1/July/2020

Python (27 Feb 1991) Version1 (1994)

- Python Basics
- GUI app with Python
- Web app with Python
- Python for data Science
- Python for AI & Machine Learning
- Bigdata Analytics with Python
- Python for IOT

- Devloper of Python
  Guido Van Rossum

```
Print ("Hello")
Print ("Hello\n Students")
Hellow
Students
```

Comments in Python

```
#   Single line comment
'''  '''''  Multiline
"""  """
```

Variables: reserved memory locations to store values
- Variables are dynamically typed.

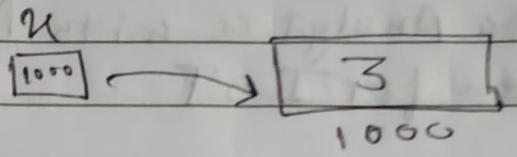
Program? → Data Processing.

identifier: The smallest identifying unit in the Program called identifier

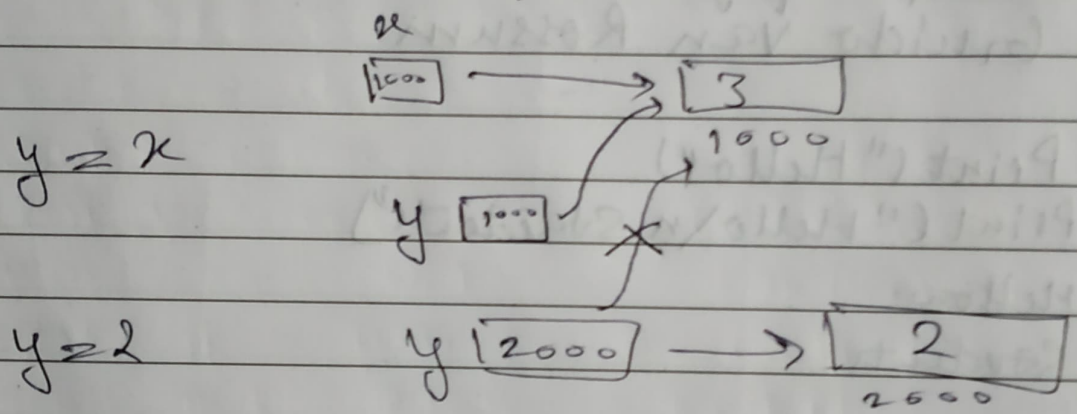
- An identifier can denote various entities like variable, type labels, ~~subordinate~~ function and so on.

Memory concept

$x = 3$



Block x: refers the actual block



$x = 3$ Print (id(x))	$x = 3.7$ Print (type(x))

- Every variable in Python is an object

Keywords in Python

- Predefined words or reserved words
- They cannot be used as variable names as their meaning is already reserved.
- Their are 33 Keywords in Python 3
- import Keyword
  Print (Keyword. kwlist)

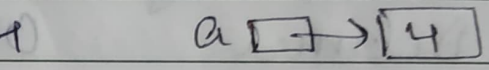
Data types in Python

- Data type is catagory of data.

Program? → ~~Data~~ to Process data

① memory
② operation
③ Representation

a = 4



Dynamicaly typed.

In Python their are 14 builtin, data type

- int a = 4
- float a = 3.7
- complex 9 + 4j
- bool x = True, x = False
- Str x = "Amit's", '''
- bytes
- bytearray
- range
- list
- tuple

Sequence types

- set
- frozenset
- dict
- None type

type Conversion in Python

Conversion function	
• int() # returns int	• dict()
• float() # returns float	• tuple()
• ord() # returns unicode	• list()
• str()	• set()
• complex()	

x = "123" > y = 5 >, x+y X > int(x) + y ↵

Handling Number System in Python

§

Number System

Decimal	0 to 9	$(572)_{10}$
Binary	0, 1	$(10100)_2$
Octal	0 to 7	$(432)_8$
Hexadecimal	0 to 9	$(3f)_{16}$
	A B C D E F	

x = 45

x = 0b45 , x = 0B45 } Binary

x = 0o45 , x = 00453 Octal

x = 0x3f , x = 0X3f3 Hex9

Print (hex(x) , (Octal(x)), (bin(x))

case sensitive

Slicing 0b, x = 0b101

y = bin(x)

Print (y[2:])

Operators in Python

- Airthemetic operator

/ → Always float
// → floor value.

-4 % 3 ⇒ 2 (Jo denomenator ka sign wahi aayega)

Order of Precedence
** > /, //, *, % > +-

Divde ki value always float

+ , - → works on both (no. or String)
* → when use with string, repetition
"ab" *3 = 'ab ab ab'

- Relational operators

True ⇒ 1 False 0
Bool(1) Bool(0)

x = True
y = int(x)

>	greater than
<	less than
>=	greater equal
<=	less tha equal to
==	equal to
!=	Not equal to

→ equality operator work with complex no. Not give errors

Boolean fun works
True + 5 = 6
False + 8 = 8

'a' == 97 False.
ord(a) == 97 // convert in unicode.
True.

3 == "three" → False , 3 == "3" → False
== checks content and type
~~== operator~~
int & float can be compared
Bool & int can be compared

True == 1 ✓ True = 1
True == 5 X False = 6
True > 5
→ False
True != False ✓

- Logical operators
C/C++/Java
&& || !
and, or → Binary operator
not → unary operator
Python
and or not

x and y
T and T → True
T and f → false
f and T → False
f and f → False

x or y
T or T → True
T or f → True
f or T → True
f or f → false

① If x and y are non boolean then result is also non boolean

(and)

Imp // if x is false then result is x otherwise y

5 and 4 → 4

T, then y 4

② (OR)

if x is false then result is y otherwise x

~~3~~ or 4 → 3

0 or 5 → 5

(Not) x>y Not T → false

not f → True

	Not
f	T
T	f

~~#~~ ~~*~~ Str

Empty "" → false

"Praveen" → True

Error

3 and x=4

No Error

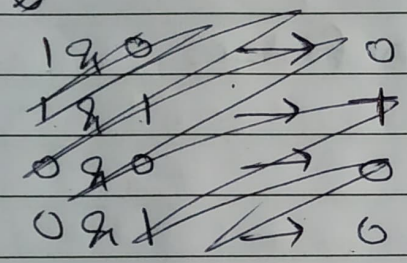
3 and 4>2

3 and 3+4

Bitwise operators : works on Bits (0, 1)

&	and
|	or
~	not
^	xor
>>	Right shift
<<	left shift

& (and)



+ tOR

- (and)

1 & 1 → 1
1 & 0 → 0
0 & 1 → 0
0 & 0 → 0

5 & 6 = 4

101 110

101
110
100 → 4

- (OR)

1 | 0 → 1
1 | 1 → 1
0 | 1 → 1
0 | 0 → 0

5 | 6 = 7

101 110

101
110
111 → 7

64 32 16 8 6 4 2 1
1 1 0 0 0 1

- (XOR) ^

5 ^ 6 = 3

1 ^ 1 → 0
1 ^ 0 → 1
0 ^ 1 → 1
0 ^ 0 → 0

101
110
011 → 3

- ~ NOT

~5 ⇒ ~~10~~1 ⇒ -6

101
0101
1010 ⇒ -K ⇒ K?

0 +ve
-1 -ve

1 0 1 0
0 1 0 1
+ 1
0 1 1 0 ⇒ 6 = K

- Right Shift (>>)

25>>2

25 ⇒ 11001

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
64 32 16 8 4 2 1

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
128 64 32 16 8 4 2 1
1 1 0 0 1

25>>2

11001
01000
⇒ 00110 → 6

11001
01100
00110 = 6

Left Shift (<<)

12 << 3 → 96

12 → 1100

Left me jagha nahi hai

```
    1100
   11000
  110000
 1100000  → 96
```

64 32

Assignment operators

=

+=, -=, *=, /=, %= } compound assignment operator

Identity operators

- is
- is not

;; Every variable in Python is an objects

;; objects are dynamically created

;; object do not have names but refrence.

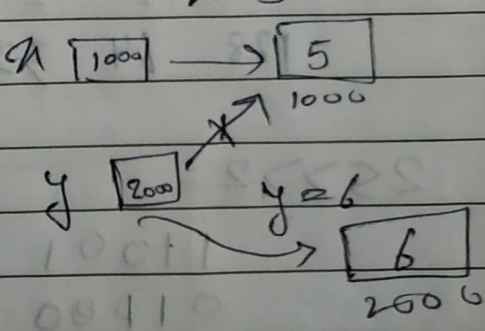
x = 5

y = 5

x == y → T

x is y → T

x is not y → F



membership operators

- in
- not in

in: To check find search data

data in x
↑ collection / iterable.

error case.
x = 256
5 in x → error

x = "abc"
"a" in ~~abc~~ x → True
x = "256"
"5" in x → T
works for string

9/8/20

Python

Taking input from user

x = input("Enter your name")
input function return string.

input function/method return value as a string
So we have to convert it in * int, float, complex
to perform mathematical operations.

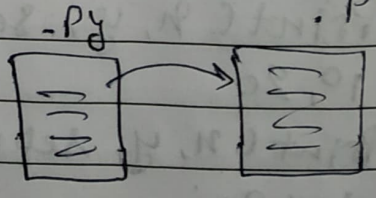
y = int(x)
Print(y)

Enter x = input("Enter no")
~~y~~ = Print(x)
y = int(x)
Print(" type(x), type(y))

Use of import, as and from

Module: Module is a Python file consisting of Python code/script.
Module can define functions, classes and variable.

import Math
math.factorial(5)
math.sqrt(5).



import Sys } to check import dir. Path
Print(Sys.Path)
Sys.Path.append('/dirname/') // append dir Path
var fun

- Module alias,
  ↳ alternative name.

```
import Math as m
m.sqrt(f)
m.factorial().
```

- from math import factorial, sqrt.
  ↓
  To access factorial directly

```
from math import *
                 ↓
                 import all math functions
gcd(4,6) → 2.
```

More on output Statement

```
Sep=","   Seprator
x=10
y=20
Print(x,y)
10 20
Point(x,",",y)
10 , 20
Print(x,y, sep=",")
10,20
Print(x,y, sep=":")
10:20
Print(x,y, sep="\n")
10
20
```

abc + def
abcdef

By default space is a seprator

Print is similer as Println (By default change the line)

```
Print("Hello", end=" . ")
Print("Students")
Print("Learn Python")
```

end = " "
└ replace new line with space.

```
x = 10
y = 20
z = 30
Print(x, y, z, sep="!", end=".")
```

- Formatted String

```
x = 10
Print("Value of x is %d", %x)
```

Nahi lagna

```
a = 3
b = 4
c = a+b
Print("Sum of %d and %d is %d" %(a, b, c))

x = 10.
y = 3.5
Print("x = %d, y = %f" %(x, y))
```

x = 10.1

~~I~~ %f = 10.1

Replacement operator { }

- Pair of curly braces is known as replacement operator.

```
x = 10
y = 3.5
z = "Amit"
Print(" Hellow {2} , x = {0}, y = {1}".format(x,y,z))

Print("Hellow {}, x = {}, y = {}".format(x, y, z))

Print("Hellow {2}, x = {x}, y = {y}".format(
                x=10, y = 3.5, z = "Amit"))
```

Decision control Statement

- Flow control Statements
- Decision control Statements

Flow control Statements

- Desicion / Selection Control Statement
  - if
  - if else
  - if elif else
- Iterating / Loop Statement
  - while
  - for

if condition :

    ____
    ____ } indentation
    ____

else :

    ____
    ____
    ____

- if elif else

if condition :

    ____
    ____
    ____

elif condition :

    ____
    ____
    ____

elif condition :

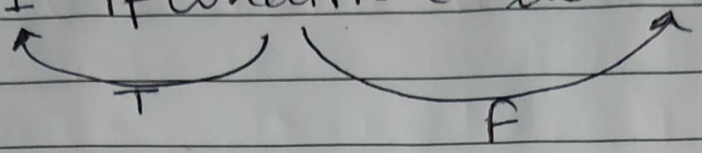
    ____
    ____
    ____
    ____

else : } optional

    ____
    ____

Ternary Operator / Single line if expression / Conditional operator

Value 1 if condition else Value 2



value of
"Conditional operators can be in a variable.

~~z = "Hellooo"~~

z = "leap year" if x%.400==0 else "leap year" if x%.4==0 ~~and~~ ~~x%.100!=0~~ else "Non leap year"

x%.100!=0

Iterative Statement - while loop

While loop is used when there is a condition

for loop is used when there is a sequence.

While condition :

_____

_____

_____

_____

14/8/20

Iterative Statement – for loop

- While
  - Iterate statements till some condition is true

- for
  - Iterate statements till for number of element in a sequence
  - Number of iteration is known in advance

C/C++/Java

```
for(i=1; i<=10; i++)
{
    ____
    ____
    ____
}
```

Python

```
for element/variable in sequence:
    ____
    ____
    ____
    ____
```

Sequence of elements : eg- Str, list, tuple, range, dict, set.

```
x = "My Str"
for e in x:
    ____
    ____
    ____
    ____
```

e = M , x = M,1
e = y , x = y,2
e = s , x = s
e = i , x = 4

iterative control - range() function

range function generates sequence of int values and only accepts int value.

range(5) → 0, 1, 2, 3, 4

① Print (range(5))
→ range(0, 5)

② for x in range(0, 5)
Print(x)
→ 0
1
2
3
4

③ range(n) 0 to n-1

④ range(start, stop, step)
↓ By default 0
↓ By default 1
Increment by 1

eg: range(0, 5, 2)
→ 0
→ 2
→ 4

⑤ To Print the result of range function for loop we use for loop.

Transfer statement in Python

break, continue and Pass → transfer statement

- Ctrl+C to stop infinite loop
- Break: The loop is immediately terminated and the program control resumes at the next statement
- Continue: we can stop the current iteration and continue with the next. skips the iteration.
- Pass: Pass keyword is used to create empty block

Use of else with loop

1) if else
2) if elif else
3) Single line if else (Ternary operator)
4) while else
5) for else

While loop terminate in two ways.

1) Break — else Block can not complied.
2) condition false. — else Block compiled

- When break is executed else will not work
- else work when condition of while is false
- In case of for loop else will execute when normal life of loop is complete.

16 | Aug | 20

Sequences

List in Python (Part - 1)

List

- Lists are very similar to arrays
- They can contain any type of variable, they can contain as many variables as you wish
- List can store heterogenous type of data
- Lists are iterable (for loop)

```
L = [ ]      // empty list
l.append(10)
Print(L) Print(L[0])
[10]
```

note - Range ~~fun~~ is used to generate sequence.

```
for i in l:
    Print (i)
```

List in Python (Part 2)

Create List

1) l = [ ]
2) l1 = [10, 20, 30]
3) l2 ⇒ ["Bhopal", "Delhi", "Jaipur"]
4) l3 ⇒ [10, 3.5, 30, "Jaipur"]
5) exception

```
l = [10, 20, 30]
    [0]  [1]  [2]
```

Print l[3] → error, index out of range

6) l2 = [ ] [0] index does not exist, value can't be inserted
l2[0] = 10
error, index out of range

- l2 = [10, 20]
→ [10, 20]
l2 = [1, 2, 3] // new values of list replaces the Previous values.
→ [1, 2, 3]

- append()
add value at the end of the list
a = []
a. append(10)
a. append(20)
→ [10, 20]

- insert()
In insert method we can specify the index and object
L. insert(index, object)

a = [10, 20]
a. insert[2, 30]
→ [10, 20, 30]
0 1 2

a. insert(4, 40)
→ [10, 20, 30, 40] // insert value at last Possible index
0 1 2 3

a. insert(1, 50) // to insert at index [1] and shifts other values by 1 index
→ [10, 50, 20, 30, 40]
0 1 2 3 4

a. insert(-1, 60) // for -ve indexing count from right to left
→ [10, 50, 20, 30, 60, 40]
-5 -4 -3 -2 -1 0
a. insert(-3, 100) // 20, left shift
→ [10, 50, 20, 100, 30, 60, 40]
-6 -5 -4 -3 -2 -1 0

- A = [10, 20, 30]
  B = [40, 50, 60]
  C = A + B
  → [10, 20, 30, 40, 50, 60]

- A + [10]
  → [10, 20, 30, 10] // Here 10 is not store in list only Prints

- A += [10] // to insert /add value
  A = A + [10] Append alternate meth.
  → [10, 20, 30, 10]
  • A += 10 // Error, int type can't be append
  • only iterable seq. can append

- D = [1, 2, 3]
  D * 3
  → [1, 2, 3, 1, 2, 3, 1, 2, 3]

- D *= 3 // to chang in D use compound assignment operator
  ~~D*~~
  D = D * 3

List in Python (Part 3)

- To modify element in List
  a = [10, 20, 30]
  a[0] = 100
  → [100, 20, 30]

- To remove element from list
  a = [10, 20, 30]
  a.remove(20) // Pass value (20)
  → [10, 30]

- D = [10, 20, 10, 20, 10, 20]
  D.remove(20) // first occurrance)
  → [10, 10, 20, 10, 20]

- List also is an object
  Sorting → Sort()
  D.Sort()
  → [10, 10, 10, 20, 20]

  t = Sorted(t)
  retures the list of sorted element.

- To remove all values from list
  D. Clear()
  → []

  Sorted(l, Key, reverse=False)
  ↑ None By Default
  optional

  D = [1, 2, 3, 4]
- To reverse values of list.
  D. reverse()
  → [4, 3, 2, 1]

  l.Sort(Key, reverse=False)

- Pop()
  extracts the value from list. By default extracts last element.
  and return the value ~~Pop~~ extracted/Poped value.
  it. D.Pop()
  → 1

  D = [4, 3, 2, 1]
  D = [4, 3, 2]

  D.Pop(0)
  → 4

  D = [3, 2]

  x = D.Pop(1)
  x = 2

  D = [3]

L.sort(key=lambda x : x.name , reverse = True)

A=[10, 20, 30, 40]
A.remove(20)
A
→ [10, 30, 40]
A.pop(2)
→ 30
A
→ [10, 30]

- index() // To determine the index of the particular value

A=[10, 20, 30, 40]
A.index(30)
→ 2
A=[10, 20, 10, 30, 40, 20, 30, 10]
A.index(10, 1, 6)

element — start index — end index

A.index(10, 1) // we can pass upto ~~3 or~~ three arguements.
→ 2

- count() // To count the occurance of a particular element.

A.count(10)
→ 3

- help(A) // A= list variable
~~help(str)~~

Command line Arguments

run Python script through command Prompt.
we can pass argument through command line.

Python Hello.py Bhopal indore Delhi
→ 'Hello.py', 'Bhopal', 'indore', 'Delhi'
accept data as string.

- From Sys import argv

Python Hello.py A B C

```
From sys import argv
For x in argv:
Print (argv, type(x))
```

```
From sys import argv
y=0
S=0
For x in argv:
    if y==0:
        y=1
    else:
        S=S+int(x)
Print("Sum is", S)
```

- Array input in Python.

```
arr= map( int, input().split())
```

Strings in Python Part-1

Strings can be written in 3 ways

- Single quote 'praveen'
- double quote "praveen"
- triple quote """praveen"""
  '''''praveen'''''

* In triple quote we can pass multiline strings.

len(s1) → to calc. length of string

To find index.
s1.index('s')

To count occurance
s1.count('e')

Strings in Python Part-2

s1 ⇒ mysirg education services.

Negative indexing.
→ Rigth to left.

mysirg education services
-8 -7 -6 -5 -4 -3 -2 -1

Slicing Operator

[beg : end : step]

include — exclude — +1
optional

```
S1[7:16]
→ 'education'
S1[-5:8:-1]
' vres noitacu'
```

* To Reverse a string. By using slicing
  S1[24: :-1] at zeroth index, it ~~count~~ not prints.

```
S1[len(S1): :-1]
```

* S1.upper() → Converts String in uppercase
* S1.lower() " " " lowercase

* S1.startswith("my")
  True.
* S1.endwith("ces")
  True

Split() → Splits the string into list of Strings.

```
S1.split(' ')
→ ["mysirg", 'education', 'Services']
```

* help(str)

Tuple Part-1

Tuple → Sequence of elements

Tuple ( ) (1, 2, 3)
0 1 2

List is mutable. (change is allowed)
Tuple is ~~non~~ immutable (no change)

t = ( ) → tuple
t = (10) → 'int'
t = (10,) → tuple
t = 10, 20, 30 → tuple
l = [11, 22, 33, 44]
t = tuple(l) l is of string type
t = tuple(10, 20, 30) X
t = tuple((10, 20, 30))

t = (10, 4.5, "xyz")
Print through idexing.
t[0] → 10
t[1] → 4.5

Printing tuple via
1) loop 2) Slicing operator
↳ while
↳ for

```
t = (10, 20, 30, 40)
i = 0
while i < len(t):
    Print( Print (t[i])
    i += 1
```

} while

```
for x in t
    Print (x)
```

} for

```
t[0 : : ]
```

} Slicing.

```
t = t + (1, 2, 3, 4)
t
  0  1  2  3 4 5
→ (10, 20, 30, 40, 1, 2, 3, 4)   // New tuple
t[ 2 : 5 : ]  → 30, 40, 1
t [ 2 : 5 ]   → 30, 40, 1
```

tuple can only concatenate tuple not 'int' to tuple

Tuple (Part 2)

Packing.

```
a = 1
b = 2
c = 3
t = (a, b, c)
```

Unpacking.

```
t = (10, 20, 30)
a, b, c = t
```

★ Concatenation

```
a = 1, 2, 3      b = 10, 20
a + b  ⇒ (1, 2, 3, 10, 20)
```

★ Repetition operator

```
t = 1, 2, 3
t * 2
  (1, 2, 3, 1, 2, 3)
```

Comparision operator

a = 1, 2, 3

b = 1, 2, 3

c = 1, 2, 4

a == b	a == c	a < c
True	False	True

Tuple Part 3

1) x = tuple(input())
2) y = eval(input())

eval means evaluate expression

x = tuple(input()) → 1,2,3

('1', ',', '2', ',', '3')

tuple function treat, its data and of string type so we use eval function to input data in tuple

Important functions in tuple

1) len(x) x = 10, 20, 10, 20, 10
2) x.count(10)
3) x.index(10)
4) max(x)
5) min(x)

	List	Tuple
1)	mutable	1) immutable
2)	[ ]	2) ( ) optional
3)	Not used as KEY values	3) used as Key values

Set in Python Part-1

1) No duplicate values
2) S={10,20,30}
3) No indexing
   S[0] ✗
4) S[1:5:2] ✗
   No. Slicing operator
5) mutable (changes are allowed)
6) Creat Set
   S={10,20,30}
   S={} → dict type not a set
   S=set() // to create empty set
7) Value in set store in random order.
   S=set({10,20,30,}) S=set((10,20))
   S → {10,20,30}
   S=set("mysirg.com")
   {'m','i','s','.','r','y','o','c','g'}

S=set()
↑ sequence enter karna hai. (list, tuple)
(int, bloat) ✗

l=[10,20,10,30,10,20]
S=set(l)
S
→ {10,20,30}

Set in Python Part-2

- S = {1, 2, 3}
  S.add(4)
  S
  {1, 2, 3, 4}

- S = Set(10)
  S = Set((10, 20))
  S
  → {10, 20}

- S = Set(3.5)
  S
  {1, 2, 3.5, 3, 4} Sequence not determind.

- S = {1, 2, 3}
  l = [1, 3, 5, 7]
  S.update(l) // update fun. Performs union.
  S = {1, 2, 3, 5, 7} (S U l)

- S.update([3, 5, 9], (5, 6, 7, 8))
  list tuple
  S = {1, 2, 3, 5, 6, 7, 8, 9}

- S.discard() // no error
  S.remove() // error when element not found.

- S={1,2,[3,4],5,6)}
  error : unhashable type: List
  // we can't use List as an element in set

- S={1,2,(3,4),(5,6)}
  S
  {1,2,(3,4),(5,6)}
  // tuple is used as element in set
  S.discard((3,4))
  S
  {1,2,(5,6)}

- S1 ⇒ {1,2,3,5,8}
  S2 ⇒ {2,3,4,6}
  S1.intersection(S2)
  {2,3}
  - ~~S.sorted()~~
  - sorted(S)

- S1.union(S2) // {2,3,4,6}
- S1.clear()

- issubset, issuperset
  S1 ⇒ {1,2,3,4}
  S2 ⇒ {2,3}
  S2.issubset(S1)
  True

- S1.pop() // in sets pop() fun. removes any1 element randomly
- S.copy() // makes copy of the set
  S1=S.copy() // element of S copies in set S1

- id(S1) // 47865504 , id(S) // 47866864

Dictionary Part-1

List	[10, 20, 30]	Object / value
Tuple	(10, 20, 30)	individual values
Set	{10, 20, 30}	

Imp.
Each element of dictionary is a pair of Key-Value

Key	Value
1) Key must be unique	1) Repetition Possible
2) ~~Any~~ Any type	2) Any type
3) Heterogenous	3) Heterogenous.

- indexing not Possible
- Slicing operator not work
- Cannot guarantee of order of storage of Key-value in the dictonary
- mutable : change is allowed. (add, delet, edit)
- grow / Shrink.

```
d={}    // To create dictionary
d=dict()  // create dict by constructor
```

- d={100:'Rahul', 101:'Samar', 102:'Ajay'}
  By constructor dict
- d=dict(one='Bhopal', two='Indre')
  → {'one':'Bhopal', 'two':'Indore'}
  (Here in dict() we use = instead of :
  and Key is Byderfault treated as string.)

- To access value ~~by~~ in dict() method.
  d['one']
  → Bhopal

- d = {1:"A", 2:"B", 3:"C", 4:"D"}
  To ~~da~~ add element in d.
  d[5] = "E"
  d
  {1:"A", 2:"B", 3:"C", 4:"D", 5:"E"}

- To edit value in dictionary
  d[1] = "a"

- for K in d:
    Print("Key=", K, "value=", d[K])

Roll no		Name.
100	—	Rahul
101	—	Samar
102	—	Ajay
↓	—	↓
Key	-	Value

	.Sort()	Sorted.()
list	✓	
Tuple		
Set	X	✓
Dict	X	✓

dictionary Part-2

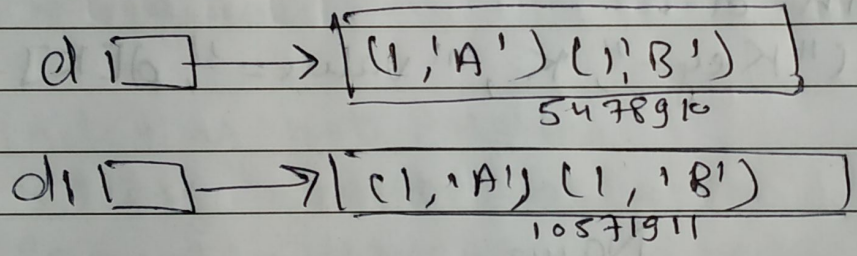
words = { }

- To delet any pair in dict
  del ['c'] X // error
  del [words ['c']]

- words.clear() // clears all pairs in dict.

- d = {1 : 'A', 2 : 'B'}
  d1 = d.copy() // Shallow copy

Here d1 & d contain same key value pair
But reffers to different key object



d is d1 // to check id's of d & d1 is same or not.
False

- d.Fromkeys() method.
  l = [1,2,3]
  d. Fromkeys (l, "ABC")
  d. ↳ optional, By deafult None
d2 = {1 : 'ABC', 2 : 'ABC', 3 : 'ABC'}.
this ~~does~~ doesn't change the value in d
we can use this to store this in another variable like (d2)

d = {1:'A', 2:'B', 3:'C', 4:'D'}

- d.get(2) // no error
'B'
d[2] // gives key error
'B'
x = d.get(3) // to store value returned by get(3)

- y=d2.pop(2) // Extract the value of key 2
y error when key is not valid
'ABC'

{2: "ABC"} dict_item
2 dicts_key
"AB" dict_value

- d.Popitem() No argument
Popitem() method returns ~~pop~~ tuple value (type
t = d.Popitem()
d
{1: 'A', 2:'B', 4:'D'}
type(t) d.items()
tuple d.key()
t d.values()
(4,'D')
t[0]
4
t[1]
'D'

- help(Dict)

7/9/2020

Organize Code

More on input Part-1

x = eval(input("Statement"))

More on input Part-2

- To take two inputs

1) x, y = input(), input()
2) x, y = input("Enter two numbers").split()

↓ By default space

input().split()

↓

returns the list split string

- d = input("Enter date").split('/')

d

['18', '7', '2020']

a, b, c = input("Enter date").split('/')

a = '18' b = '7' c = '2020' → String ~~type~~

Reading multiple values.

- a, b, c = [int(x) For x in input("Enter date").split('/')]

a, b, c = [1, 2, 3] // a = 1, b = 2, c = 3 // a, b, c = (1, 2, 3)

More on input Part-3

- Reading multiple value of diffrent type

```
a,b,c=[eval(x) for x in input().split(',')]
```
this statement accepts int, float, str.

- For List []

```
L=[eval(x) for x in input("Enter three values").
                                    split(',')]
```
→ [11,22,13]

- For tuple ()

```
t=tuple([eval(x) for x in input().split(',')])
```
→ (10,20,30)

- For ~~String~~ Set {}

```
S=Set([eval(x) for x in input().split(',')])
```
→ {33,11,12}

- for dictionary

method 1

```
d=dict(input().split('-') for _ in range(3))
```

1-'a'
2-'b'
3-'c'

→ {'1':"'a'", '2':"'b'", '3':"'c'"}

this treat 1,2,3 as string.

method 2 for dictionary input:

d = ~~dict~~ {x : input() for x in range(1,5)}

→ {1 : 'Ajay', 2 : 'Ravi', 3 : 'Rahul', 4 : 'Vinod'}

method 3

d = {input("Key") : input("Value") for x in range(3)}

→ Value Amit
Key 100
→ Value Rahul
Key 101
→ Value Ajay
Key 102
d
→ {'100' : 'Amit', '101' : 'Rahul', '102' : 'Ajay'}

* d = {int(input("Key")) : input("Value") for x in range(3)}

functions

(Part-1)

- Function is a block of code
- function has some name, but in Python we can create a function without name
- code reusebility

- Their two types of function
  - Predefined function
  - Userdefined function

- How to create a function

```
def function_name():
    statement 1
    statement 2
    ...
```

- How to call a function

```
function_name()
```

- four ways to define function

1) Take nothing, Return nothing (TNRN)
2) Take Something, Return nothing (TSRN)
3) Take Nothing, Return Something (TNRS)
4) Take Something, Return Something (TSRS)

Part-2

1) Takes Nothing, Returns Nothing

Empty

```
def add():
    Print("Enter two no.")
    a = int(input())
    b = int(input())
    S = a+b
    Print("Sum is", S)
```

local variable

function defination

```
add()
```

Empty ← function call

no. use of return

2) Takes Something, Returns Nothing

formal arguments

```
def add(a,b):
    S = a+b
    Print("Sum is", S)
add(10,20)
```

Actual arguments

3) Takes Nothing, Return Something

```
def add():
    Print("Enter two number")
    a = int(input())
    b = int(input())
    S = a+b
    return S
x = add()
Print("x=", x)
```

// we can also return multiple values

4) Takes Something, return Something

```
def add(a,b):
    s = a+b
    return s
x = add(10,20)
Print("Sum is=", x)
```

Part-3

- When function returns nothing it return None

```
def f1():
    Print("Hello")
x = f1()
Print(x, "-", type(x))
```

→ ~~Hello~~

→ none - <none type>

- Default Arguments

```
sum()          // def sum(a=0, b=0, c=0):
sum(10)        // def sum(a, b=0, c=0):
sum(10,20)     // def sum(a, b, c=0):
sum(10,20,30)  // def sum(a, b, c):
```

* after default Argument, Non default argument is not allowed

~~(a=0, b=)~~ (a=0, b, c=0) ✗

(a, b=0, c=0) ✓

Part-4

```
def f1(a,b):
    ____
    ____
    ____

f1(10, 20)
```

Positional argument

- Keyword argument.

```
def f1(a,b):
    Print("a=", a, "b=", b)
f1(10, 20)
f1(b=10, a=20)
f1(20, 10)      //Swaping
f1(10, b=20)    // mixture of Positional arg. & Keyword arg.
f1(a=10, 20)    // error : Positional arg. follows Keyword arg.
```

* if first arg is Keyword arg. then after that after that arg. every arg. must be Keyword arg.

Part -5

- Variable length arguments

```
def avg(*n):
    return(a+b)/2
x = avg(10, 20, 30)
Print("average is", x)
Print(n, type(n)) → (10,20,30) tuple.
```

*n → accepts variable lenght arguments
→ is of tuple type

```
def f1(*Points, Playername):
    Print(Playername, end=' ')
    S=0
    for x in Points:
        S = S+x
    Print("total Points =", S)

f1("Rahul", 10, 11, 11)   X
f1(10, 11, 12, Playername="Ajay")
f1(11, 12, 13, Playername = "Amit")
```

- Variable lenght Keyword Arguments

```
def f1(**K):    // **K → dict type
    Print("Person Information")
    for Key, values in K.items:
        Print(Key, "-", values)
```

f1(name="Sameer", age=22)
f1(name="Rahul", marks=87, age=23)
f1(name="Ajay", empid=125, Salary=350.00)

Recursion

Function calling itself is known as recursion
Solution of the problem is defined with the help of result of simpler version of the same problem then the solution is recursive.

```
def f1():
    ======
    f1()
    ======
```

for eg-

```
def Sum(n):
    ======
    return n + Sum(n-1)
```

n = 5

5 + Sum(4)

4 + Sum(3)

3 + Sum(2)

2 + Sum(1)

1

```
def Sum(n):
    if n==1:        # Base case
        return 1
    else:           # recursive case
        return n+Sum(n-1)
```

- Advantages
1) easy coding
2) easy to read

- Approach for recursion
1) Sum(n) // create fun.
2) n + Sum(n-1) // recursive case
3) n==1 // Base case.

* Every recursive problem can be solved with loops also

lambda expression

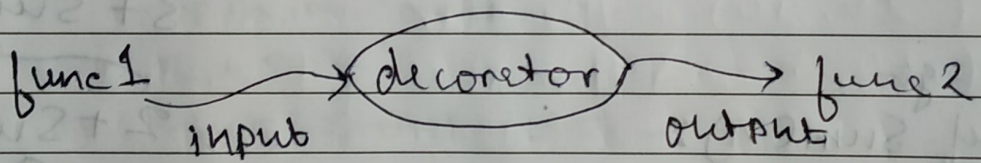
- Anonymous function, fun. without name.
- Single line function

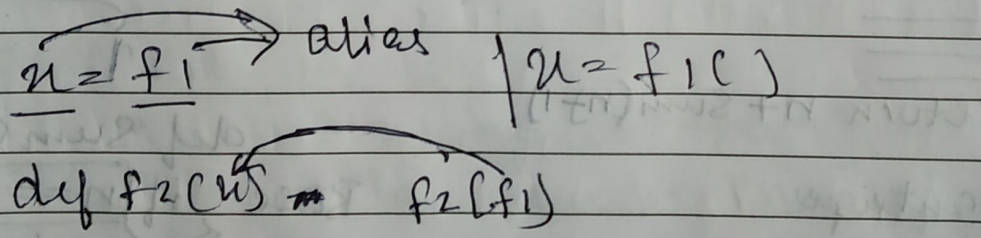
```
r=(lambda a,b : a+b) (5,3)
Print(r)
→ 8
```

- lambda fun. always return something after colon(:)

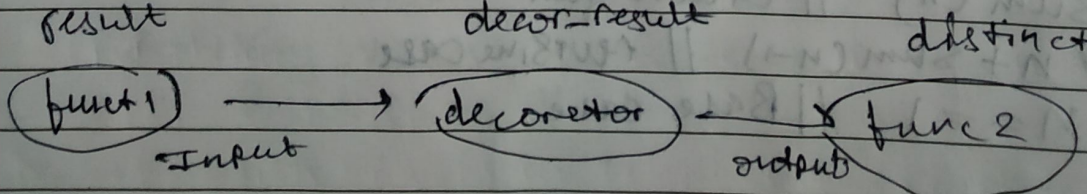
Decorators

1) function, to enhanced or to decorate another fun.





- @ decorator func name to call decorator.

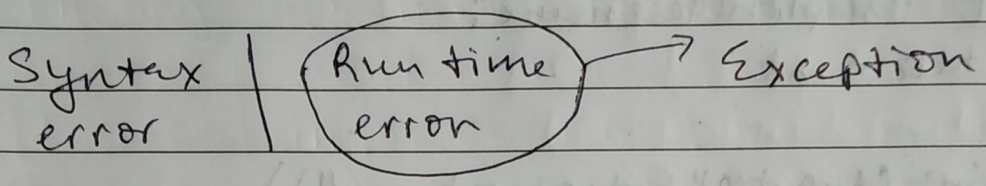


on calling result fun. distinction will execute

Exception Handling (Part-1)

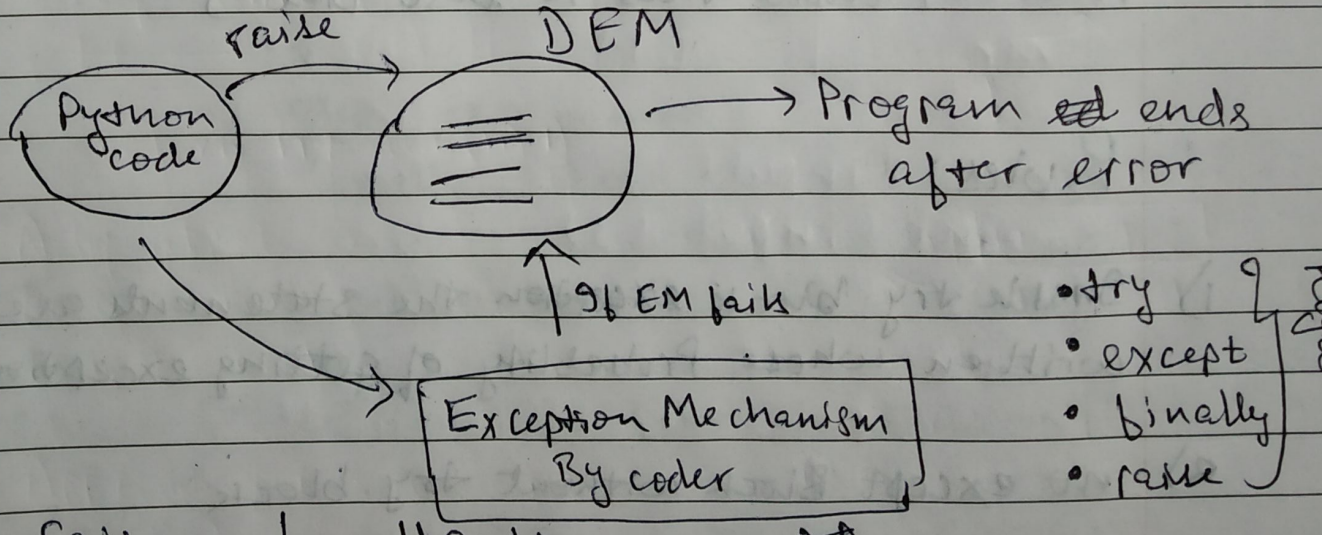
- Dealing with errors

- TypeError : unsupported operand type(s) for + : 'int' and 'str'
- ZeroDivisionError : division by zero



Typerror, zerodivision... etc → class/type in Python called exception classes

Exception classes
Deafault except mechanism (DEM)



	Raise	Handle
1)	Python	DEM
2)	Python	EM
3)	User	DEM
4)	User	EM

1) Predefined exception

2) Userdefined exception

Exception Handling Part-2

```
. . .
try:
    line1
    line2
    line3
except ExceptionClassname:
    Statement

x = int(input("Enter first no."))
y = int(input("Enter second no."))
try:
    z = x/y
    Print("Division result", z)
except ZeroDivisionError:
    Print("Invalid attempt")
Print("Hellow this is last line")
```

Rules

1) Inside try block ~~exception~~ the statements are written whose Probablity of getting exception is max
2) no except Block without try block
3) Error name and exception class name must be same.
4) No exception raised in try, then except block is skipped

5) Multiple except Block are allowed.

```
try:
    ____
    ____
    ____
except ____ :
    ____
except ____ :
    ____
except ____ :
    ____
```

a = 5+'5' → type error

num = int("String")
→ Value error

int("5") → 5

int("a") → value error: invalid literal for int()

6) If exception classname doesn't match with raised exception, the DEM (Default exception mechanism) is run.

7) Finally runs always

8) order, try
↓
except → 0 or more
↓
finally → 0 or 1

9) To handle multiple exception

except (TypeError, ValueError, ZerodivisionError):

10) Default exception Block (other than DEM)

```
except :
    Print("Default Exception")
```

11) Use of else with try, else will run when there is no exception
No exception raised then else execute

Exception Handling Part-3

- Case 3

raised	Handle
User	DEM

raise Keyword → raise Exceptionclassname ("Msg")

```
x = int(input( ))
y = int(input( ))
if y==0:
    raise ZeroDivisionerror("Denominator
                              cannot be zero")
z = x/y
Print("Divisionis", z)
```

- Case 4

raised	Handle
User	EM

```
x = int(input())
y = int(input())
try:
    if y==0:
        raise ZerodivisionError("          ")
    z = x/y
    Print("Division", z)

except ZeroDivisionError:
    Print("you can't divide by zero")
```

User defined Exception class

```
class InsufficientBalance(ZeroDivisionError):
    def __init__(self,arg):
        self.msg= arg
balance =5000
w=int(input("Enter amount to withdraw"))
try:
    if w>balance:
        raise InsufficientBalance("Insufficent
                Balance in the account")
    balance=balance-w

except InsufficentBalance as i:
    Print("Exception", i.msg)

else:
    Print("Withdraw amount", w," successfully")
finally:
    Print(" Current Balance is", Balance)
```

Exception Handling - Nesting of try block (Part-4)

```
try:
    Print("line 1")
    a)3/0  b) a+"5"
    Print("lin2")
    try:
        Print("line 3")
        c)3/0  d) 3+"5"
        Print("lin4")
```

```
        except ZeroDivisionError:
            Print("Except 1")
        finally:
            Print("finally")
        Print("line 5")
except TypeError:
        Print("Except 2")
finally:
        Print("finally 2")
```

a)
```
line 1
finally 2
```

b)
```
line 1
Except 2
finally 2
```

c)
```
line 1
line 2
line 3
Except 1
finally 1
line 5
finally 2
```

d)
```
line 1
line 2
line 3
finally 1
except 2
finally 2
```

Object Oriented Programming

1) Procedure oriented way of Programming
2) Object oriented way of Programming
   [data + function]
   → Object
3) Main aspects of OOP
   1) Classes and objects
   2) Inheritence
   3) Polymorphism
   4) Data hiding
   5) Data abstraction

Classes and Object Part-1

Class:

- class is a Keyword
- class has some name
- class encapsulates data and functions
- creating class is creating type
- class is an description of an object
- class is an blueprint of an object
- int, bloat, complex, str, tuple etc are built in class

Variable → fields } attributes
function → methods }

~~a.bit_length()~~
a.bit_length()

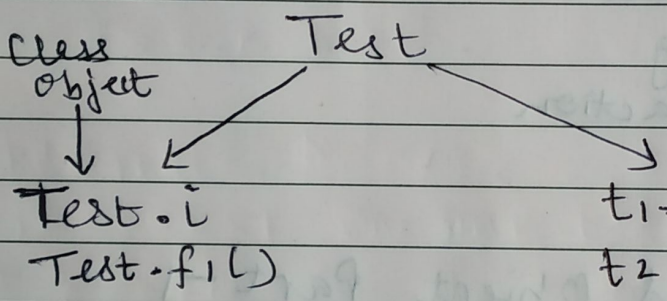
- we can create any number of ~~objects~~ objects of some class, but we have to define class first

Classes and Object Part-2

Syntax of class

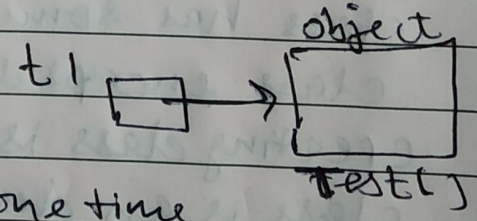
```
class classname:
    variable
    functions
```

```
Class Test :
    i = 10
    def f1():
        Print("Hello")
```



Test is used as class object

Test is used as a function



- class object can define one time but objects/instance can define more than one time

creating an object
t1 = Test()
t2 = Test()

} this process is known as instantiation

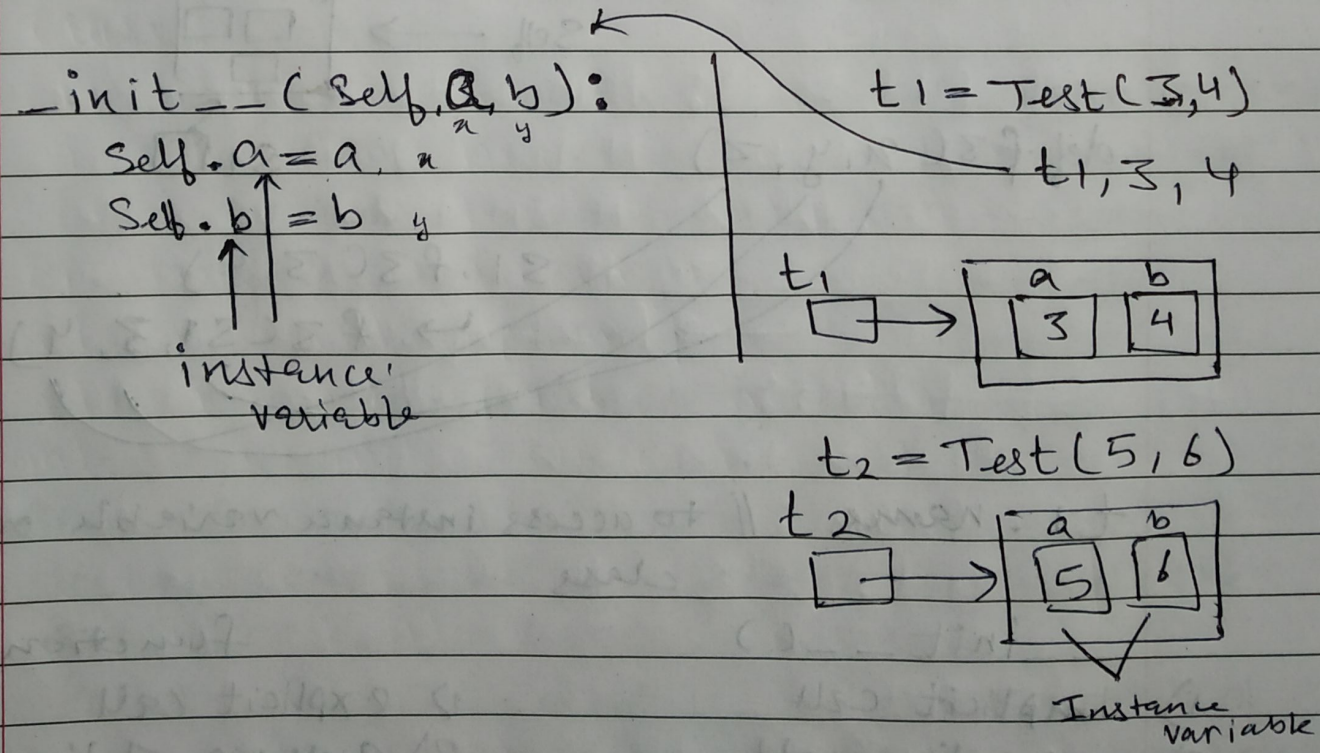
Classes and Object Part-3

Imp

__init__()

1) _init_() function automatically runs when we create object similar as constructor., and it will run on one time

2) ~~if we don't~~
__init__() function always take minimum one argument.
__init__(self)



Classes and object Part-4

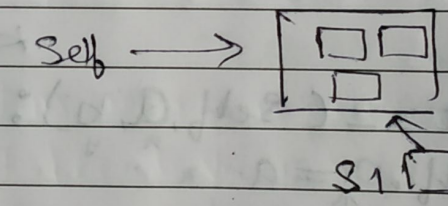
Functions in the class

```
Class Test:
    def f1(): // can't access instance variable
    ____
    ____
    ____
    def f2(self):
    def f2(Self): access instence verible
    Print(Self.name)
```

S1.f1() ← No arg.

S1.f2() ← One imlicit arg

↳ f2(S1)



def f3(x, y, z)

S1.f3(3, 4)

↳ f3(S1, 3, 4)

t1.name // to access instance variable outside class

__init__()	function
1) Implicit call	1) explicit call
2) one time call	2) any no. of time
3) To declare instence variable	3) for Problem solving logic
4) one arg is must	4) No such rule

Types of variable - instance variable

1) Instance variable
2) Static variable
3) local variable
4) Global variable

- Instance variable
  Object specific variable / Information.

- How to create instance variable
1) __init__(self)
2) methods
3) Outside the class → Objectrefrence.instance variable.

```
Class Account:
    def __init__(self,a,b):          } instance variable
        Self.accno=a                   through init()
        Self.balance=b                 func.
    def f1(self,a,b):                } through
        Self.accno=a                   method.func.
        self.balance=b

acc1=Account()
acc1.accno=102                       } through outside the
acc1.balance=6000                      class
Print(acc1.__dict__) // gives details of
                        object as dictionary
```

- When we call a ~~object~~ fun from obj outside class the one arg implicitly pass. which is current object

Types of variable static variable

- Static variable

1) ~~Ther~~ their is not static Keyword in Python
2) class specific (not object specific)
3) Shared variable among all the objects

- Creating Static variable

1) class Test :
   a = 10 # static variable

2) def __init__(self):
   self.x = 1 # instance variable
   Test.b = 20 # Static variable

3) def f1(self): # instance member function
   x = 100 # Local variable
   self.x = 2 # instance variable
   Test.c = 30 # Static variable
   → Access of S.V, this will also create S.V if not created

4) ~~def f2():~~
   @ static method
   def f2(m,n): # static method.
   Test.d = 40 # Static variable

t1 = Test()
t1.f2(3,4) # object specified calling.

---

obj = T1()
Print(obj.x) # accessing static variable
Instance

```
def f2(m,n):
    Print(m,n)
    Test.d=40

t1=Test()
Test.f2(3,4)  # calling through class object

def f2():  # static method as no args passed.
```

5)
```
def f3(
@classmethod
def f3(cls):
    cls.e=50      # static variable
    Test.f=60     # static variable

Test.f3()     # calling through class object
                implicitly pass class object
```

6) `Test.g=70` # static variable outside of class

	Class method	Static Method
1)	implicit arg. class object	No implicit arg.
2)	@classmethod	@staticmethod (optional)
	can't access instance memeber	

```
Print(Test.__dict__) // gives details of
                        Static variable as
                        dictionary.
```

Types of variable - Local and Global.

- Local variable - scope is limited to the func.

```
def f1(a):
    x=5  → local var.
```

- Global variable - declare outside of class and func.

```
y=10   # Global variable
```

we can directly use global variable in any function

```
def f1():
    global y   # using global keyword, this
    y=5          also creates global variable
    Print(y)
```

→ 5

```
y = 10  # Global variable
Print(" Outside function y=", y)
def f1():
    y=5 # local variable
    Print (" inside function local y=", y)
    Print (" inside function Global y =", globals()['y'])
f1()
Print (" Outside function y=", y)
```

→ 10
5 → inside function Preference of local variable is more than global var.
10
10

Inheritance Part-1

Encapsulation - Wrapping up of data member and member function into a single unit. eg. ~~classes and obj~~ class.

Inheritance - Defining a new class with the help of an old class.
Reusability of code.

Type → Person (name, age) → Super / old / Base / Parent

Sub Type → Student (rollno, name, age) → Sub / New / Derived / Child

Person (name, age)
↓ Inheritance
Student (rollno.)

- Stytax
  class Derived (Base class name):

eg:-
class Person :
....

class Student(Person):
....

- Inherit class from module
  first import module then

  class Derived(module.Base)
  . . .
  module.example()

- Parent class.
  init()

  Child class
  init()

  → overriding function with same name without arg. then or same arg. then the init() in child class will run.

Override → If two func. with same name and arg. defined in both Parent class and child class. then the child version of init() will execute. means child class init() overrides the Parent class init()

hiding → If two func. with same name but different arg. defined in both Parent class and child class. then the child version of init() will execute.
the child version hides the version of Parent.

- To call Parent class function inheritedly

Parentclassname.functionname(self)

→ Parent_classname.fun_neme(self)

Sl. Show Name() & Sl. Show Age() — when we call fun. of Parent class using child class object then the Rule is:-

- firstly called function ② is searched in child class and then in Parent class if not found in child class
- ~~if called~~

- if called function is ③ found in child class then it will execute without searching it in Parent class

- when the object is calling ① func. then the func. will searched in the class of object type

The child class can call Parent class function By using ~~Super.f() Supp~~ Super().

Name conflicts in Inheritance Part-1

1) Instance member variable name conflict

When their is name conflict between instance variable ~~be~~ of base class and derived class then the variable which ~~created first~~ is executed first will created first, and only one copy of variable is their in object. so when second time the ~~vari~~ instance variable encountered we were only accessed the ~~variable~~ or assigned ~~not or~~ created instance variable.

2) Static member variable name conflict

When we access static variable (x) ~~as~~ Using derived class (derived.x) then it will always access ~~child~~ static variable in child class. and when we access (x) using Base class (Base.x) then, always access ~~Static~~ Static variable of Base class. Hiding of variable.

Name conflict in Inheritance Part-2

3) Instance Member function name confict

Overriding. And hiding.

- Calling function in Parent ~~calls~~ class in overriding and hiding.
  super.f1() // Parent class function f1

Overriden : function with same name & arg.
Hiding : function with ~~diff no~~ same name diff arg.

The child version call Parent version By using super().

4) ~~Static member function Name conflict~~

4) Static member function Name Conflict

- Creating object is not necessary to call static method
- Static method can be called using class object.
- When child class object calls static method the child version of it will run.

Name Conflict between Static and instance member

Static variable and instance variable

- We can access static variable outside the class through object (obj.x)[] If their is no instance variable member]
- If we call through object obj then instance variable will access and to run static variable we have to call through class object (Derived.x)

Static function and instance function

- Their is no concept of fun. overloading in Python
- If the functions of same name is created then the last one will execute.

Accessing Base members in Derived class (Inheritance)

1) Instance member variables of Base class in Derived
2) Static member variable of Base class in Derived
3) Instance member functions of Base class in Derived
4) Static member functions of Base class in Derived

Multiple Inheritance

If a class is derived from more than one class, it is known as Multiple inheritance

Syntax:

```
Class Base1:
    ___
    ___

Class Base2:
    ___
    ___

Class Derived(Base1, Base2):
    ___
    ___
```

Data Hiding

C++/Java	Python
access specifier/modifier	class Test:
Private	x = 10 // accessible outside
Protected	__x = 200 // hidden var.
Public	

- Python changes the name of hidden var. by adding Prifix _Classname__x, (_Test__x)
  Name Mangling.

Polymorphism

Poly - many
morph - form
one words has diffrent meaning but when used in sentence it is cleared that which meaning is to be considered.

eg.

```
class A:
    def f1(self):
        Print("A - f1")
class B:
    def f2(self)
        Print("B - f2")

obj = A()
obj.f1()
obj = B()
obj.f2()
```

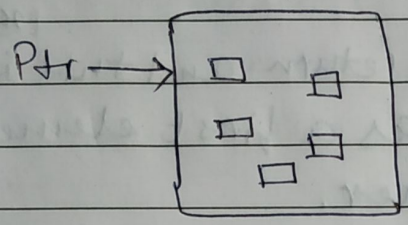
→ A - f1
→ B - f2

__name__ = __main__
Value of __name__ in current program is __main__ and in imported Program is ~~__file name__~~ imported filename

Iterators and Generators

Iterator

- similar like Pointer
- Iterator move from first element to last element and can't move backward
- it = iter (Sequence)



l = [1, 2, 3]

Ptr → (1 2 3)

for i in l:
    Print (i)

→ li = iter (l) // list_iterator

li = iter (l)
While True:
    next (li)

l = [1, 2, 3]
li = iter (l)
For i in li:
    Print (i)

1
2
3

li → (1 2 3)

next (li) → 1
next (li) → 2
next (li) → 3
next (li) → Exception

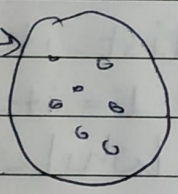
- through ~~next()~~ next() method we can access element of container one by one
- next() method return next value, after ~~of~~ value of Pointed iterator

Generators

- Special function
- it gives an iterator
- use of yield keyword it must atleast once

```
def f1():
    ==
    ==
    yield x   # Similar as return and this is treated as a first element of container.
    ==
    ==
```

```
it = f1()
```

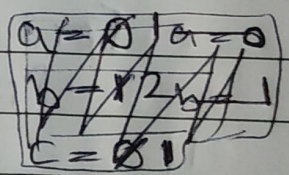
it → 

```
def fibonacci(n):
    a,b,c = 0,1,0
    while True:
        if c>n
            return
        yield a
        a,b = b, a+b
        c+=1
it = fibonacci(10)
for i in it:
    Print(i, end=' ')
it = fibonacci(10)
while True:
    Print(next(it))
```

0 → Pause

```
it = fibonacci(5)
next(it)
next(it)
next(it)
next(it)
next(it)
next(it)
```

Exception



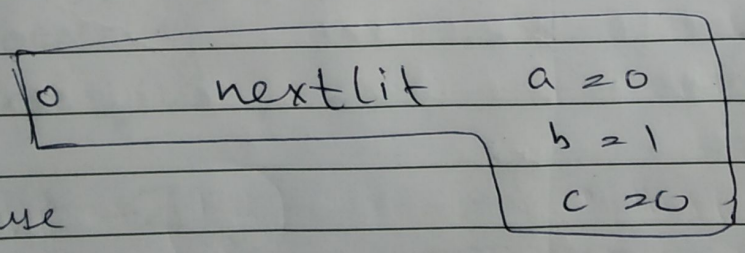
① next(it) → fun is will run from begining.

```
it = fibonalii(5)
```

①	②
0 next(it)	a = 0
1 next(it)	b = 1
1 next(it)	c = 0+1
2 next(it)	
3 next(it)	
5 next(it)	
Exception next(it)	

Note

- Value written after yeild returns next() method.
- the fun. will Pause and values of variable are preserved. and after every ~~next time~~ next() method it will resume (the fun.)
- the resumed fun. will ~~resumed~~ stop at yeild.
- when return run then the Exception is raised "StopIteration"



- The fun. will Pause after "yield a" until next(it)

File Handling in Python

File Handling Part-I

The data store in variable has lifetime less than lifetime of Program. or equal. but when Program ends the variable destroy. And the data is lost. so to Prevent data we have to create a file and save it in secondry storage.

Type of files
- 1) Text files
  - data → sequence of characters
    - .txt
    - .py
    - .dat
- 2) Binary files
  - data → seq. of bytes.
    - audio
    - video
    - image.

Logical steps of file handling
1) Open file
2) Processing
3) Close file

Part -2

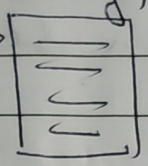
(1) Open()

f = open(filename, mode) (filename: str)

- 'file.txt' // in current dir.
- 'd:\folder\folder2\file.txt' // absolute path

Modes

- r → readonly, FileNotFoundError.
- w → write only, if no such file then it creats and Erase Previos content.
- a → append, No reading, if no such file then it create and can't Erase Previous content
- r+ → reding, writing, ~~to modify~~, FileNotFoundError



file pointer Point at beg. and move to last. and ~~to~~ then writes. but we write first then it writes at beg.

- w+ → reading & writing, if no file then create file, and erase Previous data.
  - first write then read, if we first write the data is erased, Reading not Posible on opening of a file
- a+ → reading and writing, not erased Perv. content and not useful for modify.
- x → exclusive write mode
  always create new file
  use for fresh writting
  x mode is use when their is no such file of that name.

Modes of binary file.

rb, wb, ab, r+b, w+b, a+b, xb

Part-3

f = open('file1.txt', 'r')

(3) f.close() // Closing of a file

(2). Operations

f = open('file1.text', 'r')
→ Error : FileNotFoundError
f = open('file.txt', 'w') // create a file
f.name
→ 'text1.txt'
f.mode
→ 'w'
f.closed
→ False
f.readable()
false
f.writeble()
True

Part-4

- Writing in a file

```
f = open('text1.txt','w')
text = input("Enter some text")
f.write(text)
f.close

f.writelines(l) // to write sequence
```

- Reading

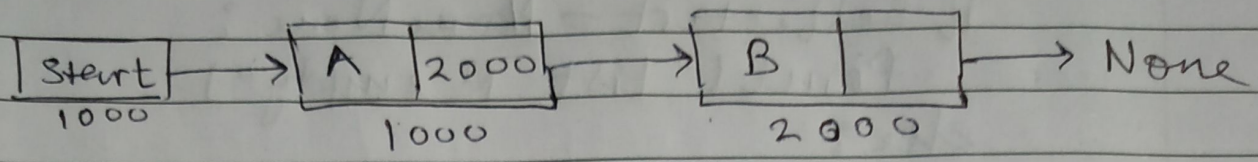
```
read()     → read all data
read(n)    → read n character
readline() → read one line, in loop read lines
             one by one
readlines() → read all lines, and return
              list of data.

f. open('text.txt', 'r')
Print(f.read())   | s = f.read()
f.close()         | Print(s)

Using for loop.
for lines in f:
    Print(line)
f.[lose()
```

Part 5

Part 6

- How to write object in a file?

Linked list




```
Class node:
    def __init__(self, data):
        self.data = data
        self.next = none
Class linked list:
    def __init__(self):
        self.start = None

    def insertLast(self, value):
        newnode = node(value)
        if self.start == None:
            self.start = newNode
        else:
            temp = self.start
            while temp.next != None:
                temp = temp.next
            temp.next = newNode

    def deleteFirst(self):
        if self.start == None:
            Print("Linked list is empty")
        else
            self.start = self.start.next
```

```
def viewList(self):
    if self.start = None:
        Print("List is empty")
    else:
        temp = self.start
        while temp != None:        || temp.next!=
            Print(temp.data, end=' ')
            temp = temp.next

myList = LinkedList()
myList. = insertdata(10)
myList = insert data(20)
myList = insert data(30)
myList = insert data(40)
myList = view data()
myList = deletefirst()
myList = ~~ins~~ viewdata()
```