type, print, sys, getsizeof

23/1/18

Python

ver 3.5, 3.6 (Object oriented programing)

It is a scripting language.

data variables:
↝ reference variable

int	a = 10	python has "type" function to know about the data type of any thing
float	b = 2.5	
String	c = "hello"	
boolean	d = True	
complex	e = 4+3j	

```
a = 10
print(type(a)) ~> int
```

o/p <class int>
<class str>

↳ type's data type is class.

features

in python there is no actual limit like in C
int has 4 bits
in java int has 32 bits space.
but in python there is actually no limit

⇒ how to print no. of bytes for any data variable

java - package
python - module

inbuilt module in python

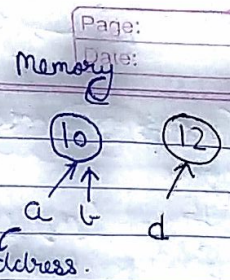
→

```
import sys
a = 9473673997 43211
print(sys.getsizeof(a))
```

↳ only possible in System module.

id

Page:
Date:



a=10
b=10 // no need to store 10 twice
d=12 // new memory allocated

// function to print the address of any variable
→ to return the address.

print (id(a)) > same output
print (id(b))
print (id(d)) — different output.

✓ Python is good for scientific and mathematical calculations application

a=2
b=3
print (a**b) // a^b = 2^3 = 8

✓ Open source, no licence issue, free of cost.

s="hello" > double quotes and single quotes both are accepted in string.
s1='hello'
s2="h"
s3='h'

print("hello") comment.

''' multiple line comment. '''

✓ It ~~supports~~ There is plenty of inbuilt module functions

✓ Supports various coding styles

Page:
Date:

no switch case available
Python use elif to optimize the switch case

if
elif
elif → else if
elif
else

multiline string

>> S = """ hello world
this is a comment """ It is a comment but it is being assigned to S

? print(s) check the output

30/1/18

Basic way how to interact with python.
interacting mode:

1) windows + R (Run dialog box)

2) cmd type || command prompt.

3) type ~~python~~ enter

you will find prompt for python

Only one line can be executed at a time hence we should not go for this method for big programs

>>> a=10 enter
>>> a don't need to write print.
>>> 10

# to know the version of Python
>>> import sys
>>> sys.version

? Python is case sensitive or not. NO

methods to print

used for

Page:
Date:

Url: python.org // download the setup

C: System Program/python/ IDLE 64 bit
IDLE 32 bit

File

>>>

To run the file we just press F5

Saving will be automatically located.

// Pydev module in Eclipse makes it possible for running python in Eclipse.

30/1/18

Taking input from users

a = int (input ("Enter a number"))

↳ Specified for taking integer values otherwise we will be storing String.

c = a+b // if data type was not mentioned in the input then string was taken as input and its value will be String concatination.

Methods to print:

print ("Result is ", c)
print ("a = {0}, b = {1} c = {2}".format(a,b,c))
print ("Result is " + str(c))
print ("a = %d b = %d c = %d" %(a,b,c))

Used for conversion

print (int ('a', 16)) // 10

String, base of String like 16 Hexa, 8 octa.

print ("{0:b} {1:b}".format(10,4)) // 1010 100

no. of variables

b - binary
o - octal
x - hexa

Binary

x - hexa
o - octal
b - binary

bin
hex
oct → inbuilt functions

a = 0b100
now it will be used as a binary 100

print(a) //4
print(bin(a))

→ Operators

a = 2000 Integer
b = 5 Integer
c = b**a (power)

★★ To print something multiple times without using loop

print ("Hello" *100)

★ a = 5
b = 2 → doubleslash gives integer part only
c = a//b //2.
c = a/b → floor value. //2.5

★★ Swapping of 2 variables

a, b = 10, 20
print(a) //10
print(b) //20
a, b = b, a
print(a) //20
print(b) //10

Membership Operator

```
s = "hello world"
s1 = "hello"
```

& → in

```
if s1 not in s :
    print ("yes")
else :
    print ("no")
```

// no

Identity Operator

compares identity

```
s = "hello"
s1 = "hello"
```

& → is not

```
if s1 is s :
    print ("yes")
else :
    print ("no")
```

```
a = 9
b = 5
b = ~b+1
print (a+b) //4
```

Subtraction without using '-' sign.

```
print (a<<2)
```

break continue pass

Page:
Date:

no switch case use if elif else

* for x in range(10, 0, -2):
  print(x)

start, end, increment/decrement

10
8
6
4
2

while loop

1/1

* String

St = "hello how are you"

#print(st.count("o")) 4
print(st.find("z", 6, 9)) -1
print(st.find"o")) 15

print(st.isupper()) // False
print(st.endswith("you")) // True
print(st.startswith("he")) // True

// counts no. of

Page:
Date:

★ 111

```
if 4==4:
#print("four")
pass
```

} if its length empty then a error is coming. to solve it just write Keyword "pass"

```
s="hello"
print(s.isalpha()) || True
      .isalnum() || False
```

→ alphabet.
→ alphanumeric. If space is there always False.

```
s=input("Enter String")
list1=s.split()
print(list1)
```

Enter String Hello my name.

['hello', 'how', 'are', 'you']

if index number is given like list1[0] // hello.

// counts no. of words.

```
if [ print(len(mylist)) ]
```

↳ gives the length of String.

```
s="hello world how are you"
print(len(s)) // no. of element ⇝ 23
```

```
s="hi, how, are, you"
mylist=s.split(",")
print(mylist)
```

→ seperates words to form list using ,

["hi", "how", "are", "you"]

Join function → joins the splited list

Page:
Date:

```
s= "hello* world *how* are *you"

mylist = s.split("*")
print(mylist)
a="*".join(mylist)
print(a)
```

```
Import string
# returning all letters
print(string.ascii_letters)
# returning lowercase letters
print(string.ascii_lowercase)
```

similarly uppercase.

```
# returning all punctuations
print(string.punctuations)

# returning whitespaces
print(string.whitespace)
```

String Slicing

```
s= "hello world"
print(s[0:7]) || hello w
```

↳ 7th index not included.

```
(s[0:700]) || index out of range not in python
```

```
Print(s[-1]) || d
(s[-3]) || r
```

print(s[0:300:2]) hlowrd

print(s[::-1]) || dlrow olleh

reversed string.

print(s[0:100:-1]) || no output.

String concatination (string + "hello")

concatination done.

```
s="hello world"
for x in s:
    print(x, end="") hello world.
```

```
s="aaltroorkhaa"
a=""
for x in s:
    if x in a:
        pass
    else
        a=a+x
print(a).
```

```
from itertools import groupby
    s="AABBBBAAZ"
    for k,g in groupby(s):
        print(k, len(list(g)))
```

a 2
b 4
a 2
z 1

Page:
Date:

```
for x in s:          ||  for x in range(0,10)
    x                ||      s[x]
```

same.

list:

```
list1 = [1, 2, "hello", 5.5]
print(list) //
print(list[0]) // 1
```

list is mutable {(can be modified)}

```
list1[0] = 100
print(list1) // list1 will be printed with a change
                [100, 2, "hello", 5.5]
```

Slicing

call by value

```
list2 = list[:]
```

value of list is copied to list2

so id(list2) ≠ id(list1)

but if list2 = list(1)

then the value is not copied

call by reference.

id(list2) = id(list1)

len

• append

if is_full()

else.

list1.append(data)

else

list1.pop()

```
list1 = [ ]
print (type (list1))
list1 [0] = 1
print (list1)  // Error
```

So to add element in the list1 {Empty list}
we use append function

```
list1.append(1)
list1.append (2)
print (list1) // [1, 2]
```

```
list1 = [None] *5
print (type (list1)) // <class 'list'>
list1 [1] = 100
print (len (list1))  5
print (list1)  [None, 100, None, None, None]
```

another way to create a list

```
list1 = list ()   ~~~> Empty list will be created
print (type (list1))
list1.append (100)
print (list1)  //[100]
```

```
List1 = [8, 7, 6, 5, 4, 3, 2, 1]
print (sum (list1)) // sum of all values in list //36
print (min (list1)) //1
print (max (list1)) //8
list1.reverse()
print (list1)  // [1, 2, 3, 4, 5, 6, 7, 8]
```

Page:
Date:

```
list1.sort()
print(list1)

list = [7,8]*2
print(list) // [7,8,7,8]

list.insert(0,100)
            ↳index  ⟶ value

list1 = [7,8,5,6,4,3,2,1]
print(list1.index(3)) // 5
      index(100) // Error.

list1 = [7,8,8,8,6,5,8,8,2]
print(list1.count(8)) // 5
      count(4) // 1
      count(100) // 0
```

agar module pura import karwana hai tab
import Counter

specific import karna hai tab

```
from collections import Counter
list1 = [7,8,8,8,5,7,7,6,8,8,5,5,4,3,2,1]
print(Counter(list1)) ⟶ return type is dictionary {}
      ↳C is upper case.

Counter({8:5, 5:3, 7:3, 1:1, 2:1, 3:1, 4
```

reverse order.

```
list1 = [1,2,3]
list2 = [4,5,6]
print(list1 + list2)  [1,2,3,4,5,6]

list1.extend(list2)
print(list1)  ## [1,2,3,4,5,6]

list1 = [1,2,3,4,5,6]
print(list1.pop())  ## 6
print(list1.pop())  ## 5
print(list1)  [1,2,3,4]

print(list1.pop(3))  ## 4   (3 → index)
print(list1)  ## [1,2,3]

print(list1.remove(3))  (3 → value)
print(list1)  ## [1,2]
```

List Comprehension

```
list1 = [x for x in range(10)]
print(list1)  [0,1,2,3,4,5,6,7,8,9]

list1 = [x*x for x in range(10)]
print(list1)  ## [0,1,4,9,25,36,49,64,81]

list1 = [1,2,3]
print(list1[::-1])  ## [3,2,1]
```

reverse order

index 3 is a complete list

```
list1 = [1,2,3,[1,2],[9,10]]
print(list1[4][1]) || 10
```

Dictionary {}:

```
d = { }   print(type(d)) || <class 'dict'>
d = {"jack": 50, "mack": 20}
```

Key — value

=> Unordered, does not supports index.

```
d = {"jack": 50, "mark": 20, "jack": 70}
print(d) || {'jack': 70, 'mark': 20}
```

only once.

with the help of Key we can access value, but if we have value, we can't access Key

```
print(d["jack"]) || 70
```

```
print(d.keys())
print(d.values())
print(d.items())
```

```
for k in d.keys():
    print(k, d[k])
```

jack 70
mark 20

★ Duplicate keys nahi ho sakti Dictionary me.

```
for v in d.values():
    print(v, d[v])  // Error
```

↳ cannot return key by knowing values

```
for k,v in d.items():
    print(k, v)
```

mark 20
jack 70

Sorting in dictionary

```
d1={"jack":40, "mack":10, "amit":30,}

for k in sorted(d1.keys()):
    print(k,d1[k])
```

↳ , reverse=True prints in reverse order.

Swapping

```
d2={ }

for k,v in d1.items():
    d2[v]=k
print d2  // {40:'jack', 10:'mack', 20:'tony'}
```

now we can perform sorting in dictionary with respect to keys.

```
for k in sorted(d2.keys()):
    print(k, d2[k])
```

To delete any item from Dictionary

```
del d1["mack"]
```

d1.clear() clears whole dictionary still its type remains dict

```
seq = ("name", "address", "prof")
d2 = {}
print(d2.fromkeys(seq,)) || {'address': None, 'name': None, 'prof': None}
```

list of dictionaries

```
list1=[]
ans='y'
while ans=='y':
    d={"name":"","age":0}
    d["name"]=input("enter name")
    d["age"]=int(input("enter age"))
    list1.append(d)
    ans=input("continuing.... (y/n)")
print(list1)
```

[{ }] => list of dictionaries

```
pets1 = {"cat": "feline"}
pets2 = {"dog": "animal"}
pets1.update(pets2)
print(pets1) || {"cat": "feline", "dog": "animal"}
print(pets2)
```

20/2/18

```
from array import array
a = array("i")
a.append("a")
```

immutable — tuples
mutable — sets

Set.

Set is unordered, every element is unique and duplicate is not allowed.
immutable are set elements but set itself is mutable.

Set creation

Empty set
```
s = set()
```

```
my_set = {1.0, "Hello", (1,2,3)}  // different data types possible
```

=> Updating a set.

```
set = {1,2,3}
print(set)          // {1,2,3}
set.update([2,3,4])   * list is passed
print(set)          // {1,2,3,4}
```

Page:
Date:

```
list1 = [1,1,2,2,3,3,4,4]
s2 = set(list1)
print(s2)

print(s2[0]) // Set does not support indexing.
```

Set is unordered.

Inbuilt functions of set.

⇒
```
items = set()
items.add("cat")
items.add("dog")
items.add("rat")
print(items)
```

⇒ discarding (deleting)

```
items.discard("man") // nothing happens as man was not in the list
```
but if
```
items.discard("cat") // element is deleted.
```

⇒ remove()

```
items.remove("man") // error unlike the discard case
items.remove("cat") // element will be deleted.
```

⇒
```
a={1,2,3,4,5}
b={4,5,6,7,8}
print(a-b)   letters in a but not in b
      (a|b)   "   "either a or b
      (a&b)   "   "both a and b
      (a^b)   "   "a or b but not both.
```

Immutable
int string
float tuple
complex frozensets

Mutable
list
dictionary
sets

Page:
Date:

```
h1 = {1,3,5,7}
h2 = {1,3}

if h2.issubset(h1):
    print("is a subset")

if h1.issuperset(h2):

print(h1.intersection(h2))

set3 = h1.union(h2)
print(set3)

print(A.symmetric_difference(B))   (A-B)U(B-A)
print(B.                    (A))
                  ↳ must

set1 = {1,2,3,4,1,2,3,4}
print(len(set1))
```

as man
list
d.

discard
eted.

Tuple

We cannot change the elements of a tuple once it is assigned whereas in list it can be changed.

```
a = (1,2,3,4,5)
print(type(a))  # <class 'tuple'>

print(a[0])  # supports indexing.

a[0] = 10  {error}.
```

```
t = (1,2)
print(type(t))   // tuple
t2 = (1)
print(type(t2))  // string
```

```
t3 = (1,)
print(type(t3))  // tuple
```

} atleast 1 comma is necessary for tuple

Inbuilt methods

```
d1 = (1,2)
d2 = ('x','y')
d3 = d1+d2   added tupples.
```

now in tuple indexing is supported.

```
del d2[0]   // Error
```

but complete tuple can be deleted by:

```
del d1
```

String index is supportable so slicing is possible

```
dat = [10,20]
data = tuple(dat)   inbuilt function.
print(data) // (10,20)
```

```
print(min(data))  // 1
      (max(data))  // 4
      (len(data))  // 4
```

Page:
Date:

Application in tuple

```
print(d1*2) || (1,2,1,2)
print("hello"*2) || hello hello
```

```
data = (1,2,3,4)
data2 = (1,2,3,4)

print(sum(data))  ||10
print(sorted(data, reverse = True)) ||[4,3,2,1]
```

Functions →

def keyword

eg.

★ no need of caring about data type.

```
def fun(name, age):        # arguments
    print("name is", name)
    print("Age is", age)

fun("amit", 30)  // function call
fun(30, "amit")  // now error  } to get rid of this
```

Keyword arguments

```
fun(age=30, name="amit")   // now perfect answer
```

acting as Keyword

→ Suppose if we write as age1 #Error

---

ast 1
mma is
ecessary
for tuple

s possible

Page:
Date:

=> Optional arguments

```
def fun(name, age=10):
    ____
    ____

fun(name="amit", age=20)
```

age 20 will be taken but if we pass only name then age=10 is automatically assumed.

---

multiple arguments

```
def fun(*var):          # * → to have any no. of arguments
                        # var has became tuple
    if len(var)==0:
        print("No arg")
    elif len(var)==1:
        print("one arg")
    elif len(var)==2:
        print("Two arg")
    elif len(var)==3:
        print("Three arg")
    else
        print("hello")
```

```
fun()
fun(3)
fun(3,4)
fun(4,4,4)
```

} function calls of multiple argument.

```
def fun(a, *var)
```

// atleast 1 argument is necessary else error

Page:
Date:

Keyword argument with variable length → dictionary type.

```
def fun(**var):
    for k in var.keys():
        print(k, var[k])

fun()
fun(name = "amit", age = 20)
fun(name = "tony", age = 23, address = "knp")
```

age 20
name amit
age 30
address knp
name tony

```
fun("amit", 20)
```
// Error as value is assigned and no key is assigned.

Return functions

```
def fun(a, b):
    return a+b, a-b, a*b
x, y, z = fun(3, 3)
print(x, y, z)
```

x ∈ a+b
y ∈ a-b
z ∈ a*b

```
total = 0
def add1(a, b):
    total = a+b
    print("in side", total)   // 6
add1(3, 3)
print("out side total", total)   // 0
```

Page:
Date:

Out now global phenomenon

```
total = 0
def add1(a,b):
    global total
    total = a+b
    print("in side", total) #6
add1(3,3)
print("out side total", total) #6
```

Recursion

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```

recursion

This is the process of stack

```
x = fact(0)
print(x)
```

To proove recursion is a stack (~~FIL~~ LIFO) process so the best example is decimal to binary converter

```
def b(x):
    if x>1:
        b(x//2)
    print(x%2, end=" ")
```

floor division (ignores fraction part, just return integer part).

b(8) → 1000

Date:
Page:

Functions: (call by value / call by reference)

```
list1 = [1,2,3,4]
def fun (list2):
    list2[0] = 100
    print (list2) // [100,2,3,4]

fun (list1)          // call by reference
print (list1)        [100,2,3,4]

fun (list1[:])       // call by value.
[1,2,3,4]
```

no changes here as it is call by value new location is created for list2 so inside function its value is different and outside it is different

⇒ Lambda Functions:

```
f = lambda x: x*2
print (f(3)) // 6

f1 = lambda a, b: a+b
print (f1(4,4)) // 8
```

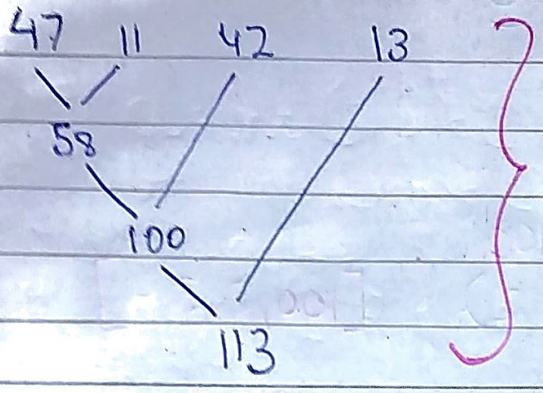
⇒ Without swapping we can sort using lamb in dictionary.

```
S1 = sorted (mydict.items(), key = lambda t: t[1])
print (S1)
S2 = sorted (mydict.items(), key = lambda t: t[0])
print (S2)
```

sorted → user defined; key → keywords; t[1] → value are passed; key = → key is passed

no new line taken

```
from functools import reduce
print(reduce(lambda x,y: x+y,[47,11,42,13]))
```



Methodology of reduce + lambda function.

map & zip

```
for x,y in zip(l1,l2):
    print(x,y)
print(list(zip(T1,T2,T3)))
```

```
items = [1,2,3,4,5]
def sqr(x): return x**2
print(list(map(sqr,items)))
```

★ we can invoke a function no. of times without loop

filter

```
fib = [0,1,1,2,3,5,8,13,21,34,55]
result = filter(lambda x: x%2!=0, fib)
print(list(result))
```

2 meth

Modules

=> import math
print (math.factorial (122))

Some other functions are:

math.pow (2,3)
math.sqrt (25)
math.ceil (3.9876)
math.floor (3.7679)
round (5.1)
abs (-50)
max (1,2,3)
min (1,2,3)

=> import random

random.randint (0,5) → within a range.
random.random ()
int (random.random ()*100)
random.choice (['red', 'black', 'green'])
L = [1,2,3,4,5,6]
random.shuffle (1)
print (1)

Creating own modules

methods.
1) import mymod → user defined
mymod.f1()
mymod.f2()
mymod.f3().

2) ~~import~~ from mymod import *
f1()
f2()
f3()

not taken as next line.

20/3/18

Date:
Page:

Exception Handling:

```
try:
    a = int(input("enter a number"))
    b = int(input(" " " "))
    c = a/b
    print("Result is", c)
except:
    print("divisor is zero")
else:
    print("i am else")
finally:
    print("i am finally")
```

- If exception occurs → except:
- otherwise if no exception → else:
- finally: it is will always be executed.
  - ↳ like database closing
    - ↳ even if exception occurs database should close
    - or closing of a file.

Suppose for the same code above:

Multiple errors we have to be specific

```
try:
    c = a/b
    list1 = [1,2,3]
    print(list1[100])
    print("Result is", c)
except ZeroDivisionError as e:
    print("divisor is zero")
except IndexDivisionError as e:
    print("error message")
```

Page:
Date:

now if we run program and a zero is input in b.

★ division is zero occurs and list error is not even reached. so the first error reached makes the program hault.

★ if in except zero division error as e :
print (e) // objects it will automatically print its own predefined error message.

★ like for IndexDivisionError as e1 :
print (e1) list index out of range

Save file :

Contents are being transferred from main memory to secondary memory.

Open file

Contents are being transferred / copied from 2nd dry memory to primary (main) memory)

★ File Handling

→ Single slash will not be considered as escape sequence as path provided it will be considered as escape sequence.

write / finish.

```
f = open ("e:\\test.txt", "w")
f.write ("hello world") //writed in test.txt file
f.close() // file is closed.
```

now for escape sequence handling in other way.

```
f = open (r"e:\test.txt", "w")
f.write (r"hello \n world")
f.write (r"hello \\n world")
```

both will print as hello \n world
no new line taken

not taken as next line.

Page:
Date:

Read:

```
f = open(r"C:\test.txt", "r")
s = f.read()
print(type(s))   # type
print(s)         # whole content printed
f.close()
```

```
s = f.read(5)    # first five element read.
s1 = f.read(5)   # next read operation will start from the place where the cursor was left after s (read option).
print(s)  # Hello
print(s1) # _Worl
```

✓ `s = f.readline()` # reads 1st whole line
* Cursor is left at the end.

✓ `s1 = f.readline()` # reads 2nd whole line

`s = f.readlines()` # reads all lines and return type is a list.

['hello world \n', 'second line \n', 'last']

- \n: It is read by computer not visible to us next line
- last: no \n at the last line

Page:
Date:

```
list1 = ["hello", "world"]
f = open(r"e:\test.txt", "w")   || Overwrites
s = f.writelines(list1)
f.close
```

file :→ hello_world
if we have given
list1 = ["hello\n", "world"]
file :→ hello
world

```
f = open(r"e:\test.txt", "w")
f.write("hello wolrd \n second line \n last)
s = f.readline()
print(f.tell())   || Current position of cursor.
```
like 13

To move the cursor.

```
f.seek(0)
print(f.tell())   || 0
```

To rename your file.

```
import os
os.rename('mno.text', 'pct.txt')
```

remove director.

```
import os
os.rmdir("e:\\____")
```

Page:
Date:

remove file.

```
import os
os.remove("e:\\pgm.txt").
```

make a directory

```
os.mkdir("e:\\new")
```

to know attributes of file.

```
fo = open("f1.txt", "rb")
```

"rb" → binary mode.

```
print (fo.name)   name of file
      (fo.close)  closed or not. (T/F)
      (fo.mode)   opening mode.
```

To get current working directory.

```
import os
print (os.getcwd())
import os.
os.chdir("e:\\new")
print (os.getcwd())
```

Regular Expression

used for validation of data

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match(r'(.*) are (.*)', line)
if matchObj:
    print("matchObj.group(0): ", matchObj.group(0))
    print("matchObj.group(1): ", matchObj.group(1))
    print("matchObj.group(2): ", matchObj.group(2))
else:
    print("No match!!")
```

( ) groups
. represent any character except new line.
* how many occurances of a character.

before this any character is applied

searching word

Cats are smarter than dogs
Cats
Smarter than dogs

.* Zero or more character of any type

```
line = "are dogs smarter than"
matchObj = re.match(r'dogs', line)
if matchObj:
    print
else:
    print("No match!!")
```

→ searches if dogs is present at the first word only else false bad case to search elements.

```
se
searchObj = re.search(r'dogs', line, re.I)
```

ignore cases

```
if searchObj:
    print(searchObj.group(0))  || now search is successfull
```

Page:
Date:

```
phone = "2004-959-559 # this is phone numbers"
# Delete python-style comments
num = re.sub(r'#.*$', "", phone)
```

→ end tak chale.

→ decide no. of occurances

→ 200

d - digits
D - other than digits

```
# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print("Phone num:", num)
```

→ small digits are removed.

→ capital other than digits are removed.

|| 2004959559

---

findall(): → r: → find all the occurances

(Returns type is a list).

↳ it does case sensitive search.

```
sentence = 'Dogs and Cats are Animals. Dog is Fan'
print(re.findall(r'Dog', sentence))
```

→ ['Dog', 'Dog']

↳ works on substring as this one comes from Dogs

\w – word character + one or more occurances
\W – other than word character ? zero or one occurance.
[] – range. * zero or more occurances

Page:
Date:

Data validation : - - - - ^ → shows the beginning
$ → shows ending.
{ } max min limit

line = " a##sd123@gmail.com "

matchobj = re.match(r'(^\d$)', line) # one digit only

= re.match(r'(^\d{1,}$)', line
min one → max any

line = " 123456 "
r'(^\d{6,6}$)', line || allowed.

[ ] range.

re.match(r'([a-z]+$)', line
line = " abcdef " → one or more occurances
|| allowed.
→ only lower case.

r'([A-Z]+$)', line
line = " ABCD " || allowed.

r'([0-9]+$)'

decimal ke baad 2 allowed hai max and min both.
(r'^\d*.\d{2,2}$', line)
7654.87 allowed.

r'(^\d?)', line # zero or more digits.

email address

r'^[a-z0-9-]+@[a-z].*\.(com)+$

has its own meaning but when followed by ^ last it means and becomes normal

Object oriented programming:

Code reusability

To describe any object we should know about its attributes.

S - State
B - behaviour
I - Identity

Class is a plan/template

7/3/18

variable ka jo scope hai use limit karo hua hai ek scope pe

user will never supply any input

This will point at current instance

method is nothing but a function created inside the class.

```
# class Class Computer:
# functions (instance method)  def writecode(self,text):
                                   print(text," written")
                               def execute(self):
                                   print(text,"execute")
```

creating object

```
com = Computer()
com1 = Computer()
```

Self is not a keyword we can replace it with any thing like def

two instances should be provided but we provide only 1).

```
com.writecode("code")
com.execute("code")
```

output: code written

1 -> It refers current instance

Self is seperating two objects com & com1 by limiting the scope for each method

```
Class C:
    def f(self):
        print(self)
o=C()
o.f()   <- main -- C object at address 1 >
o1=C()
o1.f()  <- main -- C object at address 2 >
```

its just a convention

means zero argument

self is not a keyword we can replace it with a.

```
class emp:
    #classvariable=0
    def __init__(self):
        self.c=self.c+1
    def show(self):
        print(self.c)
o=emp()
o.show()  # 1
o1=emp()
o1.show()  # 1
```

fixed interlotion for constructor

Constructor

Class variable is to be shared with all the objects whereas instance variable is non shareable data

first argument of f method

self limits C and makes it as instance variable here

Email address

Page:
Date:

```
[r'^[a-z0-9-]+@[a-z].*\.(com)+$
```

. has its own meaning but when] followed by \ lost its means and becomes normal .

Object oriented programming:

Code reusability

** self it wi

1 -> It r

* self is limiting th

Class
def

Object oriented programming:

Code reusability

To describe any object we should know about its attributes.

S – State
B – behaviour
I – Identity

Class is a plan / template

27/3/18

variable ka jo scope hai use limit kare hua hai ek scope pe

User will never supply any input This will point at current instance

```
# class Class Computer:
# functions (instance method)   def writecode(self, text):
                                    print(text, " written")
                                def execute(self, text):
                                    print(text, "execute")
```

method is nothing but a function created inside the class.

creating object

```
com = computer()
com1 = computer()
```

★★ self is not a keyword we can replace it with any thing like def

com.writecode("code") com.execute("code") — two instances should be provided but we provide only 1).

output: code written

1 → It refers current instance

★ self is seperating two objects com & com1 by limiting the scope for each method.

its just a convention

```
Class C:                    → means zero argument.
    def f(self):            self is not a keyword we can replace it with a.
        print(self)
o=C()
o.f()  <__main__.C object at address 1>
o1=C()
o1.f()  <__main__.C object at address 2>
```

```
class emp:                  → fixed intentation for constructor
#classvariable=0
    def __init__(self):     # Constructor
        self.c=self.c+1
    def show(self):
        print(self.c)
o=emp()
o.show()  # 1
```

★ Class variable is to be shared with all the objects whereas instance variable is non sharable due to self

```
o1=emp()
o1.show()  # 1
```

first argument of ∫ method → self limits c and makes it as instance variable here.

here there is no self (emp.c is used) so here how. C is a class variable / static variable.

```
class emp:
    c=0
    def __init__(self):
        emp.c = emp.c+1
    def show(self):
        print(emp.c)
```

to count no. of objects created just by using constructor (and the anti self variable method i.e class variable / static variable)

```
o = emp()
o.show()      ||1
o1 = emp()
o1.show()     ||2
o2 = emp()
o.show()      ||3.
```

```
class calc:
    def __init__(self, a, b):  # constructor
        self.a = a             # assigning to particular
        self.b = b             # a (variable)
    def add1(self):
        print(self.a + self.b)
    def mul(self):
        print(self.a * self.b)

c1 = calc(10,5)
c1.add1()   ||15
```

Static method

★ argument is not necessary in static

```
class c1:
    @staticmethod  # must to be written
    def fun():
        print("How u doin?")

c1.fun()  || How u doin
```

without creating any object, this function could be called.

Class method

★ atleast 1 argument is necessary in class method

```
class c1:
    @classmethod  # must to be written
    def fun(c):
        print(cls)

c1.fun()  || <class '__main__.c1'>
```

{ points at the class

→ Unlike self which points at the object

Inheritance (Code reusability)

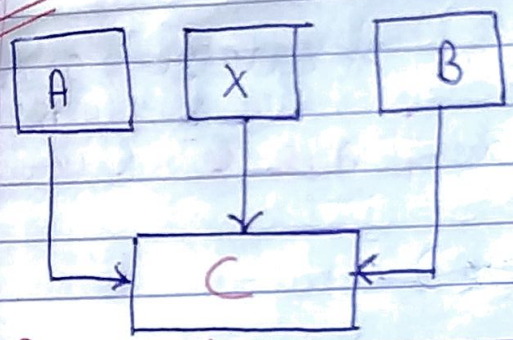
```
class c1:
    def add1(self, a, b):
        self.a = a
        self.b = b
        print(self.a + self.b)

class ch(c1):  # inheritance
    pass

och = ch()
och.add1(10,10)
```

```
class ch(ch1):
    def sub(self, x, y):
        self.x = x
        self.y = y
        print(self.x - self.y)

och = ch()
och.add1(10, 10)   # inherited from ch1
ach.sub(10, 5)     # got on its own
```



Multiple Inheritance

```
class A:
    pass
class B:
    pass
class C(A, B):
    pass

oc = C()   # object created ...
oc.fun()   # A as A is in preference higher order
```

```
def fun(self):
    print("A")

def fun(self):
    print("B")
```

→ sequence matters if there is a conflict (ambiguity) between any same named function defined in same parent classes.

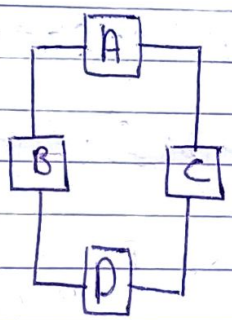
multiple inheritance

Page:
Date:

Hierarchical Inheritance

```
Class A:
    def fun(self):
        print("A")
Class B(A):
    def fun1(self):
        print("B")
Class C(A)
    pass
```

⇒ Hybrid:



Further Constructor phenomenon:

```
Class c1:
    def __init__(self):
        print("base init")
Class ch(c1):
    pass
```

but if child has its own constructor

```
    def __init__(self):
        print("Hi")
```

```
O = ch() // base init ⟶ then Hi
```

=> With the help of Super keyword we can call parent class

```
class c1:
    def __init__(self,a,b)
        self.a=a
        self.b=b
        print("Area is ", self.a * self.b)

class c2:
    def __init__(self, a,b,c):
        self.a=a
        self.b=b
        self.c=c
        print("vol is", self.a * self.b * self.c)
                                    self.a, self.b
        Static(c2,self).__init__()

com = c2(2,2,2)  # vol is 8
                 # area is 4.
```

=> Polymorphism:

no inheritance involved

=> Method overloading : Compile time polymorphism

```
class c:
    def fun(self, *var):
        if len(var)==0:
            print("zero")
        if len(var)==1
            print("one")
```

```
o=c()
o.fun()      # zero
o.fun(3)     # one
```

=> Method overrider: run time polymorphism

```
class c1:
    def fun(self):
        ----------①
class c2(c1):
    def fun(self):
        ----------②

co=c2()
co.fun() ||②
```

inheritance is involved

* In python only one constructor can be formed unlike java.

Overloading: (Constructor)

variable argument list

```
Class c:
    def __init__(self,*var):
        if len(var)==0:
            print("zero")
        if len(var)==1:
            print("one")
```

o=c() 2 2 constructors for same class
o1=c(1) s makes it overloaded.

Access modifiers:

public, protected, private } by convention

```
Class c1:
    def __init__(self,a,b,c):
        self.a=a    || public
        self._b=b   || protected
        self.__c=c  || private

o=c1(1,2,3)
print(o.a)      || public is accessible
print(o._b)     # in python private is also accessible
print(o.__c)    # error as it is private
print(o._c1__c) # now no error private is also accessible
```

// The same can be done with the methods

```
Class c1:                    but if we do, make it private
    def _fun(self):
        print("protected")

o=c1()
o._fun()  ||protected
o._c1__fun()
```

5/4/18

Date:

Abstraction : Focusing on what should be hidden

```
class veh:
    pass
class car(veh):
    pass

v = veh()
b = bus()
print (isinstance (b, veh))
```

generalization

specification

★ is-a relation is inheritance

Aggregation : (has-a relation).

eg. building & room interdepent

★ If there is strong dependency then composition

And the relationship which have no strong dependencies then its aggregation → eg. body and hand not such interdependency

```
class room:
    pass
class building:
    def __init__(self, nor):
        self.rooms = []
        for i in range(0, nor):
            r = room()
            self.rooms.append(r)
    def __del__(self):
```

Page:
Date:

del self.rooms

```
b=building(4)
print(b.rooms[0])
del b
print(b.rooms[0]) || Error means composition
```

---

aggregation

```
Class player:
    def __init__(self, name):
        self.name=name.
Class team:
    def __init__(self):
        self.players=[]
    def addplayer(self,p):
        self.players.append(p)

p1 = player("Sachin")
t=team()
t.addplayer(p1)
print(t.players[0].name)//Sachin
del t  //t destroyes but p1 still exist
```

Association (uses-a relationship) //temporary warning

```
Class computer:
    def writecode(self, text):
        print(text, "written in editor")
    def execute(self):
        print("code executed")
```

Page:
Date:

```
class stu:
    def dolabassignment (self, c, assignment)
        c.writecode ("python code")
        c.execute()

c1 = computer()
s1 = stu()
s1.dolabassignment (c1, "python").
```

Student uses computers behaviour

Database handling

Database is a collection of tables

Table

Eid	Ename	Design
fc(0)	amit	A.P
fc102	Sumit	HOD

Record

Table is a collection of records

Record is a collection of logically related schemas (Fields).

Page:
Date:

Two types of ends:

BE (Back End) Database // Oracle 11g
Cx_Oracle — Intermediate
F.E (Python)

```
import cx_Oracle

con = cx_Oracle.connect("user/pass")
print(con.version) // to check oracle is connected or not.

cur = con.cursor()
cur.execute("create table emp(eid number(3), ename varchar2(30), and so on)")
```

name of field
width
Reyword
name of table
data type
2nd field

now for eg 101.35
(we give (5,2))
decimal

```
con.commit() // after every write (operation) this is must.
```

→ it makes your changes permanent

```
con.close() // closes the connection
```

Page:
Date:

cur.execute("insert into emp values(101, 'amit')")

added to the database

Oracle

desc emp → table — show str. of table & their data type not record of table

Select * from emp → for Record

Write operation

// cur.execute("delete from emp where eid=101")

uniquely identifiable

delete from emp // all the records from the table will be deleted

cur.execute("update emp set ename='prakhar' where eid=101")

Insertion in database manually

cur.execute("insert into empt values(:1, :2)", (i,n))

input data manually

"primary key" is keyword used to make that column accept unique values only.

("delete from empt where eid = :1", (i,))

how to deal with dictionaries:

d = {"eid": 101, "ename": "Shamit"}

con.~~cursor~~execute("insert into emp2 values (:eid, :ename)", d)

dictionary

print(cur.fetchall())

to fetch all the records

type → list of tuples

print(cur.fetchone())

to fetch only one

type → tuples

print (cur.fetchmany(1))
type : list
↳ no. of elements to be fetched

Arraysize

cur.arraysize = 2 // by default it is 50. → cursor is able to hold only 2 records

print (cur.fetchmany()) // 3 elements only
(cur.rowcount) // 3

To perform actions multiple time :

cur.executemany

list1 = [{"eid": 500, "ename": "suraj"}, {"eid": 600, ...
]

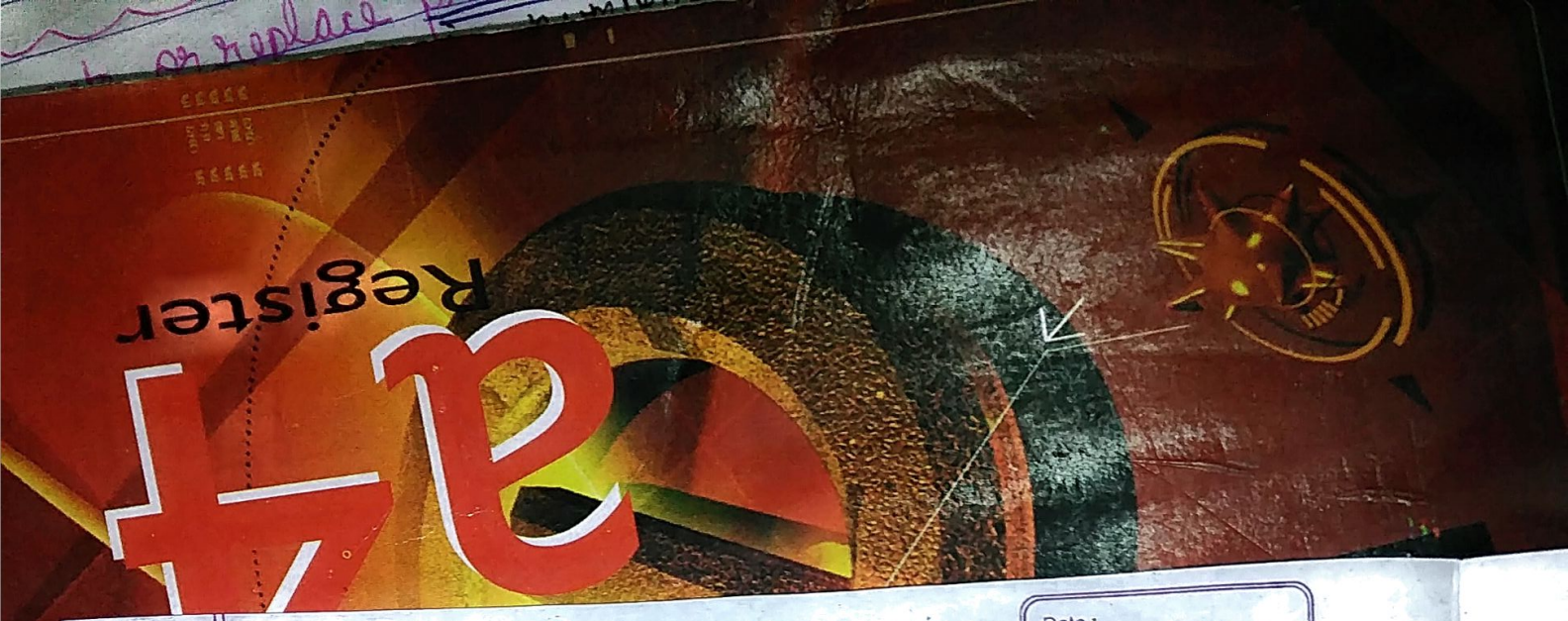
cur.executemany ("insert into emp2 values (:eid, :ename)", list1)

cur.execute ("select * from emp")
print (cur.description)
it will work with read operation

cur.prepare("select * from emp2 where id=:id")
not use this approach
because debugging becomes difficult.

GUI:

```
from tkinter import *  || contains all the methods used for GUI
master = Tk()  || form is keyword formed by this
master.title("My app")  ||setting the title
Label(master, text="Name").grid(row=0)   (inbuilt function)
Label(master, text="Id").grid(row=1)
e1 = Entry(master)   (inbuilt)
e2= Entry(master)
e1.grid(row=0, column=1)   (placements (Position))
e1.grid(row=1, column=1)
Button(master, text="Add", command=ISR).grid(row=4, column=4, sticky=W)
   (owner (first command always))  (function database)
   (Keyword)
def ISR():  || database connection here
* con.execute("insert into emp2(name,id) values {('''+name1+''','''+id1''')")
   new format for entering values
   East west north south position
```



Date.:

Page No.

mainloop() // must otherwise the form will

Procedure in Oracle

Page:
Date:

Searching name

procedure

Oracle

keyword

name of procedure

write only

```
Create or replace procedure InsertR1 (name1 out
varchar2, id1 in number)
```

variable that is asked from python

read only (cannot be modified)

(we will pass this while executing)

supplying something to procedure

```
is
begin
Select name into name1 from emp3
where id=id1;
end;
```

This table has to be created earlier

We have to write logic

```
create table emp3(name varchar2(30, id number(4))
```

now invoking some procedure in python

we can also create whole procedure directly in python by cur.execute("""

triple quotes because of multiple line

```
import cx_Oracle
con =
cur =
id= input("Enter id")
```

This variable is must

variable

varchar2 in python is called string

```
py_out = cur.var(cx_Oracle.STRING)
cur.callproc('InsertR1',[py_out, id])
print(py_out.getvalue())
con.commit()
con.close()
```

Page:
Date:

Another procedure:

both in an out

```
create or replace procedure sqr1(x in out number)
is
begin
x:=x*x;    // finding square
end;
```

Python:

```
py_inout = cur.var(cur_Oracle.NUMBER)  // for x as out
py_inout.setvalues(0, int(id))

cur.callproc('sqr1',[py_inout])

print(py_out.getvalue())
```

(4) name is not change as there is 1 variable only x.

Function (in):

At Oracle

```
create or replace function totalemp(sal1 in number) return
number is
total number(5) :=0;
begin
select count(*) into total from emp2 where sal=sal1;
return total ;
end;
```

Notes: read only (sal1 in); must return; return data type (number); width; initial value; data type; assignment syntax (:=); simple variable (total).

Page:
Date:

At python

```
import cx_Oracle
con =
cur =
sal1 = input ("Enter sal ")

list1 = [sal1]
py_returnvalue = cur.callfunc('totallemp',
                              cx_Oracle.NUMBER,
                              list1)

print (py_returnvalue)  // ★ get is not needed here
```

Function (in out)

At Oracle

```
Create or replace function inout_fn (outparm in
                                     out number)
return number is
begin
out_parm := outparm * outparm ;
return outparm;
end inout_fn ;
```

At python

```
import
con
cur
num = input ("Enter number")
py_out = cur.var(cx_Oracle.NUMBER)
(py_out = cur.callfunc ('inout_fn', cx_Oracle.NUMBER, [num])
print (py_out)
con.commit()
con.close().
```

Custon Exception

```
class Smallval (Exception):          → parentclass is Exception fixed.
    def __init__(self, arg):
        self.args = arg
                    ↳ attribute of Exception parent class.
class largeval (Exception):
    def __init__(self, arg):
        self.args = arg

try:
    a = int(input ("Enter a number"))
    if a<10:
        raise Smallval ("value is small")   ← error message
    if a>10:
        raise largeval ("value is large")
    else:
        print ("Yes the corrct value is10")
```

Page:
Date:

```
except smallval as e:
    print(type(e.args)) #tuple
    for i in e.args:
        print(i, end=" ")
    print("\n")
except largeval as e:
    print(type(e.args))
    for in e.args:
        print(i, end=" ")
    print("\n")
```

✓ "Ord" function is used to print ascii value of the word/letter/number.

```
print(ord("A"))
```