|  |
| --- |
| Ijara Management Company |
| Python for beginners |
| Learning python |

|  |
| --- |
| Noman ul Haq  4/21/2019 |

Table of Contents

[Lesson # 20: Break, continue, pass 4](#_Toc7355347)

[Lesson # 22: Array 8](#_Toc7355348)

[Lesson # 23: Installation of Numpy 17](#_Toc7355349)

[Lesson # 24: ways of creating array in Numpy 17](#_Toc7355350)

[linespace 18](#_Toc7355351)

[Arrange 18](#_Toc7355352)

[Logspace 19](#_Toc7355353)

[ZEROS 19](#_Toc7355354)

[Ones 19](#_Toc7355355)

[Lesson # 26: Copying an Array in python 20](#_Toc7355356)

[Vactorized operation 20](#_Toc7355357)

[Array addition 20](#_Toc7355358)

[Aliasing 21](#_Toc7355359)

[Shallow copy 21](#_Toc7355360)

[Deep Copy 22](#_Toc7355361)

[Working with Matrix in Python 23](#_Toc7355362)

[Matrices 23](#_Toc7355363)

[Diagnol matrix 24](#_Toc7355364)

[Addition and multiplication of two matrix 24](#_Toc7355365)

[Lesson # 28 Functions in Python 25](#_Toc7355366)

[What is function? 25](#_Toc7355367)

[Why Function? 25](#_Toc7355368)

[Creating and calling function 25](#_Toc7355369)

[Lesson # 29 Function Arguments in Python 25](#_Toc7355370)

[Function argument 25](#_Toc7355371)

[Mutable and immutable argument 26](#_Toc7355372)

[Lesson # 30 Types of Arguments in Python 26](#_Toc7355373)

[Formal argument 26](#_Toc7355374)

[Actual argument 26](#_Toc7355375)

[Position 26](#_Toc7355376)

[Keyword 26](#_Toc7355377)

[Default 26](#_Toc7355378)

[Variable length 27](#_Toc7355379)

[Lesson # 31 Keyworded Variable Length Arguments in Python | \*\*kwargs 28](#_Toc7355380)

[Lesson # 31 Global Keyword in Python | Global vs Local Variable 28](#_Toc7355381)

[Lesson # 32 Pass List to a Function in Python 29](#_Toc7355382)

[Pass list to a function Count number of even and odd number in a list 29](#_Toc7355383)

[Fibonacci Sequence in Python 30](#_Toc7355384)

[#34 Python Tutorial for Beginners | Factorial in Python 30](#_Toc7355385)

[#35 Python Tutorial for Beginners | Recursion in Python 30](#_Toc7355386)

[#36 Python Tutorial for Beginners | Factorial using Recursion in Python 31](#_Toc7355387)

[#37 Python Tutorial for Beginners | Anonymous Functions in Python | Lambda 31](#_Toc7355388)

[#38 Python Tutorial for Beginners | Anonymous Functions in Python | Lambda 31](#_Toc7355389)

[Filter 31](#_Toc7355390)

[map 31](#_Toc7355391)

[reduce 32](#_Toc7355392)

[Decorators in Python 33](#_Toc7355393)

[#39 Python Tutorial for Beginners | Modules in Python 33](#_Toc7355394)

[#40 Python Tutorial for Beginners | Special Variable \_\_name\_\_ in Python 33](#_Toc7355395)

[#41 Python Tutorial for Beginners | Special Variable \_\_name\_\_ part 2 in Python 33](#_Toc7355396)

[#42 Python Tutorial for Beginners | Object Oriented Programming in Python | Introduction 34](#_Toc7355397)

[#44 Python Tutorial for Beginners | \_\_init\_\_ method in Python 35](#_Toc7355398)

[#45 Python Tutorial for Beginners | Constructor, Self and Comparing Objects in Python 36](#_Toc7355399)

[#46 Python Tutorial for Beginners | Types of Variables in Python 37](#_Toc7355400)

[Instance variable 37](#_Toc7355401)

[Class variable 37](#_Toc7355402)

[#47 Python Tutorial for Beginners | Types of Methods in Python 38](#_Toc7355403)

[Instance method 38](#_Toc7355404)

[accessor /mutator method 39](#_Toc7355405)

[Class method 39](#_Toc7355406)

[Static method 40](#_Toc7355407)

[#48 Python Tutorial for Beginners | Inner class in Python 40](#_Toc7355408)

[#49 Python Tutorial for Beginners | Inheritance in Python 42](#_Toc7355409)

[Child Class / super class, sub class / single level inheritance 42](#_Toc7355410)

[Multilevel inheritance 43](#_Toc7355411)

[Multiple inheritance 44](#_Toc7355412)

[#50 Python Tutorial for Beginners | Constructor in Inheritance Python 45](#_Toc7355413)

[Method resolution order 45](#_Toc7355414)

[#51 Python Tutorial for Beginners | Introduction to Polymorphism in Python | Part 1 47](#_Toc7355415)

[#52 Python Tutorial for Beginners | Duck Typing in Python 47](#_Toc7355416)

[#53 Python Tutorial for Beginners | Operator Overloading in Python | Polymorphism 48](#_Toc7355417)

[Operator overloading 48](#_Toc7355418)

[#54 Python Tutorial for Beginners | Method Overloading and Method Overriding in Python 50](#_Toc7355419)

[Method overloading 50](#_Toc7355420)

[Method Overriding 50](#_Toc7355421)

[Iterator in Python 51](#_Toc7355422)

[Generators in Python 51](#_Toc7355423)

[#55 Python Tutorial for Beginners | Exception Handling in Python 51](#_Toc7355424)

## Lesson # 20: Break, continue, pass

av= 10  
x= int(input(**"How many candies you want"**))  
  
i = 1  
**while** i <= x:  
 **if** i>av:  
 print(**"Exceed candy available quantity"**)  
 **break** print(**"Candy"**, i)  
 i+=1  
  
print(**"Bye"**)

**for** i **in** range(1,101):  
 **if** i%3==0 **or** i%5==0:  
 **continue** print(i)

print(**"bye"**)

**for** i **in** range(1,101):  
 **if** i%3==0 **and** i%5==0:  
 **continue** print(i)  
print(**"bye"**)

**for** i **in** range(1,101):  
 **if** i%2!=0 :  
 **pass  
 else**:  
 print(i)  
  
print(**"bye"**)

**Difference between continue and break**

**for** a **in** range(0,5):  
 **if** a==3:  
 **continue** print(a,**' hello'**)

**0 hello  
1 hello  
2 hello  
4 hello**

**for** a **in** range(0,5):  
 **if** a==3:  
 **break** print(a,**' hello'**)

**0 hello  
1 hello  
2 hello**

**for** a **in** range(0,10):  
 **if** a **in** (3,6,5,0):  
 **continue  
 elif** a==5:  
 **continue** print(a,**' hello'**)

**--------------------------------------------------------------------------**

**Lesson # 21: Printing pattern**

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

**# # # #**

**# # # #**

**# # # #**

**# # # #**

**for** j **in** range(1,5,1):  
 **for** j **in** range(j):  
 print(**"# "**,end=**""**)  
 print()

**#   
# #   
# # #   
# # # #**

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

**# # # #   
# # #   
# #   
#**

**for** j **in** range(1,5,1):  
 **for** k **in** range(1,6-j,1):  
 print(k,end=**" "**)  
 print()

**1 2 3 4   
1 2 3   
1 2   
1**

a = **'A P Q E'**b = **'A B Q R'**c = **'A B C R'**d = **'A B C D'  
for** j **in** range(4):  
 **if** j == 0:  
 print(a)  
 **elif** j ==1:  
 print(b)  
 **elif** j ==2:  
 print(c)  
 **elif** j ==3:  
 print(d)

**A P Q E  
A B Q R  
A B C R  
A B C D**

**Lesson # 21: For else**

x = [10,23,45,6,34,87,20]  
**for** i **in** x:  
 **if** i % 5 == 0:  
 print(i)

**10**

**45**

**20**

x = [10,23,45,6,34,87,20]  
**for** i **in** x:  
 **if** i % 5 == 0:  
 print(i)  
 **break**

x = [9,23,42,6,34,87,22]  
**for** i **in** x:  
 **if** i % 5 == 0:  
 print(i)  
  
 **else**:  
 print(**"not found"**)

**not found**

**not found**

**not found**

**not found**

**not found**

**not found**

**not found**

x = [9,23,42,6,34,87,22]  
**for** i **in** x:  
 **if** i % 5 == 0:  
 print(i)  
  
**else**:  
 print(**"not found"**)

**not found**

## Lesson # 22: Array

We have list, tuple, set available with us but why an array. In array we need to have all the values in same type so we can have same list with integer, float, strings in one list. For example if we have int array so it should be int array, if float array then it should be float array.

Array in python, they don’t have specific or fix size means you can expand and shrink it, it is quite flexible to work with. You can use function like append, get index value of any particular value.

When you are using array, lets example if we have list of students and every student have marks so lets say we have one subject like python it is a language and you have subject in your college and every student will be having marks so if you have 10 student in a class then you need to create 10 different variables like marks 1, marks 2 etc. or in case you have100 students then you need to create 100 variables. Instead of doing that we can create an array.

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained. The type is specified at object creation time by using a type code, which is a single character. The following type codes are defined:

| **Type code** | **C Type** | **Python Type** | **Minimum size in bytes** |
| --- | --- | --- | --- |
| 'c' | char | character | 1 |
| 'b' | signed char | int | 1 |
| 'B' | unsigned char | int | 1 |
| 'u' | Py\_UNICODE | Unicode character | 2 (see note) |
| 'h' | signed short | int | 2 |
| 'H' | unsigned short | int | 2 |
| 'i' | signed int | int | 2 |
| 'I' | unsigned int | long | 2 |
| 'l' | signed long | int | 4 |
| 'L' | unsigned long | long | 4 |
| 'f' | float | float | 4 |
| 'd' | double | float | 8 |

**Note**

The 'u' typecode corresponds to Python’s unicode character. On narrow Unicode builds this is 2-bytes, on wide builds this is 4-bytes.

<https://docs.python.org/2/library/array.html>

**import** array  
  
vals = array.array(**'i'**, [2, 8, 9, 5, 3, 6, 7])  
  
print(vals)

array('i', [2, 8, 9, 5, 3, 6, 7])

**import** array  
  
vals = array.array(**'i'**, [2, 8, 9, 5, 3.5, 6,6, 7])  
  
print(vals)

**TypeError: integer argument expected, got float**

**import** array  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
  
print(vals)

**array('i', [2, 8, 9, 5, -3, 6, 6, 7])**

**import** array  
  
vals = array.array(**'I'**, [2, 8, 9, 5, -3, 6,6, 7])  
  
print(vals)

**OverflowError: can't convert negative value to unsigned int**

**import** array  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
  
print(vals.buffer\_info())

(6355456, 8) printing address and size of array

**import** array  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
  
print(vals.typecode)

i

**import** array  
  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
vals.reverse()  
print(vals)

array('i', [7, 6, 6, -3, 5, 9, 8, 2])

**import** array  
  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
vals.reverse()  
print(vals[1])

8

**import** array  
  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
**for** i **in** range(len(vals)):  
 print(i)

0  
1  
2

.

**import** array  
  
  
vals = array.array(**'i'**, [2, 8, 9, 5, -3, 6,6, 7])  
**for** e **in** vals:  
 print(e,end=**" "**)

**2 8 9 5 -3 6 6 7**

**import** array  
  
  
vals = array.array(**'u'**, [**'a'**, **'e'**, **'e'**, **'i'**,**'o'**,**'u'**])  
**for** e **in** vals:  
 print(e,end=**" "**)

a e e i o u

**import** array  
  
  
vals = array.array(**'i'**, [1,2,8,9,4,6,5])  
newArr = array.array(vals.typecode, (a **for** a **in** vals))  
**for** e **in** newArr:  
 print(e,end=**" "**)

1 2 8 9 4 6 5

**import** array  
  
  
vals = array.array(**'u'**, [**'a'**, **'e'**, **'e'**, **'i'**,**'o'**,**'u'**])  
newArr = array.array(vals.typecode, (a **for** a **in** vals))  
**for** e **in** newArr:  
 print(e,end=**" "**)

**import** array  
  
  
vals = array.array(**'i'**, [1,2,8,9,4,6,5])  
newArr = array.array(vals.typecode, (a\*a **for** a **in** vals))  
**for** e **in** newArr:  
 print(e,end=**" "**)

1 4 64 81 16 36 25

**import** array  
  
  
vals = array.array(**'i'**, [1,2,8,9,4,6,5])  
newArr = array.array(vals.typecode, (a\*\*a **for** a **in** vals))  
**for** e **in** newArr:  
 print(e,end=**" "**)

1 4 16777216 387420489 256 46656 3125

**import** array  
  
  
vals = array.array(**'i'**, [1,2,8,9,4,6,5])  
newArr = array.array(vals.typecode, (a\*\*a **for** a **in** vals))  
i = 0  
**while** i<len(vals):  
 print(i,end=**" "**)  
 i+=1

0 1 2 3 4 5 6

**Lesson # 23: Array values from the user**

**from** array **import** \*  
  
arr = array(**'i'**,[])  
  
x = int(input(**"Enter value for length of array"**))  
**for** i **in** range(x):  
 y = int(input(**"Enter the value"**))  
 arr.append(y)  
  
print(arr,end=**" "**)

Enter value for length of array5

Enter the value25

Enter the value365

Enter the value15

Enter the value258

Enter the value45

array('i', [25, 365, 15, 258, 45])

Process finished with exit code 0

**from** array **import** \*  
  
arr = array(**'i'**,[])  
  
x = int(input(**"Enter value for length of array"**))  
**for** i **in** range(x):  
 y = int(input(**"Enter the value"**))  
 arr.append(y)  
  
print(arr)  
  
searchval = int(input(**"Enter value for search"**))  
  
**for** a **in** arr:  
 **if** a==searchval:  
 print(a)

Enter value for length of array3

Enter the value1256

Enter the value38987

Enter the value2123

array('i', [1256, 38987, 2123])

Enter value for search38987

38987

Process finished with exit code 0

**Printing index value**

**from** array **import** \*  
  
arr = array(**'i'**,[])  
  
x = int(input(**"Enter value for length of array"**))  
**for** i **in** range(x):  
 y = int(input(**"Enter the value"**))  
 arr.append(y)  
  
print(arr)  
  
searchval = int(input(**"Enter value for search"**))  
k= 0  
**for** a **in** arr:  
 **if** a==searchval:  
 print(k)  
  
 k+=1

Enter value for length of array3

Enter the value12

Enter the value36

Enter the value98

array('i', [12, 36, 98])

Enter value for search98

Create an array with 5 values and delete the value at index number 2 without using built in function

**from** array **import** \*  
a=array(**"i"**,[])  
  
n=int(input(**"enter the length"**))  
  
**for** i **in** range(n):  
 x=int(input(**"enter the next value"**))  
 a.append(x)  
  
print(a)  
  
n=int(input(**'enter the value for delete'**))  
**for** e **in** a:  
 **if** e==n:  
 a.remove(e)  
  
print(a)

enter the length4

enter the next value12

enter the next value16

enter the next value20

enter the next value25

array('i', [12, 16, 20, 25])

enter the value for delete20

array('i', [12, 16, 25])

Process finished with exit code 0

**Write a code to reverse an array without using in-built function**

**from** array **import** \*  
arr=array(**'i'**,[1,2,3,4,5])  
arr1=array(**'i'**,[])  
**for** i **in** range(5,0,-1):  
 arr1.append(arr[i-1])  
print(arr)  
print(arr1)

## Lesson # 23: Installation of Numpy

## Lesson # 24: ways of creating array in Numpy

* Array

**from** numpy **import** \*  
arr=array([1,2,3,4,5])  
print(arr)

**from** numpy **import** \*  
arr=array([1,2,3,4,5])  
print(arr.dtype)

int32  
Process finished with exit code 0

**from** numpy **import** \*  
arr=array([1,2,3,4,5.6])  
print(arr.dtype)

print(arr)

float64  
Process finished with exit code 0

**from** numpy **import** \*  
arr=array([1,2,3,4,5.6],int)  
print(arr.dtype)  
print(arr)

### linespace

Its break the values from first value to second value and in third value mention how many parts/ patch first to last value should break.

**from** numpy **import** \*  
  
arr=linspace(0,16,17)  
print(arr.dtype)  
print(arr)

float64  
[ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16.]  
Process finished with exit code 0

**from** numpy **import** \*  
  
arr=linspace(0,16,20)  
print(arr)

[ 0. 0.84210526 1.68421053 2.52631579 3.36842105 4.21052632  
 5.05263158 5.89473684 6.73684211 7.57894737 8.42105263 9.26315789  
 10.10526316 10.94736842 11.78947368 12.63157895 13.47368421 14.31578947  
 15.15789474 16. ]

Process finished with exit code 0

* **If don’t specify range value then default breaks/ patches will be 50**

**from** numpy **import** \*  
  
arr=linspace(0,16)  
print(arr)

[ 0. 0.32653061 0.65306122 0.97959184 1.30612245 1.63265306

1.95918367 2.28571429 2.6122449 2.93877551 3.26530612 3.59183673

3.91836735 4.24489796 4.57142857 4.89795918 5.2244898 5.55102041

5.87755102 6.20408163 6.53061224 6.85714286 7.18367347 7.51020408

7.83673469 8.16326531 8.48979592 8.81632653 9.14285714 9.46938776

9.79591837 10.12244898 10.44897959 10.7755102 11.10204082 11.42857143

11.75510204 12.08163265 12.40816327 12.73469388 13.06122449 13.3877551

13.71428571 14.04081633 14.36734694 14.69387755 15.02040816 15.34693878

15.67346939 16. ]

Process finished with exit code 0

### Arrange

It is not arrange it is basically “A Range”

**from** numpy **import** \*  
  
arr=arange(1,15,3)  
print(arr)

[ 1 4 7 10 13]  
Process finished with exit code 0

**from** numpy **import** \*  
  
arr=arange(0,50,5)  
print(arr)

[ 0 5 10 15 20 25 30 35 40 45]  
Process finished with exit code 0

### Logspace

3rd values determine number of patches, first number start with 10 raise to first value till 10 raise to second value.

**from** numpy **import** \*  
  
arr=logspace(1,4,5)  
print(arr)

[ 10. 56.23413252 316.22776602 1778.27941004   
 10000. ]

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=logspace(1,4,5)  
print(**'%.2f'**%arr[0])

10.00

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=logspace(1,4,5)  
print(**'%.2f'**%arr[1])

56.23

Process finished with exit code 0

### ZEROS

**from** numpy **import** \*  
  
arr=zeros(5)  
print(arr)

[0. 0. 0. 0. 0.]

Process finished with exit code 0

### Ones

**from** numpy **import** \*  
  
arr=ones(5)  
print(arr)

[1. 1. 1. 1. 1.]

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=ones(5,int)  
print(arr)

## Lesson # 26: Copying an Array in python

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
print(sort(arr))

[1 2 2 4 5 6 8 9]

Process finished with exit code 0

### Vactorized operation

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print((arr2))

[ 3 15 3 13 15 5 13 13]

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(log(arr2))

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(sin (arr2))

### Array addition

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(sqrt(arr2))

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(sort(arr2))

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(concatenate([arr,arr1]))

[1 6 2 8 9 2 5 4 2 9 1 5 6 3 8 9]

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(sort(concatenate([arr,arr1])))

### Aliasing

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=arr  
print(arr)  
print(arr1)  
  
print(id(arr))  
print(id(arr1))

[1 6 2 8 9 2 5 4]

[1 6 2 8 9 2 5 4]

186585440

186585440

Process finished with exit code 0

### Shallow copy

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=arr.view()  
print(arr)  
print(arr1)  
  
print(id(arr))  
print(id(arr1))

[1 6 2 8 9 2 5 4]

[1 6 2 8 9 2 5 4]

189615464

189615424

Process finished with exit code 0

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=arr.view()  
arr1[1]=99  
print(arr)  
print(arr1)  
  
print(id(arr))  
print(id(arr1))

[ 1 **99** 2 8 9 2 5 4]

[ 1 **99** 2 8 9 2 5 4]

187717856

185720408

Process finished with exit code 0

### Deep Copy

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=arr.copy()  
arr1[1]=99  
print(arr)  
print(arr1)  
  
print(id(arr))  
print(id(arr1))

[1 **6** 2 8 9 2 5 4]

[ 1 **99** 2 8 9 2 5 4]

181932264

187389816

Process finished with exit code 0

1. Write a code to add 2 arrays using for loop

**from** array **import** \*  
a = array(**'i'**, [1, 2, 3])  
**for** i **in** range(len(a)):  
 b = array(**'i'**, [4, 5, 6])  
 print(a[i]+b[i],end=**" "**)

1. Write a code to find the maximum value from an array without in-built function

**from** numpy **import** \*  
  
arr=array([1,6,2,8,9,2,5,4])  
arr1=array([2,9,1,5,6,3,8,9])  
arr2 = arr+arr1  
print(max (arr2))

## Working with Matrix in Python

**from** numpy **import** \*  
arr = array([  
 [1,2,3],  
 [4,5,6]  
 ])  
print(arr)

**from** numpy **import** \*  
arr = array([  
 [1,2,3,4],  
 [4,5,6,0]  
 ])  
print(arr.size)  
print(arr.dtype)  
print(arr.shape)

**multi dimential to singal dimension**

**from** numpy **import** \*  
arr = array([  
 [1,2,3,4],  
 [4,5,6,0]  
 ])  
arr1 = arr.flatten()  
print(arr1)

**from** numpy **import** \*  
arr = array([  
 [1,2,3,4,4,5],  
 [4,5,6,0,9,8]  
 ])  
arr1 = arr.flatten()  
  
arr2 = arr1.reshape(3,4)  
print(arr2)

**from** numpy **import** \*  
arr = array([  
 [1,2,3,4,4,5],  
 [4,5,6,0,9,8]  
 ])  
arr1 = arr.flatten()  
  
arr2 = arr1.reshape(2,2,3)  
print(arr2)

### Matrices

**from** numpy **import** \*  
arr = array([  
 [1,2,3,4,4,5],  
 [4,5,6,0,9,8]  
 ])  
m = matrix(arr)  
  
print(arr)

**from** numpy **import** \*  
  
arr = array([  
 [1,2,3,4,4,5],  
 [4,5,6,0,9,8]  
 ])  
m = matrix(**'1 2 ;3 4; 5 6; 7 8'**)  
  
print(m)

**from** numpy **import** \*  
  
arr = array([  
 [1,2,3,4,4,5],  
 [4,5,6,0,9,8]  
 ])  
m = matrix(**'1 2 3; 4 5 6; 6 7 8'**)  
  
print(m)

### Diagnol matrix

**from** numpy **import** \*  
  
m = matrix(**'1 2 3; 4 5 6; 6 7 8'**)  
  
print(diagonal(m))

### Addition and multiplication of two matrix

**from** numpy **import** \*  
  
m1 = matrix(**'1 2 3; 4 5 6; 6 7 8'**)  
m2 = matrix(**'6 5 4; 9 8 6; 3 6 5'**)  
  
m3 = m1 + m2  
  
print(m3)

[[ 7 7 7]

[13 13 12]

[ 9 13 13]]

Process finished with exit code 0

**from** numpy **import** \*  
  
m1 = matrix(**'1 2 3; 4 5 6; 6 7 8'**)  
m2 = matrix(**'6 5 4; 9 8 6; 3 6 5'**)  
  
m3 = m1 \* m2  
  
print(m3)

[[ 33 39 31]

[ 87 96 76]

[123 134 106]]

Process finished with exit code 0

**from** numpy **import** \*  
  
m1 = matrix(**'1 2 3; 4 5 6'**)  
m2 = matrix(**'7 8; 9 10 ; 11 12'**)  
  
m3 = m1 \* m2  
  
print(m3)

[[ 58 64]

[139 154]]

Process finished with exit code 0

## Lesson # 28 Functions in Python

### What is function?

When we create big and complex project and we need to break project in smaller tasks, and same tasks we need to reuse in several places, then we put all of them in one place. The function which we create by our own.

### Why Function?

We can call function multiple time.

### Creating and calling function

**def** greet():  
 print(**"Hello"**)  
 print(**"welcome"**)  
  
greet()

**def** add(x,y):  
 z= x+y  
 print(z)  
add(2,5)

**def** add(x,y):  
 z= x+y  
 **return** z  
result = add(5,9)  
print(result)

**def** add\_sub(x,y):  
 z= x+y  
 y= x-y  
 **return** z,y  
  
result = add\_sub(5,9)  
print(result)

## Lesson # 29 Function Arguments in Python

### Function argument

How to pass a parameter, different type of parameter

**def** update(x):  
 x=8  
 print(id(x))  
 print(x)  
a = 8  
update(a)  
print(a)  
print(id(a))

### Mutable and immutable argument

**def** update(lst):  
 print(id(lst))  
 lst[2] = 123  
 print(id(lst))  
 print(**"x "**,lst)  
lst = [10,20,52,23,6]  
print(id(lst))  
  
update(lst)  
print(**"lst"**,lst)

## Lesson # 30 Types of Arguments in Python

### Formal argument

The variable which is not having any value yet

### Actual argument

The actual values passing to the variables

### Position

**def** person(name,age):  
 print(name)  
 print(age)  
  
person(**'Noman'**,45)

### Keyword

**def** person(name,age):  
 print(name)  
 print(age)  
  
person(age=45,name=**'Noman'**)

### Default

**def** person(name,age=18):  
 print(name)  
 print(age)  
  
person(**'Noman'**)

**def** person(name,age=18):  
 print(name)  
 print(age)  
  
person(**'Noman'**,24)

### Variable length

**def** sum(a,b):  
 c=a+b  
 print(c)  
sum(6,5,8,9,10)

Traceback (most recent call last):

File "C:/Users/user/PycharmProjects/MyPythoneProject/MyCode.py", line 5, in <module>

sum(6,5,8,9,10)

TypeError: sum() takes 2 positional arguments but 5 were given

Process finished with exit code 1

**def** sum(a,\*b):  
 c=a+b  
 print(c)  
sum(6,5,8,9,10)

Traceback (most recent call last):

File "C:/Users/user/PycharmProjects/MyPythoneProject/MyCode.py", line 5, in <module>

sum(6,5,8,9,10)

File "C:/Users/user/PycharmProjects/MyPythoneProject/MyCode.py", line 3, in sum

c=a+b

TypeError: unsupported operand type(s) for +: 'int' and 'tuple'

Process finished with exit code 1

**def** sum(a,\*b):  
 print(a)  
 print(b)  
sum(6,5,8,9,10)

6

(5, 8, 9, 10)

Process finished with exit code 0

**def** sum(a,\*b):  
 c = a  
 **for** i **in** b:  
 c = c+i  
 print(c)  
  
sum(2,2,2,2,10)

**def** sum(\*b):  
 c = 0  
 **for** i **in** b:  
 c = c+i  
 print(c)  
  
sum(2,2,2,2,10)

## Lesson # 31 Keyworded Variable Length Arguments in Python | \*\*kwargs

**def** person(name,\*\*data):  
 print(name)  
 **for** i,j **in** data.items():  
 print(i,j)  
  
person(**'Noman'**,age=45,city=**'Karachi'**,phone=33227262)

## Lesson # 31 Global Keyword in Python | Global vs Local Variable

a = 10  
**def** something():  
 a = 9  
  
  
 print(a)  
something()  
print(a)

a = 10  
**def** something():  
 **global** a  
 a = 9  
  
  
 print(a)  
something()  
print(a)

a = 10  
print(id(a))  
**def** something():  
 a = 9  
  
 x = globals()[**'a'**]  
 print(id(x))  
 print(**"in fun "**,a)  
  
 globals()[**'a'**]=15  
something()  
print(**"outside "**,a)

## Lesson # 32 Pass List to a Function in Python

### Pass list to a function Count number of even and odd number in a list

**def** CntNamechar(lst):  
 more = 0  
 less = 0  
  
 **for** i **in** range(len(lst)):  
 **if** len(lst[i]) > 5:  
 more = more+1  
 **else**:  
 less = less+1  
 print(lst)  
 **return** (more,less)  
  
list =[]  
x= int(input(**"How many name you would like to enter"**))  
  
**for** a **in** range(x):  
 list.append((input()))  
more,less = CntNamechar(list)  
print(**'Names more then 5 characters :{} and Name less then 5 characers :{}'**.format(more,less))

**def** counts(lst):  
  
 more = 0  
 less = 0  
  
 **for** i **in** range(len(lst)):  
 **if** len(lst[i]) > 5:  
 more = more + 1  
 **else**:  
 less = less + 1  
  
 **return** more, less  
  
  
list = []  
x = int(input(**"How many names you want to enter:"**))  
  
**for** k **in** range(x):  
 list.append((input()))  
  
more, less = counts(list)  
print(**'Names more than 5 characters : {} and Names less than 5 characters : {}'**.format(more, less))

## Fibonacci Sequence in Python

**def** fab(n):  
  
 a = 0  
 b = 1  
 **if** n == 1:  
 print(a)  
  
 **else**:  
 print(a)  
 print(b)  
  
 **for** i **in** range(2,n):  
 c = a + b  
 a = b  
 b = c  
 **if** c<= 100000:  
 print(c,end=**" "**)  
fab(100)

**def** fab(n):  
  
 a = 0  
 b = 1  
 print(a)  
 **for** i **in** range(0,n):  
 c = a + b  
 a = b  
 b = c  
 print(c)  
  
fab(100)

## #34 Python Tutorial for Beginners | Factorial in Python

**def** fact(n):  
 a = 1  
 **for** i **in** range(1,n+1):  
 a = i\*a  
 **return** a  
  
  
print(fact(6))

## #35 Python Tutorial for Beginners | Recursion in Python

**import** sys  
  
sys.setrecursionlimit(2000)  
print(sys.getrecursionlimit())  
a = 0  
**def** recur():  
 **global** a  
 a+=1  
 print(**"Hello"**,a)  
 recur()  
recur()

## #36 Python Tutorial for Beginners | Factorial using Recursion in Python

**def** fact(n):  
 **if** n<=0:  
 **return** 1  
 **return** n \* fact(n-1)  
result = fact(5)  
print(result)

## #37 Python Tutorial for Beginners | Anonymous Functions in Python | Lambda

f = **lambda** a:a\*a  
  
result = f(5)  
print(result)

f = **lambda** a,b,c:a+b/(c)  
  
result = f(5,9,5)  
print(result)

## #38 Python Tutorial for Beginners | Anonymous Functions in Python | Lambda

### Filter

**def** is\_even(n):  
 **return** n%2==0  
  
  
nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(is\_even,nums))  
  
print(evens)

**def** is\_even(n):  
 **return** n%2==0  
  
  
nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(**lambda** n:n%2==0,nums))  
  
print(evens)

### map

**def** is\_even(n):  
 **return** n%2==0  
  
**def** update(n):  
 **return** n\*2  
  
nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(**lambda** n:n%2==0,nums))  
  
double = list(map(update,evens))  
  
print(evens)  
print(double)

nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(**lambda** n:n%2==0,nums))  
  
double = list(map(**lambda** n:n\*2,evens))  
  
print(evens)  
print(double)

### reduce

**from** functools **import** \*  
  
**def** is\_even(n):  
 **return** n%2==0  
  
**def** update(n):  
 **return** n\*2  
  
**def** add\_all(a,b):  
 **return** a+b  
  
nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(**lambda** n:n%2==0,nums))  
  
double = list(map(**lambda** n:n\*2,evens))  
  
sums = reduce(add\_all,double)  
  
print(evens)  
print(double)  
print(sums)

**from** functools **import** \*  
  
nums = [3,8,6,2,3,7,9,8,9,3,5]  
  
evens = list(filter(**lambda** n:n%2==0,nums))  
  
double = list(map(**lambda** n:n\*2,evens))  
  
sums = reduce(**lambda** a,b:a+b,double)  
  
print(evens)  
print(double)  
print(sums)

## Decorators in Python

To changing the existing function behavior with our desires. We can take input as function within a function which is not possible in another programming. By using of decorators we can easily change the code by using of existing function in new function.

**def** dev(a,b):  
 print(a/b)  
  
  
**def** dev\_reverse(func):  
  
 **def** inside(a,b):  
 **if** a<b:  
 a,b=b,a  
 **return** func(a,b)  
 **return** inside  
  
  
  
result = dev\_reverse(dev)  
  
result(3,4)

## #39 Python Tutorial for Beginners | Modules in Python

You can easily create different modules and call them by import function

## #40 Python Tutorial for Beginners | Special Variable \_\_name\_\_ in Python

## #41 Python Tutorial for Beginners | Special Variable \_\_name\_\_ part 2 in Python

**# MyCode.py**

**def** add():  
 print(**"result 1 is"**,\_\_name\_\_)  
  
**def** sub():  
 print(**"result 2 is"**)  
  
**def** main():  
 print(**"in calc main"**)  
 add()  
 sub()

**#MyFirstPythone.py**

**from** MyCode **import** \*  
**def** fun1():  
 add()  
 print(**"from fun 1"**)  
  
**def** fun2():  
 print(**"from fun 2"**)  
  
**def** main():  
 fun1()  
 fun2()  
  
main()

**# MyCode.py**

**def** add():  
 print(**"result 1 is"**,\_\_name\_\_)  
  
**def** sub():  
 print(**"result 2 is"**)  
  
**def** main():  
 print(**"in calc main"**)  
 add()  
 sub()  
  
**if** \_\_name\_\_==**"\_\_main\_\_"**:  
 main()

**#MyFirstPythone.py**

**from** MyCode **import** \*  
**def** fun1():  
 add()  
 print(**"from fun 1"**)  
  
**def** fun2():  
 print(**"from fun 2"**)  
  
**def** main():  
 fun1()  
 fun2()  
  
main()

## #42 Python Tutorial for Beginners | Object Oriented Programming in Python | Introduction

**class** computer:  
 **def** config(self):  
 print(**"i7, 16GBRam, 1TB"**)  
  
com1 = computer()  
  
print(type(com1))  
  
computer.config(com1)

**class** computer:  
 **def** config(self):  
 print(**"i7, 16GBRam, 1TB"**)  
  
com1 = computer()  
  
print(type(com1))  
  
computer.config(com1)  
  
com1.config()

## #44 Python Tutorial for Beginners | \_\_init\_\_ method in Python

**class** computer:  
 **def** \_\_init\_\_(self,cpu,ram):  
 self.cpu = cpu  
 self.ram = ram  
  
 **def** config(self):  
 print(**"Config is "**,self.cpu,self.ram)  
  
com1 = computer(**"noman"**,**"1TB"**)  
com2 = computer(**"asim"**,**"500gb"**)  
  
com1.config()  
com2.config()

**class** computer:  
 **def** \_\_init\_\_(self,cpu,ram):  
 self.x = cpu  
 self.y = ram  
  
 **def** config(self):  
 print(**"Config is "**,self.x,self.y)  
  
com1 = computer(**"noman"**,**"1TB"**)  
com2 = computer(**"asim"**,**"500gb"**)  
  
com1.config()  
com2.config()

## #45 Python Tutorial for Beginners | Constructor, Self and Comparing Objects in Python

**class** computer:  
 **def** \_\_init\_\_(self):  
 self.name = **'Noman'** self.age = 45  
  
 **def** update(self):  
 self.age=30  
  
com1 = computer()  
com2 = computer()  
  
com2.name = **'Arsalan'**print(com1.name)  
print(com2.name)  
com2.update()  
print(com1.age)  
print(com2.age)

**class** computer:  
 **def** \_\_init\_\_(self):  
 self.name = **'Noman'** self.age = 45  
  
 **def** update(self):  
 self.age=30  
 self.name = **'Arsalan'**com1 = computer()  
com2 = computer()  
  
  
com2.update()  
  
print(com1.name)  
  
print(com1.age)  
  
print(com2.name)  
  
print(com2.age)

**class** computer:  
 **def** \_\_init\_\_(self):  
 self.name = **'Noman'** self.age = 45  
  
 **def** update(self):  
 self.age=30  
 self.name = **'Arsalan'  
  
 def** compare(self,other):  
 **if** self.age ==other.age:  
 **return True  
 else**:  
 **return False**com1 = computer()  
com2 = computer()  
  
  
com2.update()  
  
print(com1.name)  
  
print(com1.age)  
  
print(com2.name)  
  
print(com2.age)  
  
**if** com1.compare(com2):  
 print(**"They are Same"**)  
**else**:  
 print(**"They are different"**)

## #46 Python Tutorial for Beginners | Types of Variables in Python

### Instance variable

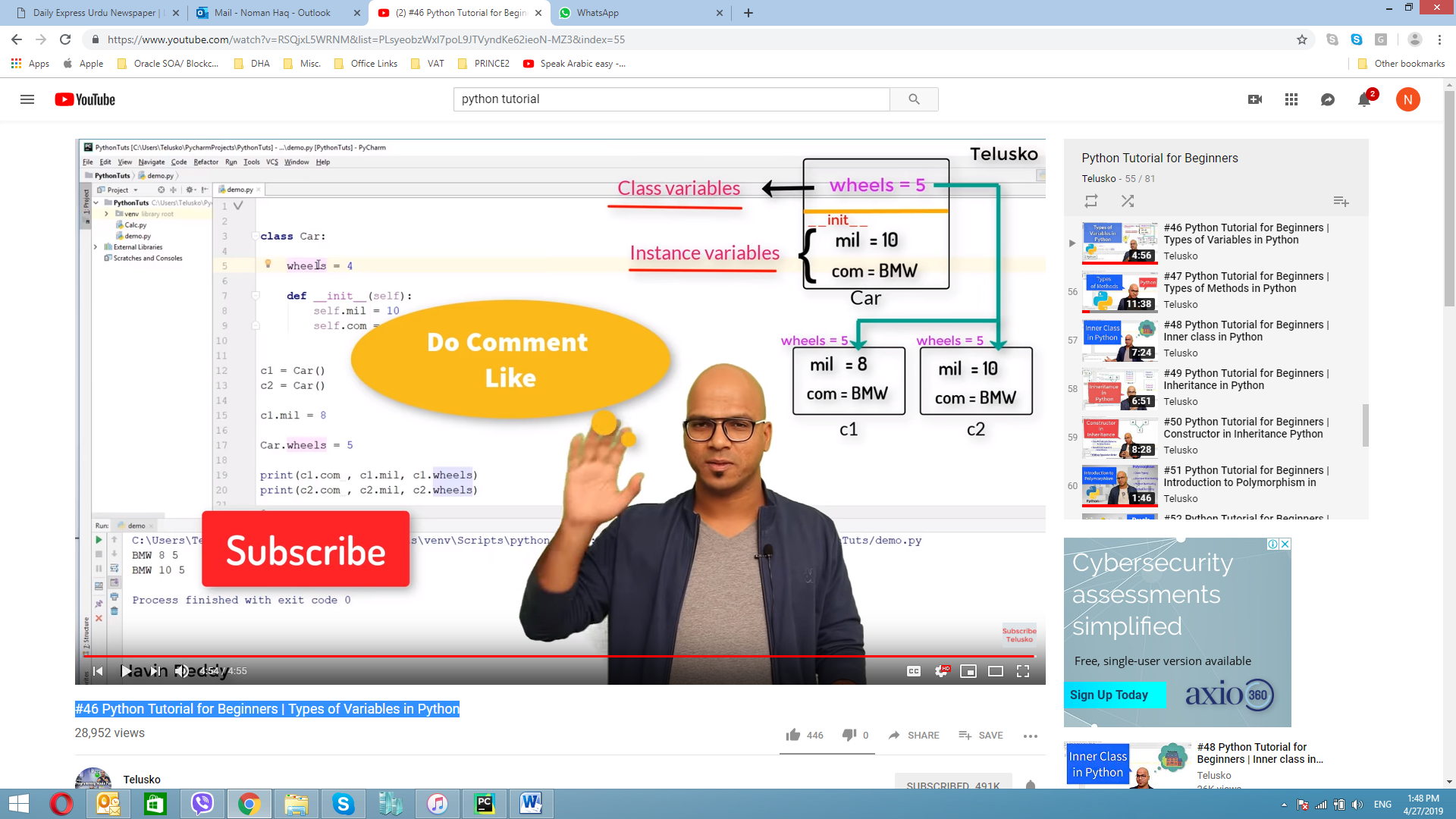
**class** car():  
  
 **def** \_\_init\_\_(self):  
 self.mil = 10  
 self.company = **"BMW"**c1 = car()  
c2 = car()  
  
c1.mil = 250  
print(c1.company,c1.mil)  
print(c2.company,c2.mil)

### Class variable

The variable which is common for all objects is called class variable

**class** car():  
  
 company = **"BMW"  
  
 def** \_\_init\_\_(self):  
 self.mil = 10  
 self.wheel = 4  
  
  
c1 = car()  
c2 = car()  
  
c1.mil = 250  
*#car.company = "Lexus"*print(c1.company,c1.mil,c1.wheel)  
print(c2.company,c2.mil,c2.wheel)

**class** car():  
  
 company = **"BMW"  
  
 def** \_\_init\_\_(self):  
 self.mil = 10  
 self.wheel = 4  
  
  
c1 = car()  
c2 = car()  
  
c1.mil = 250  
car.company = **"Lexus"**print(c1.company,c1.mil,c1.wheel)  
print(c2.company,c2.mil,c2.wheel)



## #47 Python Tutorial for Beginners | Types of Methods in Python

### Instance method

**class** student():  
 school = **"Arman School"  
 def** \_\_init\_\_(self,m1,m2,m3):  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 **def** average(self):  
 **return** (self.m1+self.m2+self.m3)/3  
  
s1 = student(36,87,39)  
s2 = student(89,54,26)  
  
print(s1.average())  
print(s2.average())

### accessor /mutator method

If you want to fetch the value you are using accessor method. If you want to modify value then we are using mutator method.

**class** student():  
 school = **"Arman School"  
 def** \_\_init\_\_(self,m1,m2,m3):  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 **def** average(self):  
 **return** (self.m1+self.m2+self.m3)/3  
  
 **def** getm1(self): **---- This method represent accssor method**   
 **return** self.m1  
  
 **def** setm1(self,value): **---- This method represent mutator method**  
 self.m1 = value  
  
s1 = student(36,87,39)  
s2 = student(89,54,26)  
  
print(s1.average())  
print(s2.average())

### Class method

When we use class method then we need to use special symbol or way of doing that is @classmethod as decorator

**class** student():  
 school = **"Arman School"  
 def** \_\_init\_\_(self,m1,m2,m3):  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 **def** average(self):  
 **return** (self.m1+self.m2+self.m3)/3  
  
 @classmethod  
 **def** info(cls):  
 **return** cls.school  
  
s1 = student(36,87,39)  
s2 = student(89,54,26)  
  
print(s1.average())  
print(s2.average())  
print(student.info())

### Static method

This method is nothing to do with class or instance variable, now where it will be usefulle for say you want use this method somewhere outside of this class or perfor any other operation. For creating static method we need to use some special decorator like we used in class method which is @statismethod

**class** student():  
 school = **"Arman School"  
 def** \_\_init\_\_(self,m1,m2,m3):  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 **def** average(self):  
 **return** (self.m1+self.m2+self.m3)/3  
  
 @classmethod  
 **def** getschoolname(cls):  
 **return** cls.school  
  
 @staticmethod  
 **def** info():  
 print(**"This is student class .. in class module "**)  
  
s1 = student(36,87,39)  
s2 = student(89,54,26)  
  
print(s1.average())  
print(s2.average())  
print(student.getschoolname ())  
student.info()

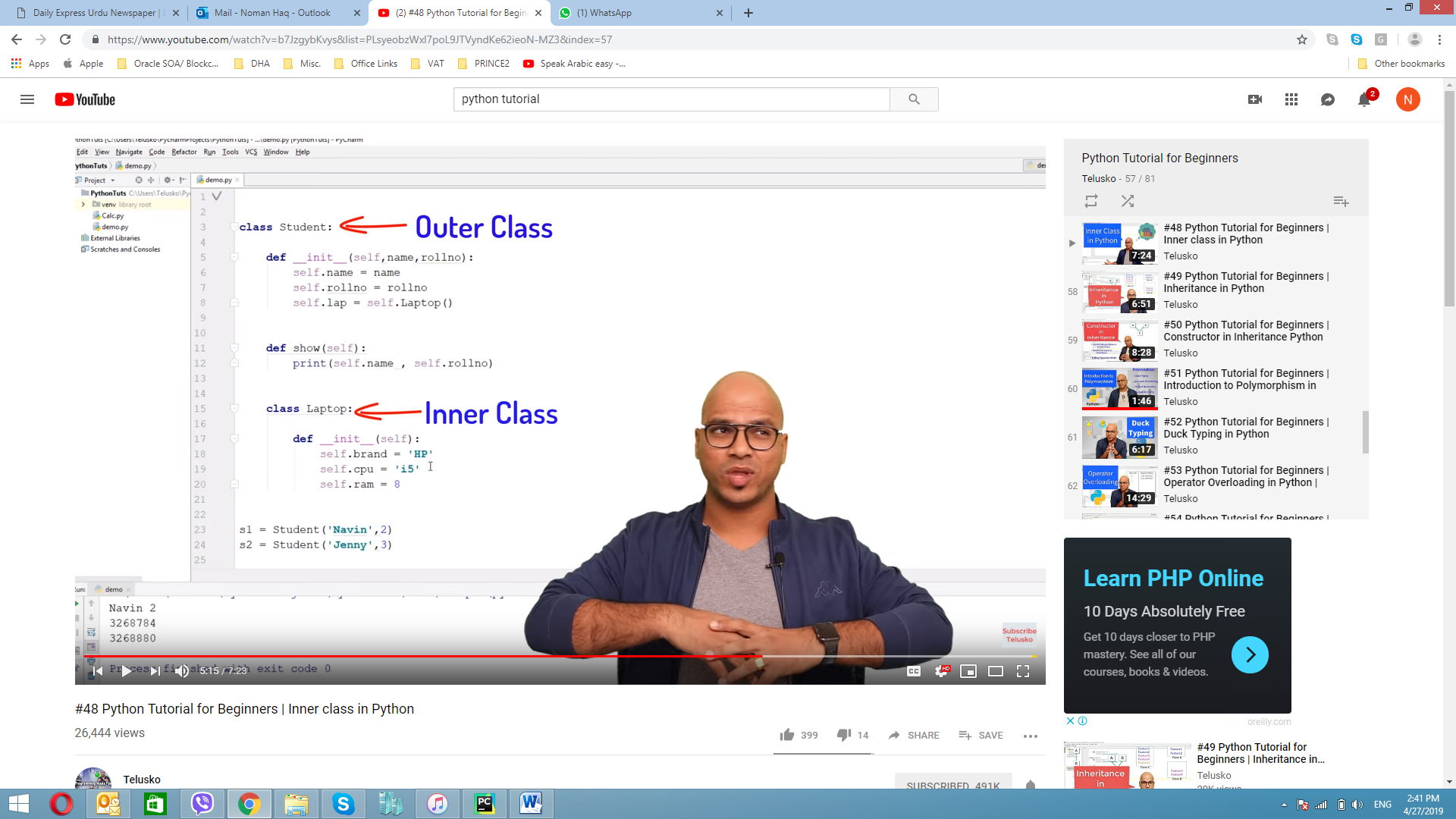
## #48 Python Tutorial for Beginners | Inner class in Python

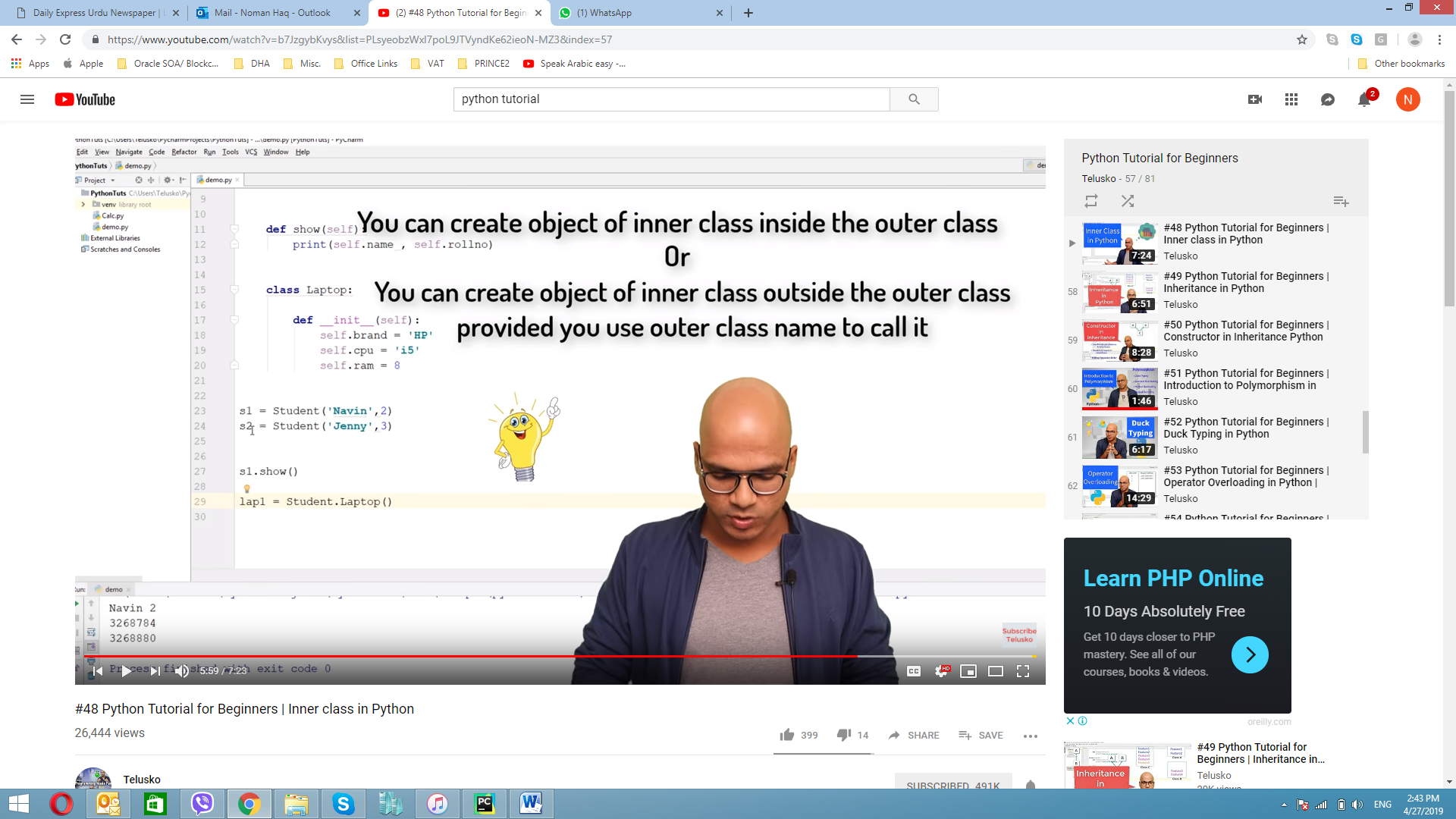
We can easily class inside a class

**class** student():  
 **def** \_\_init\_\_(self,name,rollno):  
 self.name = name  
 self.rollno = rollno  
  
  
  
s1 = student(**"A-Rahman"**,1)  
s2 = student(**"Noman"**,2)  
  
print(s1.name,s1.rollno)  
print(s2.name,s2.rollno)

**class** student():  
 **def** \_\_init\_\_(self,name,rollno):  
 self.name = name  
 self.rollno = rollno  
  
 **def** show(self):  
 print(self.name, self.rollno)  
  
s1 = student(**"A-Rahman"**,1)  
s2 = student(**"Noman"**,2)  
  
s1.show()  
s2.show()

**class** student():  
 **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():  
 **def** \_\_init\_\_(self):  
 self.brand = **'HP'** self.cpu = **"i7"** self.ram = **"16GB"**s1 = student(**"A-Rahman"**,1)  
s2 = student(**"Noman"**,2)  
  
s1.show()  
s2.show()



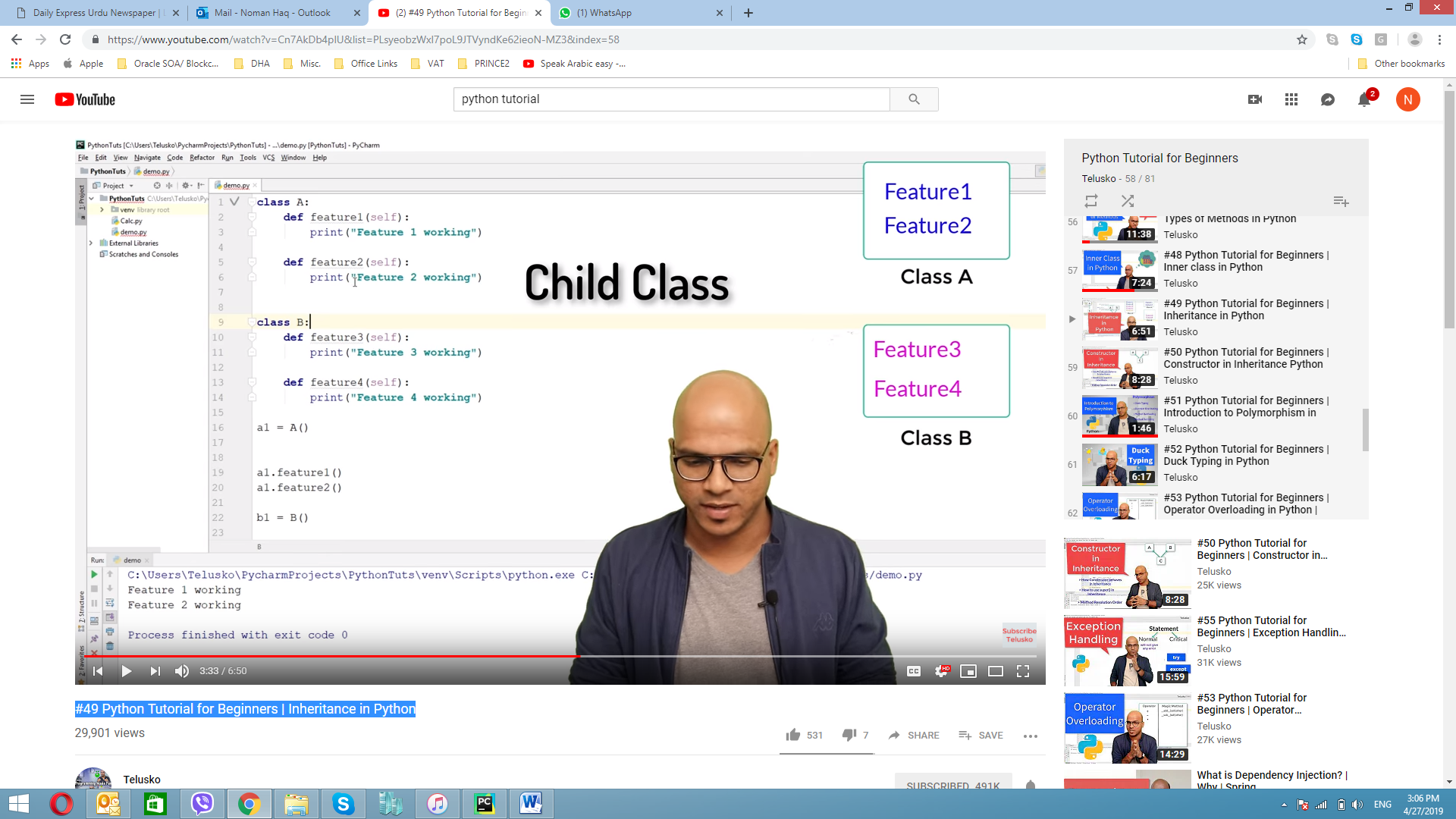
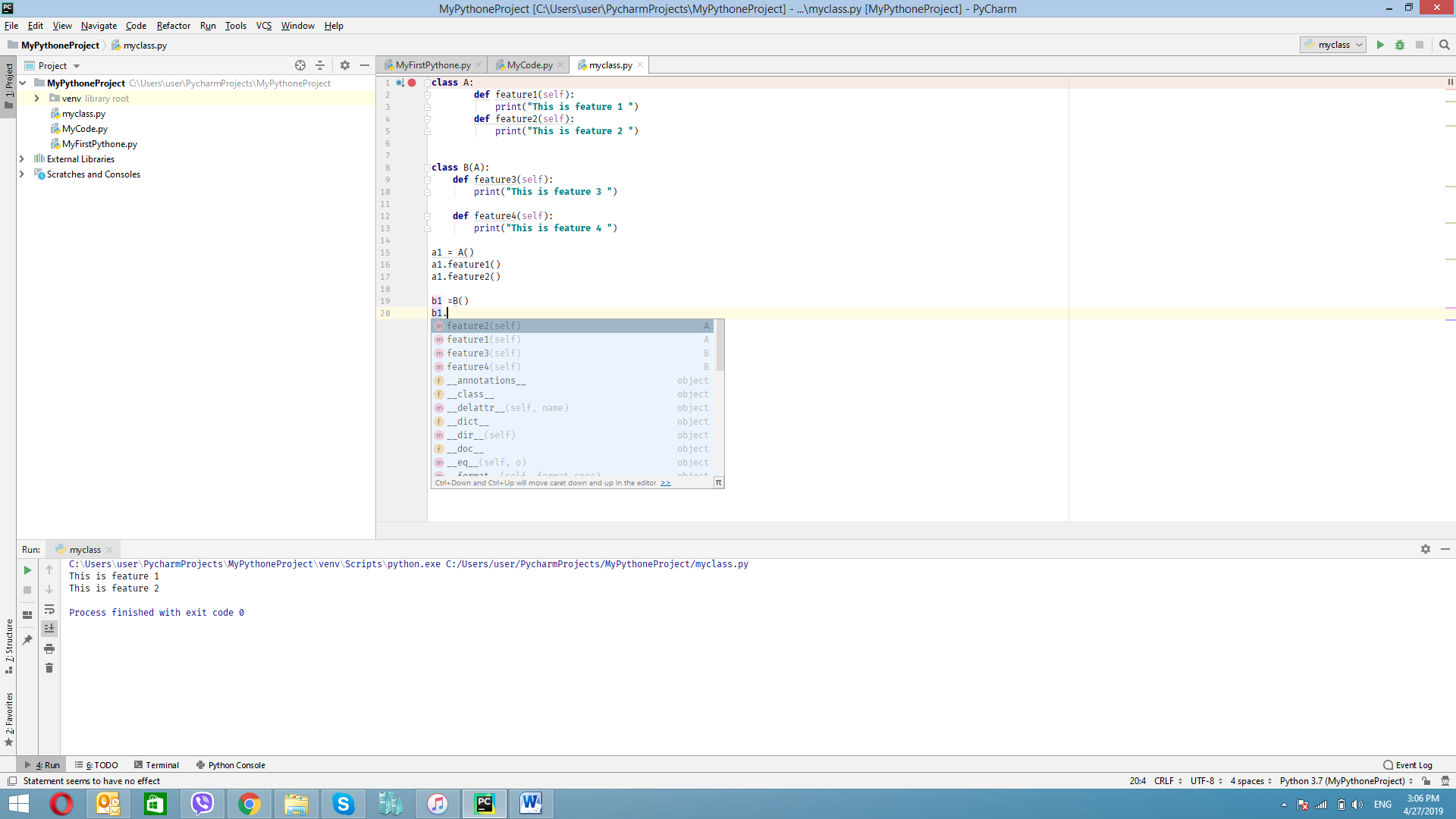


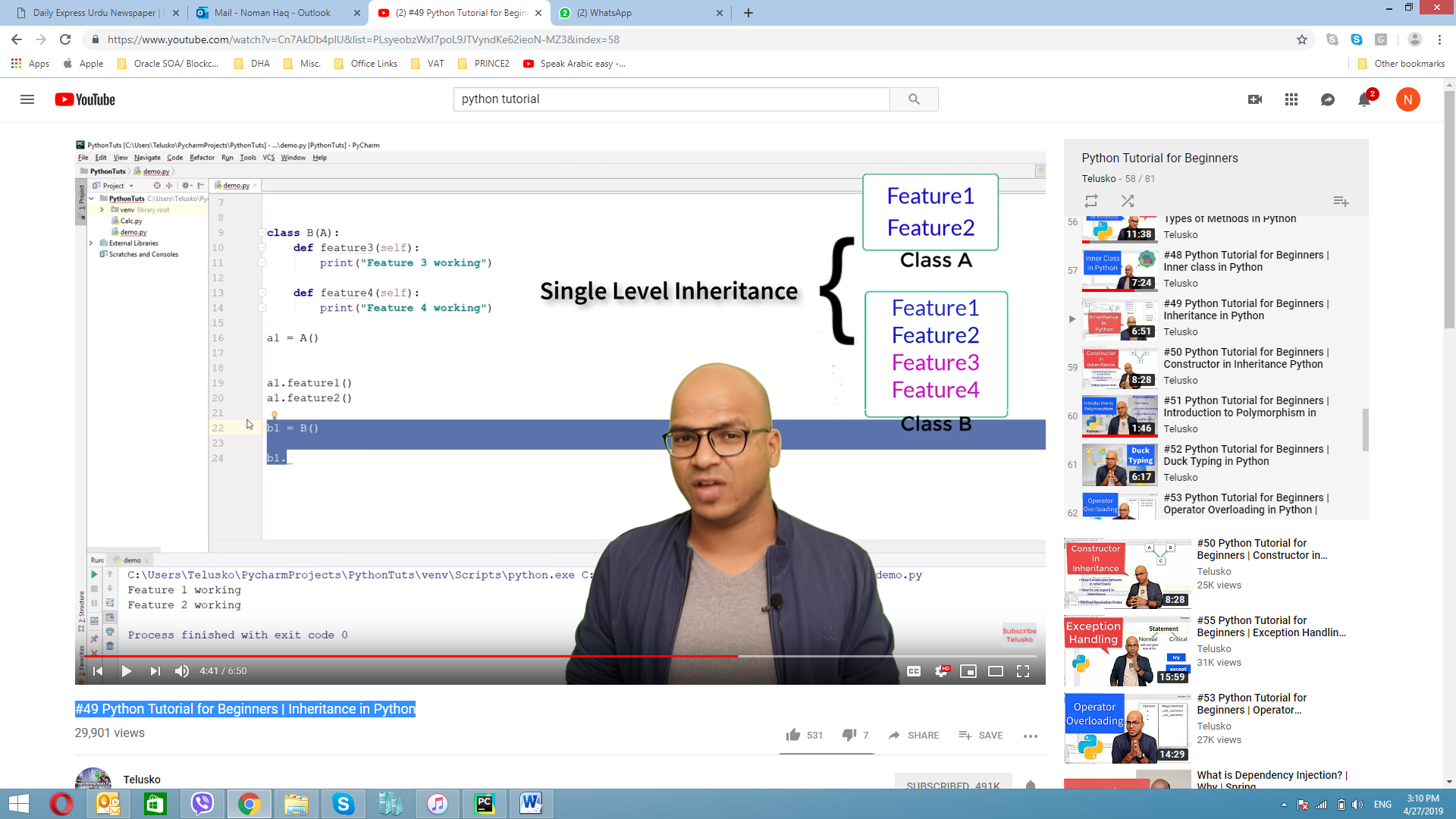
**class** student():  
 **def** \_\_init\_\_(self,name,rollno):  
 self.name = name  
 self.rollno = rollno  
 self.lap = self.laptop()  
  
 **def** show(self):  
 print(self.name, self.rollno)  
 self.lap.show()  
  
  
 **class** laptop():  
 **def** \_\_init\_\_(self):  
 self.brand = **'HP'** self.cpu = **"i7"** self.ram = **"16GB"  
 def** show(self):  
 print(self.brand,self.ram,self.cpu)  
  
s1 = student(**"A-Rahman"**,1)  
s2 = student(**"Noman"**,2)  
  
  
s1.show()  
s2.show()

**class** student():  
 **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():  
 **def** \_\_init\_\_(self):  
 self.brand = **'HP'** self.cpu = **"i7"** self.ram = **"16GB"  
 def** show(self):  
 print(self.brand,self.ram,self.cpu)  
  
s1 = student(**"A-Rahman"**,1)  
s2 = student(**"Noman"**,2)  
  
lap1 = s1.lap.brand = **"Toshiba"**lap1 = s1.lap.cpu = **"i8"**lap1 = s1.lap.ram = **"32GB"**s1.show()  
s2.show()

## #49 Python Tutorial for Beginners | Inheritance in Python

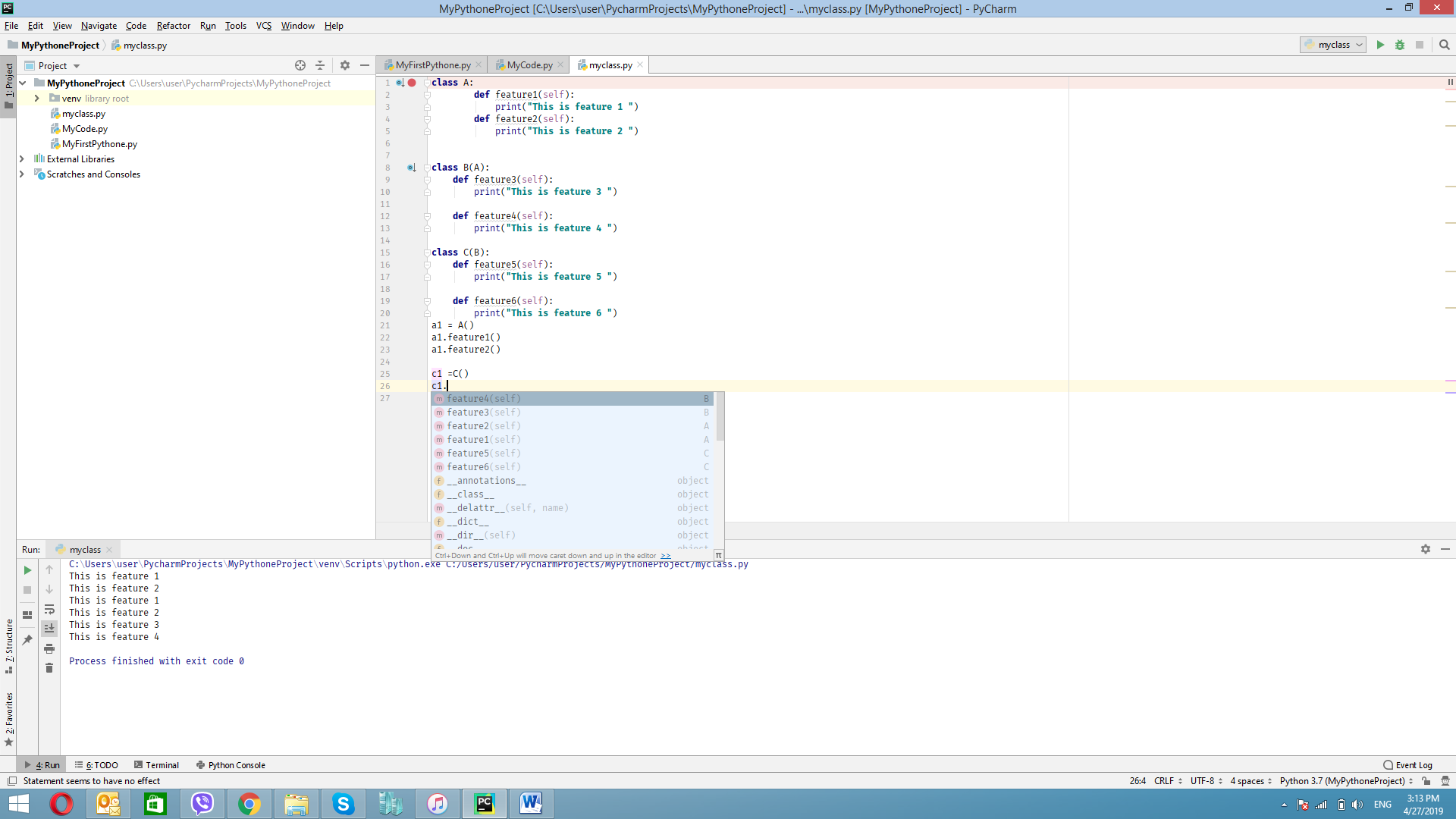
### Child Class / super class, sub class / single level inheritance





**class** A:  
 **def** feature1(self):  
 print(**"This is feature 1 "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B(A):  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
  
a1 = A()  
a1.feature1()  
a1.feature2()  
  
b1 =B()  
b1.feature1()  
b1.feature2()  
b1.feature3()  
b1.feature4()

### Multilevel inheritance



**class** A:  
 **def** feature1(self):  
 print(**"This is feature 1 "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B(A):  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
  
**class** C(B):  
 **def** feature5(self):  
 print(**"This is feature 5 "**)  
  
 **def** feature6(self):  
 print(**"This is feature 6 "**)  
a1 = A()  
a1.feature1()  
a1.feature2()  
  
c1 =C()  
c1.feature1()  
c1.feature2()  
c1.feature3()  
c1.feature4()  
c1.feature5()  
c1.feature6()

### Multiple inheritance

In case there is no relationship between two classes and we want to create relationships.



**class** A:  
 **def** feature1(self):  
 print(**"This is feature 1 "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B():  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
  
**class** C(A,B):  
 **def** feature5(self):  
 print(**"This is feature 5 "**)  
  
 **def** feature6(self):  
 print(**"This is feature 6 "**)  
  
c1 =C()  
c1.feature1()  
c1.feature2()  
c1.feature3()  
c1.feature4()  
c1.feature5()  
c1.feature6()

## #50 Python Tutorial for Beginners | Constructor in Inheritance Python

**class** A:  
  
 **def** \_\_init\_\_(self):  
 print(**"in A init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B(A):  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in B init"**)  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
  
a1 = B()

### Method resolution order

Whenever we will have multiple inheritance, first it will call init itself and then will see who is appearing on left side first. It will always prefer left side.

Same thing happen in method

**class** A:  
  
 **def** \_\_init\_\_(self):  
 print(**"in A init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B():  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in B init"**)  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
**class** C(A,B):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in C init"**)  
a1 = C()

**class** A:  
  
 **def** \_\_init\_\_(self):  
 print(**"in A init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 in A "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B():  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in B init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 in B "**)  
  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
**class** C(A,B):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in C init"**)  
a1 = C()  
a1.feature1()

**class** A:  
  
 **def** \_\_init\_\_(self):  
 print(**"in A init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 in A "**)  
 **def** feature2(self):  
 print(**"This is feature 2 "**)  
  
  
**class** B():  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in B init"**)  
 **def** feature1(self):  
 print(**"This is feature 1 in B "**)  
  
 **def** feature3(self):  
 print(**"This is feature 3 "**)  
  
 **def** feature4(self):  
 print(**"This is feature 4 "**)  
**class** C(A,B):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(**"in C init"**)  
 **def** feature1(self):  
 super().feature3()  
a1 = C()  
a1.feature1()  
*#a1.feature1()*

## #51 Python Tutorial for Beginners | Introduction to Polymorphism in Python | Part 1

One thing can take multiple forms.

## #52 Python Tutorial for Beginners | Duck Typing in Python

If it is look like a duck, swim like a duck and quack like a duck then it probably is a duck.

We have class with name of pycharm and it has function with name of execute which have 2 characterstics, when we call pycharm with execute method it give us 2 characteristics:

**class** PyCharm:  
 **def** execute(self):  
 print(**"Compiling"**)  
 print(**"Running"**)  
**class** laptop:  
 **def** code(self,ide):  
 ide.execute()  
  
ide = PyCharm()  
  
lap1 = laptop()  
lap1.code(ide)

hence if we call execute with some other method which has more characterstics it will return more means method name reman same but we are calling same method from another class:

**class** PyCharm:  
 **def** execute(self):  
 print(**"Compiling"**)  
 print(**"Running"**)

**class** MyEditor:  
 **def** execute(self):

print(**"Compiling"**)  
 print(**"Running"**)

print(**"Debugging"**)  
 print(**"Showing Alerts "**)

print(**"Others 1"**)  
 print(**"Others 2"**)

**class** laptop:  
 **def** code(self,ide):  
 ide.execute()  
  
ide = MyEditor ()  
  
lap1 = laptop()  
lap1.code(ide)

## #53 Python Tutorial for Beginners | Operator Overloading in Python | Polymorphism

### Operator overloading

a = 1  
b = 6  
  
print(a+b)  
print(int.\_\_add\_\_(a,b))

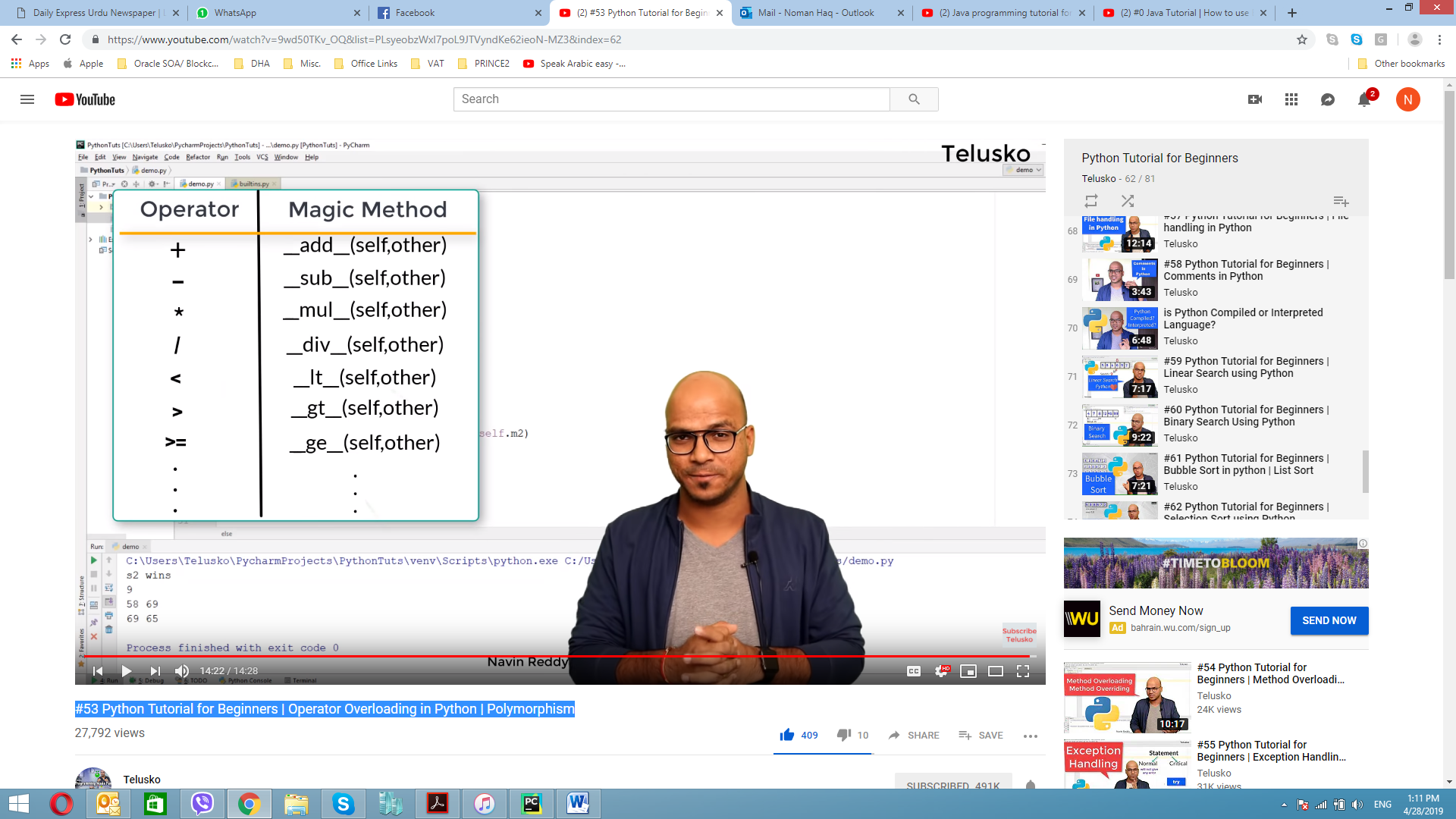
a = 8  
b = 6  
  
print(a+b)  
print(int.\_\_add\_\_(a,b))  
print(int.\_\_mul\_\_(a,b))  
print(int.\_\_sub\_\_(a,b))

**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  
  
s1 = student(86,96)  
s2 = student(54,63)  
  
s3 = s1+s2  
print(s3.m2)

**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): *# for greater then* g1 = self.m1 + self.m1  
 g2 = self.m2 + self.m2  
  
 **if** g1 > g2:  
 **return True  
 else**:  
 **return False**s1 = student(86,96)  
s2 = student(54,63)  
  
s3 = s1+s2  
print(s3.m2)  
  
**if** s1 > s2:  
 print(**"s1 wins"**)  
**else**:  
 print(**"s2 wins"**)

**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): *# for greater then* g1 = self.m1 + self.m1  
 g2 = self.m2 + self.m2  
  
 **if** g1 > g2:  
 **return True  
 else**:  
 **return False  
 def** \_\_str\_\_(self):  
 **return** self.m1, self.m2  
s1 = student(86,96)  
s2 = student(54,63)  
  
s3 = s1+s2  
print(s3.m2)  
  
**if** s1 > s2:  
 print(**"s1 wins"**)  
**else**:  
 print(**"s2 wins"**)  
  
print(s1.\_\_str\_\_())

**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): *# for greater then* g1 = self.m1 + self.m1  
 g2 = self.m2 + self.m2  
  
 **if** g1 > g2:  
 **return True  
 else**:  
 **return False  
 def** \_\_str\_\_(self):  
 **return'{} {}'**.format( self.m1, self.m2)  
s1 = student(86,96)  
s2 = student(54,63)  
  
s3 = s1+s2  
print(s3.m2)  
  
**if** s1 > s2:  
 print(**"s1 wins"**)  
**else**:  
 print(**"s2 wins"**)  
  
print(s1)  
print(s2)



## #54 Python Tutorial for Beginners | Method Overloading and Method Overriding in Python

### Method overloading

**class** student():  
 **def** \_\_init\_\_(self,m1,m2):  
 self.m1 = m1  
 self.m2 = m2  
  
 **def** sum(self,a=**None**,b=**None**,c=**None**):  
 **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(58,63)  
print(s1.sum(5,6,8))

### Method Overriding

**class** A:  
 **def** show(self):  
 print(**"This is my father phone Samsung"**)  
**class** B:  
 **pass**s1 = B()  
s1.show()

AttributeError: 'B' object has no attribute 'show' Process finished with exit code 1

**class** A:  
 **def** show(self):  
 print(**"This is my father phone Samsung"**)  
**class** B(A):  
 **pass**s1 = B()  
s1.show()

**class** A:  
 **def** show(self):  
 print(**"This is my father phone Samsung"**)  
**class** B(A):  
 **def** show(self):  
 print(**"This is my own phone iphone"**)  
  
s1 = B()  
s1.show()

## Iterator in Python

**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(next(values))  
  
**for** i **in** values:  
 print(i)

## Generators in Python

**def** topten():  
 n = 1  
 **while** n<= 10:  
 sq = n\*n  
 **yield** sq  
 n+= 1  
  
values = topten()  
  
**for** i **in** values:  
 print(i)

## #55 Python Tutorial for Beginners | Exception Handling in Python

Compile time error

Logical error

Run time error

a = 5  
b = 0  
  
**try**:  
 print(a/b)  
**except** Exception:  
 print(**"Hey, you cannot divide value with zero "**)

a = 5  
b = 0  
  
**try**:  
 print(a/b)  
**except** Exception **as** e:  
 print(**"Hey, you cannot divide value with zero "**, e )

finally will be execute when we got error or we don’t get an error

a = 5  
b = 0  
  
**try**:  
 print(**"Resource Open"**)  
 print(a/b)  
**except** Exception **as** e:  
 print(**"Hey, you cannot divide value with zero "**, e )  
**finally**:  
 print(**"Resource Closed"**)

a = 5  
b = 0  
  
**try**:  
 print(**"Resource Open"**)  
 print(a/b)  
**except** ZeroDivisionError **as** e:  
 print(**"Hey, you cannot divide value with zero "**, e )  
**except** ValueError **as** e:  
 print(**"Hey, you enter invalid value"**, e )  
**except** Exception **as** e:  
 print(**"Other error"**, e )  
  
**finally**:  
 print(**"Resource Closed"**)

## #56 Python Tutorial for Beginners | MultiThreading in Python

**class** Hello:  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hello"**)  
  
**class** Hi:  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hi"**)  
  
  
s1 = Hello()  
s2 = Hi()  
  
s1.run()  
s2.run()

**from** threading **import** \*  
**class** Hello(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hello"**)  
  
**class** Hi(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hi"**)  
  
  
s1 = Hello()  
s2 = Hi()  
  
s1.run()  
s2.run()

**from** time **import** \*  
**from** threading **import** \*  
**class** Hello(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hello"**)  
 sleep(1)  
  
**class** Hi(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hi"**)  
 sleep(1)  
  
s1 = Hello()  
s2 = Hi()  
  
s1.run()  
s2.run()

**from** time **import** \*  
**from** threading **import** \*  
**class** Hello(Thread):  
 **def** run(self):  
 **for** i **in** range(500):  
 print(**"Hello"**)  
 sleep(1)  
  
**class** Hi(Thread):  
 **def** run(self):  
 **for** i **in** range(500):  
 print(**"Hi"**)  
 sleep(1)  
  
s1 = Hello()  
s2 = Hi()  
  
s1.start()  
s2.start()

Hello  
Hi

Hello  
Hi

HiHello

Hello  
Hi

**from** time **import** \*  
**from** threading **import** \*  
**class** Hello(Thread):  
 **def** run(self):  
 **for** i **in** range(500):  
 print(**"Hello"**)  
 sleep(1)  
  
**class** Hi(Thread):  
 **def** run(self):  
 **for** i **in** range(500):  
 print(**"Hi"**)  
 sleep(1)  
  
s1 = Hello()  
s2 = Hi()  
  
s1.start()  
sleep(.2)  
s2.start()

**from** time **import** \*  
**from** threading **import** \*  
**class** Hello(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hello"**)  
 sleep(1)  
  
**class** Hi(Thread):  
 **def** run(self):  
 **for** i **in** range(5):  
 print(**"Hi"**)  
 sleep(1)  
  
s1 = Hello()  
s2 = Hi()  
  
s1.start()  
sleep(.2)  
s2.start()  
  
s1.join()  
s2.join()  
print(**"Bye"**)

## #57 Python Tutorial for Beginners | File handling in Python

## #58 Python Tutorial for Beginners | Comments in Python

## is Python Compiled or Interpreted Language?