**Day-7 Morning Assessment**

**Exception Handling**

1. except Exception as e: catches only exceptions that inherit from Exception and gives access to the exception object and except: catches all exceptions, including SystemExit, KeyboardInterrupt and has no access to the exception details.

2. try:  
   num = int(input("Enter a number: "))  
   result = 100 / num  
   print("Result:", result)  
except ValueError:  
   print("Invalid input! Please enter a number.")  
except ZeroDivisionError:  
   print("Cannot divide by zero.")  
except Exception as e:  
   print("An unexpected error occurred:", e)  
  
3. The finally block is always executed after the try and except blocks, regardless of whether an exception was raised or not. It's used for cleanup actions like closing files or releasing resources.  
Example:

try:  
   file = open("example.txt", "r")  
   data = file.read()  
   print(data)  
except FileNotFoundError:  
   print("File not found.")  
finally:  
   file.close()  
   print("File closed.")

4. class InvalidAgeError(Exception):  
   pass  
age = int(input("Enter your age: "))  
if age < 18:  
   raise InvalidAgeError("Age must be at least 18.")  
else:  
   print("Access granted.")  
  
5. o/p:

Divided by zero

Done

6. attempts = 0  
while attempts < 3:  
   try:  
       num = int(input("Enter a number: "))  
       print("Result:", 100 / num)  
       break  
   except ValueError:  
       print("Invalid input! Please enter a number.")  
       attempts += 1  
   except ZeroDivisionError:  
       print("Cannot divide by zero.")  
       break  
else:  
   print("Too many invalid attempts. Exiting.")

o/p: Enter a number: 5

Result: 20.0  
  
7. raise is used to manually raise exceptions and assert is used to debug assumptions during development.

Example of raise:  
if age < 0:  
   raise ValueError("Age cannot be negative")

Example of assert:  
assert age >= 0, "Age must be non-negative"

**Regular Expressions**

8. pattern = r'^(?=.\*[A-Z])(?=.\*\d)(?=.\*[@#$%&]).{8,}$'  
  
9. re.match() matches only at the beginning of the string whereas re.search() scans entire string for a match.  
Example:  
import re  
text = "hello123"  
m1 = re.match(r'\d+', text)  
print(m1)     
m2 = re.search(r'\d+', text)  
print(m2.group())

o/p:

None

123

10. import re  
text = "Email me at test123@gmail.com or hr@openai.org"  
emails = re.findall(r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}', text)  
print(emails)

o/p: ['test123@gmail.com', 'hr@openai.org']  
  
11. import re  
def is\_valid\_indian\_number(number):  
 return re.fullmatch(r'[6-9]\d{9}', number) is not None  
  
print(is\_valid\_indian\_number("9876543210"))  
print(is\_valid\_indian\_number("1234567890"))

o/p:

True

False    
  
12. ^ → Start of string  
[A-Za-z0-9\_] → Letters, digits, or underscore  
{3,15} → Length between 3 and 15 characters  
$ → End of string  
It validates a username-like input: only alphanumeric characters/underscores, 3 to 15 chars long.  
  
13. import re  
text = "I love #Python and #MachineLearning! #AI"  
hashtags = re.findall(r'#\w+', text)  
print(hashtags)

o/p: ['#Python', '#MachineLearning', '#AI']

14. re.match() checks if a pattern matches at the start of a string.  
Using compiled regex with re.compile() improves performance when using the same pattern repeatedly.

import re  
pattern = re.compile(r'[A-Z][a-z]+')  
print(pattern.match("Hello"))  
print(pattern.match("World"))

o/p: <re.Match object; span=(0, 5), match='Hello'>

<re.Match object; span=(0, 5), match='World'>

15. import re  
def check\_password():  
   pwd = input("Enter password: ")  
   if re.fullmatch(r'(?=.\*[A-Z])(?=.\*[a-z])(?=.\*\d)(?=.\*[@#$%&]).{8,}', pwd):  
       print("✅ Strong password")  
   else:  
       print("❌ Weak password")  
check\_password()

o/p:

Enter password: amitha

❌ Weak password