**Day-5 morning assessment**

**Decorators**

1. def log\_start\_end(func):  
   def wrapper(\*args, \*\*kwargs):  
       print("Function started")  
       result = func(\*args, \*\*kwargs)  
       print("Function ended")  
       return result  
   return wrapper  
  
2. def multiply\_return\_by\_2(func):  
   def wrapper(\*args, \*\*kwargs):  
       result = func(\*args, \*\*kwargs)  
       return result \* 2  
   return wrapper  
  
3. def log\_function\_name(func):  
   def wrapper(\*args, \*\*kwargs):  
       print(f"Calling function: {func.\_\_name\_\_}")  
       return func(\*args, \*\*kwargs)  
   return wrapper  
  
4. def check\_two\_args(func):  
   def wrapper(\*args, \*\*kwargs):  
       if len(args) + len(kwargs) != 2:  
           raise ValueError("Function must be called with exactly 2 arguments")  
       return func(\*args, \*\*kwargs)  
   return wrapper  
  
5. def count\_calls(func):  
   count = 0  
   def wrapper(\*args, \*\*kwargs):  
       nonlocal count  
       count += 1  
       print(f"Function has been called {count} times")  
       return func(\*args, \*\*kwargs)  
   return wrapper  
  
6. def run\_once(func):  
   has\_run = False  
   def wrapper(\*args, \*\*kwargs):  
       nonlocal has\_run  
       if not has\_run:  
           has\_run = True  
           return func(\*args, \*\*kwargs)  
       else:  
           print("Function has already run once. Skipping.")  
   return wrapper  
  
7. def require\_authentication(func):  
   def wrapper(\*args, \*\*kwargs):  
       if kwargs.get('is\_authenticated', False):  
           return func(\*args, \*\*kwargs)  
       else:  
           print("User not authenticated")  
   return wrapper  
  
8. def repeat(n):  
   def decorator(func):  
       def wrapper(\*args, \*\*kwargs):  
           for \_ in range(n):  
               func(\*args, \*\*kwargs)  
       return wrapper  
   return decorator  
  
9. import time  
  
def measure\_time(func):  
   def wrapper(\*args, \*\*kwargs):  
       start = time.time()  
       result = func(\*args, \*\*kwargs)  
       end = time.time()  
       print(f"Execution time: {end - start:.6f} seconds")  
       return result  
   return wrapper  
  
10. def suppress\_exceptions(func):  
   def wrapper(\*args, \*\*kwargs):  
       try:  
           return func(\*args, \*\*kwargs)  
       except Exception as e:  
           print(f"Exception caught: {e}")  
           return None  
   return wrapper

**Logging**

11. def simple\_log(func):  
   def wrapper(\*args, \*\*kwargs):  
       print(f"Function {func.\_\_name\_\_} was called")  
       return func(\*args, \*\*kwargs)  
   return wrapper  
  
12. import logging  
  
logging.basicConfig(filename='app.log', level=logging.INFO, format='%(asctime)s - %(message)s')  
  
def file\_logger\_func():  
   logging.info("This is a log message to a file.")  
  
13. def log\_args\_and\_return(func):  
   def wrapper(\*args, \*\*kwargs):  
       logging.info(f"Calling {func.\_\_name\_\_} with args={args}, kwargs={kwargs}")  
       result = func(\*args, \*\*kwargs)  
       logging.info(f"{func.\_\_name\_\_} returned {result}")  
       return result  
   return wrapper  
  
14. def calculator(op, a, b):  
   if op == 'add':  
       result = a + b  
   elif op == 'subtract':  
       result = a - b  
   elif op == 'multiply':  
       result = a \* b  
   elif op == 'divide':  
       result = a / b  
   else:  
       raise ValueError("Invalid operation")  
  
   logging.info(f"{op}({a}, {b}) = {result}")  
   return result  
  
15. logging.basicConfig(level=logging.WARNING, format='%(levelname)s: %(message)s')  
  
# Example:  
logging.debug("This won't show")  
logging.warning("This is a warning")  
logging.error("This is an error")  
  
16.import time  
  
def log\_execution\_time(func):  
   def wrapper(\*args, \*\*kwargs):  
       start = time.time()  
       result = func(\*args, \*\*kwargs)  
       duration = time.time() - start  
       logging.info(f"{func.\_\_name\_\_} took {duration:.4f} seconds")  
       return result  
   return wrapper  
  
17. def log\_exceptions(func):  
   def wrapper(\*args, \*\*kwargs):  
       try:  
           return func(\*args, \*\*kwargs)  
       except Exception as e:  
           logging.exception(f"Exception occurred in {func.\_\_name\_\_}")  
           raise  # Re-raise if you want the program to still crash  
   return wrapper  
  
18. def log\_ip(func):  
   def wrapper(\*args, \*\*kwargs):  
       ip = kwargs.get('ip', 'unknown')  
       logging.info(f"{func.\_\_name\_\_} called from IP: {ip}")  
       return func(\*args, \*\*kwargs)  
   return wrapper  
  
19. from logging.handlers import RotatingFileHandler  
  
logger = logging.getLogger("RotatingLogger")  
logger.setLevel(logging.INFO)  
  
handler = RotatingFileHandler("rotating.log", maxBytes=2000, backupCount=3)  
formatter = logging.Formatter('%(asctime)s - %(message)s')  
handler.setFormatter(formatter)  
  
logger.addHandler(handler)  
  
# Example logging  
logger.info("Rotating log message")  
  
20. def log\_processing\_time(func):  
   def wrapper(\*args, \*\*kwargs):  
       start\_time = time.strftime('%Y-%m-%d %H:%M:%S')  
       logging.info(f"{func.\_\_name\_\_} started at {start\_time}")  
  
       result = func(\*args, \*\*kwargs)  
  
       end\_time = time.strftime('%Y-%m-%d %H:%M:%S')  
       logging.info(f"{func.\_\_name\_\_} ended at {end\_time}")  
       return result  
   return wrapper

**Authorization**

21. def require\_admin(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('role') == 'admin':  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Admins only.")  
   return wrapper

22. def has\_view\_reports\_permission(user):  
   return 'view\_reports' in user.get('permissions', [])

23. def block\_if\_inactive(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('status') == 'inactive':  
           print("Access blocked: User is inactive.")  
           return  
       return func(user, \*args, \*\*kwargs)  
   return wrapper

24. AUTHORIZED\_EMAILS = {'admin@example.com', 'user@domain.com'}  
  
def email\_authorized(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('email') in AUTHORIZED\_EMAILS:  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Unauthorized email.")  
   return wrapper

25. def valid\_token\_required(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       token = user.get('token')  
       if token and token.startswith("valid-"):  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Invalid token.")  
   return wrapper

26. def premium\_required(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('subscription') == 'premium':  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Premium members only.")  
   return wrapper

27. def authenticated\_only(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('is\_authenticated'):  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: User not authenticated.")  
   return wrapper

28. def within\_working\_hours(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       current\_hour = datetime.datetime.now().hour  
       if 9 <= current\_hour < 17:  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Outside working hours.")  
   return wrapper

29. import logging  
  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')  
  
def log\_unauthorized\_access(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if not user.get('is\_authenticated'):  
           logging.warning(f"Unauthorized access attempt by user: {user}")  
           print("Access denied: Not authenticated.")  
           return  
       return func(user, \*args, \*\*kwargs)  
   return wrapper

30. ALLOWED\_COUNTRIES = {'IN', 'US', 'UK'}  
  
def country\_restricted(func):  
   def wrapper(user, \*args, \*\*kwargs):  
       if user.get('country') in ALLOWED\_COUNTRIES:  
           return func(user, \*args, \*\*kwargs)  
       else:  
           print("Access denied: Country not allowed.")  
   return wrapper

**Lambda Functions**

31. square = lambda x: x \* x  
print(square(5))

o/p: 25  
  
32. numbers = [1, 2, 3, 4, 5, 6]  
evens = list(filter(lambda x: x % 2 == 0, numbers))  
print(evens)

 o/p: [2, 4, 6]  
  
33. numbers = [1, 2, 3, 4]  
cubes = list(map(lambda x: x \*\* 3, numbers))  
print(cubes)

 o/p: [1, 8, 27, 64]  
  
34. is\_palindrome = lambda s: s == s[::-1]  
print(is\_palindrome("madam"))

o/p: True  
  
35. pairs = [(1, 3), (2, 1), (4, 2)]  
sorted\_pairs = sorted(pairs, key=lambda x: x[1])  
print(sorted\_pairs)

o/p: [(2, 1), (4, 2), (1, 3)]  
  
36. from functools import reduce  
  
factorial = lambda n: reduce(lambda x, y: x \* y, range(1, n + 1))  
print(factorial(5))

 o/p: 120  
  
37. div\_by\_3\_and\_5 = lambda x: x % 3 == 0 and x % 5 == 0  
print(div\_by\_3\_and\_5(15))

 o/p: True  
  
38. words = ['apple', 'banana', 'cherry']  
uppercase = list(map(lambda s: s.upper(), words))  
print(uppercase)

o/p: ['APPLE', 'BANANA', 'CHERRY']  
  
39. people = [{'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}]  
sorted\_people = sorted(people, key=lambda person: person['age'])  
print(sorted\_people)

 o/p: [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}]  
  
40. maximum = lambda a, b: a if a > b else b  
print(maximum(10, 20))

o/p: 20