## **Day 11 - 23.06.25**

## **Assignment**

## Section A: Basic Understanding (Short Answer Questions)

**1) What is a Python package? How is it different from a module?**

A package is a folder that contains one or more Python files (modules) and an \_\_init\_\_.py file.  
 A module is just a single Python file (like math.py or my\_module.py).

So:

* module = file
* package = folder of modules

**2)What is the purpose of \_\_init\_\_.py in a package directory?**

It tells Python, “Hey, this folder is a package!”

Also, you can use it to run setup code or define what gets imported when using from package import \*.

**3)What happens when you use from package import \* in Python?**

It imports everything listed in the package’s \_\_all\_\_ variable (defined in \_\_init\_\_.py).

If \_\_all\_\_ is not defined, nothing gets imported by default.

**4)What is the effect of defining \_\_all\_\_ in a package’s \_\_init\_\_.py file?**

It controls what gets imported when someone uses from package import \*.

Example: If \_\_all\_\_ = ['basic\_ops'], then only basic\_ops will be imported

.

**5)How can you create and use a subpackage in Python?**

Just make a folder inside your package with its own \_\_init\_\_.py and modules.  
 Then import like:

from main\_package.subpackage import module\_name

## Section B: Coding-Based Questions

**6)Create a package called math\_utils with the following modules:**

**basic\_ops.py – containing functions for add, subtract**

**advanced\_ops.py – containing functions for power and factorial**

**Demonstrate how to import and use all functions using from math\_utils import \*.**

Folder structure:

math\_utils/

├── \_\_init\_\_.py

├── basic\_ops.py

└── advanced\_ops.py

math\_utils/basic\_ops.py

def add(a, b):

return a + b

def subtract(a, b):

return a - b

math\_utils/advanced\_ops.py

def power(a, b):

return a \*\* b

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

math\_utils/init.py

from . import basic\_ops

from . import advanced\_ops

\_\_all\_\_ = ['basic\_ops', 'advanced\_ops']

main.py (to use it):

from math\_utils import \*

print(basic\_ops.add(2, 3))

print(advanced\_ops.factorial(5))

**7) Write Python code to show how intra-package references work when module\_a.py imports a function from module\_b.py inside the same package.**

Folder: my\_package/

my\_package/

├── \_\_init\_\_.py

├── module\_a.py

└── module\_b.py

module\_b.py

def greet():

return "Hello from module B"

module\_a.py

from .module\_b import greet # <- relative import

def call\_greet():

return greet()

**8) Create a package shapes with a subpackage area. In area, create modules circle.py and rectangle.py. Show how to import area\_of\_circle from circle.py in rectangle.py using relative import.**

Structure:

shapes/

└── area/

├── \_\_init\_\_.py

├── circle.py

└── rectangle.py

circle.py

def area\_of\_circle(radius):

return 3.14 \* radius \* radius

rectangle.py

from .circle import area\_of\_circle # <- relative import

def show\_circle\_area():

print("Circle area:", area\_of\_circle(5))

**9)Modify your package math\_utils to include \_\_all\_\_ = ['basic\_ops'] in the \_\_init\_\_.py. What will happen if you run from math\_utils import \* after that?**

from . import basic\_ops

\_\_all\_\_ = ['basic\_ops']

from math\_utils import \*

Only basic\_ops will be imported. advanced\_ops won't be available unless you import it manually:

from math\_utils import advanced\_ops

# **Exception Handling – Question Paper**

## **Section A: Basic Try-Except (2 marks each)**

**1. Write a program to divide two numbers entered by the user. Handle ZeroDivisionError using try-except.**

try:

a = int(input("Enter first number: "))

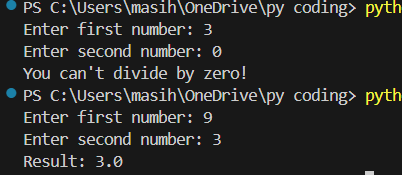
b = int(input("Enter second number: "))

result = a / b

print("Result:", result)

except ZeroDivisionError:

print("You can't divide by zero!")

****

**2. Write a program to convert a string to an integer. Handle ValueError if the input is not a valid number.**

try:

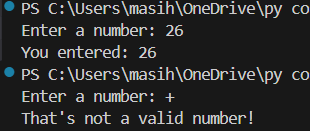
user\_input = input("Enter a number: ")

num = int(user\_input)

print("You entered:", num)

except ValueError:

print("That's not a valid number!")



**3. Accept two numbers from the user and perform addition. Use try-except to handle invalid input types.**

try:

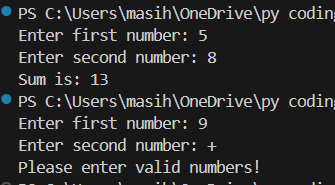
x = int(input("Enter first number: "))

y = int(input("Enter second number: "))

print("Sum is:", x + y)

except ValueError:

print("Please enter valid numbers!")



**4. Write a program to read an element from a list using an index entered by the user. Handle IndexError.**

my\_list = [10, 20, 30, 40, 50]

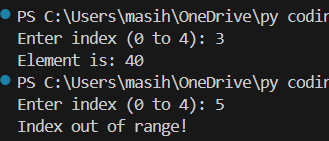
try:

idx = int(input("Enter index (0 to 4): "))

print("Element is:", my\_list[idx])

except IndexError:

print("Index out of range!")

****

**Section B: Try-Except-Else (4 marks each)  
5. Create a program that accepts a number from the user and prints its square. Use try-except-else to handle ValueError and ensure successful computation is shown only if there's no error.**

try:

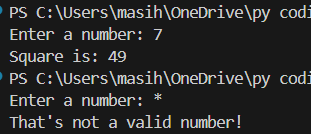
num = int(input("Enter a number: "))

except ValueError:

print("That's not a valid number!")

else:

print("Square is:", num \*\* 2)



**6. Write a program to open a file and read contents. Use try-except-else to handle FileNotFoundError.**

try:

file = open("sample.txt", "r")

except FileNotFoundError:

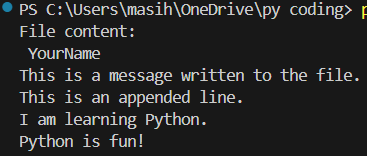
print("File not found!")

else:

content = file.read()

print("File content:\n", content)

file.close()

****

**7. Write a Python program to convert a number to its binary format. Use try-except-else to handle any invalid input.**

try:

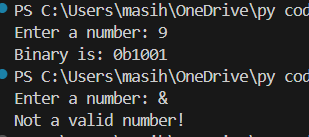
num = int(input("Enter a number: "))

except ValueError:

print("Not a valid number!")

else:

print("Binary is:", bin(num))



## **Section C: Try-Finally (5 marks each)**

**8. Write a program that opens a file and ensures it gets closed, whether or not an exception occurs. Use try-finally.**

file = None

try: file = open("sample.txt", "r")

data = file.read()

print(data)

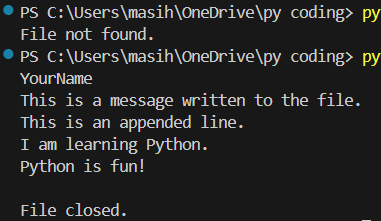
except FileNotFoundError:

print("File not found.")

finally:

if file:

file.close()

print("File closed.")

**9. Simulate a login process where the user input is handled in a try block and a log message is printed in finally regardless of success or failure.**

try:

username = input("Enter username: ")

password = input("Enter password: ")

if username == "Raja" and password == "1234":

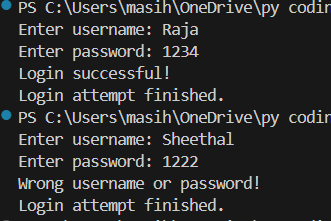
print("Login successful!")

else:

print("Wrong username or password!")

finally:

print("Login attempt finished.")



**10. Write a program that divides two numbers, catching errors with try-except, and printing a clean-up message using finally.**

try:

a = int(input("Enter number 1: "))

b = int(input("Enter number 2: "))

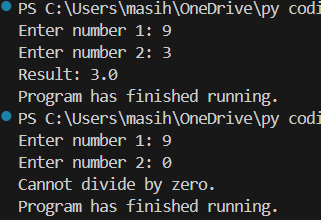
print("Result:", a / b)

except ZeroDivisionError:

print("Cannot divide by zero.")

finally:

print("Program has finished running.")

****

## **Section D: Combined Exception Handling (6 marks each)**

**11. Create a program that handles multiple exceptions: ZeroDivisionError, ValueError, and always prints "Execution complete" using finally.**

try:

x = int(input("Enter number 1: "))

y = int(input("Enter number 2: "))

result = x / y

print("Result:", result)

except ZeroDivisionError:

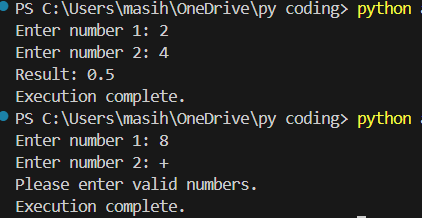
print("Can't divide by zero!")

except ValueError:

print("Please enter valid numbers.")

finally:

print("Execution complete.")



**12. Write a program to simulate bank withdrawal. Use try-except-else-finally to handle incorrect amount input, and always print a message whether the transaction succeeded or failed.**

balance = 5000

try:

amount = int(input("Enter amount to withdraw: "))

if amount > balance:

print("Not enough balance.")

else:

balance -= amount

except ValueError:

print("Invalid amount entered!")

else:

print("Withdrawal successful. Remaining balance:", balance)

finally:

print("Transaction attempt complete.")

