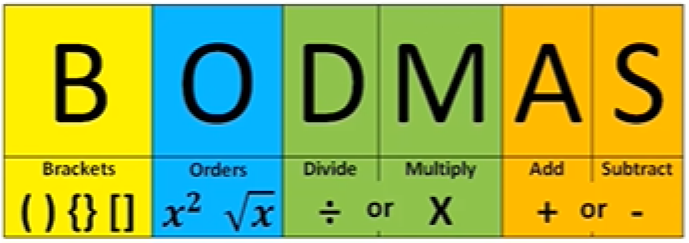
Python Interpreter
IDE --> Integrated Development Environment   (Pycharm,.....)

One Tab Space == 4 Spaces

IDLE →How to clear screen

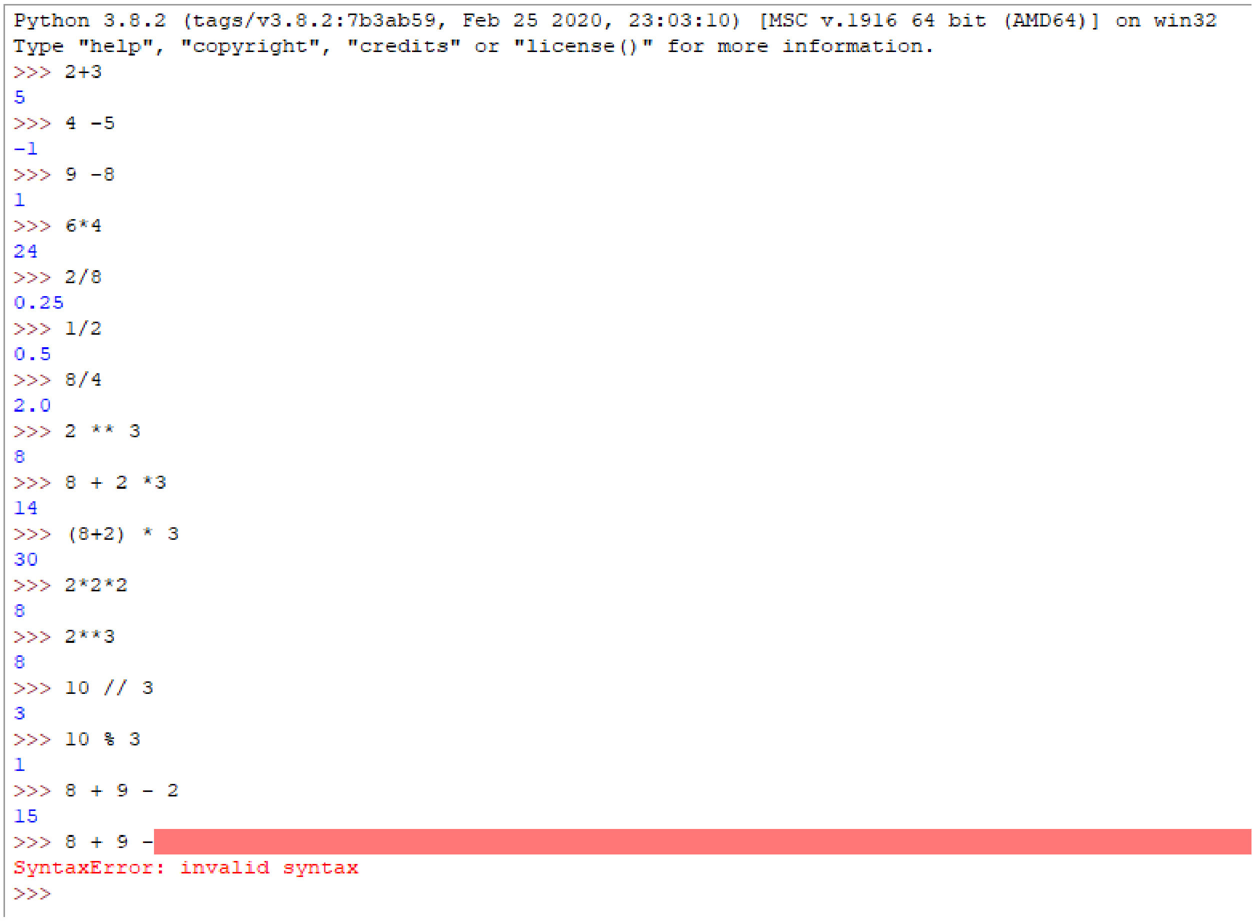
```
//for windows
import os
def cls():
...    os.system("cls")

cls( ) →call function

//for Windows
import os
os.system('cls')
//for linux
import os
os.system('clear')
```

```
>> 8/4
  2.0  →Float point Representation by default

→5/2 = 2.5
>> 5 // 2 = 2       →Integer Division (or) Float Division
>> 8 + 2*3 → 8 + 6 = 14   (It follows BODMAS rule)
```



```
>> 2 ** 3 = 8   (2 power of 3 = 2^3)
```

IDE



```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 2+3
5
>>> 4 -5
-1
>>> 9 -8
1
>>> 6*4
24
>>> 2/8
0.25
>>> 1/2
0.5
>>> 8/4
2.0
>>> 2 ** 3
8
>>> 8 + 2 *3
14
>>> (8+2) * 3
30
>>> 2*2*2
8
>>> 2**3
8
>>> 10 // 3
3
>>> 10 % 3
1
>>> 8 + 9 - 2
15
>>> 8 + 9 -
SyntaxError: invalid syntax
>>>
```

>>> print('raghava's laptop')

SyntaxError: invalid syntax

solution:

>>> print("raghava's laptop")

raghava's laptop

If need to print double quotes use single quote for entire string and vice versa

>>> print('raghava "laptop" ')

raghava "laptop"

>>> print('raghava/'s "laptop" ')

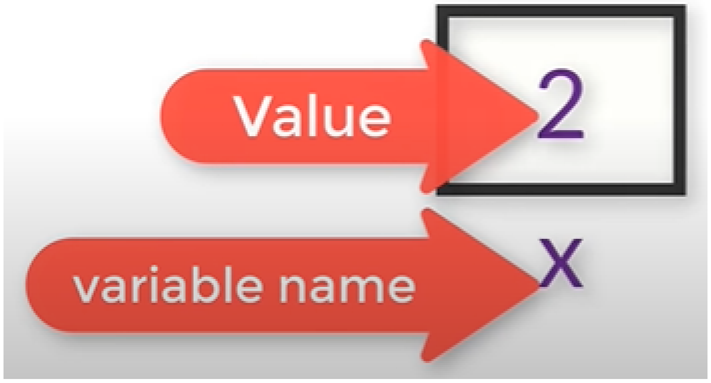
```
SyntaxError: invalid syntax
>>> print('raghava\'s "laptop" ')   → backslash
raghava's "laptop"
```

Concatenation of strings

```
>>> 'raghava' + 'gudiwada'
'raghavagudiwada'
>>> 10 * "Raghava "
'Raghava Raghava Raghava Raghava Raghava Raghava Raghava Raghava Raghava
Raghava '

>>> print('c:\doc\raghava')
c:\docaghava
>>> print('c:\doc\gudiwada')
c:\doc\gudiwada
>>> print('c:\doc\navin')    → here \n means newline
c:\doc
avin
>>> print(r 'c:\doc\navin')  →r Raw string
c:\doc\navin
```

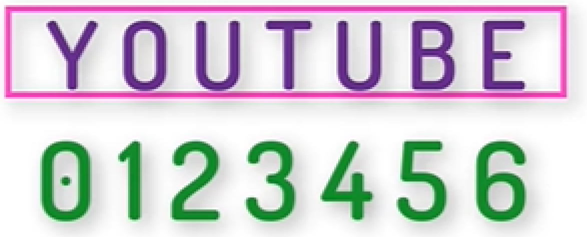
Variables in Python

Variables : It is nothing but a container storing the value in it. We change the value of x.



```
>>> x = 2
>>> x + 3
5
>>> y = 4
>>> x + y
6
>>> x = 8   (The value of x is updated)
>>> x + y
12
>>> x
8
>>> x + 10
18
>>> _ +  y   → (Here underscore will take previous output 18+ 4 = 22)
22


>>> name = "youtube"
>>> name
```

```
'youtube'
>>> name + 'rocks'
'youtuberocks'
```



Single Character Print

```
>>> name[0]
'y'
>>>
>>> name[6]
'e'

>>> name[9]
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    name[9]
IndexError: string index out of range
>>> name[-1]       →negative numbers it start from end ‘-1’
'e'
>>> name[-2]
'b'
>>> name[-7]
'y'
```

Multiple characters Print

```
>>> name[0:2]
'yo'
>>> name[1:4]
'out'
```

```
>>> name[1:]  →If you don’t specify the ending index it goes till the end
'Outube'
>>> name[3:10] →It won’t give any error
'tube'
>>> name[:5] →If you don’t specify the starting Index it will start from beginning
'youtu’
```

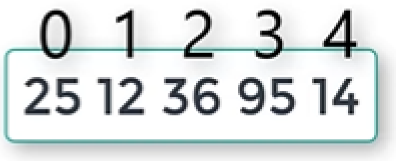
→Strings in python are Immutable(it can’t change)

```
>>> name[1] = 'R'
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    name[1] = 'R'
TypeError: 'str' object does not support item assignment

>>> 'my ' + name[3:]
'my tube'
>>> myname = 'Gudiwada Raghava'
>>> len(myname)    →’len’ default inbuilt function
16
```

List in Python

For list we need to use [ ] Square Bracket like Array

```
>>> nums =[25, 12, 36, 95, 14]
>>> nums
[25, 12, 36, 95, 14]
```



0 1 2 3 4

25 12 36 95 14

```
>>> nums[0]
25
```

```
>>> nums[3]
95
>>> nums[10]
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    nums[10]
IndexError: list index out of range


>>> nums[2:]
[36, 95, 14]
>>> nums[-1]
14
>>> nums[-5]
25
>>> names = ['raghava', 'raju', 'john', 'ramu']
>>> names
['raghava', 'raju', 'john', 'ramu']
>>> values = [9.5, 'Raghava', 25]
>>> values
[9.5, 'Raghava', 25]
```

List we have different types of data to present like integer, float, string …

List of Lists

```
>>> mixed = [nums, names, values]
>>> mixed
[[25, 12, 36, 95, 14], ['raghava', 'raju', 'john', 'ramu'], [9.5, 'Raghava', 25]]

>>> nums.append(45)
>>> nums
[25, 12, 36, 95, 14, 45]
>>> nums.insert(2, 77)
>>> nums
```

```
[25, 12, 77, 36, 95, 14, 45]

>>> nums.remove(14)
>>> nums
[25, 12, 77, 36, 95, 45]

>>> nums.pop(1)
12
>>> nums
[25, 77, 36, 95, 45]
>>> nums.pop()   →it will remove last value (like stack data structure)
45
```

How to delete multiple values

```
>>> del nums[2:]
>>> nums
[25, 77]
```

How to ADD multiple values

```
 >>> nums.extend([25, 77, 12, 14, 36])
>>> nums
[25, 77, 25, 77, 12, 14, 36]


>>> min(nums)
12
>>> max(nums)
77
>>> sum(nums)
266
```

Ascending order of List

```
>>> nums.sort()
>>> nums
[12, 14, 25, 25, 36, 77, 77]
```

Tuples & sets in Python

List & Tuple both are almost similiar. In List we can change the value(Mutable). In Tuple we can’t change value (Immutable).
For Tuple we need to use ( ) Round Bracket.

```
>>> tup = (21, 36, 14, 25)
>>> tup
(21, 36, 14, 25)
```

```
# accessing tuple elements using indexing
>>> tup[1]
36
>>> tup[1] = 34     →we can’t change the value.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Iteration is faster in tuple when compared to List. →Tuple is used when we can’t change the value.

```
>>> tup.index(14)   →It will show the index of the element in tuple.
2
>>> tup.count(25)  →It will count how many times element present in the tuple
1
```

SET

It is the collection of **unique** elements.

For Set we need to use { } Curly Bracket.

```
>>> set = {22, 25, 14, 21, 5}
>>> set        →Value are not in sequence/ not sorted also random order
{5, 21, 22, 25, 14}
```

→Set uses the hash function because of fastness.

Set is the same as List. Difference is the Set can't allow duplicate values. It can't follow the sequence order.

- the discard(x) method removes x from the set, but *doesn't* raise any error if x is not present in the set.
- the pop() method removes and returns a random element from the set.
- the clear() method removes all elements from a set

1. Union

```
>>> first_set = {1, 2, 3}
>>> second_set = {3, 4, 5}
>>> first_set.union(second_set)
{1, 2, 3, 4, 5}
>>>
>>> first_set | second_set     # using the `|` operator
{1, 2, 3, 4, 5}
```

2. Intersection

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.intersection(second_set)
{4, 5, 6}
>>>
>>> first_set & second_set     # using the `&` operator
{4, 5, 6}
```

3. Difference

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.difference(second_set)
{1, 2, 3}
>>>
>>> first_set - second_set     # using the `-` operator
{1, 2, 3}
>>>
>>> second_set - first_set
{8, 9, 7}
```

4. Symmetric Difference

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.symmetric_difference(second_set)
{1, 2, 3, 7, 8, 9}
>>>
>>> first_set ^ second_set     # using the `^` operator
{1, 2, 3, 7, 8, 9}
```

Example of Modify sets

```
>>> a = {1, 2, 3, 4, 5, 6}
>>> b = {4, 5, 6, 7, 8, 9}
>>>
>>> a.update(b)          # a "union" operation
>>> a
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>>
>>> a &= b               # the "intersection" operation
>>> a
{4, 5, 6, 7, 8, 9}
>>>
>>> a -= set((7, 8, 9))  # the "difference" operation    → a= a - set((7, 8,  9))
>>> a
{4, 5, 6}
>>>
>>> a ^= b                # the "symmetric difference" operation
>>> a
{7, 8, 9}
```

the **a.issubset(b)** method or **<=** operator returns true if the a is a subset of b
the **a.issuperset(b)** method or **>=** operator returns true if the a is a superset of b
the **a.isdisjoint(b)** method return true if there are **no common** elements between sets a and b

Dictionary in Python

```
>>> data = {1: 'Raghava', 2: "ramu", 4 : "Rani"}
>>> data
{1: 'Raghava', 2: 'ramu', 4: 'Rani'}
```

Accessing an Element in Dictionary

```
>>> data[1]
'Raghava'
>>> data[2]
'ramu'
>>> data[4]
'Rani'
>>> data[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

Another way of accessing data in dictionary

```
>>> data.get(1)
'Raghava'
>>> data.get(3)
>>> print(data.get(3))
None
>>> data.get(1, "Not Found")
'Raghava'
>>> data.get(3, "Not Found")    →Value is not present for a key then print “Not Found”
'Not Found'
```

Merge two lists into Dictionary

```
>>> keys = ['Raghava', 'Kiran', 'Harsh']
>>> values = ['Python', 'Java', 'javascript']
>>> data = dict(zip(keys, values))
>>> data
{'Raghava': 'Python', 'Kiran': 'Java', 'Harsh': 'javascript'}
```

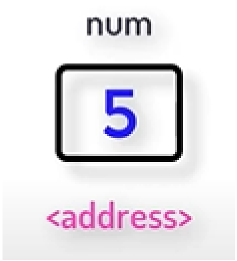
Accessing & Adding & Deleting key value pair into Dictionary

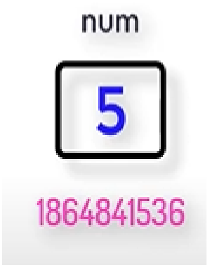
```
>>> data['Kiran']
'Java'
>>> data['Monika']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Monika'
>>> data['Monika'] = 'CSS'
>>> data
{'Raghava': 'Python', 'Kiran': 'Java', 'Harsh': 'javascript', 'Monika': 'CSS'}
>>> del data['Harsh']
>>> data
{'Raghava': 'Python', 'Kiran': 'Java', 'Monika': 'CSS'}
```

```
>>> prog = {'JS' : 'Atom', 'CS' : 'VS', 'Python' : ['Pycharm', 'Sublime'],
'Java':{'JSE': 'Netbeans', 'JEE':'Eclipse'}}
>>> prog
{'JS': 'Atom', 'CS': 'VS', 'Python': ['Pycharm', 'Sublime'], 'Java': {'JSE': 'Netbeans',
'JEE': 'Eclipse'}}
>>> prog['JS']
'Atom'
>>> prog['Python']
['Pycharm', 'Sublime']
>>> prog['Python'][0]
'Pycharm'
>>> prog['Python'][1]
'Sublime'
>>> prog['Java']
{'JSE': 'Netbeans', 'JEE': 'Eclipse'}
>>> prog['Java']['JEE']
'Eclipse'
>>> prog['Java']['JSE']
'Netbeans'
```

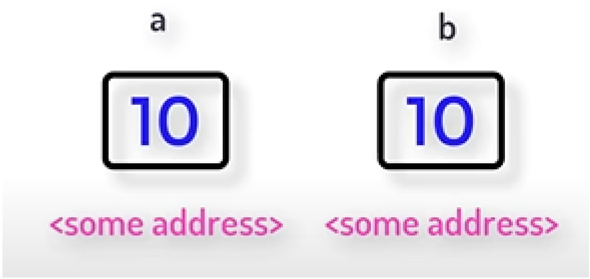
```
>>help()
help> topics
help>modules
….
help>quit
Back to python
>>>
```

More on Variables



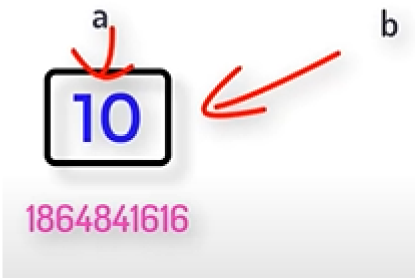


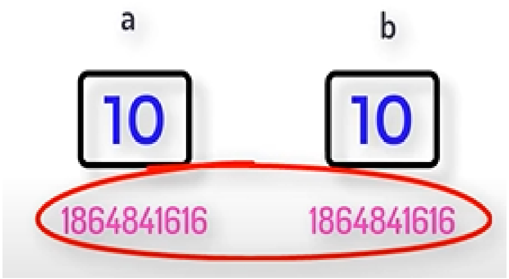
```
>>> num = 5
>>> id(num)          —>getting address of num variable
140736308765608
>>> name = 'raghava'
>>> id(name)
2409476206128
```



```
>>> a = 10
>>> b = a
>>> a
10
>>> b
10
>>> id(a)
140736308765768
```

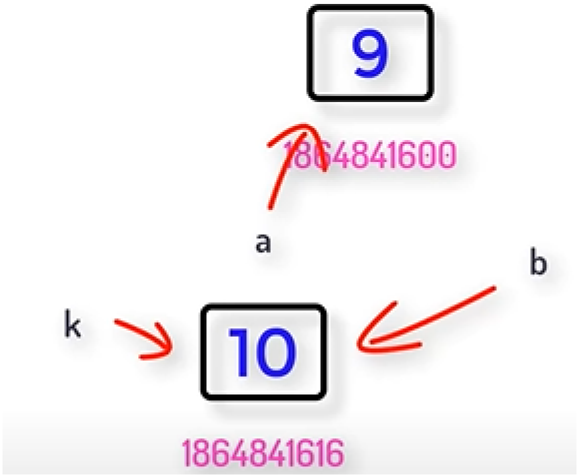
```
>>> id(b)
140736308765768
>>> id(10)
140736308765768
>>> k = 10
>>> id(k)
140736308765768
```





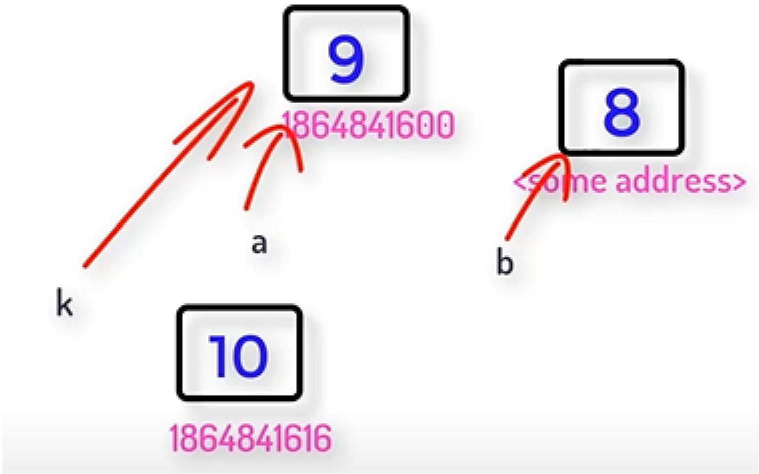
You create multiple variables if they both have the same data then both will point to the same box(Same address).

Python is more memory efficient.

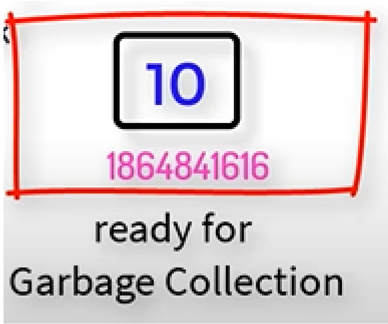


```
>>> a = 9
>>> id(a)
```

```
140736308765736
>>> id(b)
140736308765768
>>> k = a
>>> id(k)
140736308765736
>>> id(b)
140736308765704
```



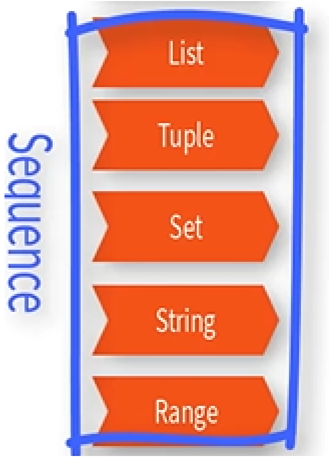
Here, no variable is assigned to 10. Then in python we have **Garbage collection**

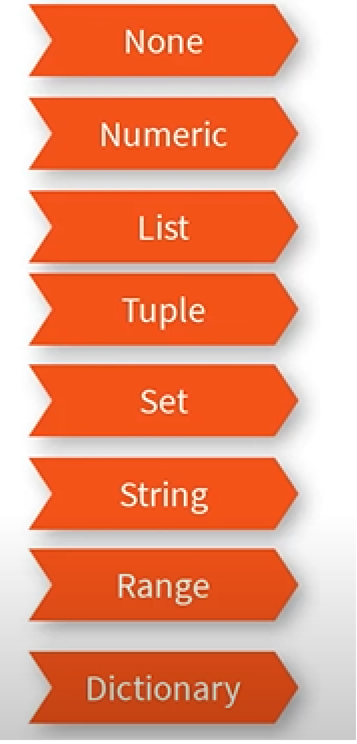


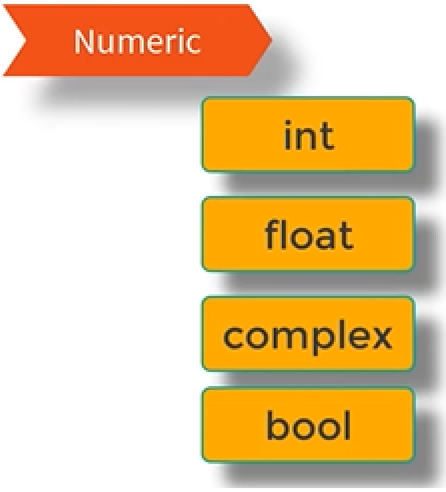
In python we can’t make variable as constant

```
>>> PI = 3.14
>>> PI
3.14
>>> PI = 3.17
>>> PI
3.17
>>> type(PI)
<class 'float'>
>>> type(a)
<class 'int'>
>>> type(b)
<class 'int'>
```

Data Types in Python







```
>>> num =2.5
>>> type(num)
<class 'float'>
>>> num = 9
>>> type(num)
<class 'int'>
>>> num = 6 + 9j
>>> type(num)
<class 'complex'>
>>> a = 5.6
>>> b = int(a)                →Type conversion float to int
>>> type(b)
<class 'int'>
>>> b
5
>>> k = float(b)              →Type conversion int to float
>>> k
5.0
```

```
>>> type(k)
<class 'float'>
>>> k = 6
>>> c = complex(b, k)            →conversion of complex number
>>> c
(5+6j)
>>> b < k
True
>>> bool = b < k
>>> bool
True
>>> type(bool)
<class 'bool'>
>>> b > k
False
>>> int(True)
1
>>> int(False)
0
>>> lst = [25, 36, 45, 12]
>>> type(lst)
<class 'list'>
>>> s = {25, 36, 45, 12, 15, 25}
>>> s
{36, 25, 12, 45, 15}
>>> type(s)
<class 'set'>
>>> t = (25, 36, 45, 12)
>>> t
(25, 36, 45, 12)
>>> type(t)
<class 'tuple'>
>>> str = "raghava"
>>> type(str)
<class 'str'>
```

```
>>> st = 'r'          →There is no type as char in python
>>> type(st)
<class 'str'>
>>> range(10)
range(0, 10)
>>> list(range(10))    →For printing of range we are using list or else we can use for loop
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

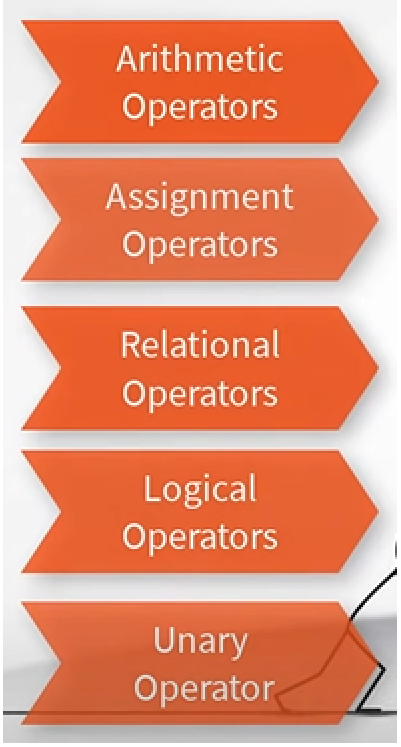
Range take 3 parameters →start, end, difference

→The **range()** function in Python works by taking **three** inputs as parameters which are **start_value, stop_value, and step value.** The range() function returns an immutable sequence of numbers starting from the start_value, and increments the values by the value given in the step_value, and stops at the stop_value.

```
>>> list(range(2, 20, 3) )
[2, 5, 8, 11, 14, 17]
>>> type(range(10))
<class 'range'>
```

```
>>> d = {'raghava': 'samsung', 'kc': 'iphone', 'surya':'realme'}
>>> d
{'raghava': 'samsung', 'kc': 'iphone', 'surya': 'realme'}
>>> d.keys()
dict_keys(['raghava', 'kc', 'surya'])
>>> d.values()
dict_values(['samsung', 'iphone', 'realme'])
>>> d['kc']    →fetching value
'iphone'
>>> d.get('raghava')    →other way of fetching value
'samsung'
```

Operators in Python



Arithmetic Operators

Assignment Operators

Relational Operators

Logical Operators

Unary Operator

```
>>> x = 6
>>> y = 19
>>> x +y
25
>>> x -y
-13
>>> x *y
114
>>> x /y
0.3157894736842105
>>> x = x +2
>>> x
8
>>> x += 2                    →shorthand operator
>>> x
10
```

```
>>> x *= 3                    →shorthand operator
>>> x
30
>>> a, b = 5, 6           →shorthand operator  for assignment
>>> a
5
>>> b
6
```

Unary operator

```
>>> n = 7
>>> -n
-7
>>> n
7
>>> n = -n               →shorthand operator
>>> n
-7
```

Relational operators

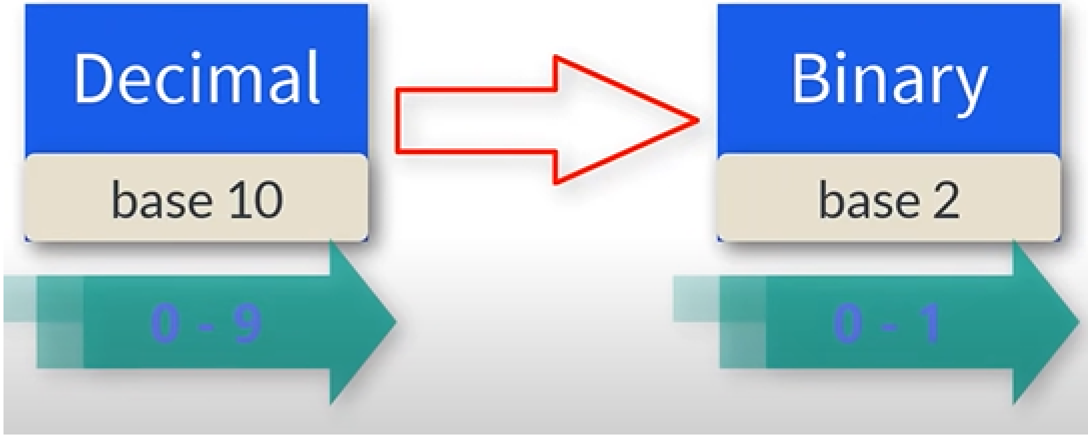
```
>>> a < b
True
>>> a > b
False
>>> a == b
False

>>> a = 6
>>> a == b
True
>>> a <= b
True
>>> a >= b
True
>>> a != b
False
>>> b = 7
>>> a != b
True
```

Logical Operators

```
>>> a = 5
>>> b = 4
>>> a < 8 and b < 5
True
>>> a < 8 and b < 2
False
>>> a < 8 or b < 2
True
>>> x = True
>>> x
True
>>> not x
False
>>> x = not x
>>> x
False
```

Number system conversion in Python

1. Binary
2. Decimal
3. Octal
4. Hexadecimal



```
>>> bin(25)                →Decimal to binary conversion
'0b11001'
>>> 0b0101
5
>>> oct(25)              →Decimal to octal conversion
'0o31'
>>> hex(25)                →Decimal to headecimal conversion
'0x19'
>>> hex(10)
'0xa'
>>> 0xf
15
```

Swap 2 variables in Python

Control +B →to run the program in the Sublime Editor

→**Using the 3rd variable.**

```
a = 5
b = 6

temp = a
a = b
b = temp
```

```
print(a)
print(b)
```

→without using the 3rd variable.

```
a = 5
b = 6

a = a + b          #11      →4 bit
b = a - b          # 5       →3 bit
a = a - b          # 6        →3 bit

print(a)
print(b)
```

→we are wasting **one extra bit**

→Using Xor operator

```
a = 5   #101
b = 6   #110

a = a ^ b
b = a ^ b
a = a ^ b

print(a)
print(b)
```

→Python one line swap 2 variables

```
a = 5   #101
b = 6   #110
```

a, b = b, a

```
print(a)
print(b)
```

Python BitWise Operators

Complement →We will use 2's complement →(1's complement +1)

```
>>> ~12
-13
>>> ~1
-2
>>> ~0
-1
```

Bitwise And (&)

```
>>> 12 & 13
12
>>> 25 & 30
24
```

Bitwise OR (|)

```
>>> 12 | 13
13
```

Bitwise XoR (^)

```
>>> 12 ^ 13
1
>>> 25 ^ 30
7
```

LeftShift (<<)

```
>>> 5 << 1  → value 5 is left shift 1 time (5*2 in one time = 10)
10
```

RightShift (>>)

```
>>> 5 >> 1 → value 5 is right shift 1 time (5/2 in one time = 2)
2
```

Import Math function in Python

```
>>> x = sqrt(25)
Traceback (most recent call last):
  File "<pyshell#108>", line 1, in <module>
    x = sqrt(25)
NameError: name 'sqrt' is not defined
>>> import math
>>> x = math.sqrt(25)
>>> x
5.0
>>> x = math.sqrt(15)
>>> x
3.872983346207417
>>> print(math.floor(2.9))   #floor gives the lowest integer
2
>>> print(math.ceil(2.9))   #ceil gives the highest integer
3
>>> 3 ** 2
9
>>> print(math.pow(3, 2))
9.0
>>> print(math.pi)
3.141592653589793
>>> print(math.e)
2.718281828459045
>>> import math as m
>>> math.sqrt(25)
5.0
>>> m.sqrt(25)
5.0
>>> from math import sqrt, pow            →we imported only 2 functions
>>> pow(2, 10)
1024.0
```

User input in python

```
x = input("Enter 1st number")
print(type(x))
y = input("Enter 2nd number")
z = x + y
print(z)
```

```
Enter 1st number8
<class 'str'>
Enter 2nd number9
89
```

```
x = input("Enter 1st number")    →it is taking as string
a = int(x)                         →converted to integer
y = input("Enter 2nd number")
b = int(y)
z = a + b
print(z)
```

```
x =int(input("Enter 1st number"))
y = int(input("Enter 2nd number"))
z = x + y
print(z)
```

solution:

```
Enter 1st number4
Enter 2nd number3
7
```

```
ch =input("Enter 1st number")
print(ch[0])
```

solution:

```
Enter 1st numberpqr
```

```
p
```

```
ch =input("Enter 1st number")[0]
print(ch)
```

solution:

```
Enter 1st numberpqr
p
```

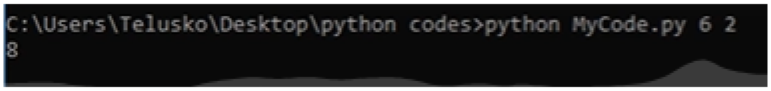
```
result =eval(input("Enter an expr"))
print(result)
```

Solution:

```
Enter an expr2 + 6 -2
6
```

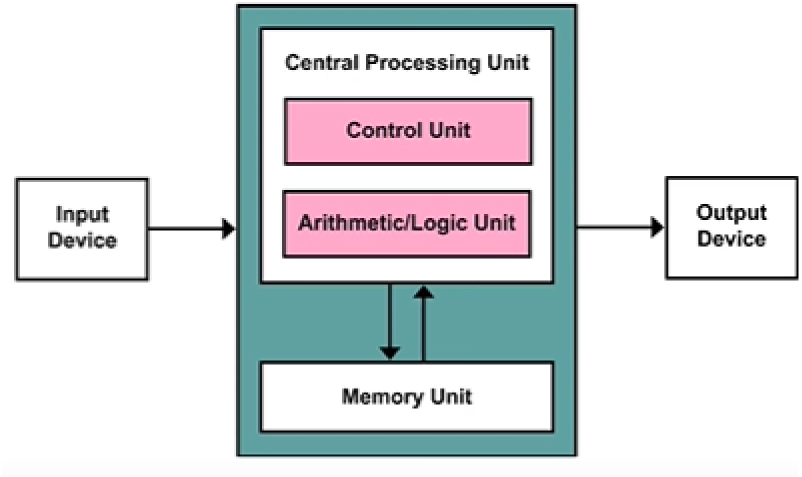
```
import sys                              →argv is present sys module
x = int(sys.argv[1])
y = int(sys.argv[2])

z = x +y
print(z)
```

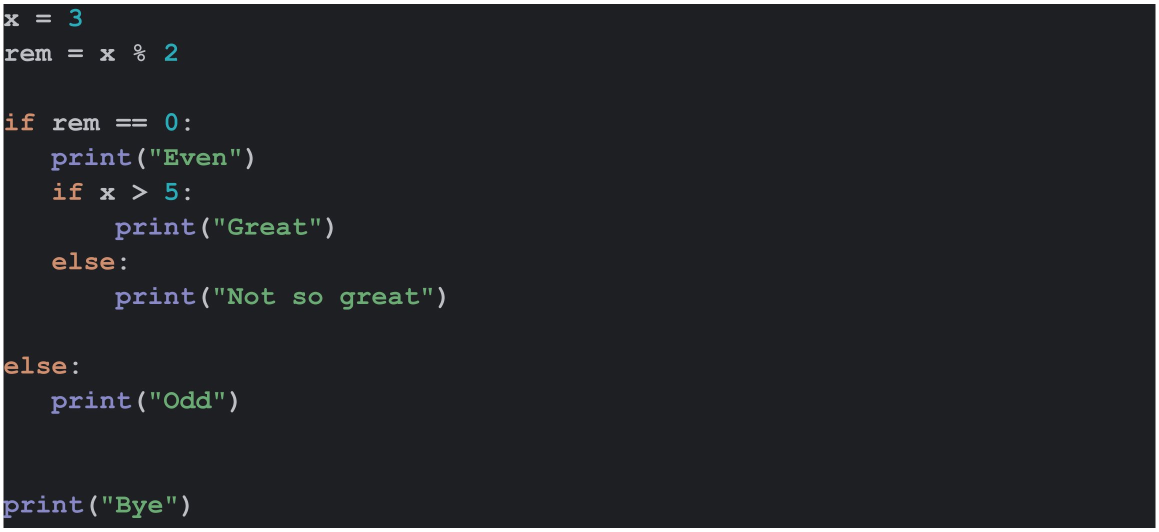
→file is saved as MyCode.py. While running it is passing values.



```
C:\Users\Telusko\Desktop\python codes>python MyCode.py 6 2
8
```

If Elif Else statements in Python





```
x = 3
rem = x % 2

if rem == 0:
   print("Even")
   if x > 5:
       print("Great")
   else:
       print("Not so great")

else:
   print("Odd")


print("Bye")
```

O/p:

Odd
Bye

```
x = 8
rem = x % 2

if rem == 0:
   print("Even")
   if x > 5:
       print("Great")
   else:
       print("Not so great")

else:
   print("Odd")


print("Bye")         #This will print every time
```

O/p:

Even

Great

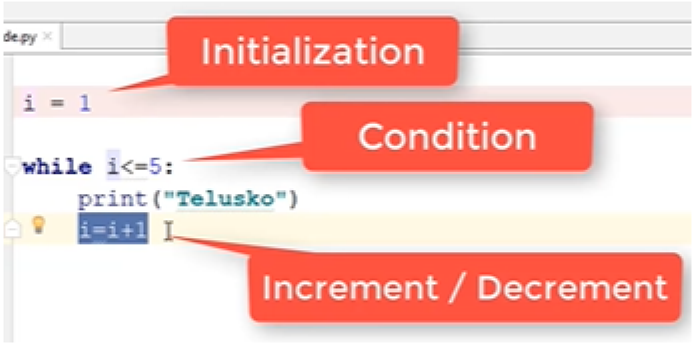
Bye

```
x = 4

if x == 1:
   print("One")

elif x == 2:       #this will execute only when above if condition fail
   print("Two")

elif x == 3:
   print("Three")

elif x == 4:
   print("Four")

else:
   print("Wrong Input")
```

O/P:

Four

While Loop in Python



```
i = 1

while i<=5:
    print("Telusko")
    i=i+1
```

```
i = 1

while i <= 5:
   print("Hello")
   i += 1
```

O/p:

```
Hello
Hello
Hello
Hello
Hello
```

```
i = 1

while i <= 5:
   print("Hello", i)
   i += 1
```

O/P:

```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

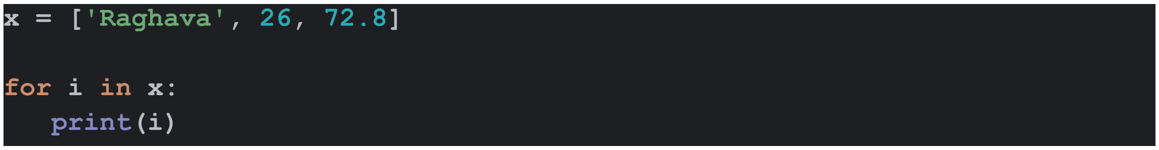
end: string appended after the last value, default a newline

```
i = 1

while i <= 5:
    print("Hello", end=" ")
    j = 1
    while j <= 4:
        print("Raghava", end=" ")
        j += 1

    i += 1
    print()
```

O/P:

```
Hello Raghava Raghava Raghava Raghava
Hello Raghava Raghava Raghava Raghava
Hello Raghava Raghava Raghava Raghava
Hello Raghava Raghava Raghava Raghava
Hello Raghava Raghava Raghava Raghava
```

For Loop in Python



```
x = ['Raghava', 26, 72.8]

for i in x:
   print(i)
```

O/P:

```
Raghava
26
72.8
```

```
for i in ['Raghava', 26, 72.8]:
    print(i)
```

O/P:

```
Raghava
26
72.8
```

```
x = 'Raghava'

for i in x:
   print(i)
```

O/P:

```
R
a
g
h
a
v
a
```

```
for i in range(10, 21, 1):
  print(i)
```

O/P:

```
10
11
12
13
14
15
16
17
18
19
20
```

```
for i in range(1, 21):
    if i%5 != 0:
        print(i)
```

O/P:

```
1
2
3
4
6
7
8
9
11
12
13
14
16
17
18
19
```

Break Continue Pass in Python

```
x = int(input("How many Candies you want?"))
i = 1
while i <= x:
    print("Candy")
    i += 1
```

O/P:

```
 How many Candies you want?5
Candy
Candy
Candy
Candy
Candy
```

```
available = 5

x = int(input("How many Candies you want?"))

i = 1
while i <= x:

   if i > available:
       print("Out of stock")
       break

   print("Candy")
   i += 1
```

O/P:

```
How many Candies you want?7
Candy
Candy
Candy
Candy
Candy
Out of stock
```

Break → it comes out of the loop or skip entire loop

```
for i in range(1, 16):

   if i % 3 == 0 or i%5 == 0:
       continue

   print(i)
```

O/p:

```
1
2
4
7
8
11
13
14
```

Continue: It will skip that part/iteration of execution

```
for i in range(1, 10):

    if i % 2 != 0:
        pass
    else:
        print(i)
```

O/P:

2
4
6
8

If there is no code in the “if” condition it gives an error. That is why just we used “pass”

Printing Patterns in Python

```
for i in range(4):
   for j in range(4):
       print("# ", end ="")
   print()
```

ans:

```
# # # #
# # # #
# # # #
# # # #
```

→range function the value starts from 0.

```
for i in range(4):
  for j in range(i+1):
      print("# ", end ="")
  print()
```

ans:

```
#
# #
# # #
# # # #
```

```
for i in range(4):
  for j in range(4-i):
     print("# ", end ="")
  print()
```

ans:

```
# # # #
# # #
# #
#
```

For else in Python

```
nums = [12, 16, 18, 20, 25]

for num in nums:

   if num % 5 == 0:
       print(num)
```

ans:

```
20
25
```

→Printing only 1st number

```
nums = [12, 16, 18, 20, 25]

for num in nums:

   if num % 5 == 0:
       print(num)
       break
```

ans:

20

```
nums = [12, 16, 18, 21, 26]

for num in nums:

   if num % 5 == 0:
       print(num)
       break
else:
   print("Not Found")
```

ans:

Not Found

Prime Number in Python

```
num = 7

for i in range(2, num):
   if num % i == 0:
       print("Not Prime")
       break
else:
   print("Prime")
```

ans:

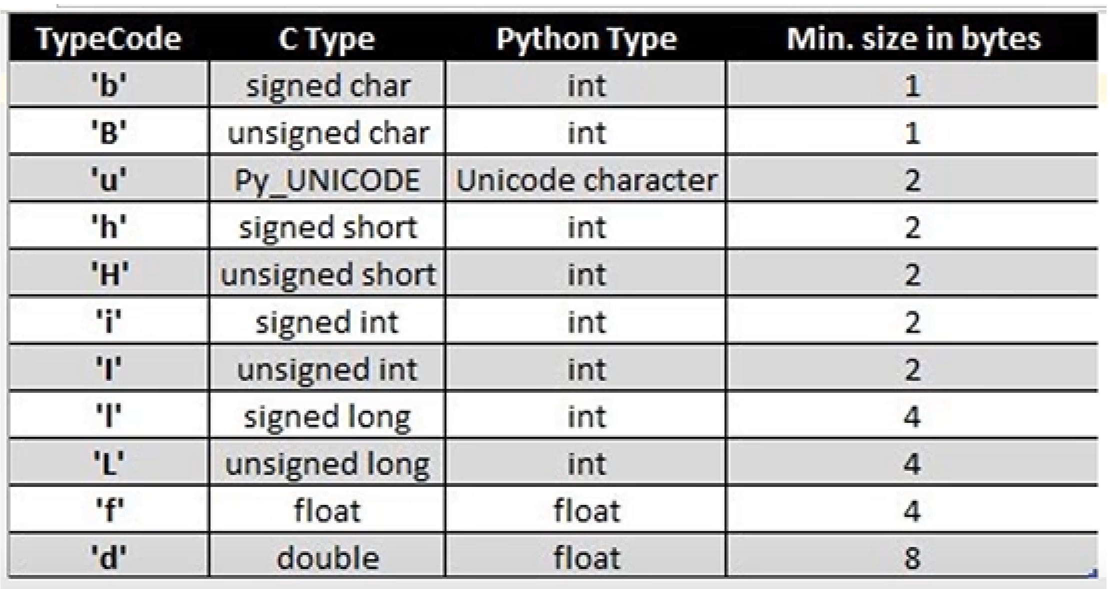
Prime

→efficient way to check prime number

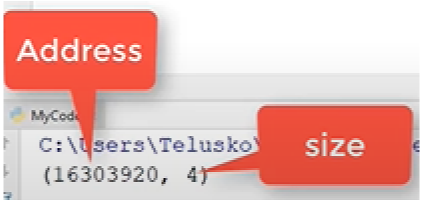
```
def is_prime(n):
    # 0 and 1 are not primes
    if n < 2:
        return False

    # 2 is the only even prime number
    if n == 2:
        return True

    # All other even numbers are not primes
    if n % 2 == 0:
        return False

    # Check for odd factors up to the square root of n
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False

    return True

>>> is_prime(5)
True
>>> is_prime(10)
False
>>> is_prime(97)
True
```

Array in Python

In Array should contain same data type →Homogeneous data type



TypeCode	C Type	Python Type	Min. size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8



```
C:\Users\Telusko\
(16303920, 4)
```

→ Buffer info will give address & Size.

from array import *

vals = array('i', [5, 9, -8, 4, 2])

newArr = array(vals.typecode, (a for a in vals))

```
for x in newArr:
    print(x)
print("Need some Space")
```

sqr = array(vals.typecode, (a*a for a in vals)) #printing square of a number

```
####we will try while loop for printing
i = 0
while i < len(sqr):
    print(sqr[i])
    i= i+1

print("Empty line")

print(vals)

print(vals[0])

print(vals.buffer_info())

print(vals.typecode)

for i in range(len(vals)):          #print all values in array one by one
    print(vals[i])
print("Newline")

#####one more way to print elements one by one

for x in vals:
    print(x)
vals.append(10)
vals.append(50)

print(vals)

vals.remove(2)

print(vals)

vals.pop(2)      #will remove third element
```

```
print(vals)

vals.reverse()

print(vals)

arr = array('u', ['a', 'd', 'z'])

for x in arr:
    print(x)
```

Answer:

```
5
9
-8
4
2
Need some Space
25
81
64
16
4
Empty line
array('i', [5, 9, -8, 4, 2])
5
(2472167127856, 5)
i
5
9
-8
4
2
```

```
Newline
5
9
-8
4
2
array('i', [5, 9, -8, 4, 2, 10, 50])
array('i', [5, 9, -8, 4, 10, 50])
array('i', [5, 9, 4, 10, 50])
array('i', [50, 10, 4, 9, 5])
a
d
z
[Finished in 75ms]
```

Array values from User in Python | Search in Array

```
from array import *

arr = array('i', [])

n = int(input("Enter the length of the array :"))

for i in range(n):
    x = int(input("Enter the next value :"))
    arr.append(x)


print(arr)

val = int(input("Enter the value for search :"))

k = 0
for e in arr:
    if e == val:
        print(k)        #printing index of the value in array
        break

    k = k+1
if k == len(arr):
    print("Not Found")

print("New line")

####for search we can use function

print(arr.index(val))
```

Answer:

```
Enter the length of the array :4
Enter the next value :12
Enter the next value :56
Enter the next value :86
Enter the next value :97
array('i', [12, 56, 86, 97])
Enter the value for search :86
2
New line
2
```

Why Numpy? Installing Numpy in Pycharm

For Multidimensional array & Scientific calculation of array we need **numpy package**

→In numpy data type mention in array is optional
6 ways to create an array

```
from numpy import *

arr = array([1, 3, 4, 6, 9, 10])

print(arr)
```

O/P:

```
[ 1  3  4  6  9 10]
```

1. array()

```
from numpy import *

arr = array([1, 2, 3, 4, 5])

print(arr)
print(arr.dtype)

arr = array([1, 2, 3, 4, 5.0])

print(arr)
print(arr.dtype)    #all values are converted into float -->Implicit Conversion

arr = array([1, 2, 3, 4, 5], float)

print(arr)
print(arr.dtype)
```

O/P:

```
[1 2 3 4 5]
int32
[1. 2. 3. 4. 5.]
float64
[1. 2. 3. 4. 5.]
float64
```

2. linspace()

```
from numpy import *

arr = linspace(0, 15, 16) #start, stop-->here last element is also included,
parts

print(arr)  #16 different parts -->0 to 15 = 16
print(arr.dtype)

print()

arr = linspace(0, 15, 20) #start, stop-->here last element is also included,
parts

print(arr)  #16 different parts -->0 to 15 = 16
print(arr.dtype)
print()

#if you don't specify by default 50 parts
arr = linspace(0, 15) #start, stop-->here last element is also included, parts

print(arr)  #16 different parts -->0 to 15 = 16
print(arr.dtype)
```

O/P:

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15.]
float64

[ 0.          0.78947368  1.57894737  2.36842105  3.15789474  3.94736842
  4.73684211  5.52631579  6.31578947  7.10526316  7.89473684  8.68421053
  9.47368421 10.26315789 11.05263158 11.84210526 12.63157895 13.42105263
 14.21052632 15.        ]
```

```
float64

[ 0.          0.30612245  0.6122449   0.91836735  1.2244898   1.53061224
  1.83673469  2.14285714  2.44897959  2.75510204  3.06122449  3.36734694
  3.67346939  3.97959184  4.28571429  4.59183673  4.89795918  5.20408163
  5.51020408  5.81632653  6.12244898  6.42857143  6.73469388  7.04081633
  7.34693878  7.65306122  7.95918367  8.26530612  8.57142857  8.87755102
  9.18367347  9.48979592  9.79591837 10.10204082 10.40816327 10.71428571
 11.02040816 11.32653061 11.63265306 11.93877551 12.24489796 12.55102041
 12.85714286 13.16326531 13.46938776 13.7755102  14.08163265 14.3877551
 14.69387755 15.        ]
float64
```

3. arange()

```
from numpy import *

arr = arange(1, 15, 2) #start, stop, steps

print(arr)
print(arr.dtype)
```

O/P:

```
[ 1  3  5  7  9 11 13]
int32
```

4. logspace()

```
from numpy import *

arr = logspace(1, 40, 5) #start, stop, steps

print(arr)
print(arr.dtype)
print('%.2f' %arr[0])
print('%.2f' %arr[2])
```

O/P:

```
[1.00000000e+01 5.62341325e+10 3.16227766e+20 1.77827941e+30
 1.00000000e+40]
```

float64
10.00
31622776601683793296.00

5. zeros() →all values by default zeros

```
from numpy import *

arr = zeros(5, int)     # mention size

print(arr)
print(arr.dtype)
print()

arr1 = zeros(5)

print(arr1)
print(arr1.dtype)
```

O/P:

```
[0 0 0 0 0]
int32

[0. 0. 0. 0. 0.]
float64
```

6. ones() →all values by default ones

```
from numpy import *

arr = ones(5, int)  # mention size

print(arr)
print(arr.dtype)
print()

arr1 = zeros(5)

print(arr1)
print(arr1.dtype)
```

O/P:

```
[1 1 1 1 1]
int32

[0. 0. 0. 0. 0.]
float64
```

Copying an Array in Python

→Adding every element in array by some number

```
from numpy import *

arr = array([1, 2, 3, 4, 5])

arr = arr + 5

print(arr)
```

O/P:

```
[ 6  7  8  9 10]
```

→Addition of two Arrays

```
from numpy import *

arr1 = array([1, 2, 3, 4, 5])

arr2 = array([6, 1, 2, 8, 7])

arr3 = arr1 + arr2

print(arr3)
```

O/P:

```
[ 7  3  5 12 12]
```

→Finding Sin, Cos, log values of the array

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(sin(arr))
```

O/P:

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

```
from numpy import *

arr = array([1, 2, 3, 4, 5])

print(cos(arr))
```

O/P:

```
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
```

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(log(arr))
```

O/P:

```
[0.         0.69314718 1.09861229 1.38629436 1.60943791]
```

→Square root of each element in array

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(sqrt(arr))
```

O/P:

```
[1.         1.41421356 1.73205081 2.         2.23606798]
```

→Addition of all elements in an array

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(sum(arr))
```

O/P:

```
15
```

—-->

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(min(arr))            →Minimum element in an array
```

O/P:

```
1
```

```
from numpy import *

arr = array([1, 2, 3, 4, 5])


print(max(arr))            →Maximum element in an array
```

O/P:

```
5
```

→Sort an array

```
from numpy import *

arr = array([1, 25, 32, 13, 5])


print(sort(arr))
```

O/P:

```
[ 1  5 13 25 32]
```

→Concatenation of two arrays

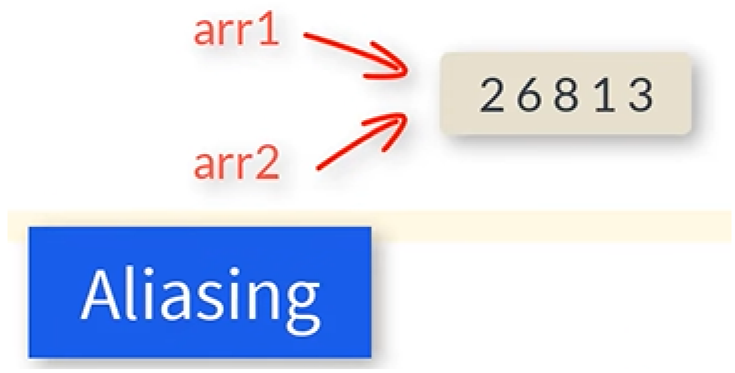
```
from numpy import *

arr1 = array([1, 25, 32, 13, 5])

arr2 = array([1, 2, 3, 4, 5])


print(concatenate([arr1, arr2]))
```

O/P:

```
[ 1 25 32 13  5  1  2  3  4  5]
```

→1. copy of an array (Assigning or Aliasing)

Then still we have only an array that exists. Both are pointing to same array

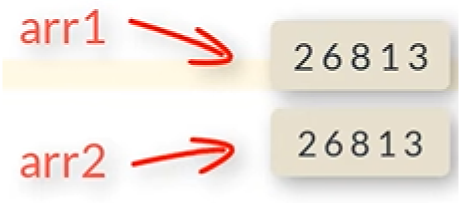


```
from numpy import *

arr1 = array([2, 6, 8, 1, 3])

arr2 = arr1

print(arr1)
print(arr2)

print(id(arr1))
print(id(arr2))
```

O/P:

```
[2 6 8 1 3]
[2 6 8 1 3]
140194761505360
140194761505360
```

→Two different address for two arrays



```
from numpy import *

arr1 = array([2, 6, 8, 1, 3])

arr2 = arr1.view()

print(arr1)
print(arr2)

print(id(arr1))
print(id(arr2))
```

O/P:

```
[2 6 8 1 3]
[2 6 8 1 3]
139935743916624
139935743917296
```

2. Shallow Copy:

When you're changing the element in one array it is changing in other array also. →that os we don’t want

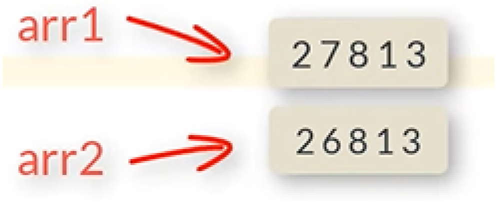
```
from numpy import *

arr1 = array([2, 6, 8, 1, 3])

arr2 = arr1.view()
```

```
arr1[1] = 7

print(arr1)
print(arr2)

print(id(arr1))
print(id(arr2))
```

O/P:

```
[2 7 8 1 3]
[2 7 8 1 3]
139896625871440
139896625872112
```

3. Deep Copy:



→The above error can be modified by using the copy()

```
from numpy import *

arr1 = array([2, 6, 8, 1, 3])

arr2 = arr1.copy()

arr1[1] = 7

print(arr1)
print(arr2)

print(id(arr1))
print(id(arr2))
```

O/P:

```
[2 7 8 1 3]
[2 6 8 1 3]
139896625871440
139896625872112
```

Working with Matrix in Python

```
from numpy import *

arr1 = array([
             [1, 2, 3, 6, 2, 9],
             [4, 5, 6, 7, 5, 3]

         ])

print(arr1.dtype)
print(arr1.ndim)    #dimension
print(arr1.shape)   #no of rows & columns
print(arr1.size)

arr2 = arr1.flatten()  #convert from 2D to 1D array

print(arr2)

arr3 = arr2.reshape(3, 4)      #convert from 1D to 2D array

print(arr3)

print(arr3.ndim)

arr4 = arr2.reshape(2, 2, 3)      #convert from 1D to 3D array

print(arr4)
print(arr4.ndim)
```

O/P:

```
int64
2
(2, 6)
12
[1 2 3 6 2 9 4 5 6 7 5 3]
[[1 2 3 6]
```

```
 [2 9 4 5]
 [6 7 5 3]]
2
[
        [[1 2 3]
         [6 2 9]]

        [[4 5 6]
        [7 5 3]]
]
3
```

→

```
from numpy import *

arr1 = array([
                [1, 2, 3, 6],
                [4, 5, 6, 7]

            ])

m = matrix(arr1)

m1 = matrix('1 2 3 6; 4 5 6 7')

print(m)
print(m1)

m2 = matrix('1 2 3; 6 4 5;1 6 7')
print(m2)
print(diagonal(m2))
print(m2.min())
print(m2.max())
```

O/P:

```
[[1 2 3 6]
 [4 5 6 7]]

[[1 2 3 6]
 [4 5 6 7]]

[[1 2 3]
 [6 4 5]
 [1 6 7]]

[1 4 7]

1
7
```

→Matrix Multiplication

```
from numpy import *

m1 = matrix('1 2 3; 6 4 5; 1 6 7')
m2 = matrix('1 2 3; 6 2 5; 8 6 7')

m3 = m1 + m2

print(m3)

m4 = m1 * m2

print(m4)
```

O/P:

```
[[ 2  4  6]
 [12  6 10]
 [ 9 12 14]]

[[37 24 34]
 [70 50 73]
 [93 56 82]]
```

Functions in Python

→Creating our own functions

```
def greet():
   print("Hello")
   print("Good Morning")

greet()
```

O/P:

```
Hello
Good Morning
```

→Addition

```
def add(x, y):
    c = x + y
    print(c)

add(3, 4)
```

O/P:

```
7
```

→This function return value

```
def add(x, y):
    c = x + y
    return c

result = add(3, 4)
print(result)
```

O/P:

```
7
```

→This will return two values

```
def add_sub(x, y):
    c = x + y
    d = x - y
    return c, d

result1, result2  = add_sub(3, 4)
print(result1, result2)
```

O/P:

```
7    -1
```

→Here only result2 is printing

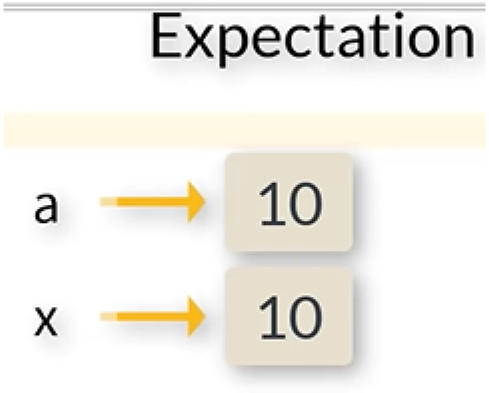
```
def add_sub(x, y):
    c = x + y
    d = x - y
    return c, d

_, result2  = add_sub(3, 4)
print(result2)
```

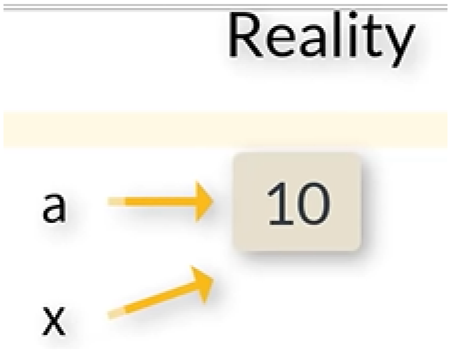
O/P:

```
-1
```

Function Arguments in Python

→In python we don’t have pass by value & pass by reference concept

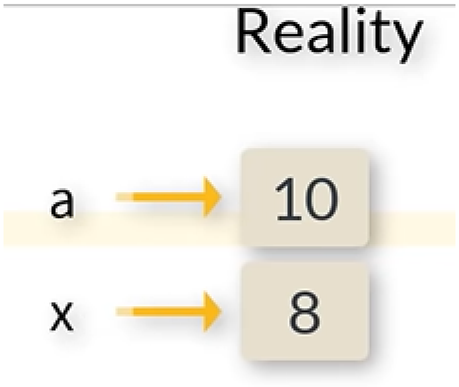




```
def update(x):
    print(id(x))
    x = 8
    print("x", x)

a = 10
print(id(a))
update(a)
print("a", a)
```

O/P:

```
140604243788304
140604243788304
x 8
a 10
```



```
def update(x):
    print(id(x))
    x = 8
    print(id(x))      #This line is updated from above code
    print("x", x)

a = 10
print(id(a))
update(a)
print("a", a)
```

O/P:

```
140305439162896
140305439162896
140305439162832
x 8
a 10
```

→When we update the value it will change the address of the variable.

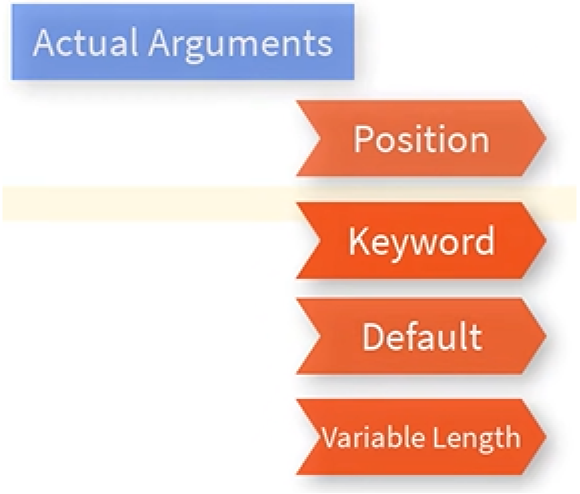
→**LIST**

```
def update(x):
    print(id(x))
    x[1] = 15
    print(id(x))
    print("x", x)

lst = [10, 20, 30]
print(id(lst))
update(lst)
print("lst", lst)
```

O/P:

```
140167234996032
140167234996032
140167234996032
x [10, 15, 30]
lst [10, 15, 30]
```

Types of Arguments in Python



```
def person(name, age=18):
    print(name)
    print(age)

person('Raghava', 28)              #position

person(age = 28, name = 'Raghava')            #keyword
person('Raghava')                    #default
```

O/P:

```
Raghava
28
Raghava
28
Raghava
18
```

→Variable length

```
def sum(a, *b):    #*b takes multiple values
    print(a)
    print(b)          # a integer, b tuple
    c = a

    for i in b:
        c = c + i

    print(c)

sum(3, 6, 34, 70)          #variable length
```

O/P:

```
3
(6, 34, 70)
113
```

→modify the above code

```
def sum(*b):    #*b takes multiple values

    c = 0

    for i in b:
        c = c + i

    print(c)

sum(3, 6, 34, 70)
```

O/P:

```
113
```

Keyworded Variable Length Arguments in Python

```
def person(name, **data):    #double star it will accept keyword arguments
    print(name)
    print(data)

    for i, j in data.items():
        print(i, j)

person("Raghava", age = 26, city ="Vijayawada", mob=8341613068)
```

O/P:

```
Raghava
{'age': 26, 'city': 'Vijayawada', 'mob': 8341613068}
age 26
city Vijayawada
mob 8341613068
```

Global Keyword in Python | Global vs Local Variable

```
a = 10

def something():
    a = 15
    print("in fun", a)

something()

print("outside", a)
```

O/P:

```
in fun 15
outside 10
```

—-------------------------------->

```
a = 10

def something():
    global a
    a = 15                    #it is also global
    print("in fun", a)

something()

print("outside", a)
```

O/P:

```
in fun 15
outside 15
```

—---→

```
a = 10
print(id(a))

def something():

    a = 9
    x = globals()['a']              #x, a both pointing to same location
    print(id(x))
    print("in fun", a)

something()

print("outside", a)
```

O/P:

```
139665239720464
139665239720464
in fun 9
outside 10
```

→we want to change the global variable without affecting the local variable.

```
a = 10
print(id(a))

def something():

    a = 9
    x = globals()['a']
    print(id(x))
    print("in fun", a)
    globals()['a'] = 16


something()

print("outside", a)
```

O/P:

```
140016148431376
140016148431376
in fun 9
outside 16
```

Pass List to a Function in Python

```
def count(lst):

    even = 0
    odd = 0

    for i in lst:
        if i%2 == 0:
            even += 1
        else:
            odd += 1
    return even, odd

lst = [12, 7, 3, 19, 18, 23, 27, 88, 64]
even, odd = count(lst)

print("Total no of even numbers in the list", even)
print("Total no of odd numbers in the list", odd)

print("Even : {} and Odd : {}".format(even, odd))
```

O/P:

```
Total no of even numbers in the list 4
Total no of odd numbers in the list 5
Even : 4 and Odd : 5
```

→ Modified the above code the list is taken from user as input

```
def count(lst):

    even = 0
    odd = 0

    for i in lst:
        if i%2 == 0:
            even += 1
        else:
            odd += 1
    return even, odd

lst = []

n = int(input("Enter the number of elements in list: "))

for i in range(0, n):
    ele = int(input())
    # adding element in the list
    lst.append(ele)

even, odd = count(lst)


print("Total no of even numbers in the list", even)
print("Total no of odd numbers in the list", odd)

print("Even : {} and Odd : {}".format(even, odd))
```

O/P:

```
Enter the number of elements in list: 7
12
13
16
18
90
15
28
Total no of even numbers in the list 5
Total no of odd numbers in the list 2
Even : 5 and Odd : 2
```

Fibonacci Sequence

```
def fib(n):
    a = 0
    b = 1
    if n <= 0:
        print("Invalid Number")
    elif n == 1:
        print(a)
    else:
        print(a)
        print(b)

        for i in range(2, n):
            c = a + b
            a = b
            b = c
            print(c)
```

```
fib(5)
```

O/P:

```
0
1
1
2
3
```

Factorial

Iterative Approach:

```
def fact(n):


    res = 1
    for i in range(1, n+1):
        res *= i

    return res

x = 5

result = fact(x)
print(result)
```

O/P:

```
120
```

```
import sys

print(sys.getrecursionlimit())
```

O/P:

```
1000
```

→To Increase the Recursion limit

```
import sys

sys.setrecursionlimit(3000)
print(sys.getrecursionlimit())
```

O/P:

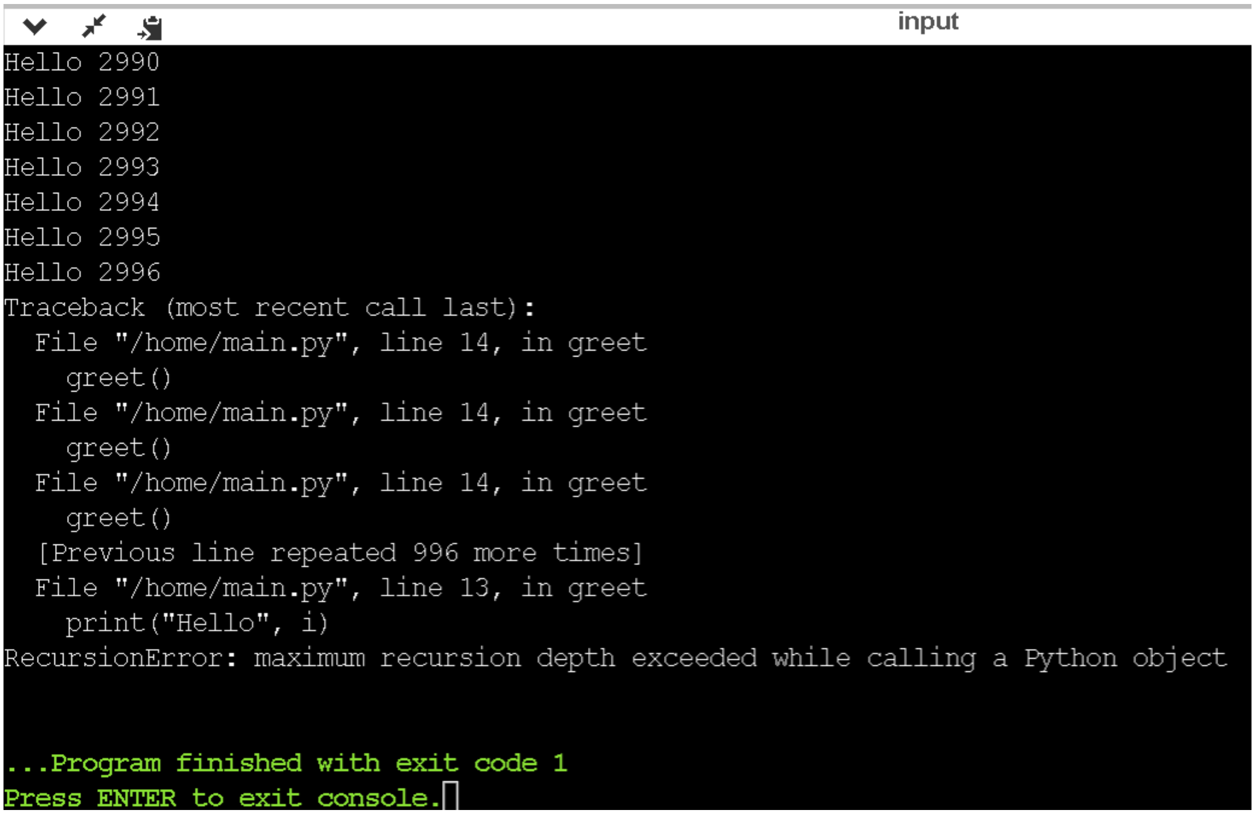
```
3000
```

—--->

```
import sys

sys.setrecursionlimit(3000)
print(sys.getrecursionlimit())

i = 0



def greet():
   global i
   i += 1
   print("Hello", i)
   greet()

greet()
```

O/P:

The limit is 3000 but it is printing upto 2996



```
Hello 2990
Hello 2991
Hello 2992
Hello 2993
Hello 2994
Hello 2995
Hello 2996
Traceback (most recent call last):
  File "/home/main.py", line 14, in greet
    greet()
  File "/home/main.py", line 14, in greet
    greet()
  File "/home/main.py", line 14, in greet
    greet()
  [Previous line repeated 996 more times]
  File "/home/main.py", line 13, in greet
    print("Hello", i)
RecursionError: maximum recursion depth exceeded while calling a Python object


...Program finished with exit code 1
Press ENTER to exit console.
```

Factorial using Recursion

```
def fact(n):

   if n == 0:
      return 1

   return n*fact(n-1)

x = 5

result = fact(x)
print(result)
```

O/P:

120

Anonymous Functions

Functions without name or lambda

→Functions are object in python

→Square of a number

```
f = lambda a : a*a

result = f(5)
print(result)
```

O/P:

```
25
```

→Addition of two Numbers

```
f = lambda a, b : a + b

result = f(5, 6)
print(result)
```

O/P:

```
11
```

Filter Map Reduce

→Filter takes two arguments one is function, the other one is list

Filter

```
def is_even(n):
    return n%2 == 0



nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(is_even, nums))

print(evens)
```

O/P:

```
[2, 4, 8, 10]
```

→The above function is replaced with lambda

```
nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(lambda n : n%2 == 0, nums))

print(evens)
```

O/P:

```
[2, 4, 8, 10]
```

Map

```
def update(n):
    return n*2

nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(lambda n : n%2 == 0, nums)) #function, list/iterable

double = list(map(update, evens))   #function, list/iterable

print(double)
```

O/P:

```
[4, 8, 16, 20]
```

→The above function is replaced with lambda

```
nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(lambda n : n%2 == 0, nums))

double = list(map(lambda n : n*2, evens))

print(double)
```

O/P:

```
[4, 8, 16, 20]
```

Reduce

```
from functools import reduce

def add_all(a, b):
    return a + b


nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(lambda n : n%2 == 0, nums))

double = list(map(lambda n : n*2, evens))
print(double)
sum = reduce(add_all, double)

print(sum)
```

O/P:

```
[4, 8, 16, 20]
48
```

→The above function is replaced with lambda

```
from functools import reduce

nums = [2, 3, 5, 4, 8, 1, 10]

evens = list(filter(lambda n : n%2 == 0, nums))

double = list(map(lambda n : n*2, evens))

print(double)

sum = reduce(lambda a, b : a +b, double)
```

```
print(sum)
```

O/P:

```
[4, 8, 16, 20]
48
```

Decorators

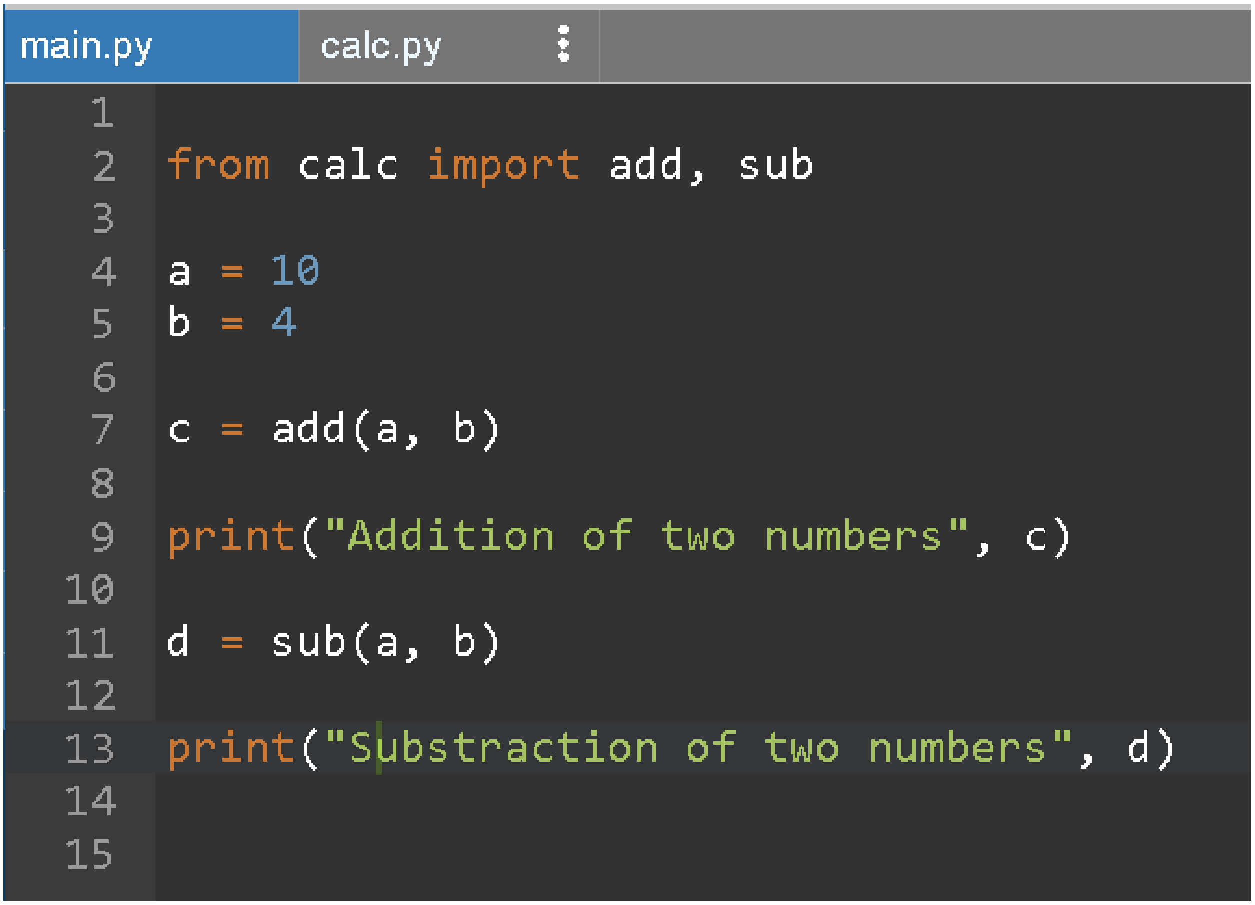
→we can add extra features to the existing code
The condition if a <b then we need to swap a, b we don’t not have access to the original function. By using decorators we need to modify.

```
def div(a,b):
    print(a/b)


def smart_div(func):

    def inner(x, y):   #will take 2 arguments that is same as original function
        if x < y:
            x, y = y, x
        return func(x, y)  #the current x, y is after modified

    return inner

div1 = smart_div(div)

div1(2,4)
```

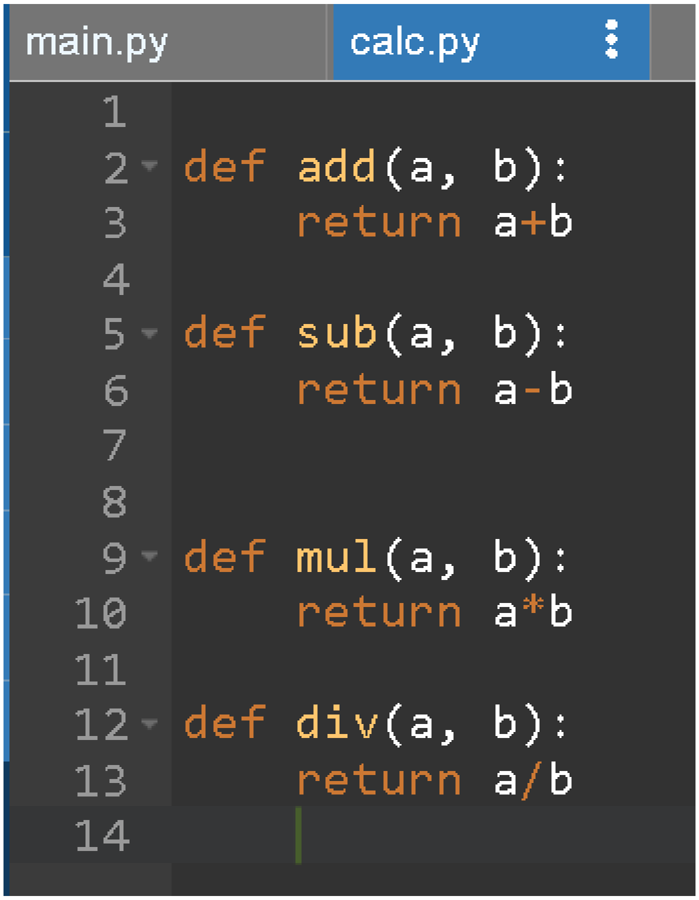
O/P:

```
2.0
```

Modules



```python

from calc import add, sub

a = 10
b = 4

c = add(a, b)

print("Addition of two numbers", c)

d = sub(a, b)

print("Substraction of two numbers", d)


```



```

def add(a, b):
    return a+b

def sub(a, b):
    return a-b


def mul(a, b):
    return a*b

def div(a, b):
    return a/b

```

O/P:

Addition of two numbers 14
Substraction of two numbers 6

Special Variable __name__

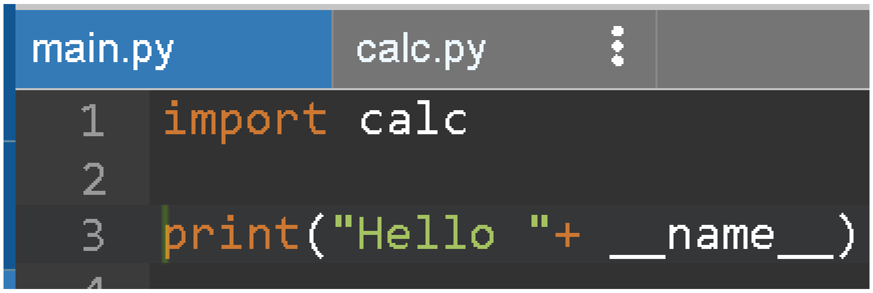
print("Hello"+ __name__)

O/P:

Hello__main__

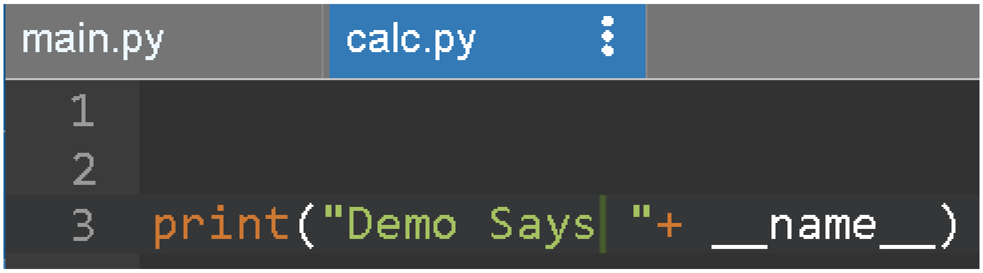
If you're importing in the other function then it will print the module name.

main.py



```python
import calc

print("Hello "+ __name__)
```

calc.py

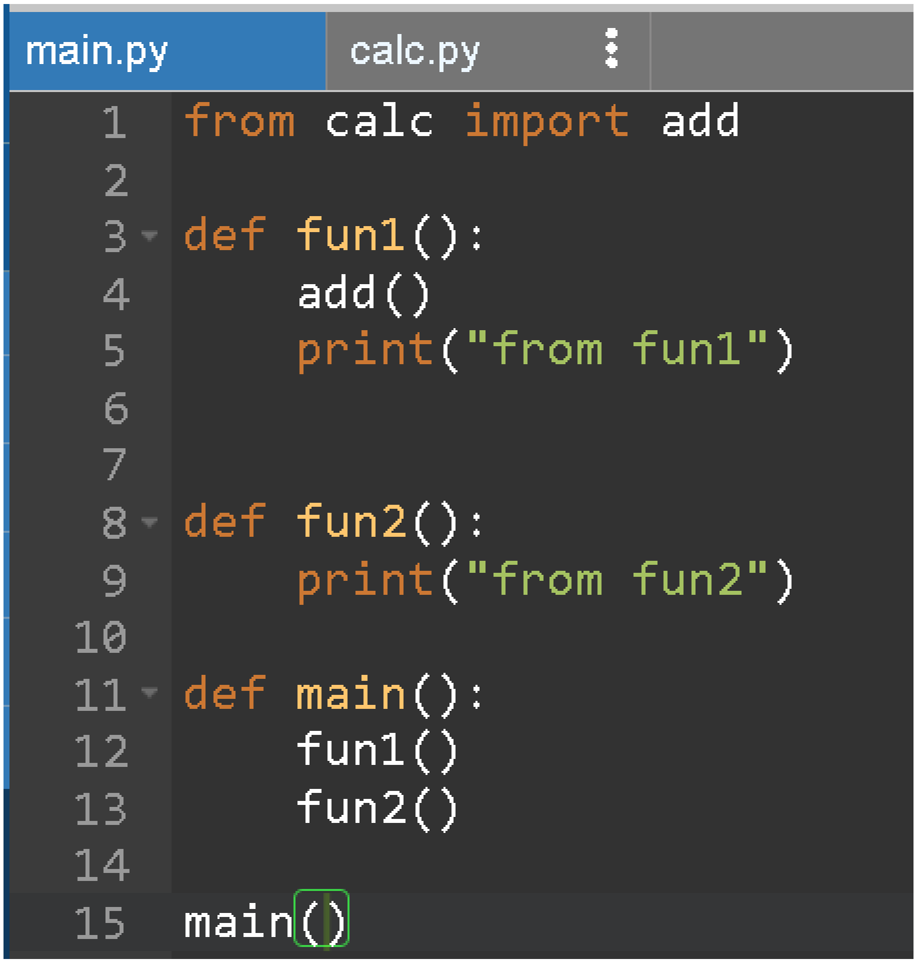


```python


print("Demo Says "+ __name__)
```

O/P:

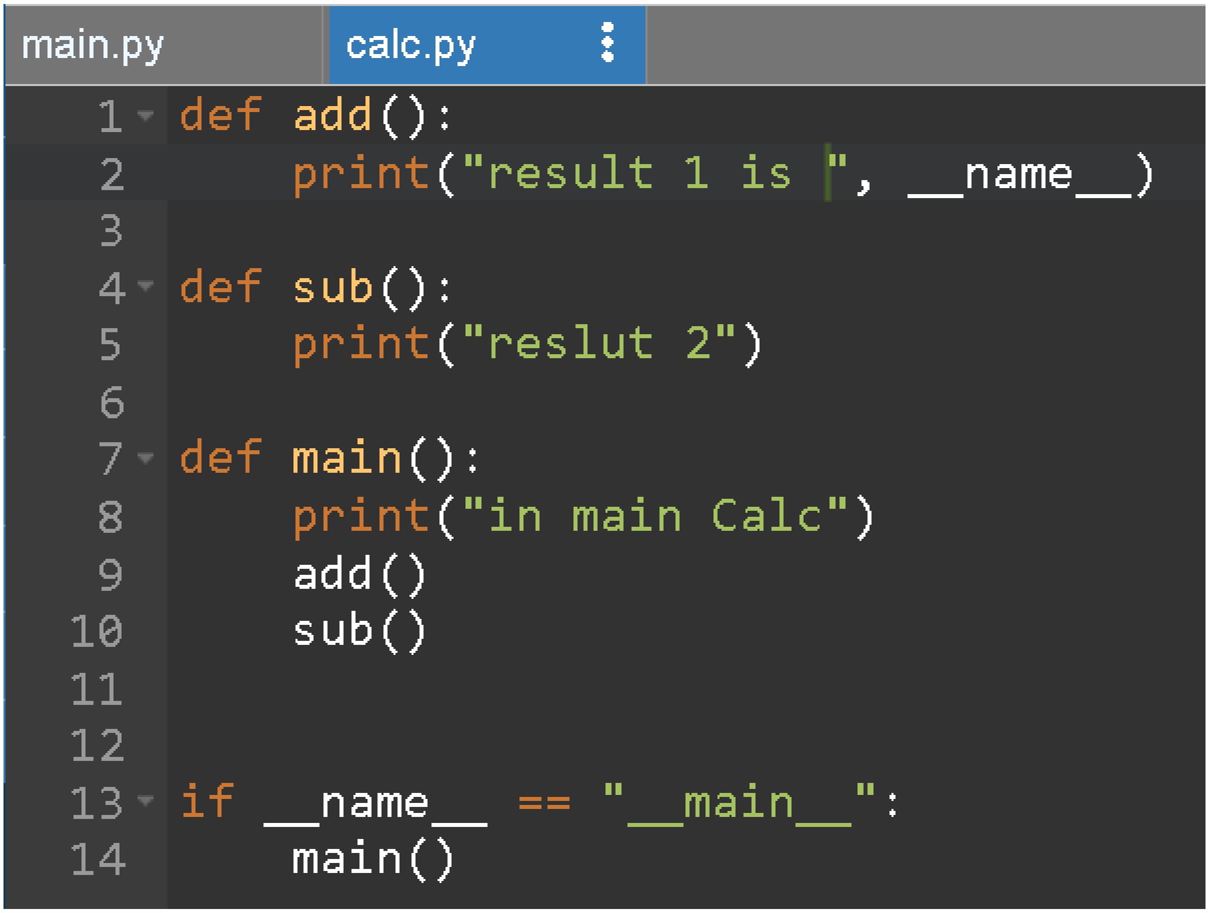
```
Demo Says calc
Hello __main__
```

main.py



```python
from calc import add

def fun1():
    add()
    print("from fun1")


def fun2():
    print("from fun2")

def main():
    fun1()
    fun2()

main()
```

calc.py



```python
def add():
    print("result 1 is ", __name__)

def sub():
    print("reslut 2")

def main():
    print("in main Calc")
    add()
    sub()


if __name__ == "__main__":
    main()
```

O/P:

```
result 1 is  calc
from fun1
from fun2
```

Object Oriented Programming

→Functions in object oriented programming is called Methods
Class →Design/Blueprint
Object →Instance

Class and Object

Class —> Attribute →Variables
—>Behaviour →Methods(Function)

Example:

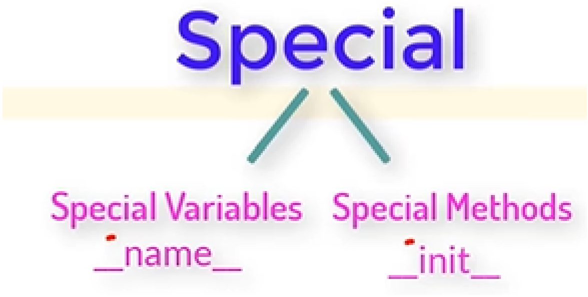
```
class Computer:

    def config(self):                    #Method
        print("i5, 16GB, 1TB")



com1 = Computer()        #Object
com2 = Computer()



com1.config()     #By using object calling the method
com2.config()

print()
#Other way of calling the method in class
Computer.config(com1)
Computer.config(com2)
```

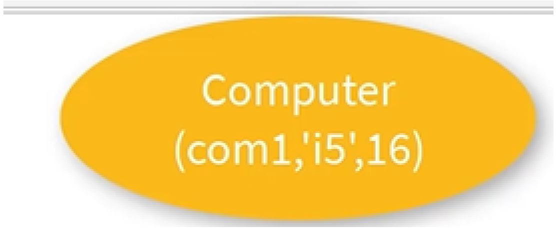
O/P:

```
i5, 16GB, 1TB
i5, 16GB, 1TB
```

i5, 16GB, 1TB
i5, 16GB, 1TB

__init__ method





com1 = Computer() →by default you are passing “com1”

```
class Computer:

   def __init__(self, cpu, ram):    #called automatically no need to call
explicitly for every object
       self.cpu = cpu
       self.ram = ram


   def config(self):                              #Method
       print("config is", self.cpu, self.ram)
```

```
com1 = Computer('i5', 16)        #Object
com2 = Computer('Ryzen', 8)


com1.config()     #By using object calling the method
com2.config()
```

O/P:

```
config is i5 16
config is Ryzen 8
```

Constructor, Self and Comparing

1. Every time you create an object it is allocated to new space

   Size of object ?
   →Depends on the no. of Variables and size of each variable.

   Who allocates the size to the object?
   →Constructor

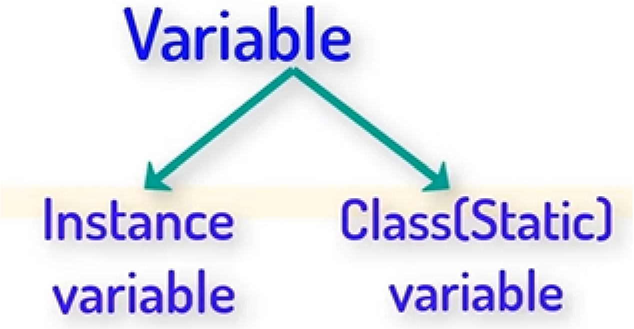
   ```
   class Computer:

       def __init__(self):
           self.name = "Raghava"
           self.age = 26

       def update(self):
           self.age = 30

       def compare(self, other):
           if self.age == other.age:
               return True
           else:
   ```

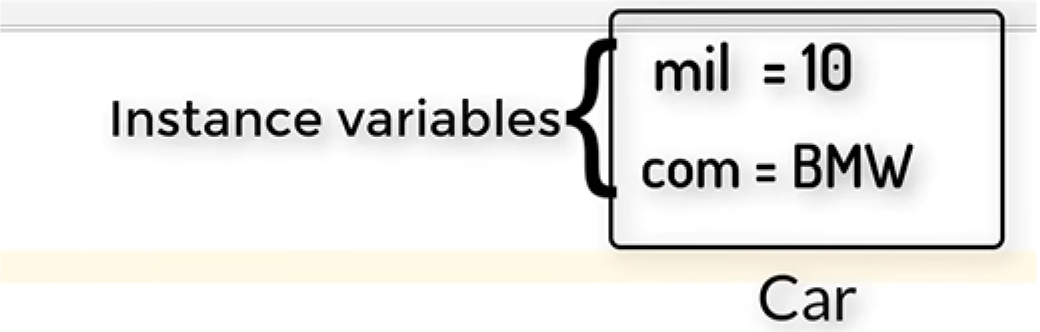
```
            return False

c1 = Computer()
c1.age = 24
c2 = Computer()

#c1.name = "karthik"
#c1.age = 36

if c1.compare(c2):
    print("they are same")
else:
    print("they are different")

c1.update()

print(c1.name)
print(c1.age)
print(c2.name)
print(c2.age)
```

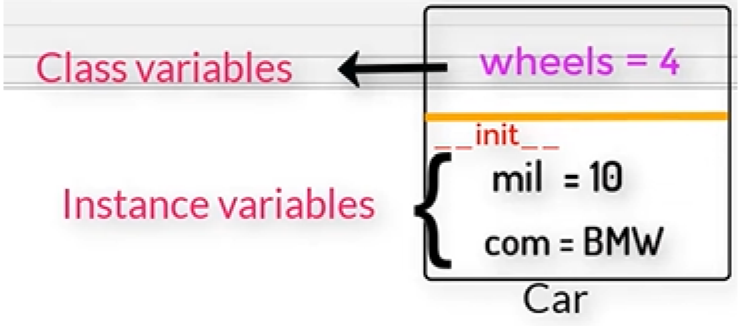
O/P:

```
they are different
Raghava
30
Raghava
26
```

Types of Variables

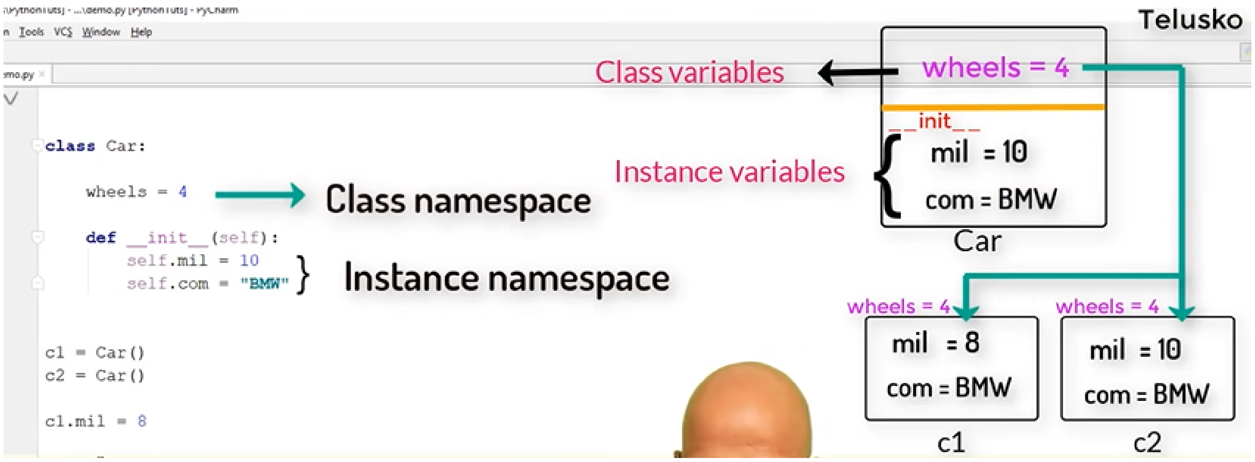






NameSpace is an area where you create and store objects/variables.

1. Class namespace
2. Object / Instance namespace



```
class Car:

    wheels = 4

    def __init__(self):
        self.mil = 10
        self.com = "BMW"

c1 = Car()
c2 = Car()

c1.mil = 8
```

```
class car:

    wheels = 4

    def __init__(self):
        self.mil = 10
        self.com = 'BMW'

c1 = car()
c2 = car()

c1.mil = 8

car.wheels = 5

print(c1.com, c1.mil, c1.wheels)
print(c2.com, c2.mil, c2.wheels)
```

O/P:

```
BMW 8 5
BMW 10 5
```

Types of Methods

Object → 1. Variables → To store data
→ 2. Methods → behaviors /to perform some operation

Three types of Methods

1. Instance methods
2. Class methods
3. Static methods

We are passing “self” that is why Instance the avg method we used here is the “Instance Method”

Accessor Method →read/access the value of variable →getters

Mutator Method   →modify/change the value of variable →setters

```
class Student:

    school = 'Telusko'                  #class variable

    #method
    def __init__(self, m1, m2, m3):
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3

    def avg(self):
        return (self.m1 + self.m2 + self.m3)/3

    def get_m1(self):                         #accessor
        return self.m1

    def set_m1(self, value):                  #mutator
        self.m1 = value
    @classmethod                         #decorators for class method
    def getSchool(cls):                  #Class Method
        return cls.school


    @staticmethod                      #decorators for static method
    def info():                        #static Method
        print("This is Student class.. in abc method")

#object

s1 = Student(34, 47, 32)
s2 = Student(89, 32, 12)

print(s1.avg())
print(s2.avg())
print(Student.getSchool())

Student.info()
```

O/P:

```
37.666666666666664
44.333333333333336
Telusko
This is Student class.. in abc method
```

INNER CLASS

1. You can create object of inner class inside the outer class
   (OR)
2. You can create object of inner class outside the outer class provided you use outer class name to call it.

1.

```
class Student:                          #Outer Class

    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        self.lap = self.Laptop()

    def show(self):
        print(self.name , self.rollno)

    class Laptop:                            #INNER class

        def __init__(self):
            self.brand = 'HP'
            self.cpu = 'i5'
            self.ram = 8

        def show(self):
            print(self.brand , self.cpu, self.ram)


s1 = Student("Raghava", 2)
s2 = Student("Gudiwada", 3)

print (s1.lap.brand)

lap1 = s1.lap
lap2 = s2.lap

print(id(lap1))
print(id(lap2))
```

O/P:

```
HP
140643286007376
140643286007184
```

2.

```
class Student:                                      #Outer Class

    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        #self.lap = self.Laptop()

    def show(self):
        print(self.name , self.rollno)
    class Laptop:                                  #INNER class

        def __init__(self):
            self.brand = 'HP'
            self.cpu = 'i5'
            self.ram = 8

        def show(self):
            print(self.brand , self.cpu, self.ram)


s1 = Student("Raghava", 2)
s2 = Student("Gudiwada", 3)

lap1 = Student.Laptop()

print(id(lap1))
```

O/P:

```
140311361715648
```

→

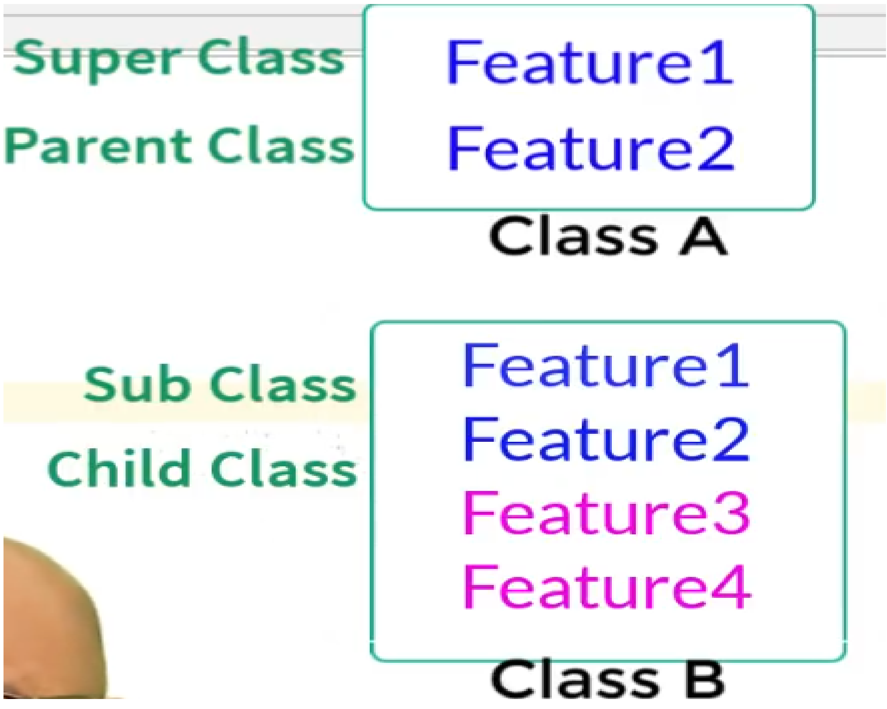
```
class Student:                                     #Outer Class

    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        self.lap = self.Laptop()

    def show(self):
        print(self.name , self.rollno)
        self.lap.show()

    class Laptop:                                   #INNER class
```

```
        def __init__(self):
            self.brand = 'HP'
            self.cpu = 'i5'
            self.ram = 8

        def show(self):
            print(self.brand , self.cpu, self.ram)


s1 = Student("Raghava", 2)
s2 = Student("Gudiwada", 3)

s1.show()
```

O/P:

```
Raghava 2
HP i5 8
```

INHERITANCE



Super Class

Parent Class

Feature1

Feature2

Class A

Sub Class

Child Class

Feature1

Feature2

Feature3

Feature4

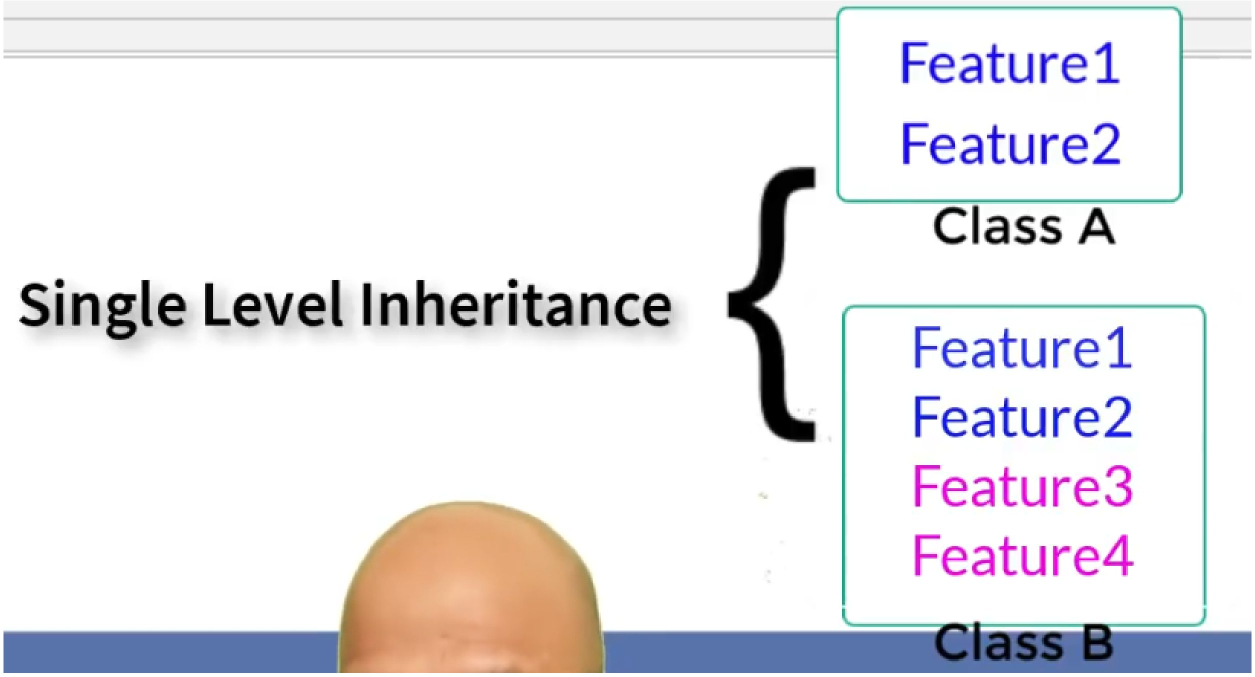
Class B

1. Single Inheritance

```
class A:
    def feature1(self):
        print("Feature 1 working")

    def feature2(self):
        print("Feature 2 working")

class B(A):                         #child/subclass
    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")

a1 = A()


a1.feature1()
a1.feature2()

print()
b1 = B()
b1.feature2()
```

O/P:

```
Feature 1 working
Feature 2 working

Feature 2 working
```



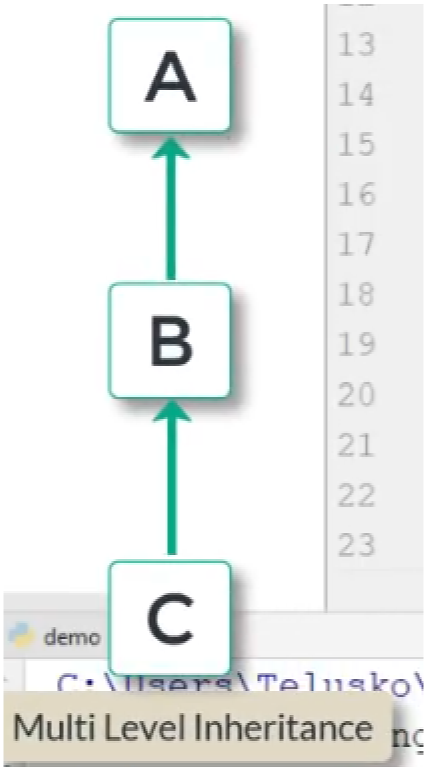
Single Level Inheritance

Feature1
Feature2
Class A

Feature1
Feature2
Feature3
Feature4
Class B

2. Multi level Inheritance

Child can Inherits all the features from parent & grandparent class as well.



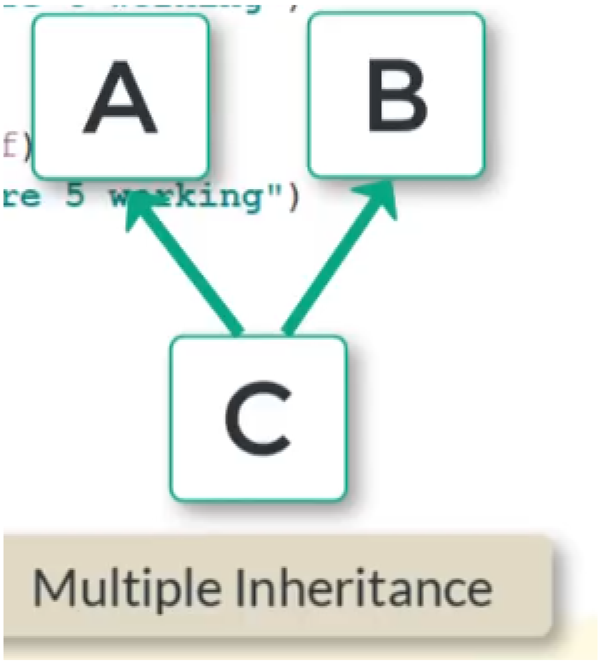
```
 class A:
  def feature1(self):
    print("Feature 1 working")

  def feature2(self):
    print("Feature 2 working")

class B(A):
  def feature3(self):
    print("Feature 3 working")

  def feature4(self):
    print("Feature 4 working")
```

```
class C(B):
    def feature5(self):
        print("Feature 5 working")

a1 = A()


a1.feature1()
a1.feature2()

print()
b1 = B()
b1.feature2()

print()

c1 = C()
c1.feature1()
c1.feature4()
```

O/P:

```
Feature 1 working
Feature 2 working

Feature 2 working

Feature 1 working
Feature 4 working
```

3. Multiple Inheritance



```
class A:
    def feature1(self):
        print("Feature 1 working")

    def feature2(self):
        print("Feature 2 working")

class B:
    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")

class C(A, B):
    def feature5(self):
        print("Feature 5 working")
```

```
a1 = A()

b1 = B()
b1.feature3()

print()

c1 = C()
c1.feature1()
c1.feature3()
```

O/P:

```
Feature 3 working

Feature 1 working
Feature 3 working
```

Constructor in Inheritance

Subclass can access all the features of superclass
But
Superclass can not access any features of subclass

If you create object of subclass it will first try find init of subclass if it is not found in subclass then it will call “init” of superclass.

```
class A:
    def __init__(self):
        print("in A __init__")

    def feature1(self):
        print("Feature 1 working")
    def feature2(self):
        print("Feature 2 working")

class B(A):

    def __init__(self):
        print("in B __init__")

    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")


a1 = B()
```

O/P:

```
in B __init__
```

When you create object of subclass it will call “init” of subclass first
If you have call super then it will first call init of superclass then call init of subclass

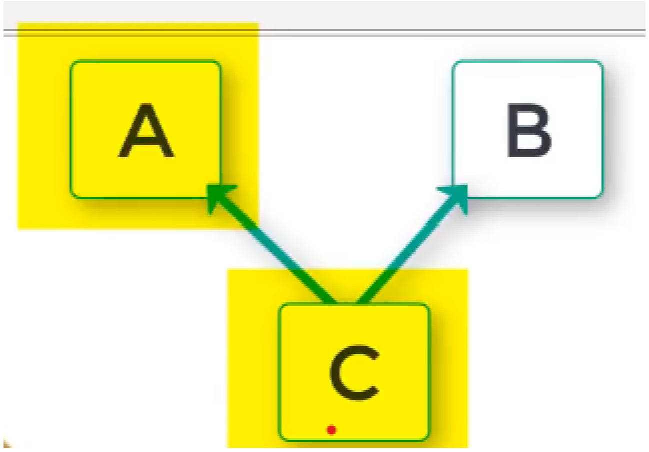
```
class A:
    def __init__(self):
        print("in A __init__")

    def feature1(self):
        print("Feature 1 working")

    def feature2(self):
        print("Feature 2 working")

class B(A):

    def __init__(self):
        super().__init__()
        print("in B __init__")

    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")


a1 = B()
```

O/P:

```
in A __init__
in B __init__
```

Method Resolution Order (MRO)

→Always start from left to right



```
class A:
    def __init__(self):
        print("in A __init__")

    def feature1(self):
        print("Feature 1 working")

    def feature2(self):
        print("Feature 2 working")

class B:

    def __init__(self):
        print("in B __init__")

    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")

class C(A,B):

    def __init__(self):
        super().__init__()
        print("in C init")

a1 = C()
```

O/P:

```
in A __init__
in C init
```

→The above concept is the same for the method also.

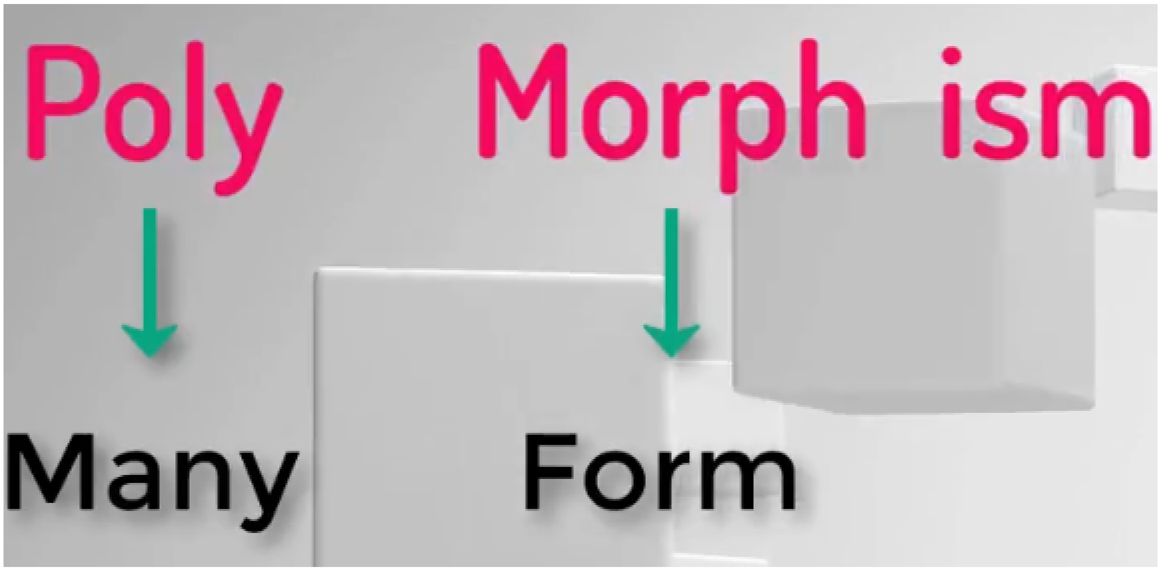
```
class A:
    def __init__(self):
        print("in A __init__")

    def feature1(self):
        print("Feature 1 working")

    def feature2(self):
        print("Feature 2 working")

class B:

    def __init__(self):
        print("in B __init__")

    def feature3(self):
        print("Feature 3 working")

    def feature4(self):
        print("Feature 4 working")

class C(A,B):

    def __init__(self):
        super().__init__()
        print("in C init")

    def feat(self):
        super().feature2()

a1 = C()
a1.feat()
```

O/P:

```
in A __init__
in C init
Feature 2 working
```

→To represent superclass we use super method

Introduction to Polymorphism



Polymorphism means one thing with many forms.

Implementation of polymorphism in 4 ways

1. **Duck Typing (Only in python)**
2. **Operator Overloading**
3. **Method Overloading**
4. **Method overriding**

Duck Typing:

```
class PyCharm:

    def execute(self):
        print("Compiling")
        print("Running")

class MyEditor:

    def execute(self):
        print("Spell Check")
        print("Convention Check")
        print("Compiling")
        print("Running")


class Laptop:

    def code(self, ide):
        ide.execute()

ide = MyEditor()

lap1 = Laptop()
lap1.code(ide)
```

O/P:

```
Spell Check
Convention Check
Compiling
Running
```

Operator Overloading:

```
__add__()
```

```
__sub__()
```

```
__mul__()
```

—>**Magic Methods**

```
a = 5
b = 6

print(a + b)

print(int.__add__(a, b))
```

O/P:

```
11
11
```

→

```
class Student:

    def __init__(self, m1, m2):
        self.m1 = m1
        self.m2 = m2

    def __add__(self, other):
        m1 = self.m1 + other.m1
        m2 = self.m2 + other.m2
        s3 = Student(m1, m2)
        return s3

    def __gt__(self, other):
        r1 = self.m1 + self.m2
        r2 = other.m1 + other.m2
        if r1 > r2:
            return True
        else:
            return False

    def __str__(self):
        #return self.m1, self.m2
        return '{} {} '.format(self.m1, self.m2)


s1 = Student(58, 69)
s2 = Student(60, 65)

s3 = s1 + s2

print(s3.m1)
print(s3.m2)
print(s3.__str__())

#print(s1)      #printing address

print(s1.__str__())

if s1 > s2:
    print("s1 wins")
else:
    print("s2 wins")
```

O/P:

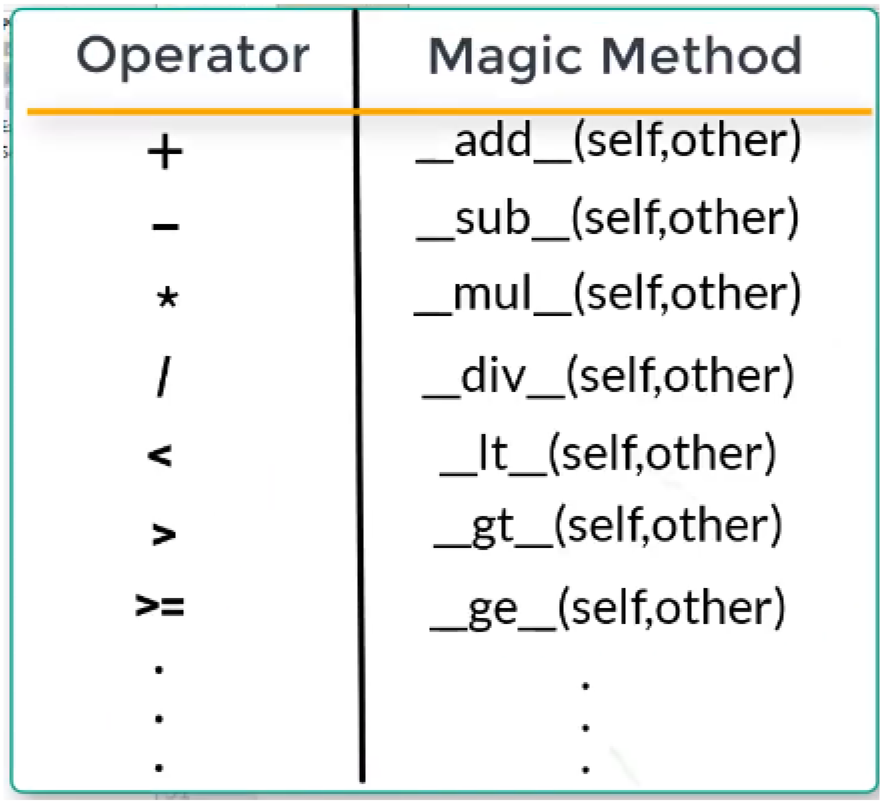
```
118 134
58 69
s1 wins
```

→Static Methods:

```
print(dir(int))
```

O/P:

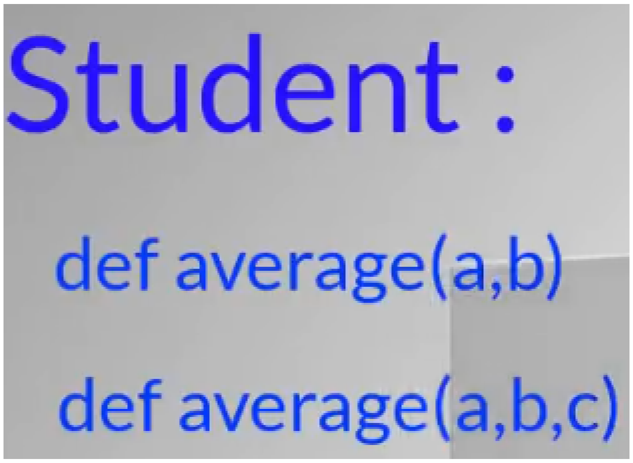
```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__',
'__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__',
'__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__',
'__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
'as_integer_ratio', 'bit_count', 'bit_length', 'conjugate', 'denominator', 'from_bytes',
'imag', 'numerator', 'real', 'to_bytes']
```



Operator	Magic Method
+	__add__(self,other)
–	__sub__(self,other)
*	__mul__(self,other)
/	__div__(self,other)
<	__lt__(self,other)
>	__gt__(self,other)
>=	__ge__(self,other)
.	.
.	.
.	.

Method Overloading and Method Overriding

Method overloading:



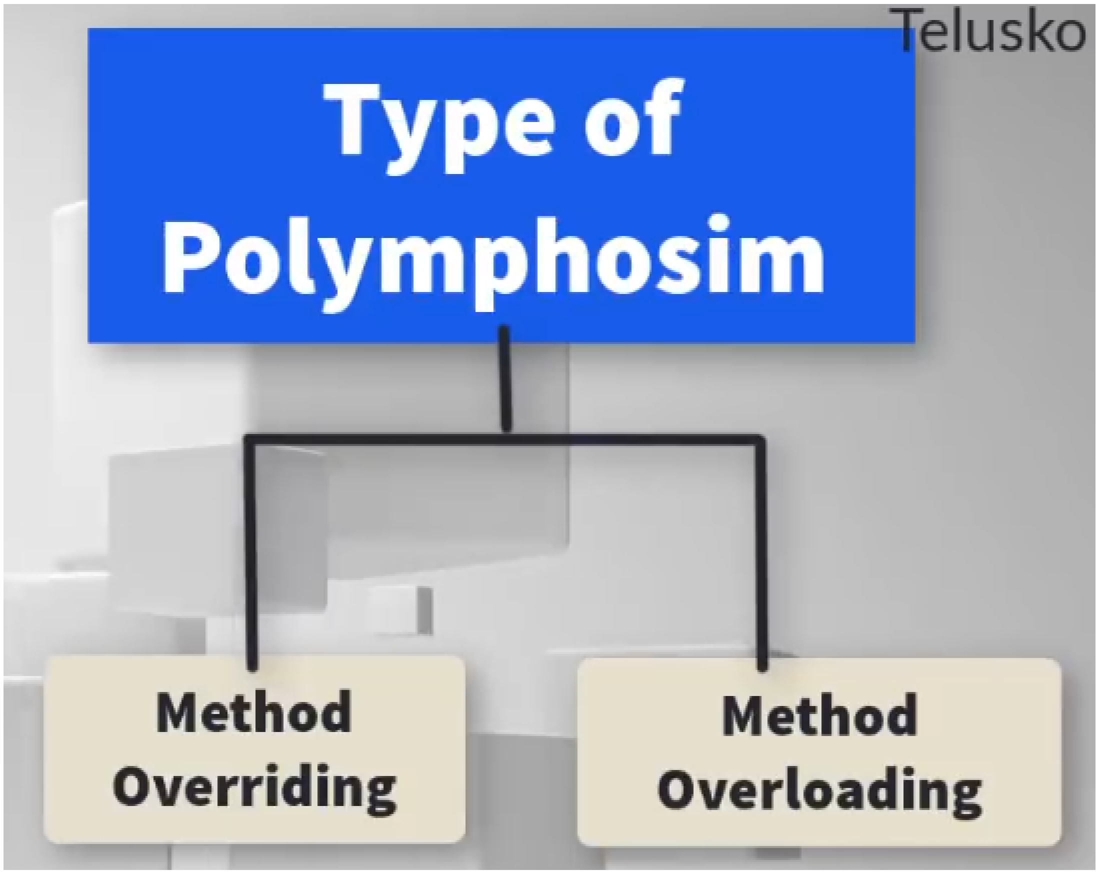
```
Student :
  def average(a,b)
  def average(a,b,c)
```

The concept is not there in python

```
class Student:

    def __init__(self, m1, m2):
        self.m1 = m1
        self.m2 = m2

    def sum(self, a=None, b=None, c=None):

        s = 0

        if a!=None and b!=None and c!=None:
            s = a+b+c
        elif a!=None and b!=None:
            s = a+b
        else:
            s = a

        return s

s1 = Student(58, 69)

print(s1.sum(5))
```

O/P:

```
5
```



Type of Polymphosim

Method Overriding

Method Overloading

Method overriding:

```
class A:

    def show(self):
        print("in A Show")


class B(A):

    def show(self):
        print("in B Show")


a1 = B()
```

```
a1.show()
```

O/P:

```
In B Show
```

→show method of class A is overridden by the show method of class B.

Iterator

```
nums = [7, 8, 9, 5]

for i in nums:
    print(i)

it = iter(nums)  #convert list to iterator

print()

print(it.__next__())

print(next(it))
```

O/P:

```
7
8
9
5

7
8
```

```
class TopTen:

    def __init__(self):
        self.num = 1

    def __iter__(self):
        return self

    def __next__(self):

        if self.num <= 10:
            val = self.num
            self.num += 1

            return val
        else:
            raise StopIteration

values = TopTen()

print(values.__next__())

for i in values:
   print(i)
```

→Because using the iterator once it is printed by using the next it cannot be repeated by using the for loop.

O/P:

```
1
2
3
4
5
6
```

```
7
8
9
10
```

Generators

→Yield is the same as return but return will terminate the function.

```
def topten():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5


values = topten()

print(values.__next__())

for i in values:
    print(i)
```

O/P:

```
1
2
3
4
5
```

→

```
def topten():

    n = 1

    while n <= 10:
        sq = n*n
        yield sq
        n += 1

values = topten()

for i in values:
    print(i)
```

O/P:

```
1
4
9
16
25
36
49
64
81
100
```

Exception handling

Errors → 1.Compile time error
2. Logical
3. Run time

Compile time error → Syntactical Errors (easy to find)
E.g.
Missing (:)
Wrong Spelling

Logical Error → e.g. (software Bug)
Wrong output
Like 2 +3 = 4

Run time Error → e.g.
Divide by Zero

→

```
a = 5
b = 2

try:
   print("resource open")
   print(a/b)
   k = int(input("Enter a number"))
   print(k)

except ZeroDivisionError as e:
   print("Hey, You cannot divide a Number by Zero", e)

except ValueError as e:
    print("Invalid Input")

except Exception as e:
    print("Something went wrong....!")


finally:
    print("resource closed")

print("Bye")
```

O/P:

```
resource open
2.5
Enter a numberp
Invalid Input
resource closed
Bye
```

→

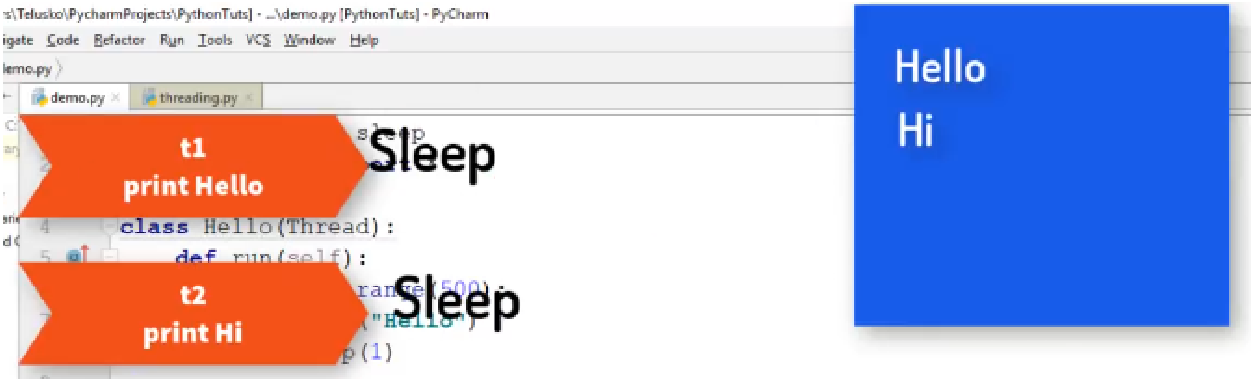
```
a = 5
b = 0

try:
    print("resource open")
    print(a/b)
    k = int(input("Enter a number"))
    print(k)

except ZeroDivisionError as e:
    print("Hey, You cannot divide a Number by Zero", e)

except ValueError as e:
    print("Invalid Input")
```

```
except Exception as e:
    print("Something went wrong....!")


finally:
    print("resource closed")

print("Bye")
```

O/P:

```
resource open
Hey, You cannot divide a Number by Zero division by zero
resource closed
Bye
```

MultiThreading



s\Telusko\PycharmProjects\PythonTuts] - ...\demo.py [PythonTuts] - PyCharm
igate Code Refactor Run Tools VCS Window Help
lemo.py
demo.py
threading.py
t1
print Hello
Sleep
t2
print Hi
Sleep
Hello
Hi

```
class Hello(Thread):
    def run(self):
```

```
from threading import *
from time import sleep

class Hello(Thread):

    def run(self):
        for i in range(7):
            print("Hello")
            sleep(1)

class Hi(Thread):

    def run(self):
        for i in range(7):
            print("Hi")
            sleep(1)
t1 = Hello()
t2 = Hi()

t1.start()
sleep(0.2)
t2.start()

t1.join()
t2.join()
print("Bye")
```

O/P:

```
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Bye
```

File Handling

```
f = open('MyData', 'r')    #read

print(f)     #print entire data

print(f.readline())    #read line by line
print(f.readline(), end="#")


f1 = open('abc', 'w')     #write

f1.write("Something")

f1.write("People")

f1.write("Laptop")

f1 = open('abc', 'a')      #append

f1.write("mobile")

for data in f:   #from MyData it copied to abc file
    f1.write(data)


f2 = open('IMG_6309.JPG', 'rb')   #read binary

f3 = open('MyPic.JPG', 'wb')


for i in f2:     #copy image to Mypic.JPG
    f3.write(i)
```

Comments

In Python have single line comments →# →parser will ignore this line

Don’t do multi line comments →parser will not ignore

Multiline comments used for documentation
Ex:
““““““
This is a comments
Using the
Multiline comments

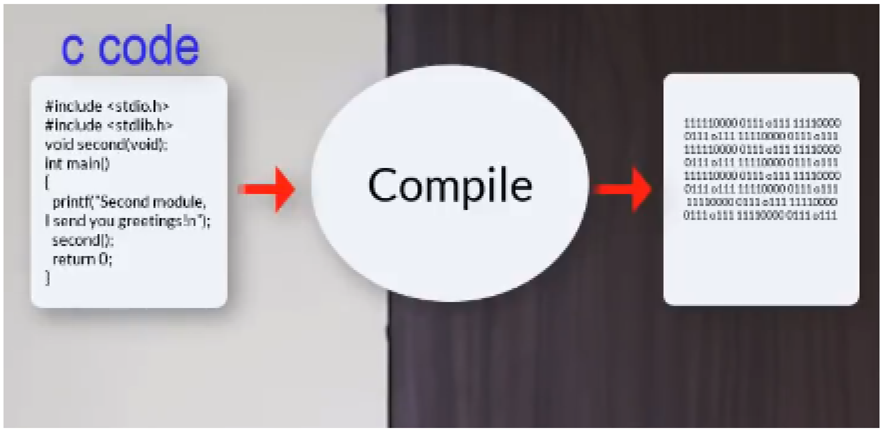
””””””

Is Python Compiled or Interpreted Language?

Compilers can convert from any language to any other language.

C, C++ → Compiler (High level language)

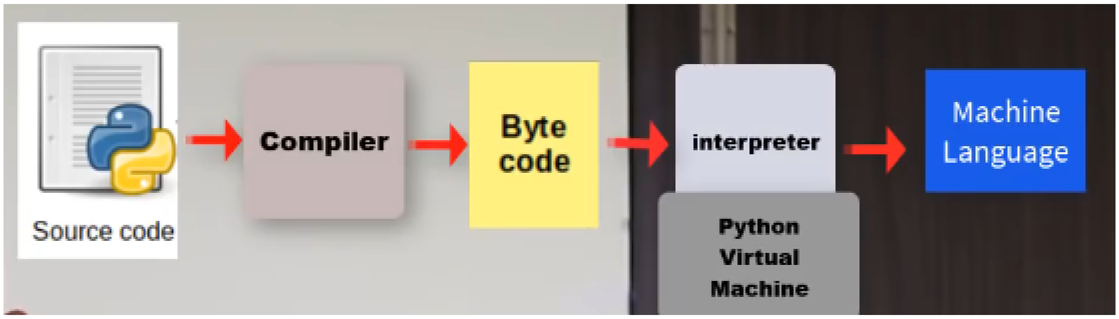
C- code —> compile → machine language



```
#include <stdio.h>
#include <stdlib.h>
void second(void);
int main()
{
  printf("Second module,
I send you greetings!\n");
  second();
  return 0;
}
```

```
11110000 0111 o111 11110000
0111 o111 11110000 0111 o111
11110000 0111 o111 11110000
0111 o111 11110000 0111 o111
11110000 0111 o111 11110000
0111 o111 11110000 0111 o111
11110000 0111 o111 11110000
0111 o111 11110000 0111 o111
```

Python



Interpreter → line by line execution

Byte code for Portability

Python has different implementations

Cpython → Implementation is done with C language (widely used)
JYpython → JAva
IRONpython → Dot Net

Python is both compiled & Interpreted language

Linear Search Using Python

→ Using For loop

```
pos = -1       #global

def search(list, ele):
    for i in range(len(list)):
        if list[i] == ele:
            globals()['pos'] = i
            return True

    return False



list = [5, 8, 4, 6, 9, 2]
ele = 4

if search(list, ele):
    print("Found at", pos+1)
else:
    print("Not Found")
```

O/P:

```
Found at 3
```

→Using While loop

```
pos = -1       #global

def search(list, ele):
    i = 0

    while i < len(list):
        if list[i] == ele:
            globals()['pos'] = i
            return True
        i = i +1

    return False



list = [5, 8, 4, 6, 9, 2]
ele = 4

if search(list, ele):
    print("Found at", pos+1)
else:
    print("Not Found")
```

O/P:

```
Found at 3
```

Binary Search Using Python

```
pos = -1       #global

def search(list, ele):
    l = 0
    u = len(list)-1

    while l <= u:
        mid = (l+u) // 2

        if list[mid] == ele:
            globals()['pos'] = mid
            return True
        elif list[mid] < ele:
            l = mid + 1
        else:
            u = mid - 1

    return False

list = [4, 7, 8, 12, 45, 99, 102, 702, 10987, 56666]
ele = 702

if search(list, ele):
    print("Found at", pos+1)
else:
    print("Not Found")
```

O/P:

```
Found at 8
```

Bubble sort using Python

```
def sort(nums):
    for i in range(len(nums)-1, 0, -1):  #5, 0 everytime decrement -1
        for j in range(i):
            if nums[j] > nums[j+1]:
                temp = nums[j]
                nums[j] = nums[j+1]
                nums[j+1] = temp


nums = [5, 3, 8, 6, 7, 2]
sort(nums)


print(nums)
```

O/P:

```
[2, 3, 5, 6, 7, 8]
```

Selection sort using Python

```
def sort(nums):
    for i in range(len(nums)-1):
        minpos = i
        for j in range(i+1, len(nums)):
            if nums[j] < nums[minpos]:
                minpos = j

        temp = nums[i]
        nums[i] = nums[minpos]
        nums[minpos] = temp

        print(nums)



nums = [5, 3, 8, 6, 7, 2]
sort(nums)



print(nums)
```

O/P:

```
[2, 3, 8, 6, 7, 5]
[2, 3, 8, 6, 7, 5]
[2, 3, 5, 6, 7, 8]
[2, 3, 5, 6, 7, 8]
[2, 3, 5, 6, 7, 8]
[2, 3, 5, 6, 7, 8]
```

Abstract Class and Abstract Method in Python

Abstract Method:

This method has only declaration but not definition in

A class which have abstract methods are called abstract classes.

We can't create object for abstract class.

A class can have multiple/atleast one abstract Methods it also include normal methods.

```
from abc import ABC, abstractmethod  #abc -->abstarct base class


class Computer(ABC):
    @abstractmethod
    def process(self):
        pass

class Laptop(Computer):
    def process(self):
        print("its running")


class Programmer:
    def work(self, com):
        print("Solving Bugs")
        com.process()

#com = Computer()
#com.process()
com1 = Laptop()
```

```
prog1 = Programmer()

prog1.work(com1)
```

O/P:

```
Solving Bugs
its running
```

Zip function in Python

→List

```
names = ("Raghava", "Kiran", "Harsh", "Navin")
comps = ("Dell", "Apple", "MS", "Dell")

zipped = list(zip(names, comps))
print(zipped)
```

O/P:

```
[('Raghava', 'Dell'), ('Kiran', 'Apple'), ('Harsh', 'MS'), ('Raghava', 'Dell')]
```

→Set

```
names = ("Raghava", "Kiran", "Harsh", "Raghava")
comps = ("Dell", "Apple", "MS", "Dell")

zipped = set(zip(names, comps))
print(zipped)
```

O/P:

```
{('Raghava', 'Dell'), ('Kiran', 'Apple'), ('Harsh', 'MS')}
```

→Dict

```
names = ("Raghava", "Kiran", "Harsh", "Raghava")
comps = ("Dell", "Apple", "MS", "Dell")

zipped = dict(zip(names, comps))
print(zipped)
```

O/P:

```
{'Raghava': 'Dell', 'Kiran': 'Apple', 'Harsh': 'MS'}
```

→For loop

```
names = ("Raghava", "Kiran", "Harsh", "Raghava")
comps = ("Dell", "Apple", "MS", "Dell")

zipped = zip(names, comps)

for (a,b) in zipped:
    print(a, b)
```

O/P:

```
Raghava Dell
Kiran Apple
Harsh MS
Raghava Dell
```

Socket Programming Using Python

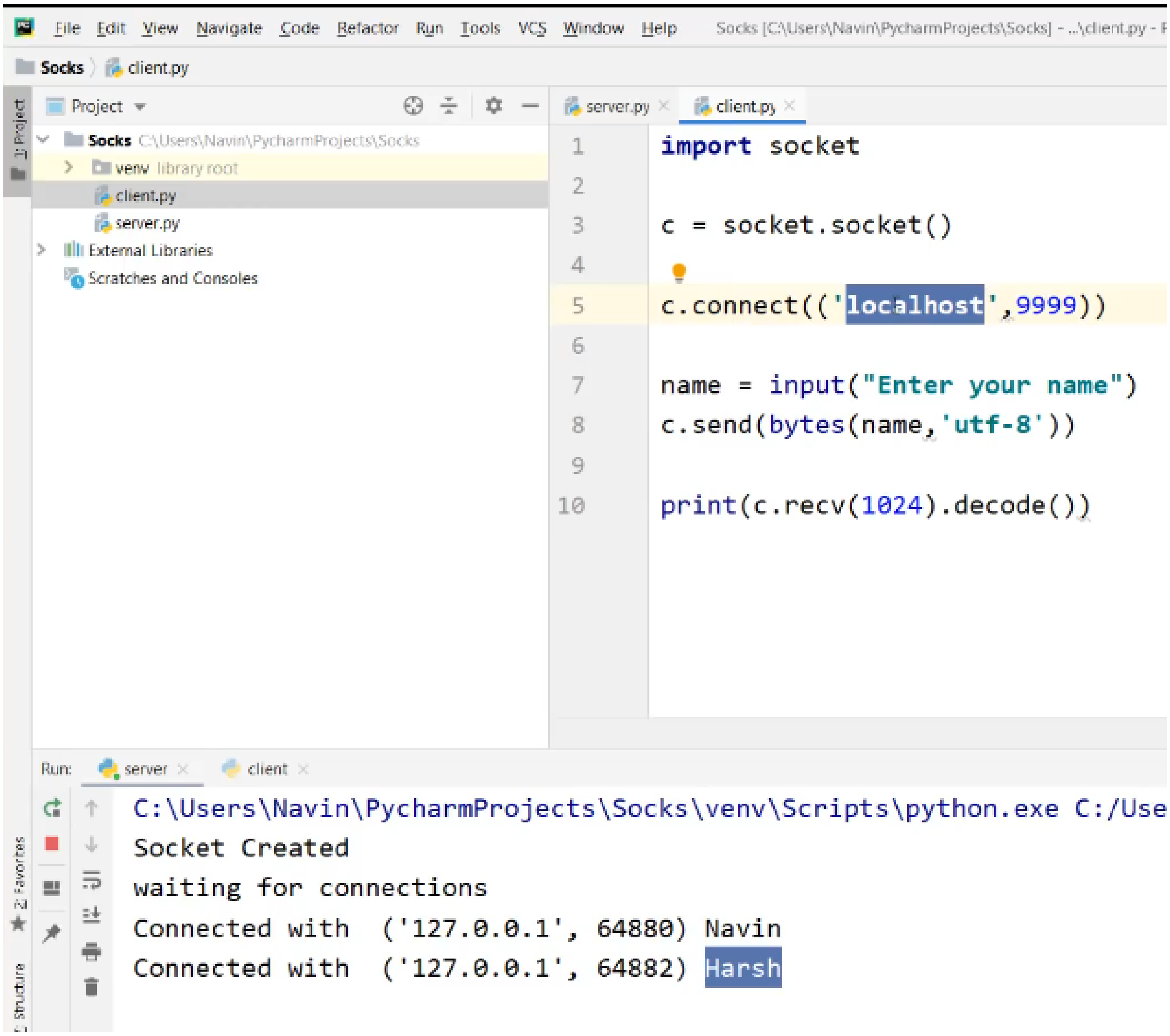
Server.py

```
import socket

s = socket.socket()   #Ipv4/Ipv6-->typeof ip, TCP?UDP

print("Socket Created")

s.bind(('localhost', 9999))

s.listen(3)

print("waiting for connections")

while True:
    c, addr = s.accept()
    name = c.recv(1024).decode()
    print("Connected with ", addr, name)


    c.send(bytes("Welcome to Telusko", 'utf-8'))

    c.close()
```

Client.py

```
import socket

c = socket.socket()   #Ipv4/Ipv6-->typeof ip, TCP?UDP
```

```
c.connect(('localhost', 9999))

name = input("Enter your name")
c.send(bytes(name, 'utf-8'))

print(c.recv(1024).decode())
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Socks [C:\Users\Navin\PycharmProjects\Socks] - ...\client.py - F
Socks
client.py
1: Project
Project
Socks C:\Users\Navin\PycharmProjects\Socks
venv library root
client.py
server.py
External Libraries
Scratches and Consoles
server.py
client.py
Run:
server
client
2: Favorites
Z: Structure

```
import socket

c = socket.socket()

c.connect(('localhost',9999))

name = input("Enter your name")
c.send(bytes(name,'utf-8'))

print(c.recv(1024).decode())
```

```
C:\Users\Navin\PycharmProjects\Socks\venv\Scripts\python.exe C:/Use
Socket Created
waiting for connections
Connected with  ('127.0.0.1', 64880) Navin
Connected with  ('127.0.0.1', 64882) Harsh
```

Sending Email using Python in 5 statements

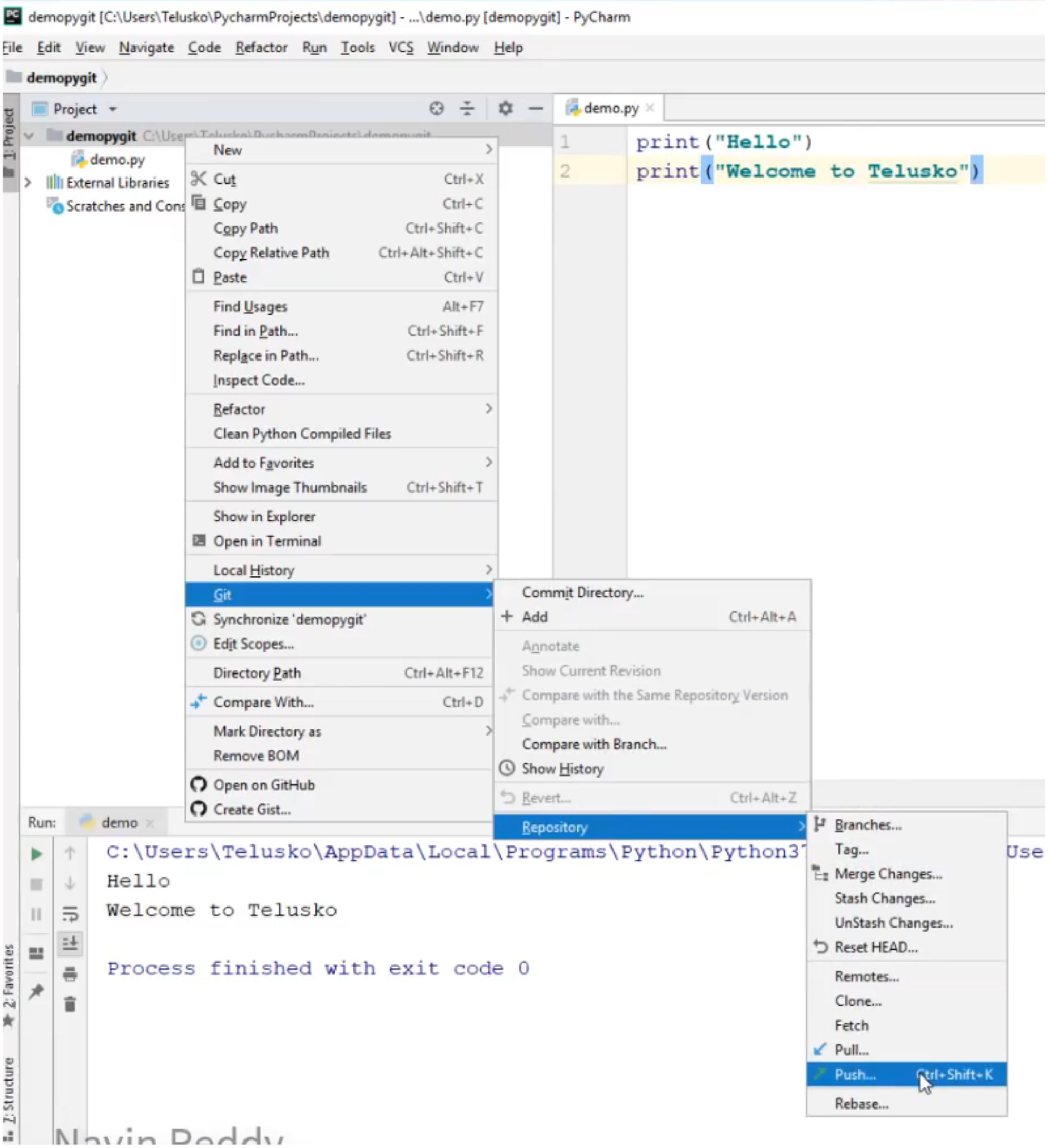
→Check once more

```
import smtplib

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

server.starttls()

server.login('gudiwadaraghava999@gmail.com', ‘$$$$$')

server.sendmail('gudiwadaraghava999@gmail.com',
'raghava.212is009@nitk.edu.in', 'Mail sent from python code')

print('Mail sent')
```

Sql installer
Password: 1234
User : Raghava
Password: 1234

GitHub & Pycharm

→pycharm to github



```
print ("Hello")
print("Welcome to Telusko")
```

```
C:\Users\Telusko\AppData\Local\Programs\Python\Python3                    Use
Hello
Welcome to Telusko

Process finished with exit code 0
```

1. Commit Directory
2. Push message send to github /Pull message is received from github to pycharm

SQL Connector installer

→ Already in my laptop
→Go to command prompt →pip3 install mysql-connector

```
import mysql.connector

mydb = mysql.connector.connect(host="localhost", user ="navin", passwd
="1234", database="telusko")


mycursor = mydb.coursor()

#mycursor.execute("show databases")

mycursor.execute("select * from student")

result = mycursor.fetchall()

for i in result:
    print(i)
```