
>>> a.extend({13,15,14})
>>> a
[1, 2, 3, 4, 5, 6, 7.5, 8, (9,), 10, 11, 12, 13, 14, 15]
>>> #set are stord in sorted order internally
>>> a.extend('hello')
>>> a
[1, 2, 3, 4, 5, 6, 7.5, 8, (9,), 10, 11, 12, 13, 14, 15, 'h', 'e', 'l', 'l', 'o']
>>> #string is also a iterable, character-by-character
>>> a=[7,6,5,8,1,2,9,4,3,0]#recreate list
>>> b=a.copy()
>>> b.sort(reverse=True)
>>> b
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> a
[7, 6, 5, 8, 1, 2, 9, 4, 3, 0]
>>> a.count(9)
1
>>> a.index(9)
6
>>> b.index(9)
0
>>> b.insert(3,'python')
>>> b
[9, 8, 7, 'python', 6, 5, 4, 3, 2, 1, 0]
>>> b.pop()#removes & return last element
0
>>> b.pop()#removes & return last element
1
>>> b.pop(3)#removes & return given index
'python'
>>> b
[9, 8, 7, 6, 5, 4, 3, 2]
>>> b.reverse()
>>> b
[2, 3, 4, 5, 6, 7, 8, 9]
>>> #that much power in your hand, go exploit the list type
>>> a[2]
5
>>> a
[7, 6, 5, 8, 1, 2, 9, 4, 3, 0]
>>> #slicing is discussed after strings
>>>
>>>              #STRINGS
>>> #string in python can be enclosed both in ' or ", ''' & """ are used for special
purposes(multi-line strings & comments)
>>> a='Hindustan College of Science & Technology'
>>> a.capitalize()
'Hindustan college of science & technology'
>>> #it just capitalizes the first char, everything else b lowercase
>>> 'my'.capitalize()
'My'
>>> 'hello'.upper()
'HELLO'
```

```
>>> a='MiXedStrIng'
>>> a.lower()
'mixedstring'
>>> a
'MiXedStrIng'
>>> # so string is immutable, lower or upper just return another stirng
>>> a.endswith('g')
True
>>> a.endswith('Ing')
True
>>> a.islower()
False
>>> a.isupper()
False
>>> a.isidentifier()
True
>>> '5'.isdigit()
True
>>> '5.5'.isdigit()
False
>>> '5.5'.replace(5,6)
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    '5.5'.replace(5,6)
TypeError: Can't convert 'int' object to str implicitly
>>> '5.5'.replace('5','hello')
'hello.hello'
>>> a='HeLlO'
>>> a.swapcase()
'hElLo'
>>> #now we'll discuss split, join & strip
>>> 'i am the programmer'.split()
['i', 'am', 'the', 'programmer']
>>> 'i am the programmer'.split('m')
['i a', ' the progra', '', 'er']
>>> input().split('$')
here$is that$$part of trial
['here', 'is that', '', 'part of trial']
>>> #i don't get, it is just a trial of at where ridhima got stuck
>>> #so split creates a list on basis of character passed , default is ' '
>>> '$'.join(['Ritchie','rich'])
'Ritchie$rich'
>>> '$'.join(['Ritchie',5])
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    '$'.join(['Ritchie',5])
TypeError: sequence item 1: expected str instance, int found
>>> '$'.join({'Ritchie','rich'})
'rich$Ritchie'
>>> #so join is the reverse of split, if join the elements of iterable on teh basis
of character taken at the beginning
>>> a='      hello                '
>>> a.rstrip()
'      hello'
>>> a
'      hello                '
```

```
>>> a.lsrtip()
Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    a.lsrtip()
AttributeError: 'str' object has no attribute 'lsrtip'
>>> #spell error!!!!
>>> a.lstrip()
'hello                '
>>> a
'      hello                '
>>> a.strip()
'hello'
>>> #understand by yourself, it is simple
>>> a='''this
is
a
multiline
string'''
>>> a
'this\nis\na\nmultiline\nstring'
>>> print(a)
this
is
a
multiline
string
>>> a.splitlines()
['this', 'is', 'a', 'multiline', 'string']
>>> a='i am a student of HCST & HCST is an enginnering college'
>>> a.count('HSCT')
2
>>> a.title()
'I Am A Student Of Hcst & Hcst Is An Enginnering College'
>>>
>>>              #Common operation on TUPLES,LISTS &STRINGS
>>> #Concatenation using '+'
>>> [1,2]+[3,4]
[1, 2, 3, 4]
>>> (1,2)+(3,4)
(1, 2, 3, 4)
>>> 'Hello'+'World!'
'HelloWorld!'
>>> [1,2]+(3,4)
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    [1,2]+(3,4)
TypeError: can only concatenate list (not "tuple") to list
>>> #they are common opeartions, doesn't mean you abuse them
>>> [1,2]==(1,2)
False
>>> #so type must be same, then they can be compared for similarity
>>> 'hello'.upper()=='HELLO'
True
>>> #Repitition using '*'
>>> [1,2]*2
[1, 2, 1, 2]
```

```
>>> (1,2)*2
(1, 2, 1, 2)
>>> 'Span!'*4
'Span!Span!Span!Span!'
>>> #i want Spam above, but typing mistake..... :(
>>> #membership using in operator
>>> 3 in [1,2,3,4,5]
True
>>> 6 in [1,2,3,4,5]
False
>>> #similar for tuple
>>> [1,2,3] in [[0,0,0],[1,2,3]]
True
>>> [1,2,3] in [1,2,3]*2
False
>>> 'hello' in 'hello world!'#substring search
True
>>> 'yellow' in 'hello world!'#substring search
False
>>> #index based access
>>> a=[1,2,3,4,5]
>>> a[2]
3
>>> a=(1,2,3,4,5)
>>> a[3]
4
>>> a='computer'
>>> a[2]
'm'
>>> #see indexing of string returns a string, where list & tuple, return the native
member type
>>> #index based assignment(for LIST only)
>>> a=[1,2,3,4,5,6]
>>> a[2]=11
>>> a
[1, 2, 11, 4, 5, 6]
>>> #Slicing(creates a part of list,string ot tuple, into a new memory)
>>> a=[1,2,3,4,5,6,7,8,9,10]
>>> a[1:7]
[2, 3, 4, 5, 6, 7]
>>> a[1:]#upto end
[2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a[:6]#upto 5th index
[1, 2, 3, 4, 5, 6]
>>> a[1:6]
[2, 3, 4, 5, 6]
>>> a[1:8:3]#like range function
[2, 5, 8]
>>> #negative indices work from back
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a[-1]
10
>>> a[1:-2]
[2, 3, 4, 5, 6, 7, 8]
>>> a[1:-2:2]
```

```
[2, 4, 6, 8]
>>> a[-5:-2:2]
[6, 8]
>>> a[-5:-6:2]
[]
>>> #validate the above slice
>>> a='i am my favourite'
>>> a[3:11]
'm my fav'
>>> a[3:11:2]
'mm a'
>>> #Slice Assignment(LIST only)
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a[3:7]=[6,5,4]
>>> a
[1, 2, 3, 6, 5, 4, 8, 9]
>>> a[3:7]=(5,7,0)
>>> a
[1, 2, 3, 5, 7, 0, 9]
>>>a[2:3]=['name']
>>>a
[1,2,'name',5,7,0,9]
>>>a[2:2]=[11]
>>>a
[1,2,,11,'name',5,7,0,9]
>>> #here contigous slice is replaced
>>> #it can be used to grow or shrink list in the middle, no built-in function exist
for these operations
>>> #used in my code mrgsrt2.py
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a[2:4]=['here','is','python3']
>>> a
[1, 2, 'here', 'is', 'python3', 5, 6, 7, 8, 9]
>>> a[2:3]=[]
>>> a
[1, 2, 'is', 'python3', 5, 6, 7, 8, 9]
>>> #both growth & shrink above
>>> a[2:4]='python3'
>>> a
[1, 2, 'p', 'y', 't', 'h', 'o', 'n', '3', 5, 6, 7, 8, 9]
>>> #as i told earlier, string is also iterable
>>>
>>>                    #DICTIONARY
>>> #Python's dictionaries are kind of hash table type which consist of key-value pairs
of unordered elements
>>> #Python Dictionaries are mutable objects that can change their value
>>> #A dictionary is enclosed by curly braces ({ }), the items are separated by commas,
and each key is separated from its value by a colon(:).
>>> a={}
>>> type(a)
<class 'dict'>
>>> a[1]='hp'
>>> a['lenovo']='K3 note'
>>> a
{1: 'hp', 'lenovo': 'K3 note'}
>>> #look above the, way of entering the key:value pairs into dictionary
```

```
>>> a[1]
'hp'
>>> a['lenovo']
'K3 note'
>>> a.keys()
dict_keys([1, 'lenovo'])
>>> type(a.keys())
<class 'dict_keys'>
>>> list()
[]
>>> list(a.keys())
[1, 'lenovo']
>>> a.values()
dict_values(['hp', 'K3 note'])
>>> type(a.values())
<class 'dict_values'>
>>> #keys &values are subclass of dict class, see this in OOP
>>> a.items()#return dictionary as (key,value) tuples
dict_items([(1, 'hp'), ('lenovo', 'K3 note')])
>>> list(a.items)
Traceback (most recent call last):
  File "<pyshell#155>", line 1, in <module>
    list(a.items)
TypeError: 'builtin_function_or_method' object is not iterable
>>> list(a.items())
[(1, 'hp'), ('lenovo', 'K3 note')]
>>> del a[1]#removeing key:value pair
>>> a
{'lenovo': 'K3 note'}
>>> a.clear()#clearing entire dictionary
>>> a
{}
>>> del a #deleting entire dictionary
>>> a
Traceback (most recent call last):
  File "<pyshell#162>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a={1: 'hp', 'lenovo': 'K3 note'}
>>> a
{1: 'hp', 'lenovo': 'K3 note'}
>>> a.get(1,'not found!')
'hp'
>>> a.get(2,'not found!')
'not found!'
>>> #returns value, if key is found, else reurn second parameter as default
>>> a.get(1)
'hp'
>>> a.get(2)
>>> #no second parameter, nothing returned
>>> a.setdefault(3,'hello')
'hello'
>>> a
{1: 'hp', 3: 'hello', 'lenovo': 'K3 note'}
>>> a.setdefault(3)
'hello'
```

```
>>> a.get(2)
>>> a.setdefault(2)
>>> a
{1: 'hp', 2: None, 3: 'hello', 'lenovo': 'K3 note'}
>>> #get this method setdefault(), by your own, see the docstring
>>>          #set & frozenset
>>> #befor i'll elaborate these two, let us discuss hashable
>>> #python need key to be immutable, such values are called hashable, & only these
values can be part of keys of dictonary, set or frozenset
>>> type({})
<class 'dict'>
>>> type(set())#set function is used to create a set from iterable objects
<class 'set'>
>>> set([1,2,3,4])
{1, 2, 3, 4}
>>> set('hello')
{'o', 'h', 'l', 'e'}
>>> {1,2,3,4}#directly this way, as it is different from dictionary
{1, 2, 3, 4}
>>> a={1,2,3,4}
>>> a.add(5)
>>> a.add([6])
Traceback (most recent call last):
  File "<pyshell#188>", line 1, in <module>
    a.add([6])
TypeError: unhashable type: 'list'
>>> #got this, unhashability
>>> a
{1, 2, 3, 4, 5}
>>> a.union({7,8})
{1, 2, 3, 4, 5, 7, 8}
>>> a
{1, 2, 3, 4, 5}
>>> a.intersection({3,4,7})
{3, 4}
>>> a
{1, 2, 3, 4, 5}
>>> a.difference({3,4,5,6})
{1, 2}
>>> a.sy
Traceback (most recent call last):
  File "<pyshell#196>", line 1, in <module>
    a.sy
AttributeError: 'set' object has no attribute 'sy'
>>> a.symmetric_difference({4,5,6,7})
{1, 2, 3, 6, 7}
>>> #search symmetric difference, if you don't know it
>>> a.remove(5)
>>> a
{1, 2, 3, 4}
>>> a.remove(5)
Traceback (most recent call last):
  File "<pyshell#201>", line 1, in <module>
    a.remove(5)
KeyError: 5
>>> #similar method, which doesn't raise, if element is not found
```

```
>>> a
{1, 2, 3, 4}
>>> a.discard(4)
>>> a
{1, 2, 3}
>>> a.discard(4)
>>> a
{1, 2, 3}
>>> # it doesn't raises KeyError like remove did
>>> a.isdisjoint({4,5,6})
True
>>> a.isdisjoint({3,5,6})
False
>>> a.issubset({1,2,3,4,5,6})
True
>>> a.issuperset({1,2})
True
>>> a.issuperset({1})
True
>>> a.pop()#remove & return arbitrary set element
1
>>> a
{2, 3}
>>> a={1,2,3,4,5,6,7,8}
>>> a.update({7,8,9,10})#union_update
>>> a
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>> a.intersection_update({1,2,3,4,5,6,7})
>>> a
{1, 2, 3, 4, 5, 6, 7}
>>> a.difference_update({6,7,8})
>>> a
{1, 2, 3, 4, 5}
>>> a.symmetric_difference_update({4,5,6,7,8})
>>> a
{1, 2, 3, 6, 7, 8}
>>> #notice frozen set i similar to set, but can't be updated(immutable)
>>> a={1,2,3}
>>> a=a+{4,5}
Traceback (most recent call last):
  File "<pyshell#232>", line 1, in <module>
    a=a+{4,5}
TypeError: unsupported operand type(s) for +: 'set' and 'set'
>>> #concatenation doesn't work for dictionary, set & frozenset
>>>
>>> #Boolean opeartors
>>> #and, or ,not
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> False or False
False
>>> not True
```

```
False
>>> not False
True
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> type(None)
<class 'NoneType'>
>>>
>>> #Loop control statement
>>> #continue, Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating
>>> #break, Terminates the loop statement and transfers execution to the statement immediately following the loop
>>> #pass, Used when a statement is required syntactically but you do not want any command or code to execute
>>> def fact(n):
    '''this is the docs string which can be used for
documentation & help at runtime
shown by python when ( of function call is pressed'''
    try:
        k=int(n)
        s=1
        for i in range(1,k+1):s=s*i
        return s
    except:
        print('Invalid Data type passed')

>>> fact(5.0)
120
>>> fact('12')
479001600
>>> fact(7)
5040
>>> fact('2')
2
>>> fact('5')
120
>>> fact([1,2])
Invalid Data type passed
>>> #Strong typing: But Python's not casual about types, it enforces the types of objects
>>> x = 'the answer is'
>>> y=23
>>> x+y
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    x+y
TypeError: Can't convert 'int' object to str implicitly
>>> x+str(y)
'the answer is23'
>>>
>>> #function have default return type as None, which can't be printed by interpreter
>>> None
>>> True
```

```
True#notice that None is not printed
>>>
>>> #There is no function overloading in Python
    #Unlike C++, a Python function is specified by its name alone
    #The number, order, names, or types of its arguments cannot be used to distinguish
between two functions with the same name
    #Two different functions can't have the same name, even if they have different
arguments
>>> #Functions are over-written at each new definition
>>> def myfun(a, b, c=20):
    print(c)
    return a-b

>>> myfun(1,2,3)
3
-1
>>> myfun(1,2)
20
-1
>>> myfun(c=1,a=2,b=3)
1
-1
>>> myfun(1,c=2,b=3)
2
-2
>>> myfun(b=1,2,c=3)
SyntaxError: positional argument follows keyword argument
>>> #all positional arguments must be at left, then all keyword arguments
>>> #similarly default arguments must be on rigt hand side
>>> #functions are first-class data types
>>> #like this
>>> def square(x):
    return x*x

>>> def applier(a,b):
    return a(b)

>>> def inc(x):
    return x+1

>>> applier(inc,5)
6
>>> applier(square,5)
25
>>> #So, function can be:
>>> #Arguments to function
    #Return values of functions
    #Assigned to variablesParts of tuples, lists, etc
>>> #Python uses a lambda notation to create anonymous functions
>>> applier(lambda z: z * 4, 7)
28
>>> f = lambda x,y : 2 * x + y
>>> f
<function <lambda> at 0x00000049474A7BF8>
>>> f(3,4)
10
```

```
>>> v = lambda x: x*x(100)
>>> v
<function <lambda> at 0x0000004944997F28>
>>> v = (lambda x: x*x)(100)
>>> v
10000
>>> s=0
>>> v = (lambda x: x+s)([1,2,3,4,5])
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    v = (lambda x: x+s)([1,2,3,4,5])
  File "<pyshell#60>", line 1, in <lambda>
    v = (lambda x: x+s)([1,2,3,4,5])
TypeError: can only concatenate list (not "int") to list
>>> def square(x):
    return x*x

>>> def twice(f):
    return lambda x: f(f(x))

>>> twice
<function twice at 0x00000049474A79D8>
>>> quad=twice(square)
>>> quad
<function twice.<locals>.<lambda> at 0x00000049474A7D08>
>>> quad(5)
625
>>> #above occurs, function compostition
>>> def counter(start=0, step=1):
    x = [start]
    def _inc():
        x[0] += step
        return x[0]
    return _inc

>>> c1 = counter()
>>> c2 = counter(100, -10)
>>> c1()
1
>>> c2()
90
>>> c1
<function counter.<locals>._inc at 0x00000049474A7F28>
>>>
>>> #Compare with X is Y:
#X and Y are two variables that refer to the identical same object
>>> a=[1,2,3]
>>> b=a
>>> c=[1,2,3]
>>> a==b
True
>>> a==c
True
>>> a is b
True
>>> a is c
```

```
False
>>> #Note: Actually 'and' and 'or' don't return True or False but value of one of their sub-expressions, which may be a non-Boolean value.
>>>#Empty iterables are treated as false
>>>#example: 0, 0.0, (), [], '', {}, set, frozenset()
>>> 1 and 2
2
>>> 1 or 2
1
>>> [] and 3
[]
>>> [] or (1,2)
(1, 2)
>>>1 and 2 and 3
3
>>> #and & or uses short circuit evaluation
>>> #An assert statement will check to make sure that something is true during the course of a program
>>> a=5
>>> assert a>4
>>> assert a>5
Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    assert a>5
AssertionError
>>>
>>>                  #Python's Higher Order Function
>>> def square(x):
    return x*x

>>> def even(x):
return 0 == x % 2
SyntaxError: expected an indented block
>>> def even(x):
    return 0 == x % 2

>>> list(map(square, range(10,20)))
[100, 121, 144, 169, 196, 225, 256, 289, 324, 361]
>>> tuple(filter(even, range(10,20)))
(10, 12, 14, 16, 18)
>>> list(map(square, filter(even, range(10,20))))
[100, 144, 196, 256, 324]
>>>
>>> #But many Python programmers prefer to use list comprehensions, instead
>>> #A list comprehension is a programming language construct for creating a list based on existing lists
#Haskell, Erlang, Scala and Python have them
>>> #Python's notation:[ expression for name in iterable if condition]
>>> [int(x) for x in input().split() if int(x)%2==0]
1 2 3 4 5 6 7 8 9 10 11 12
[2, 4, 6, 8, 10, 12]
>>> li=[3,6,2,7]
>>> [elem*2 for elem in li]
[6, 12, 4, 14]
>>> li = [('a', 1), ('b', 2), ('c', 7)]
>>> [ n * 3 for (x, n) in li]
```

```
[3, 6, 21]
>>> [2*x+1 for x in[10, 20, 30]]
[21, 41, 61]
>>> list(map( lambda x:2*x+1, [10, 20, 30]))
[21, 41, 61]
>>> [2*x+1 for x in [10, 20, 30] if x > 0]
[21, 41, 61]
>>> list(map( lambda x:2*x+1,filter( lambda x:x > 0, [10, 20, 30])))
[21, 41, 61]
>>> #else block with loops
>>> a=['hsct','aec','hitm']
>>> for i in a:
    if i=='hcst':#which i misspelled as hsct
        break
else:print('Search failed')

Search failed
>>> for i in a:
    if i=='aec':
        print('value found')
        break
else:print('Search failed')

value found
>>> #in the same way else is used with while loop, also
>>>
>>> #Taking user input using input() function, which returns a string
>>> a=input()
hindustan college
>>> a
'hindustan college'
>>> a.split()
['hindustan', 'college']
>>> a
'hindustan college'
>>> a=int(input())*10
7
>>> a
70
>>> a=input()
[1,2,3,4]
>>># as is not a list, it is taken as a string
>>> a
'[1,2,3,4]'
>>> a=list(input())#verified here
[1,2,3,4]
>>> a
['[', '1', ',', '2', ',', '3', ',', '4', ']']
>>> #so the way to input the native python object is to use eval() function
>>> a=eval(input('Enter the composite data type, for elaborating power'))
Enter the composite data type, for elaborating
power{'list':[1,2,3,4.5],'lumia':1020,(1,2,3):'tuple'}
>>> a
{(1, 2, 3): 'tuple', 'lumia': 1020, 'list': [1, 2, 3, 4.5]}
>>> type(a)
<class 'dict'>
```

```
>>> #see, how python automatically detects tha data types
>>> type(a['list'])
<class 'list'>
>>> type(a['lumia'])
<class 'int'>
>>> type(a[(1,2,3)])
<class 'str'>
>>> #i'll explain need for eval function , while discussing file handling, before that,
let us go with exception handling
>>> a=4
>>> a/0
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    a/0
ZeroDivisionError: division by zero
>>> try:
    a/0
except:#this will catch all errors
    print('divison by zero occurs')


divison by zero occurs
>>> try:
    b=[1,2,3,4]
    a/0
except ZeroDivisionError as ZDE:#this will all errors
    print(ZDE)
except :print('all other errors here')

division by zero
>>> a=[1,2,3,4]
>>> a[4]
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    a[4]
IndexError: list index out of range
>>> #exception handling is similar to what is already popular in Java,
>>> #except is used as catch
>>> #raise is used as throw
>>> try:
    raise IndexError('my_string')
except IndexError:print(IndexError)

<class 'IndexError'>
>>> try:
    raise IndexError('my_string')
except IndexError as ID:print(ID)

my_string
>>> try:
    raise IndexError('my_string2')
except Exception as EID:print(EID)#Exception is superclass of all exceptions

my_string2
>>> #also, there is a finally clause, as it was in most other programming languages
>>> try:
```

```
    a=int(input())
    a/0
except ZeroDivisionError:print('you tried divide by zero')
except:print('all other exceptions caught here')
finally:print('bye-bye')


all other exceptions caught here
bye-bye
>>> try:
    a=int(input())
    a/0
except ZeroDivisionError:print('you tried divide by zero')
except:print('all other exceptions caught here')
finally:print('bye-bye')

52
you tried divide by zero
bye-bye
>>> #there are large number of in built exceptions, also we can create our own exception, by inheriting exception class, or any of its subclass
>>> raise TypeError('i wish, so i can throw error')
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    raise TypeError('i wish, so i can throw error')
TypeError: i wish, so i can throw error
>>> #Whenever a runtime error occurs, it creates an exception object. The program stops running at this point and Python prints out the traceback, which ends with a message describing the exception that occurred.
>>> For example, dividing by zero creates an exception:
>>> print(55/0)
Traceback (most recent call last):
File "<interactive input>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
So does accessing a non-existent list item:
>>> a = []
>>> print(a[5])
Traceback (most recent call last):
File "<interactive input>", line 1, in <module>
IndexError: list index out of range
Or trying to make an item assignment on a tuple:
>>> tup = ("a", "b", "d", "d")
>>> tup[2] = "c"
Traceback (most recent call last):
File "<interactive input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
SyntaxError: invalid syntax
>>> def get_age():
    age = int(input("Please enter your age: "))
    if age < 0:
        # Create a new instance of an exception
        my_error = ValueError("{0} is not a valid age".format(age))
        raise my_error
    return age
#line 5 creates an exception object
```

```
>>> get_age()
Please enter your age: 5
5
>>> get_age()
Please enter your age: -1
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    get_age()
  File "<pyshell#78>", line 6, in get_age
    raise my_error
ValueError: -1 is not a valid age
>>> def recursion_depth(number):
    print("Recursion depth number", number)
    try:
        recursion_depth(number + 1)
    except:
        print("I cannot go any deeper into this wormhole.")

>>> recursion_depth(0)
Recursion depth number 0
Recursion depth number 1
Recursion depth number 2
Recursion depth number 3
Recursion depth number 4
Recursion depth number 5
Recursion depth number 6
Recursion depth number 7
Recursion depth number 8
Recursion depth number 9
Recursion depth number 10
Recursion depth number 11
Recursion depth number 12
.
.
.
.
Recursion depth number 975
Recursion depth number 976
Recursion depth number 977
Recursion depth number 978
I cannot go any deeper into this wormhole.
>>> #so, we see there is a limit on recursion depth, you can't go in infinite recursion
anyhow, but you can change this limit manually.
>>> #upto here i had discussed everything, that i had done in previous 5 days of
workshop, onwards from now, i am going to describe file handling, in python3, which
doesn't seems to be that much important for Novice Programmer, as it slightly
un-manageable in other languages like C/C++, but with python it is simple00
>>> import sys
>>> def f(n):
    print(n,end=' ')
    if n%10==0:print()
    f(n+1)


>>> sys.getrecursionlimit()
1000
```

```
>>> sys.getwindowsversion()
sys.getwindowsversion(major=6, minor=3, build=9600, platform=2, service_pack='')
>>> sys.setrecursionlimit(100)
>>> f(0)
0 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78#Find out why recursion ends here at 78, while limit is 100
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    f(0)
  File "<pyshell#9>", line 3, in f
    f(n+1)
  File "<pyshell#9>", line 3, in f
    f(n+1)
    .
    .
    .
    .
    .
  File "<pyshell#9>", line 2, in f
    print(n)
  File "D:\Python\Python35\lib\idlelib\PyShell.py", line 1344, in write
    return self.shell.write(s, self.tags)
RecursionError: maximum recursion depth exceeded while pickling an object
>>> sys.setrecursionlimit(5)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    sys.setrecursionlimit(5)
RecursionError: cannot set the recursion limit to 5 at the recursion depth 5: the limit
is too low
>>> sys.setrecursionlimit(25)
>>> #breaking multiple loops using try except mechanism
>>> try:
    for i in range(1,6):#[1,5]
        for j in range(6,11):#[6,10]
            if i+j==13:
                print('i=',i,'j=',j)
                raise Exception
except:
    pass#so we get to know the use of pass keyword too
finally:
    print('hello, we just used pass')


i= 3 j= 10
hello, we just used pass
>>>
```