PYTHON NOTES

Contents

Lecture#14 - Python as calculator

```
print(9/5)      #will generate floating value
print(9//5)     #will generate integer value
print (2**3)    #will generate 2 power 3
print(2**0.5)   #will generate answer till about
10 value after decimal value
print(round(2**0.5, 4)) #will generate answer till
four round off
print(2**3**2)  #will generate first 3^2 = 9, then
2^9 = 512
```

Lecture#15 - Variables in Python

```
# we can store number and string in variable in
python
_name = "awinash";
print(_name)
# example of snake_case_writing
# example of camelCase
```

Lecture#16 - String Concatation

```
firstName = "awi"
lastName = "goswami"
fullName = firstName + " " + lastName
print(fullName)
# string can be concatenated with string not with
number
# print(fullName + 3)   #error
print(fullName + "3") #no error
print(fullName + str(3)) #no error
print(fullName * 4) #string multiplication
```

Lecture#17 - User Input

```
# input function
name = input("type your name: ")
print("hello " + name)

# input function will always accept string
age = input("what is your age? ")
print(age)
```

Lecture#18 - Int() and float function

```
number1 = float(input("enter first number: "))
number2 = float(input("enter second number: "))
sum = number1 + number2
print(str(sum))

# individually
val1 = str(4)
val2 = float("54")
val3 = int("25")
print(val2 + val3)      #output will be in float
```

Lecture#19 - More about variables

```
# multiple variable in single line
name, age = "awi ", 25
print(name + str(age))

x=y=z = 1
```

Lecture#20 - Two or more input in one line

```
name, age = input("enter your name and
age").split()
print(name)
print(age)

name, age = input("enter your name and age with
comma between in").split(",")
print(name)
print(age)
```

Lecture#20 - String Indexing

```
name = "awinash"
print(name[0])           # will print a
# print(name[6]) == print(name[-1])
```

Lecture#21 - String Slicing

```
lang = "python"
#syntax - [start argument: stop argument]
print(lang[1:5])    # will print ytho
print(lang[:6])     # will print python
print(lang[2:]      # will print thon
print(lang[:]       # will print python
```

Lecture #22 - Step Argument

```
#syntax - [start argument : stop argument : step]
print("awinash_goswami"[2:11:2])    # will print
iahgs
print("awinash_goswami"[::-1])      # will print
imawsog_hsaniwa, reverse string
```

Lecture#23 - String Method Part I

```
lower = "awinash_goswami"
upper = "AWINASH_GOSWAMI"

print(len(lower))       # will count length of
lower variable including spaces
print(lower.upper())    # will make all characters
in uppercase
print(upper.lower())    # will make all characters
in lowercase
print(lower.count("a")) # will count number of
occurrence of specific character
```

Lecture#24 - Strip Method

```
name = "    Awinash       "
name1 = "   Awi       nash"

print(name)                     # will print as
usual with space
print(name.lstrip())            # will remove
spaces from left side
print(name.rstrip())            # will remove
spaces from right side
print(name.strip())             # will remove
spaces from both sides
print(name1.replace(" ", ""))   # will replace
spaces with no spaces
```

Lecture#25 - replace() and find() method

```
string = "My name is Awinash"
print(string.replace("a","i"))              #
will replace a with i
print(string.lower().replace("a", "i", 2))  #
will replace a with i after index 2
#find method is used to find position of specific
character and word
print(string.find("n"))                     #
will find character n and return n's index
print(string.find("n", 7))                  #
will find character after 7th index
```

Lecture#26 - Center method with program

```
name = "Awinash"
print(name.center(15, "*"))     #len().name is 7,
15-7 = 8. It will print eight characters, four at
left and four at right
```

Lecture#27 - Strings are immutable

```
string = "awinash"
string.replace('i', 'I')
new_string = string.replace('i', 'I')
print(string)       # will print awinash rather
awInash because string are immutable
print(new_string)   # will print awInash now
because now new string has been created
```

Lecture#28 -More Assignment Operators

```
name = "awinash"
name = name + "_goswami"
print(name)         #will print awinash_goswami
```

Lecture#29 -If Statement

```
age = int(input("Enter your age: "))
if age >= 14:
    print("you are above 14")
```

Lecture#30 - Pass Statement

```
x = 18
if x>18:
    pass        #pass is keyword in python, if you don't want to write anything in if then write pass otherwise if you leave it blank, an error will occur
```

Lecture#31 - else Statement

```
age = int(input("Enter your age: "))
if age > 14:
    print("you are above 14")
else:
    print("you are below 14")
```

Lecture#32 - nested if else

```
winning_number = int(input("Enter any number below 10: "))
if 10==winning_number:
    print("You won")
else:
    if 10<winning_number:
            print("too high")
    if 10>winning_number:
            print("too low")
```

Lecture#33 - and, or operator

```
#checking two conditions at same time
# and, or

# both variable should match for true condition
name1 = 'able'
age1 = 19
if name1=='able' and age1==19:
    print("approved")
else:
    print("disapproved")

# either anyone or both variable should match for true condition
name2 = "disable"
age2 = 69
if name2=="disable" or age2==99:
    print("able")
else:
    print("disable")
```

Lecture#34 - if...elif...else statement

```
age = int(input("Enter your age: "))

if age==0 or age<0:
    print("Invalid input")
elif 1<=age<=3:
    print("Free")
elif 3<age<=10:
    print("Ticket Price: 150")
elif 10<age<=20:
    print("Ticket Price: 250")
else:
    print("Ticket Price: 300")
```

Lecture#35 - in keyword

```
# to find presence of a specific letter in a word
name = "awinash"
if 'a' in name:
    print("yes")
else:
    print("no")
```

Lecture#36 - check empty or not

```
#check empty or not
name = input("Enter your name: ")
if name:
#true if string is not empty
    print(f"your name is {name}")
else:
    print("you did not type anything")
```

Lecture#36 - while loop

```
#while loop
i = 1
while i<=10:
    print(f" {i} - awi jani")
    i += 1
```

Lecture#37 - for loop

```
for i in range(10):
    print(f"{i+1} - hello")  # i's by default value is 0
```

Lecture#38 - break and continue keyword

```
#break and continue keyword

for i in range(11):
    print(i)
    if (i==5):
        break       #loop will stop and program
terminate

print("-------------")

for i in range(11):
    if (i==5):
        continue
    print(i)
           #loop will not print 5 and program
continues
```

Number Guessing Game

```
import random
random_number = random.randint(1,100)
print(random_number)
user_guess = int(input("Enter your guess between 1
and 100: "))
frequency = 1
game_over = False
while not game_over:
    if user_guess==random_number:
        print(f"you win, you guessed {frequency}
times")
        game_over = True
    else:
        if user_guess < random_number:
            print("too low")
        else:
            print("too high")
        frequency += 1
        user_guess = int(input("Enter your guess
between 1 and 100: "))
```

Lecture#39 - Step Argument in Range function

```
for i in range(0,11,2):
    print(i)            # will print 0,2,4,6,8,10
print("-----------")

for i in range(10,-1,-2):
    print(i)            # will print 10,8,6,4,2,0
```

Lecture#40 - for loop in string

```
print("----------way 1------------")
name = "awinash"
for i in name:
    print(i)

print("----------way 2------------")

number = input("enter a number: ")
total = 0
for i in number:
    total += int(i)
print(total)
```

Lecture#41 - function intro

```
a = int(input("first: "))
b = int(input("second: "))
def add_two_numbers(x,y):
    return x+y
print(add_two_numbers(a,b))
```

Palindrome Example

```
def is_palindrome(word):
    return word == word[::-1]

name = input("Enter: ")
print(is_palindrome(name))
```

Fibonacci Example

```
def fibonacci(digit):
    a = 0
    b = 1
    if digit == 1:
        print(a)
    elif digit == 2:
        print(a, b)
    else:
        print(a,b, end = ", ")
        for i in range(digit-2):
            c = a+b
            a = b
            b = c
            print(b, end = ", ")

fibonacci(10)
```

Lecture#42 - variable scope

```
# scope
x = 5 # global variable

def func():
    global x
    x = 7 # local variables
    return x

print(x)
print(func())
print(x)
```

Lecture#43 - Intro to List

```
#list is used for storing more than one variable in
single storage
#variable can be both mixed and of same type
numbers = ["awinash", "goswami", 4,5,6,7, 4.3,
None]
print(numbers)
```

Lecture#44 - Adding data into list

```
fruits = ["apple", "grapes"]
print(fruits)
fruits.append("mango")     #will add mango in end
of list
print(fruits)
```

Lecture#45 - More methods to add data

```
#insert method is used to add data at any position
fruits1 = ["mango", "orange"]
fruits1.insert(1, "grapes")
print(fruits1)


# + is used to concatenate lists
fruits2 = ["papaya", "watermelon"]
fruits = fruits1 + fruits2
print(fruits)

# extend method is used to add all elements of one
list in another list
fruits1.extend(fruits2)
print(fruits1)

# append method is also used to append list in list
fruits1.append(fruits2)
print(fruits1)
```

Lecture#46 - Delete data from list

```
fruits = ['orange', 'apple', 'mango', 'banana',
'chikoo']

#pop method is used to remove last element from list
fruits.pop()
print(fruits)

#now pop method will remove 2 index element
fruits.pop(2)
print(fruits)

#del method is also used to remove element
del fruits[1]
print(fruits)

#remove method used to remove element by value
fruits.remove("banana")
print(fruits)
```

Lecture#47 - In keyword with list

```
fruits = ['orange', 'apple', 'mango', 'banana',
'chikoo']
if 'apple' in fruits:
    print("available")
else:
    print("not")
```

Lecture#48 - Some more list method

```
#count method will count number of occurrence of
particular element
fruits = ['orange', 'apple', 'mango', 'banana',
'chikoo', 'guava', 'apple']
print(fruits.count("apple"))

#sort method to sort elements in alphabetical order
fruits.sort()
print(fruits)

#sorted method used for only printing after sort
number = [4,6,2,1,8,9,4,3]
print(sorted(number))

#clear method will empty the list
number.clear()
print(number)

#copy method used to copy all elements in list
copy_fruits = fruits.copy()
print(copy_fruits)
```

Lecture#48 - is vs equal

```
#list comparison
#is, ==
fruits1 = ['orange', 'apple', 'mango']
fruits2 = ['banana', 'chikoo', 'guava', 'apple']
fruits3 = ['orange', 'apple', 'mango']

print(fruits1 == fruits2) #values are different
hence False
print(fruits1 == fruits3) #values are same hence
True

print(fruits1 is fruits2) # will print false
because 'is' is used to check whether two object are
stored at same place in memory
```

Lecture#48 - split and join method

```
#split method used to convert string to list
user_info_split = "awinash 24".split()
print(user_info_split)

# join method used to convert list to string
user_info_join = ["awinash", "24"]
print(','.join(user_info_join))
```

Lecture#49 - List inside list

```
# accessing all elements of list
matrix =  [[1,2,3], [4,5,6], [7,8,9]]
for sublist in matrix:
    for i in sublist:
        print(i, end = " ")


# accessing single element of list
print("\n")
print(matrix[1][2])


#type function is used find out data’s type
name = "awi"
print(type(name))
```

Lecture#49 - More about list

```
# generate lists with range function
numbers = list(range(1,11))  # will generate a list
from 1 to 10
print(numbers)

# pop method also returns pop value
poped_value = numbers.pop()
print(poped_value)

# index method used to find position of element
print(numbers.index(1))

# passing list in function
def negative_list(l):
    temp_list = []
    for i in l:
        temp_list.append(-i)
    return temp_list

print(negative_list(numbers))
```

Lecture#50 – Intro to Tuples

```
# tuple data structure
# tuple can store any data type
# most important tuples are immutable, once tuple
is created you can't update
# data inside tuple
# tuple are faster than list in performance
# tuples are used when we know data is not be changed
e.g. name of days

example = ("Monday", "Tuesday", "Wednesday")

# methods for tuples
# count, index, len, slicing
```

Lecture#51 – More about Tuples

```
# looping in tuples
# tuple with one element
# tuple without parenthesis
# tuple unpacking
# list inside tuple
# some functions we can use with tuples

mixed = (1,2,3,4.0)

# for loop
for i in mixed:
    print(i, end = " ")


# tuple with one element
numIn = (1)      # is not tuple but integer
numTu = (1,)     # is tuple, you must add ,


# tuple without parenthesis
fruits = "banana", "chikoo", "orange"
print(type(fruits))


# tuple unpacking
bands = ("vital signs", "string", "junoon")
band1, band2, band3 = (bands)
print(band1)


# list inside tuples
countries = ("Pakistan", ["India", "Germany"])
print(countries)
popped = countries[1].pop()
print(popped)
countries[1].append("France")
print(countries)
```

Lecture#52 – More about Tuples Part II

```
# something more about tuples, list, str

nums = tuple(range(1,11))
print(nums)

# from tuple to list
nums = list((1,2,3,4,5))
print(nums)
print(type(nums))

# from tuple to str
nums = str((1,2,3,4,5))
print(nums)
print(type(nums))
```

Lecture#53 – Intro to Dictionary

```
# dictionaries intro
# we use dictionaries because of limitation of
lists, lists are not enough


# Q. what are dictionaries
# A. unordered collections of data in key: values
pair.

# how to create dictionaries
user1 = {'name':'awi', 'age':24}
print(user1)
print(type(user1))


# second method to create dictionary
user2 = dict(name = "awi", age="24")
print(user2)


# how to access data from dictionary
# Note: there is no indexing in dict because of
unordered collection
```

```
print(user1['name'])
print(user1['age'])


# what type of data can be stored in dictionary
# anything, i.e. numbers, strings, lists,
dictionary

user_info = {
    'name' : 'awi',
    'age'  : '24',
    'fav_movies' : ["kungfu panda1", "big hero"],
    'fav_tones'  : ["iphone", "morning"],
}
print(user_info['fav_movies'])

# how to add data to empty dictionary
user_info2 = {}
user_info2['name'] = "awinash"
user_info2['age']  = 34
print(user_info2)
```

Lecture#54 – Looping and in keyword

```
# in keyword and iterations in dictionary

user_info = {
    'name' : 'awi',
    'age'  : 24,
    'fav_movies' : ["kungfu panda1", "big hero"],
    'fav_tones'  : ["iphone", "morning"],
}

# check if key exist in dictionary
if 'name' in user_info:
    print("present")
else:
    print("not present")


# check if value exists in dictionary
if ["kungfu panda1", "big hero"] in
user_info.values():
    print("present")
else:
    print("not present")

# loops in dictionaries
for i in user_info.values():
    print(i)

# values method
user_info_values = user_info.values()
print(user_info_values)
print(type(user_info_values))

# keys method
user_info_keys = user_info.keys()
print(user_info_keys)
print(type(user_info_keys))

# items method
user_items = user_info.items()
print(user_items)
print(type(user_items))

for key, value in user_info.items():
    print(f "key is {key} and value is {value}")
```

Lecture#54 – Add and delete data from dictionary

```
# # add and delete data

user_info = {
    'name' : 'awi',
    'age'  : 24,
    'fav_movies' : ["kungfu panda1", "big hero"],
    'fav_tones'  : ["iphone", "morning"],
}

#  how to add data
user_info['fav_songs'] = ['song1', 'song2']
print(user_info)


# pop method
popped_item = user_info.pop('fav_movies')
print(user_info)


# popitem method, popitem() will return tuple
popped_item = user_info.popitem()   #will
randomly delete any key value pair
print(user_info)
```

Lecture#55 – update method dictionary

```
# update method()

user_info = {
    'name' : 'awi',
    'age'  : 24,
    'fav_movies' : ["kungfu panda1", "big hero"],
    'fav_tones'  : ["iphone", "morning"],
}

more_info = {'State': 'Sindh', 'hobbies':
['learning python', 'working hard']}

user_info.update(more_info)
print(user_info)
```

Lecture#56 – fromkeys, get, clear, copy method

```
# fromkeys

# d = {'name': 'unknow', 'age':'unknow'}

d = dict.fromkeys(('name', 'age', 'height'), ' ')
print(d)

# get method (useful), used to get key that is not
in dictionary rather to return error
d = {'name': 'unknown', 'age':'unknown'}
print(d['name'])
print(d.get('names'))


if d.get('name') in d:
    print("present")
else:
    print('false')

# to empty dictionary
d.clear()
print(d)

#copy method
d1 = d.copy()
print(d1)

d1 = d  #same dictionary but not copy
```

Lecture#57 – More about get() method

```
user = {'name' : 'awi', 'age':24}
print(user.get('name'))
print(user.get('names'))    #will return none
print(user.get('names', 'not found'))    #now it
will return “not found” instead of none


user = {'name' : 'awi', 'age':24, 'age': 34}
print(user) # second age will override first one
```

Lecture#57 – Word Counter Dictionary

```
def word_counter(s):
    count = {}
    for i in s:
        count[i] = s.count(i)
    return count

print(word_counter("awinash"))
```

Lecture#58 – Intro to Sets

```
# set data type
# unordered collection of unique items

s = {1,2,2,3}
print(s)
# print(s[i])         #error, set does not support
indexing because of unordered collection

l = [1,2,3,4,5,5,5,5,6,7,7,8]

s2 = set(l)   #will convert l list into set, hence
items will be unique
s3 = list(set(l))   #now after set, list has all
unique items
print(s2)
print(s3)


# add method
s.add(4)
print(s)
```

```
# remove method
s.remove(3)
print(s)

# remove vs discard method
s.remove(10)        # 10 is not in set, it will
generate error
s.discard(10)       # 10 is not in set, it will not
generate error
print(s)


# clear method
s.clear()           # will empty set
print(s)

# copy method
s4 = s.copy()
print(s4)


# only int, float, and strings can be stored in set
# tuple,list and dict cannot be saved in set

s9 = {1,2.4,"awi"}
print(s9)
```

Lecture#58 – More about Sets

```
s = {'a', 'b', 'c', 'd'}

# if in set
if 'a' in s:
    print("present")
else:
    print('not present')


# for loop in set
for i in s:
    print(i)


# set math
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}

# union
print(s1 | s2)
# Note s1.union(s2) == s1 | s2

# intersection
print(s1 & s2)
# Note s1.intersection(s2) == s1 & s2
```

Lecture#59 – What is list comprehension?

```
# list comprehension
# with help of list comprehension we can create list
in one line

# create a list of squares from 1 to 10

# example1
# general method
square = []
for i in range(1,11):
    square.append(i**2)
print(square)

# using list comprehension method

square2 = [i**2 for i in range(1,11)]
print(square2)


# example2
# general method
negative_cube1 = []
for i in range(1,11):
    negative_cube1.append(-i**3)
print(negative_cube1)


# using list comprehension method
negative_cube2 = [-i**3 for i in range(1,11)]
print(negative_cube2)


# example3
# general method
names = ["awinash", 'goswami', 'tandojam']
first = []
for name in names:
    first.append(name[0])
print(first)
```

```python
# using list comprehension
new_list = [name[0] for name in names]
print(new_list)
```

Lecture#60 – Exercise 1

```python
# define a function that takes list of string
# list containing reverse of every string

# NOTE - USE LIST COMPREHENSION

def reverse_string(l):
    return [name[::-1] for name in l]

strings = ["awinash", "goswami", "maharaj"]
print(reverse_string(strings))
```

Lecture#60 – List Comprehension with if method

```python
# list comprehension with if statement

numbers  = list(range(1,11))
nums = []
# general method
for i in numbers:
    if i%2==0:
        nums.append(i)
print(nums)


# list comprehension method
even_num = [i for i in numbers if i%2==0]
odd_num  = [i for i in range(1,11) if i%2!=0]
print(even_num)
print(odd_num)
```

Lecture#60 – List Comprehension with if method

```python
def filtered(l):
    return [str(i) for i in l if (type(i)== int or type(i) == float)]

print(filtered([True, False, [1,2,3], 1,1.0,5]))
```

Lecture#61 – List Comprehension with if else

```python
# list comprehension with if else

nums = list(range(1,11))
news_list = []

# general method
for i in nums:
    if i%2==0:
        news_list.append(i*2)
    else:
        news_list.append(-i)

print(news_list)


# using list comprehension
news_list2 = [i*2 if (i%2==0) else -i for i in nums]
print(news_list2)
```

Lecture#62 – List Comprehension with nested list

```python
example = [[1,2,3], [1,2,3], [1,2,3]]

nested_comp = [[i for i in range(1,4)] for j in range(3)]
print(nested_comp)
```

Lecture#63 – Dictionary Comprehension

```python
# dictionary comprehension

# example 1
square  = {num:num**2 for num in range(1,11)}
```

```
print(square)

# example 2
string = "awinash"
word_count = {char:string.count(char) for char in
string}
print(word_count)
```

Lecture#63 – Dictionary Comprehension with if-else

```
# dictionary comprehension with if else
# d = {1:'odd', 2:'eve'}
odd_even = {i:('even' if i%2==0 else 'odd') for i
in range(1,11)}
print(odd_even)
```

Lecture#64 – Sets Comprehension

```
# sets comprehension

# example 1
s = {k**2 for k in range(1,11)}
print(s)

# example 2
names = ['Awinash', 'Goswami', 'Tando jam']
first = {name[0] for name in names}
print(first)
```

Lecture#65 – Intro to *args

```
# make flexible functions ------------(self
assigned name)
# *operator
# *args
# *args behaves as tuple

# example 1
def all_total(*args):
    print(args)
    print(type(args))

all_total(1,2,3,4,5,6)
print(all_total)
```

```
# example 2
def sum_all(*args):
    sum = 0
    for i in args:
        sum += i
    return sum

print(sum_all(3,4,4,7,11,98))
```

Lecture#66 – *args with normal parameter

```
# *args with normal parameter

def multiply_nums(num, *args):
    multiply = 1
    print(num)
    print(args)
    for i in args:
        multiply *= i
    return multiply

print(multiply_nums(2,3,4,5))
```

Lecture#67 – *args as argument

```
def multiply_nums(*args):
    multiply = 1
    print(args)
    for i in args:
        multiply *= i
    return multiply

nums1 = [2,3,4,5]
nums2 = (3,4,5,6)

print(multiply_nums(*nums1))
print(multiply_nums(*nums2))

# Note: while using list or tuple as argument, we
must attach '*'
```

Lecture#68 – Exercise

```
def multiply(num, *args):
    if args:
        return [i**num for i in args]
    else:
        return "you did not pass any arg"

nums = [2,3,4,5]
print(multiply(3,*nums))
```

Lecture#69 – **kwargs

```
# kwargs (keyword arguments)
# **kwargs (double start operator)

# kwargs as a parameter
# **kwargs behaves as dictionary

# method 1
def func1(**kwargs):
    print(kwargs)
    print(type(kwargs))

func1(first_name="awi", last_name="goswami")


# method 2
def func2(**kwargs):
    for k,v in kwargs.items():
        print(f"{k} : {v}")

func1(first_name="John", last_name="Smith")



# method 3
def func3(name, **kwargs):
    print(name)
    for k,v in kwargs.items():
        print(f"{k} : {v}")

# dictionary unpacking
d = {
    'name': 'awi',
    'age' : 24
}


func3(**d)
```

Lecture#69 – function with all type of parameters

```
#  function will all parameters
# very important to understand

# PADK

# parameters
# args
# default parameters
# kwargs
# order must be followed as per above

def func(name, *args, last_name = 'unknown',
**kwargs):
    print(name)
    print(args)
    print(last_name)
    print(kwargs)

print(func('awinash', 345, a = 1, b = 2))
```

Lecture#70 – Exercise

```
# function
# list, reverse_str = True
# list

def func(l, **kwargs):
    if kwargs.get('reverse_str') ==  True:
        return [name[::-1].title() for name in l]
    else:
        return [name.title() for name in l]


names = ['awinash', 'goswami']
print(func(names, reverse_str = True))
```

Lecture#70 – Lambda function/expression

```python
# lambda expression (anonymous function)

# normal function
def add(a,b):
    return a+b

# lambda function/expression
# lambda function/expression is helpful for
declaring function in one line

add2 = lambda a,b : a+b

print(add2(3,4))

multiply = lambda a,b : a*b
print(multiply(5,3))
```

Lecture#71 – Lambda function/expression practice

```python
# lambda expression practice

# normal function
def is_even(a):
    return a%2==0
print(is_even(9))


# lambda function
is_even2 = lambda a: a%2==0
print(is_even2(9))


def last_char(s):
    return s[-1]

last_char1 = lambda s : s[-1]
print(last_char1("stop"))


# lambda with if, else
def func(string):
    return len(string) > 5

length = lambda strings : len(strings)> 5
print(length("awinash"))
```

Lecture#72 – Enumerate function

```python
# enumerate function

# we use enumerate function with for loop to track
position of our
# item in iterable

# without enumerate function

names = ['awinsh', 'goswami', 'tandojam']
pos = 0
for i in names:
    print(f"{pos} - {i}")
    pos += 1

# with enumerate function

for pos, name in enumerate(names):
    print(f"{pos} - {names}")


# Define a function that takes two arguments
# 1. list containing string
# 2. string that want to find in your list
# and this function will return the index of string
in your
# list and if string is not present then return -1

def func(l, strings):
    for pos, name in enumerate(l):
        if name == strings:
            return pos
    return -1

cities = ["Tando Jam",  "Hyderabad"]
print(func(cities, "Hyderabad"))
```

Lecture#73 – map function

```python
# map function

def squares(a):
    return a**2

numbers = [1,2,3,4]

square = list(map(squares, numbers))
```

```
print(square)


# using lambda expression

new_squares = list(map(lambda a:a**2, numbers))
print(new_squares)
```

Lecture#73 – map function

```
# filter function

numbers = [1,2,3,4,5,6,7,8,9]

def is_even(a):
    return a%2 == 0

evens = tuple(filter(is_even, numbers))
print(evens)
```

Note: Please Google iterator VS iterable

Lecture#74 – Zip function

```
# zip function

users_id = ['user1', 'user2', 'user3']
first_names = ['awinash', 'parshant', 'govinda']
last_names = ['goswami', 'goswami', 'goswami']
print(dict(zip(users_id, first_names)))
print(dict(zip(users_id, first_names,
last_names)))  #error, dict takes only two
parameters
print(list(zip(users_id, first_names,
last_names)))
```

Lecture#75 – Zip function part II

```
l1 = [1,3,5,7]
l2 = [2,4,6,8]

l = [(1,2), (3,4), (5,6), (7,8)]

# * operator with zip
l1, l2 = list(zip(*l))
print(list(l1))
print(list(l2))

new_list = []
for pair in zip(l1, l2):
    new_list.append(max(pair))

print(new_list)
```

Challenge

```
# Challenge

def average_finder(*args):
    average = []
    for pair in zip(*args):
        average.append(sum(pair)/len(pair))
    return average

l1, l2, l3 = [1,2,3], [4,5,6], [7,8,9]
print(average_finder(l1,l2,l3))

# using lambda expression

average_finder_lambda = lambda *args:
[sum(pair)/len(pair) for pair in zip(*args)]
print(average_finder_lambda(l1,l2,l3))
```

Lecture#76 – all and any function

```
# any, all function

numbers1 = [2,4,6,8,10]
numbers2 = [1,3,5]

print([all(num%2==0 for num in numbers1)])
print([any(num%2==0 for num in numbers2)])
```

Lecture#77 – all and any function practice

```
def my_sum(*args):
    if all([type(arg) ==  int or type(arg) == float
for arg in args]):
        total = 0
        for arg in args:
            total += arg
        return total
    else:
        return "Wrong input"

print(my_sum(1,2,3,4,5.7))
print(my_sum(1,2,3,4,5.7, "awi"))
```

Lecture#77 – advance min and max function

```
# advance min() and mix()

students1 = {
    'awinash' : {'score':50, 'age': 23},
    'labesh'  : {'score':60, 'age': 24},
    'parshant'  : {'score':70, 'age': 25}
}

print(max(students1, key = lambda item:
students1[item]['score']))

students2 = [
    {'name': 'awinash', 'score': 90, 'age':24},
    {'name': 'labesh', 'score': 100, 'age':25},
    {'name': 'rahul', 'score': 110, 'age':26},
]

print(max(students2, key = lambda item:
item.get('score'))['name'])
```

Lecture#78 – sorted function in advance

```
# Advance sorted function

cars = ('BMW', 'AUDI', 'Chevrolet', 'Mitsubhi')
sorted(cars)
print(cars)

# Note: tuples are immutable, hence they are showing
same result
# However here we can see them sorted using sorted
function with print
print(sorted(cars))

guitars = [
    {'model': 'yamaha f310', 'price': 8400},
    {'model': 'faith naptune', 'price': 54000},
    {'model': 'faith apollo venus', 'price':
35000},
    {'model': 'taylor 814ce', 'price': 450000},
]

sorted_guitars =  sorted(guitars, key = lambda d:
d['price'])
# sorted in ascending order w.r.t price
print(sorted_guitars)

sorted_guitars2 = sorted(guitars, key = lambda d:
d['price'], reverse=True)
# sorted in descending order w.r.t price
print(sorted_guitars)
```

Lecture#79 – More about functions

```
# what are doc strings
# how to write docstrings
# how to see docstrings
# what is help function

def add(a,b):
    '''this function takes 2 numbers and return
their sum \n'''
    return a+b


print(add.__doc__)
print(sorted.__doc__)
print(help(sum))
```

Lecture#80 – Decorator chapter intro

```
#  First class function/closure
# then finally we will learn about decorators

def square(a):
    return a**2
```

```
s = square          # now s will be treated as
square()
print(s(8))
print(s.__name__)
print(square.__name__)
print(s)
print(square)       # that's why both s and square
have same memory location
```

Lecture#81 – Function as argument

```
# function as argument

l = [1,2,3,4]

def square(a):
    return a**2

def my_map(func, l):
    new_list = []
    for item in l:
        new_list.append(func(item))
    return new_list

print(my_map(square, l))
```

Lecture#82 – Function returning function

```
# function returning function

def outer_func(msg):
    def inner_func():
        print(f" message is {msg}")
    return inner_func

func_var = outer_func("Hello")
func_var()
```

Lecture#83 – Closure Practice

```
# function returning function (closures) practice
# also called first class function
# practical example

def to_power(x):
    def cal_power(n):
        return n**x
    return cal_power

cube = to_power(3)
print(cube(2))

square = to_power(2)
print(square(4))
```

Lecture#84 – Decorator Intro

```
# Decorators - enhance the functionality of other
functions
# @ use for decorator - called syntactic sugar

def decorator_function(any_function):
    def wrapper_function():
        print("This is awesome function")
        any_function()
    return wrapper_function

@decorator_function
def func1():
    print("This is function 1")

func1()
```

Lecture#85 – Decorator Intro Part II

```
def decorator_function(any_function):
    def wrapper_function(*args, **kwargs):
        print("This is awesome function")
        return any_function(*args, **kwargs)
    return wrapper_function

@decorator_function
def func1(x):
    print(f"This is function1 with argument {x}")

func1(5)

@decorator_function
def add(a,b):
    return a+b
```

```python
print(add(5,3))
```

Lecture#86 – Decorator Intro Part III

```python
from functools import wraps

def decorator_function(any_function):
    @wraps(any_function)
    def wrapper_function(*args, **kwargs):
        """this is wrapper function"""
        print("This is awesome function")
        return any_function(*args, **kwargs)
    return wrapper_function


@decorator_function
def add(a,b):
    '''this is add function'''
    return a+b

print(add.__doc__)
print(add.__name__)
```

Lecture#87 – Decorator Practice

```python
# decorator practice

from functools import wraps

def print_function_data(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        print(f"You are calling
{function.__name__} function")
        print(f"{function.__doc__}")
        return function(*args, **kwargs)
    return wrapper

@print_function_data
def addition(a,b):
    '''This function takes two numbers as argument
and return their sum'''
    return a+b

print(addition(4,9))
```

Lecture#88 – Decorator Exercise

```python
# exercise decorator

from functools import wraps
import time

def calculate_time(function):
    @wraps(function)
    def wrapper_function(*args, **kwargs):
        print(f"Executing function
{function.__name__}")
        t1 = time.time()
        returned_val = function(*args, **kwargs)
        t2 = time.time()
        t = t2 - t1
        print(f"This function took {t} sec")
        return returned_val
    return wrapper_function

@calculate_time
def square_finder(n):
    return [i**99 for i in range(1,n+1)]

square_finder(100000)
```

Lecture#89 – Decorator Practice

```python
from functools import wraps
def only_int_allow(function):
    @wraps(function)
    def wrapper(*args, **kwargs):

        # method 1
        data_types = []
        for arg in args:
            data_types.append(type(arg)==int)
        if all(data_types):
            return function(*args, **kwargs)
        else:
            print("Invalid arguments")
    return wrapper

        # method 2
        # if all([type(arg) == int for arg in
args]):
        #     return function(*arg, **kwargs)
        # print("Invalid argument")

@only_int_allow
def add_all(*args):
    total = 0
    for i in args:
        total += i
    return total

print(add_all(1,2,3,4,5,6,7))
```

Lecture#89 – Decorator with arguments

```
from functools import wraps
def only_data_type_allow(data_type):
    def decorator(function):
        @wraps(function)
        def wrapper(*args, **kwargs):
            if all([type(arg) == data_type for arg
in args]):
                return function(*args, **kwargs)
            print("Invalid argument")
        return wrapper
    return decorator

only_data_type_allow(str)
def string_join(*args):
    string = ''
    for i in args:
        string = string + " " + i
    return string

print(string_join("awinash", "goswmai"))
```

Lecture#90 – Generator Example

```
# create your first generator with generator
function

def nums(n):
    for i in range(1, n+1):
        yield i

numbers = nums(10)

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

Lecture#91 – Generator Comprehension

```
# Generator comprehension

square = (i**2 for i in range(1,11))
print(square)

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

Lecture#92 – OOP – Create your first class

```
# OOP - Create your first class

# WHAT IS CLASS
# HOW TO CREATE A CLASS
# WHAT IS INIT METHOD, constructor
# WHAT ARE ATTRIBUTES, INSTANCE VARIABLE
# HOW TO CREATE OUR OBJECT


class Person:
    def __init__(self, first_name, last_name,
age):
        # instance variable
        print("init method called")
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

p1 = Person("Awinash", "Goswami", 25)
p2 = Person("Parshant", "Goswami_sahab", 22)

print(p1.first_name)
print(p2.last_name)
```

Lecture#93 – OOP – Instance Method

```
# Instance Method

class Person:
    def __init__(self, first_name, last_name,
age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def full_name(self):
        return f"{self.first_name}
{self.last_name}"

    def is_above_18(self):
        return self.age>18

p1 = Person("awi", "goswami", 24)
print(p1.is_above_18())
# print(Person.is_above_18(p1)) is same as
print(p1.is_above_18())
```

Lecture#94 – OOP – Class variable

```
# class variable

class Circle:
    pi = 3.14
    def __init__(self, radius):
        self.radius = radius

    def cal_circumference(self):
        return 2*Circle.pi*self.radius

    def cal_area(self):
        return Circle.pi*self.radius**2

c1 = Circle(4)
print(f"area is {c1.cal_area()}")
print(f"circumference is
{c1.cal_circumference()}")
```

Lecture#95 – OOP – Class Method

```
class Person:
    count_ins = 0
    def __init__(self, first_name, last_name,
age):
        Person.count_ins += 1
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    @classmethod
    def count_instances(cls):
        return f"You have created {cls.count_ins}
instances of {cls.__name__} class"

    def full_name(self):
        return f"{self.first_name}
{self.last_name}"

    def is_above_18(self):
        return self.age>18

p1 = Person("awi", "goswami", 24)
p2 = Person("Parshant", "goswami", 22)

print(Person.count_instances()) # class method
```

Lecture#96 – OOP – Static Method

```
class Person:
    count_ins = 0
    def __init__(self, first_name, last_name,
age):
        Person.count_ins += 1
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    @classmethod
    def count_instances(cls):
        return f"You have created {cls.count_ins}
instances of {cls.__name__} class"

    @staticmethod
    def hello_example():
        print("Hello static method")

    def full_name(self):
        return f"{self.first_name}
{self.last_name}"

    def is_above_18(self):
        return self.age>18

p1 = Person("awi", "goswami", 24)
p2 = Person("Parshant", "goswami", 22)
```

```
print(Person.count_instances()) # class method
print(Person.hello_example())
```

Lecture#97 – OOP – property_setter_decorator

```
class Phone:
    # constructor
    def __init__(self, brand, model_name, price):
        self.brand = brand
        self.model_name = model_name
        self._price = max(price,0)

    @property
    def complete_specification(self):
        return f"{self.brand} {self.model_name}
and price is {self._price}"

    # Note: In python, first write getter then
instantly setter after it

    # getter()
    @property
    def price(self):
        return self._price

    # setter()
    @price.setter
    def price(self, new_price):
        self._price = max(new_price, 0)

phone1 = Phone("Nokia", '1100', 1000)
print(phone1.complete_specification)
```

Lecture#98 – OOP – Inheritance Intro

```
# inheritance intro

class Phone:                    #Base/Parent
Class
    # constructor
    def __init__(self, brand, model_name, price):
        self.brand = brand
        self.model_name = model_name
        self._price = max(price,0)

    def full_name(self):
        return f"{self.brand} {self.model_name}"

    def make_a_call(self, number):
        return f"calling {number}..."

class SmartPhone(Phone):        #Derived/Child
class
    def __init__(self, brand, model_name, price,
ram, internal_memory, rear_camera):
        super().__init__(brand, model_name,
price)
        self.ram = ram
        self.internal_memory = internal_memory
        self.rear_camera = rear_camera


phone = Phone("Nokia", '1100', 1000)
smartphone = SmartPhone('Samsung', 'A7', 30000,
'3GB', '32GB', '13MP')
print(phone.full_name())
print(smartphone.full_name() + f" and price is
{smartphone._price}")
```

Lecture#99 – OOP – Multilevel Inheritance, MRO, method overriding, isinstance(), issubclass()

```
# can we derive more than one class from base class?
# multilevel inheritance
# method resolution order MRO
# method overriding
# isinstance(), issubclass()


class Phone: #base/parent Class
    # constructor
    def __init__(self, brand, model_name, price):
        self.brand = brand
        self.model_name = model_name
        self._price = max(price,0)

    def full_name(self):
        return f"{self.brand} {self.model_name}"
    def make_a_call(self, number):
        return f"calling {number}..."

class SmartPhone(Phone): #derived/child class
    def __init__(self, brand, model_name, price, 
ram, internal_memory, rear_camera):
        super().__init__(brand, model_name, 
price)
        self.ram = ram
        self.internal_memory = internal_memory
        self.rear_camera = rear_camera

    def full_name(self):
        return f"{self.brand} {self.model_name} 
and cost  is {self._price}"

class FlagshipPhone(SmartPhone):
    def __init__(self, brand, model_name, price, 
ram, internal_memory, rear_camera, front_camera):
        super().__init__(brand, model_name, 
price, ram, internal_memory, rear_camera)
        self.front_camera = front_camera



flagshipPhone = FlagshipPhone('OnePlus', '5', 
50000, '6GB', '64GB', '13MP', '16MP')
print(flagshipPhone.full_name())

#Method Resolution Order - MRO
# print(help(flagshipPhone))

# isinstance()
print(isinstance(flagshipPhone, SmartPhone))


# issubclass()
print(issubclass(SmartPhone, Phone))
```

Lecture#100 – OOP – Multiple Inheritance

```
# multiple inheritance
class A:

    def class_a_method(self):
        return 'I am just a class A method'

    def hello(self):
        return 'hello from class A'

class B:

    def class_b_method(self):
        return 'I am just a class B method'

    def hello(self):
        return 'hello from class B'

class C(A,B):
    pass

instance_c = C()

# Class A hello() method will be printed because see 
the class C inheritance order (A,B)
print(instance_c.hello())
```

Lecture#101 – OOP – Magic/Dunder methods, operator overloading, polymorphism

```
# special magic/methods dunder methods
# operator overloading
# polymorphism

class Phone:
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price

    def phone_name(self):
        return f"{self.brand} {self.brand}"

    # for common user __str__ dunder method
    def __str__(self):
        return f"{self.brand} {self.model} and 
price is {self.price}"

    # for python developer __repr__ dunder method
    def __repr__(self):
        return f"Phone(\'{self.brand}\', 
\'{self.model}\', {self.price})"

    # operator overloading example
    def __add__(self, other):
        return self.price + other.price
```

```
phone1 = Phone("nokia", "1100", 1000)
phone2 = Phone("nokia", "1600", 1200)

print(phone1.__str__())
print(phone1.__repr__())
print(phone1 + phone2)
```

Lecture#102 – Raise errors

```
def add(a,b):
        if (type(a) is int) and (type(b) is int):
                return a+b
        raise TypeError('Oops! You have entered
wrong input. Please enter integer only')

print(add('3', '6'))
```

Lecture#103 – Raise errors Example1

```
# raise errors example 1
# NotImplementError
# abstract method

class Animal:
        def __init__(self, name):
                self.name = name

        # abstact method example. A method in which
we perform nothing but only delivers a message.
        # In python, there is no concept of abstact
method, it has come from Java.
        def sound(self):
                raise NotImplementedError('You
have to define this method in subclass')

class Dog(Animal):
        def __init__(self, name, breed):
                super().__init__(name)
                self.breed = breed

        def sound(self):
                return 'bhow bhow'

class Cat(Animal):
        def __init__(self, name, breed):
                super().__init__(name)
                self.breed = breed

        def sound(self):
                return 'meow meow'


doggy = Dog('sammy', 'german_shepherd')
print(doggy.sound())

billi = Cat('yonna', 'preety_beautiful')
print(billi.sound())
```

Lecture#104 – Raise errors Example2

```
# raise errors example 2

class Mobile:
        def __init__(self, name):
                self.name = name

class MobileStore:
        def __init__(self):
                self.mobiles = []

        def add_mobile(self, new_mobile):
                if isinstance(new_mobile,
Mobile):

self.mobiles.append(new_mobile)
                else:
                        raise TypeError('new
mobile should be object of Mobile class')


samsung = 'samsung galaxy s8'

onePlus = Mobile('one plus 6')
moboStore = MobileStore()

# moboStore.add_mobile(samsung)         #will
raise error because samsung is not object of Mobile
class
moboStore.add_mobile(onePlus)

mobo_phones = moboStore.mobiles
print(mobo_phones[0].name)
```

Lecture#105 – Raise errors Example2

```
# Exception handling

# Exception are error that occur execution time

while True:
    try:
        age = int(input("Enter your age: "))
        break
    except ValueError:
        print('Please enter integer value')
    except:
        print('unexpected error...')

if age < 18:
    print('you can\'t play this game')
else:
    print('you can play this game')
```

Lecture#106 – Else finally with try except

```
# else and finally clause in exception handling

while True:
    try:
        number = int(input('enter any integer
value: '))
    except ValueError:
        print('Please enter integer value')
    except:
        print('unexpected error!')
    else:
        print(f'you entered {number} integer')
        break
    finally:
        print('Finally blocks always execute
whether error occur or not')
```

Lecture#107 – Custom Exception

```
# python custom exceptions
# Q - Why custom exceptions?
# A - To increase the readibility of code.
```

```
class NameTooShortError(ValueError):
    pass

def validate(name):
    if len(name) < 8:
        raise NameTooShortError('Name is too
short. Please enter min 8 letter name')

username = input('Enter your name: ')
validate(username)
print(f' hello {username}')
```

Lecture#108 – Exercise

```
def divide(a,b):
    try:
        return a/b
    except ZeroDivisionError as err:
        print(err)
    except TypeError as err:
        print(err)

print(divide(10,0))
```

Lecture#109 – Read Text Files

```
# readfile
# open function     - to open file
# read method       - to read file
# seek method       - to change cursor position
# tell method       - to find cursor current
position
# readline method   - to read single line
# readlines method  - to put each line in list
# close method      - to close file


# open function
f = open('file.txt')

# read method
print(f.read())

# seek method
print(f.seek(4))
print(f.read())
```

```
# tell method
print(f" cursor position: {f.tell()}")

# readline method
f.seek(0)
print(f.readline())

# readline method
f.seek(0)
print(f.readlines())

print(f.closed)
f.close()
```

Lecture#110 – With blocks

```
# with block
# use: it will read file, close by itself and correct
damaged file


with open("file.txt") as f:
    data = f.read()
    print(data)
```

Lecture#111 – Writing into file

```
# Write in file
# w  - write method
# r  - read method
# a  - append method
# r+ - for both read and write
# Note: mode 'a', 'r+' will create file there is no
file existed with name
# whereas mode 'w' does not create file

# writing in existing file deleting already existed
data
# with open("file.txt", 'w') as f:
    # f.write("\n\nNew line added")


# writing in existing file not deleting already
existed data
# with open("file.txt", 'a') as f:
#     f.write("\n\nNew line added")
```

```
# reading and writing in existing file
# with open("file.txt", 'r+') as f:
#     f.seek(len(f.read()))
#     f.write("\nNew line added")
```

Lecture#112 – Reading and writing together

```
# reading one file and pasting its data into other

with open("file1.txt", 'r') as rf:
    with open("file2.txt", 'w') as wf:
        wf.write(rf.read())
```

Lecture#113 – work with csv file

```
# work with csv files

from csv import reader

with open('file.csv', 'r') as f:
    csv_reader = reader(f)
    next(csv_reader)        # it will not print
first row of the file
    for row in csv_reader:
        print(row)
```

Lecture#114 – read csv file with DictReader

```
from csv import DictReader

with open('test.csv', 'r') as f:
    csv_reader = DictReader(f)
    for row in csv_reader:
        print(row['email'])
```

Lecture#115 – write to csv file

```
# writer, DictWriter

from csv import writer
```

```
with open('test2.csv', 'w', newline='') as f:
    csv_writer = writer(f)
    # methods - writerow, writerows
    # csv_writer.writerow(['name', 'country'])
    # csv_writer.writerow(['awi', 'pakistan'])
    # csv_writer.writerow(['john', 'US'])


csv_writer.writerows([['name','country'],['awi',
'pakistan'],['john', 'US']])
```

Lecture#116 – write to csv file using DictWriter

```
from csv import DictWriter

with open('test3.csv', 'w', newline='') as f:
    csv_writer = DictWriter(f,
fieldnames=['firstname', 'lastname','age'])
    csv_writer.writeheader()

    # method1

    # csv_writer.writerow({
    #     'firstname': 'awinash',
    #     'lastname': 'goswami',
    #     'age':'25'
    # })
    # csv_writer.writerow({
    #     'firstname': 'awinash',
    #     'lastname': 'goswami',
    #     'age':'25'
    # })
    # csv_writer.writerow({
    #     'firstname': 'awinash',
    #     'lastname': 'goswami',
    #     'age':'25'
    # })

    # method2

    csv_writer.writerows([
        {'firstname':'awi',
'lastname':'goswami', 'age': 25},
        {'firstname':'awi',
'lastname':'goswami', 'age': 25},
        {'firstname':'awi',
'lastname':'goswami', 'age': 25}
    ])
```

Lecture#117 – read and write both to csv file

```
from csv import DictReader, DictWriter

with open('test3.csv', 'r') as rf:
    with open('test4.csv', 'w', newline='') as wf:
        csv_reader = DictReader(rf)
        csv_writer = DictWriter(wf,
fieldnames=['first_name', 'last_name', 'age'])
        csv_writer.writeheader()

        for row in csv_reader:
            fname, lname, age = row['firstname'],
row['lastname'], row['age']
            csv_writer.writerow({
                'first_name': fname.upper(),
                'last_name': lname.upper(),
                'age':age
             })
```

Lecture#118 – OS module part I

```
import os

# to get current working directory
os.getcwd()

# to create folder
os.mkdir('movies')

# to check whether a folder already exists
os.path.exists('movies')

# to create a file
open('file.txt', 'a').close()

# to enlist all files and folder
os.listdir()
```

Lecture#118 – OS module part II and shutil method

```
import os
import shutil

# to check file in depth

fileiterator =
os.walk(r'D:\Gallery\drama_seriel')
for current_path, folder_name, file_name in
fileiterator:
    print(f'current_path: {current_path}')
    print(f'folder_name: {folder_name}')
    print(f'file_name: {file_name}')

# to delete folder that is empty
os.rmdir('any_folder_name')

# create a folder in a folder
os.makedirs('new/movies')

# will permenantly delete folder
shutil.rmtree('movies')

# to copy one folder in an other. first parameter
is folder to be copied and second is where to copy
shutil.copytree('new', 'documents/new')

# to copy one file in an other. first parameter is
file to be copied and second is where to copy
shutil.copy('new_file', 'documents/new')

# to move file or folder
shutil.move('file.txt', 'documents/file.txt')
```

Lecture#119 – edit images using python

```
# installation of pillow library
# change the image extension
# resize image files
# resize multiple images using for loop
# Sharpness
# Brightness
# Color
# Contrast
```

```
# Image blur, GaussianBlur

from PIL import Image, ImageEnhance, ImageFilter
import os


# img1.save('dog1.pdf')
# img1.show()
# 250
# MAX_SIZE = (250,250)
# img1.thumbnail(MAX_SIZE)
# img1.save('dog1small.jpg')

# for item in os.listdir():
#     if item.endswith('.jpg'):
#         img1 = Image.open(item)
#         filename,extension =
os.path.splitext(item)
#         img1.save(f'{filename}.png')
# img1 = Image.open('dog1.jpg')
# enhancer = ImageEnhance.Sharpness(img1)
# enhancer.enhance(5).save('dog1edited.jpg')
# 0 : blurry
# 1: original image
# 2 : image with increased sharpness

# -------color ---------
# img1 = Image.open('dog1.jpg')
# enhancer = ImageEnhance.Color(img1)
# enhancer.enhance(2).save('dog1edited.jpg')

# --------brightness -----------
# img1 = Image.open('dog1.jpg')
# enhancer = ImageEnhance.Brightness(img1)
# enhancer.enhance(1.5).save('dog1edited.jpg')


img1 = Image.open('dog1.jpg')
enhancer = ImageEnhance.Contrast(img1)
enhancer.enhance(1.5).save('dog1edited.jpg')

# image blur

#
img1.filter(ImageFilter.GaussianBlur(radius=4)).
save('dog1edited.jpg')


# from PIL import Image,ImageFilter, ImageEnhance
# image = Image.open('cute.jpg')
```

```
#
image.filter(ImageFilter.MaxFilter(size=0)).show
()
# # enhancer = ImageEnhance.Brightness(image)
# # enhancer.enhance(4).show()
```