Contents

Python Notes

[Lecture#14 - Python as calculator 6](#_Toc534732154)

[Lecture#15 - Variables in Python 6](#_Toc534732155)

[Lecture#16 - String Concatation 7](#_Toc534732156)

[Lecture#17 - User Input 7](#_Toc534732157)

[Lecture#18 - Int() and float function 8](#_Toc534732158)

[Lecture#19 - More about variables 8](#_Toc534732159)

[Lecture#20 - Two or more input in one line 8](#_Toc534732160)

[Lecture#20 - String Indexing 9](#_Toc534732161)

[Lecture#21 - String Slicing 9](#_Toc534732162)

[Lecture#22 - Step Argument 9](#_Toc534732163)

[Lecture#23 - String Method Part I 10](#_Toc534732164)

[Lecture#24 - Strip Method 10](#_Toc534732165)

[Lecture#25 - replace() and find() method 10](#_Toc534732166)

[Lecture#26 - Center method with program 11](#_Toc534732167)

[Lecture#27 - Strings are immutable 11](#_Toc534732168)

[Lecture#28 -More Assignment Operators 11](#_Toc534732169)

[Lecture#29 -If Statement 12](#_Toc534732170)

[Lecture#30 - Pass Statement 12](#_Toc534732171)

[Lecture#31 - else Statement 12](#_Toc534732172)

[Lecture#32 - nested if else 13](#_Toc534732173)

[Lecture#33 - and, or operator 13](#_Toc534732174)

[Lecture#34 - if...elif...else statement 14](#_Toc534732175)

[Lecture#35 - in keyword 14](#_Toc534732176)

[Lecture#36 - check empty or not 15](#_Toc534732177)

[Lecture#36 - while loop 15](#_Toc534732178)

[Lecture#37 - for loop 15](#_Toc534732179)

[Lecture#38 - break and continue keyword 16](#_Toc534732180)

[Number Guessing Game 16](#_Toc534732181)

[Lecture#39 - Step Argument in Range function 17](#_Toc534732182)

[Lecture#40 - for loop in string 17](#_Toc534732183)

[Lecture#41 - function intro 18](#_Toc534732184)

[Palindrome Example 18](#_Toc534732185)

[Fibonacci Example 19](#_Toc534732186)

[Lecture#42 - variable scope 19](#_Toc534732187)

[Lecture#43 - Intro to List 20](#_Toc534732188)

[Lecture#44 - Adding data into list 20](#_Toc534732189)

[Lecture#45 - More methods to add data 20](#_Toc534732190)

[Lecture#46 - Delete data from list 21](#_Toc534732191)

[Lecture#47 - In keyword with list 22](#_Toc534732192)

[Lecture#48 - Some more list method 22](#_Toc534732193)

[Lecture#48 - is vs equal 23](#_Toc534732194)

[Lecture#48 - split and join method 24](#_Toc534732195)

[Lecture#49 - List inside list 24](#_Toc534732196)

[Lecture#49 - More about list 25](#_Toc534732197)

[Lecture#50 – Intro to Tuples 25](#_Toc534732198)

[Lecture#51 – More about Tuples 26](#_Toc534732199)

[Lecture#52 – More about Tuples Part II 27](#_Toc534732200)

[Lecture#53 – Intro to Dictionary 28](#_Toc534732201)

[Lecture#54 – Looping and in keyword 29](#_Toc534732202)

[Lecture#54 – Add and delete data from dictionary 31](#_Toc534732203)

[Lecture#55 – update method dictionary 32](#_Toc534732204)

[Lecture#56 – fromkeys, get, clear, copy method 32](#_Toc534732205)

[Lecture#57 – More about get() method 33](#_Toc534732206)

[Lecture#57 – Word Counter Dictionary 34](#_Toc534732207)

[Lecture#58 – Intro to Sets 34](#_Toc534732208)

[Lecture#58 – More about Sets 36](#_Toc534732209)

[Lecture#59 – What is list comprehension? 37](#_Toc534732210)

[Lecture#60 – Exercise 1 38](#_Toc534732211)

[Lecture#60 – List Comprehension with if method 39](#_Toc534732212)

[Lecture#60 – List Comprehension with if method 39](#_Toc534732213)

[Lecture#61 – List Comprehension with if else 40](#_Toc534732214)

[Lecture#62 – List Comprehension with nested list 40](#_Toc534732215)

[Lecture#63 – Dictionary Comprehension 41](#_Toc534732216)

[Lecture#63 – Dictionary Comprehension with if-else 41](#_Toc534732217)

[Lecture#64 – Sets Comprehension 42](#_Toc534732218)

[Lecture#65 – Intro to \*args 42](#_Toc534732219)

[Lecture#66 – \*args with normal parameter 43](#_Toc534732220)

[Lecture#67 – \*args as argument 44](#_Toc534732221)

[Lecture#68 – Exercise 44](#_Toc534732222)

[Lecture#69 – \*\*kwargs 45](#_Toc534732223)

[Lecture#69 – function with all type of parameters 46](#_Toc534732224)

[Lecture#70 – Exercise 47](#_Toc534732225)

[Lecture#70 – Lambda function/expression 47](#_Toc534732226)

[Lecture#71 – Lambda function/expression practice 48](#_Toc534732227)

[Lecture#72 – Enumerate function 49](#_Toc534732228)

[Lecture#73 – map function 50](#_Toc534732229)

[Lecture#73 – map function 51](#_Toc534732230)

[Lecture#74 – Zip function 51](#_Toc534732231)

[Lecture#75 – Zip function part II 52](#_Toc534732232)

[Challenge 53](#_Toc534732233)

[Lecture#76 – all and any function 53](#_Toc534732234)

[Lecture#77 – all and any function practice 54](#_Toc534732235)

[Lecture#77 – advance min and max function 54](#_Toc534732236)

[Lecture#78 – sorted function in advance 55](#_Toc534732237)

[Lecture#79 – More about functions 56](#_Toc534732238)

[Lecture#80 – Decorator chapter intro 57](#_Toc534732239)

[Lecture#81 – Function as argument 57](#_Toc534732240)

[Lecture#82 – Function returning function 58](#_Toc534732241)

[Lecture#83 – Closure Practice 58](#_Toc534732242)

[Lecture#84 – Decorator Intro 59](#_Toc534732243)

[Lecture#85 – Decorator Intro Part II 60](#_Toc534732244)

[Lecture#86 – Decorator Intro Part III 60](#_Toc534732245)

[Lecture#87 – Decorator Practice 61](#_Toc534732246)

[Lecture#88 – Decorator Exercise 62](#_Toc534732247)

[Lecture#89 – Decorator Practice 63](#_Toc534732248)

[Lecture#89 – Decorator with arguments 64](#_Toc534732249)

[Lecture#90 – Generator Example 65](#_Toc534732250)

[Lecture#91 – Generator Comprehension 65](#_Toc534732251)

[Lecture#92 – OOP – Create your first class 66](#_Toc534732252)

[Lecture#93 – OOP – Instance Method 67](#_Toc534732253)

[Lecture#94 – OOP – Class variable 67](#_Toc534732254)

[Lecture#95 – OOP – Class Method 68](#_Toc534732255)

[Lecture#96 – OOP – Static Method 69](#_Toc534732256)

[Lecture#97 – OOP – property\_setter\_decorator 70](#_Toc534732257)

[Lecture#98 – OOP – Inheritance Intro 71](#_Toc534732258)

[Lecture#99 – OOP – Multilevel Inheritance, MRO, method overriding, isinstance(), issubclass() 72](#_Toc534732259)

[Lecture#100 – OOP – Multiple Inheritance 74](#_Toc534732260)

[Lecture#101 – OOP – Magic/Dunder methods, operator overloading, polymorphism 75](#_Toc534732261)

[Lecture#102 – Raise errors 76](#_Toc534732262)

[Lecture#103 – Raise errors Example1 77](#_Toc534732263)

[Lecture#104 – Raise errors Example2 78](#_Toc534732264)

[Lecture#105 – Raise errors Example2 79](#_Toc534732265)

[Lecture#106 – Else finally with try except 80](#_Toc534732266)

[Lecture#107 – Custom Exception 81](#_Toc534732267)

[Lecture#108 – Exercise 81](#_Toc534732268)

[Lecture#109 – Read Text Files 82](#_Toc534732269)

[Lecture#110 – With blocks 83](#_Toc534732270)

[Lecture#111 – Writing into file 83](#_Toc534732271)

[Lecture#112 – Reading and writing together 84](#_Toc534732272)

[Lecture#113 – work with csv file 85](#_Toc534732273)

[Lecture#114 – read csv file with DictReader 85](#_Toc534732274)

[Lecture#115 – write to csv file 86](#_Toc534732275)

[Lecture#116 – write to csv file using DictWriter 86](#_Toc534732276)

[Lecture#117 – read and write both to csv file 88](#_Toc534732277)

[Lecture#118 – OS module part I 88](#_Toc534732278)

[Lecture#118 – OS module part II and shutil method 89](#_Toc534732279)

[Lecture#119 – edit images using python 90](#_Toc534732280)

# 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)

# using list comprehension

new\_list = [name[0] for name in names]

print(new\_list)

# Lecture#60 – Exercise 1

# 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

# 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

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

# 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

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

# 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

# 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

# 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

# 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

# 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

print(add(5,3))

# Lecture#86 – Decorator Intro Part III

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

# 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

# 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

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()

# 