Section A: Basic Understanding

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

A module is a single .py file with Python code.

A package is a directory that contains a special \_\_init\_\_.py file and can include multiple modules or sub-packages. Essentially, packages are a way to organize related modules.

2.Purpose of \_\_init\_\_.py in a package directory:

It tells Python that the directory should be treated as a package. It can also be used to execute initialization code or define what gets imported when using from package import .

3. What happens when you use from package import in Python?

Python imports all the symbols defined in the package’s \_\_all\_\_ list from the \_\_init\_\_.py . If \_\_all\_\_ is not defined, nothing is imported by default.

4. Effect of defining \_\_all\_\_ in a package’s \_\_init\_\_.py file:

It limits what gets imported when you use from package import . Only the names in the \_\_all\_\_ list will be brought into the namespace.

5. How to create and use a subpackage in Python:

- Create a subdirectory inside a package.

- Add an \_\_init\_\_.py to it.

- Place your module inside and import using:

from package.subpackage.module import something .

Section B: Coding-Based

6. Create a package math\_utils :

math\_utils/

│

├── \_\_init\_\_.py

├── basic\_ops.py # functions: add, subtract

└── advanced\_ops.py # functions: power, factorial

basic\_ops.py

python

def add(a, b):

return a + b

def subtract(a, b):

return a - b

advanced\_ops.py

python

def power(a, b):

return a b

def factorial(n):

return 1 if n == 0 else n factorial(n - 1)

Usage:

from math\_utils import basic\_ops, advanced\_ops

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

print(advanced\_ops.factorial(5))

7. Intra-package reference (module\_a.py using module\_b.py):

package/

├── \_\_init\_\_.py

├── module\_a.py

└── module\_b.py

module\_b.py:

python

def greet(name):

return f"Hello, {name}!"

module\_a.py:

python

from .module\_b import greet

print(greet("Jothiswaran"))

8. Relative import in subpackage shapes.area

shapes/

├── \_\_init\_\_.py

└── area/

├── \_\_init\_\_.py

├── circle.py # contains: area\_of\_circle

└── rectangle.py # imports from circle

circle.py:

def area\_of\_circle(radius):

return 3.14 radius radius

rectangle.py:

from .circle import area\_of\_circle

print(area\_of\_circle(5))

9.Modify math\_utils/\_\_init\_\_.py with \_\_all\_\_ = ['basic\_ops'] :

\_\_init\_\_.py:

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

effect

When you run from math\_utils import , only basic\_ops is imported. advanced\_ops is excluded unless imported explicitly.

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

1. Divide two numbers & handle ZeroDivisionError:

try:

a = int(input("Enter numerator: "))

b = int(input("Enter denominator: "))

print("Result:", a / b)

except ZeroDivisionError:

print("Cannot divide by zero.")

2. Convert string to integer & handle ValueError:

try:

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

print("Number is:", num)

except ValueError:

print("Invalid input! Not a number.")

3. Addition with input validation:

try:

x = int(input("First number: "))

y = int(input("Second number: "))

print("Sum:", x + y)

except ValueError:

print("Please enter valid numbers.")

4. Access list element with index handling:

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

try:

index = int(input("Enter index: "))

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

except IndexError:

print("Index out of range.")

Section B: Try-Except-Else (4 marks each)

5. Print square with try-except-else:

try:

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

except ValueError:

print("Invalid input!")

else:

print("Square:", num 2)

6. Open file & handle FileNotFoundError:

try:

f = open("example.txt", "r")

except FileNotFoundError:

print("File not found.")

else:

print("Contents:\n", f.read())

f.close()

7. Binary conversion with error handling:

try:

number = int(input("Enter an integer: "))

except ValueError:

print("That's not a valid integer.")

else:

print("Binary:", bin(number))

Section C: Try-Finally (5 marks each)

8. Ensure file is closed with finally:

python

try:

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

print(file.read())

finally:

file.close()

print("File closed.")

9. Login simulation with final log:

python

try:

username = input("Username: ")

password = input("Password: ")

if password != "admin123":

raise ValueError("Wrong password.")

print("Login successful!")

except ValueError as e:

print(e)

finally:

print("Login attempt logged.")

10. Division with try-except-finally:

python

try:

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

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

print("a/b =", a / b)

except Exception as e:

print("Error:", e)

finally:

print("End of division operation.")

Section D: Combined Exception Handling (6 marks each)

11. Multiple exception handling with finally:

try:

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

y = int(input("Enter divisor: "))

print("Result:", x / y)

except ZeroDivisionError:

print("Cannot divide by zero.")

except ValueError:

print("Invalid input type.")

finally:

print("Execution complete.")

12. Bank withdrawal with all blocks:

try:

balance = 1000

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

if withdraw > balance:

raise ValueError("Insufficient funds.")

except ValueError as e:

print("Error:", e)

else:

balance -= withdraw

print(f"Withdrawal successful. New balance: {balance}")

finally:

print("Transaction attempt logged.")