Day – 5 Morning Assessment

1. def outer(func):

def inner():

func()

print(“Function Ended”)

return inner

@outer

def func():

pass

func()

1. def outer(func):

def inner():

print(“Function Started”)

result = func()

print(result\*2)

print(“Function Ended”)

return inner()

@outer

def func():

return 3

func()

1. def outer(func):

def inner():

print(f“calling function:{func.\_\_name\_\_}”)

return inner

@outer

def func():

pass

func()

1. def outer(func):

def inner(\*args):

if len(\*args) == 2:

print(f”function is called with exactly 2 arguments”)

else:

print(f”function is called with {len(\*args)} arguments”)

return func(\*args)

return inner

@outer

def func(a,b):

pass

func(1,100)

1. def outer(func):

count = 0

def inner():

nonlocal count

count += 1

print(f"the function is called {count} times")

return func()

return inner

@outer

def func():

pass

func()

func()

func()

1. def outer(func):

count = 0

def inner():

nonlocal count

count += 1

if count >1:

print(f"You cannot run this function more than once")

else: return f”You can run this function only once”)

return func()

return inner

@outer

def func():

pass

func()

func()

func()

1. def outer(func):

def inner(\*args,\*\*kwargs):

if kwargs.get(“is\_authenticated”) == True:

return “Access granted”

else:

return” Access denied”

return func(\*args,\*\*kwargs)

return inner

@outer

Def func():

Return “welcome to func!”

func(is\_authenticated = True)

func(is\_authenticated = False)

1. def repeat(x):

def outer(func):

def inner(\*args,\*\*kwargs):

for i in range(x):

result = func(\*args,\*\*kwargs)

return result

return inner

return outer

@repeat(3)

def func():

print(“Hello”)

func()

1. import time

def outer(func):

def inner(\*args,\*\*kwargs):

start = time.time()

result = func(\*args,\*\*kwargs)

end = time.time()

print(f”time required for execution is {end-start:.6f} seconds”)

return result

return inner

def func():

print(“Hello World!)

func()

1. def outer(func):

def inner(func):

try:

return func(\*args,\*\*kwargs)

except exception:

return none

return inner

def func():

return 1/0

func()

Logging:

1. def logger(func):

def wrapper(\*args,\*kwargs):

print(f” {func.\_\_name\_\_} with {args} {kwargs}”)

Return func(\*args,\*\*kwargs)

Return wrapper

@logger

def sub(a,b,dividend):

return a\*b

print(sub(26,12,dividend = 5))

1. import logging

logging.basicConfig(filename='app.log',level = logging.INFO)

logging.info("this ia a log message")

1. def logger(func):

def inner(\*args,\*\*kwargs):

print(f”calling {func.\_\_name\_\_} with {\*args, \*\*kwargs}”)

result = func(\*args,\*\*kwargs)

return result

return inner

@logger

def func(a,b):

return a\*b

func(4,7)

1. from functools import wraps

def log\_operations(func):

def wrapper(\*args,\*\*kwargs):

print(f”operation {func.\_\_name\_\_} with {\*args,\*\*kwargs}”)

result = func(\*args,\*\*kwargs)

return result

return wrapper

@log\_operations

def add(a,b):

return a+b

@log\_operations

def sub(a,b):

return a-b

@log\_operation

def prod(a,b):

return a\*b

@log\_operation

def div(a,b):

return a/b

sub(9,5)

add(8,2)

prod(1,7)

div(5,3)

1. logging.basicConfig(filename='warnings.log',level = logging.WARNING)

logging.warning("this ia a warning message")

logging.error("this is an error")

1. import logging

import time

logging.basicConfig(filename = ‘timing.log’, level = logging.INFO)

def log\_execution(func):

def wrapper(\*args,\*\*kwargs):

start = time.time()

result = func(\*args,\*\*kwargs)

end = time.time()

print(f”the execution time for the {func.\_\_name\_\_} is {end-start:’4f} seconds”)

return result

return wrapper

@log\_execution

def func(a,b):

return a+b

func()

1. import logging

logging.basicConfig(filename=’exceptions.log’, level = logging.ERROR)

def error\_handled\_function():

try:

1/0

except Exception:

logging.error(”error occurred: %s,e”)

raise

error\_handled\_function()

1. import logging

logging.basicConfig(filename=’ip address.log’, level = logging.INFO)

def log\_ip(ip):

def decorator(func):

def wrapper(\*args, \*\*kwargs):

logging.info(f”function {func.\_\_name\_\_} called by user with ip {ip}’)

return func(\*args,\*\*kwargs)

return wrapper

return decorator

@log\_ip(“192.168.0.0”)

def sayhello():

print(“hello”)

sayhello()

1. import logging

from logging.handlers import RotatingFileHandler

logger = logging.getLogger(‘rotating\_logger’)

logger.setLevel(logging.INFO)

handler = RotatingFileHandler(‘rotate.log’ , maxBytes = 1000, backupCount = 2)

logger.addHandler(handler)

logger.info(“this is a rotating log message”)

1. import logging

import time

logging.basicConfig(filename=’process.log’, level = logging.INFO)

def log\_process(func):

def wrapper(\*args,\*\*kwargs):

logging.info(f”function {func.\_\_name\_\_} started”)

result = func(\*args, \*\*kwargs)

return wrapper

@log\_process

def func():

print(“processing data….”)

time.sleep(20)

print(“data processed successfully”)

func()

Authorization

1. def authorize(user\_role):

def decorator(func):

def wrapper(\*args,\*\*kwargs):

if user\_role == “admin”:

print(f”Access granted to {user\_role}”)

return func(\*args,\*\*kwargs)

else:

print(f”Access denied to {user\_role}”)

return “Unauthorized Access”

return wrapper

return decorator

@authorize(“admin”)

def view\_dashboard():

return “dashboard data”

@authorize(“guest”)

def view\_settings():

Return “settings data”

Print(view\_dashoard())

Print(view\_settings())

1. def authorize(user\_permit):

def decorator(func):

def wrapper(\*args,\*\*kwargs):

if user\_permit == “Permit”:

print(f”Permission granted to {user\_permit}”)

return func(\*args,\*\*kwargs)

else:

print(f”Permission denied to {user\_permit}”)

return “No user permit”

return wrapper

return decorator

@authorize(“Permit”)

def view\_reports():

return “reports data”

print(view\_reports())

1. def block\_if\_inactive(func):

def wrapper(user\_status,\*args,\*\*kwargs):

if user\_status.lower() == “inactive”:

print(f”user status is ‘{user\_permit}.function ‘{func.\_\_name\_\_}’ will not be executed.”)

return func(user\_status,\*args,\*\*kwargs)

return wrapper

@block\_if\_inactive

def view\_profile(user\_status):

print(f”user profile is being viewed.status: {user\_status}”)

view\_profile(‘inactive’)

view\_profile(‘active’)

print(view\_reports())

1. def authorize(email\_allowed):

def decorator(func)

def wrapper(user\_email,\*args,\*\*kwargs):

if user\_email == email\_allowed:

print(f”Access granted to {user\_email}”)

return func(user\_email, \*args,\*\*kwargs)

else:

print(f”ccess denied to {user\_email}”)

return “Unauthorized Access”

return wrapper

return decorator

@authorize([admin@cgi.com](mailto:admin@cgi.com))

def view\_dashboard(user\_email):

return f”welcome {user\_email}! Here is your dashboard”

@authorize(“guest”)

def view\_settings(user\_email):

return f”{user\_email}, can change settings”

1. def authorize(user\_token):

def decorator(func)

def wrapper(user\_token,\*args,\*\*kwargs):

if user\_token== valid\_token:

print(f”Access granted to {user\_tokenl}”)

return func(user\_token, \*args,\*\*kwargs)

else:

print(f”Access denied to {user\_token}”)

return “Unauthorized Access”

return wrapper

return decorator

@authorize(valid\_token)

def view\_dashboard(user\_token):

return f”welcome {user\_tokenl}! Here is your dashboard

1. def premium(func):

def wrapper(\*args,\*\*kwargs):

subscription = kwargs.get(“subscription”)

if subscription == “premium”:

return func(\*args,\*\*kwargs)

else:

print(“Acess denied as premium subscription needed”)

return none

return wrapper

@premium

def access\_premium(\*\*kwargs):

print(“welcome to the premium feature!”)

access\_premium(subscription=”premium”)

1. def authentication(func):

def wrapper(\*args,\*\*kwargs):

if not kwargs.get(“is\_authenticated”):

return “Authentication Required”

return func(\*args,\*\*kwargs)

return wrapper

@authentication

def func(\*args,\*\*kwargs):

return “is authenticated”

func()

1. import datetime from datetime

def hours(start=9, end =17):

def decorator(func):

def wrapper(\*args,\*\*kwargs):

now = datetime.now().hour()

if start <= now <end:

return func(\*args,\*\*kwargs)

else:

return “Access Denied: Only allowed in working hours”

return wrapper

return decorator

@hours

def func():

return “company’s data”

func()

1. def authentication(func):

def wrapper(\*args,\*\*kwargs):

if not kwargs.get(“is\_authenticated”):

return “Authentication Required”

return func(\*args,\*\*kwargs)

return wrapper

@authentication

def func(\*args,\*\*kwargs):

return “is authenticated”

func()

1. def countries(allowed\_codes)

def decorator(func):

def wrapper(\*args,\*\*kwargs):

country = kwargs.get(‘country\_code’)

if country not in allowed\_codes:

return “Authentication Required”

return func(\*args,\*\*kwargs)

return wrapper

return decorator

@countries(“IN”,”US”)

def func(\*args,\*\*kwargs):

return “Service data allowed”

func()

Lambda Functions:

31. x = lambda y: y\*y

Print(f”the square of {y} is {x}”)

32. num = list(map(int,input(“enter the numbers: “).split()))

Even = list(filter(lambda x: x%2 == 0, num))

Print(even)

33. num = list(map(int,input(“enter the numbers: “).split()))

Cubes = list(map(lambda x :x\*\*3,num)))

Print(cubes)

34. str = lambda s : s == s[::-1]

35. tup = ((1,2),(2,1))

t = sorted(tup,key = lambda x: x[1])

print(t)

36. from functools import reduce

fact = lambda x: reduce(lambda y,z : y\*z, range(1,x+1)) if n>0 else 1

37.num = lambda x : x%3 ==0 and x%5 == 0

38. str = [“apple”,”mango”,”banana”]

x = list(map(lambda s : s.upper(),strings))

print(x)

39. dict = [{“name”:“Ramesh”, “age”:22},

{“name”:“Mohan”, “age”:30},

{“name”:”Pallavi”,”age”:28}]

d = sorted(dict,key = lambda y = y[“age”])

Print(d)

40. num = lambda x,y : x>y?x:y

Print(num)