Day 9

Date 17 June 2024

Daily Report

Today's traninig session was based on Exaption Handing, File Handling and some Main module of Python - Random and Math Module.

Today's Topic covered

Exception Handling

Exception handling in Python is a mechanism to respond to runtime errors, preventing the program from crashing and allowing the program to handle errors gracefully. It helps in debugging, maintaining clean code, and providing user-friendly error messages.

Key Concepts

1. Exception: An exception is an error that occurs during the execution of a program. When an exception is raised, the normal flow of the program is interrupted.
2. Try Block: The code that might raise an exception is placed inside a try block.
3. Except Block: The code that handles the exception is placed inside an except block.
4. Else Block: The code inside the else block is executed if no exceptions are raised.
5. Finally Block: The code inside the finally block is executed regardless of whether an exception is raised or not.
6. Raise: Used to raise an exception manually.

File Handling

File handling is an essential aspect of programming that involves reading from and writing to files. Python provides built-in functions and modules to handle files, allowing you to create, read, write, and manipulate files in various ways.

File Methods:

- open(): Opens a file and returns a file object.
- read(): Reads the entire content of a file.
- readline(): Reads a single line from a file.
- readlines(): Reads all lines from a file and returns them as a list.
- write(): Writes a string to a file.
- writelines(): Writes a list of strings to a file.
- close(): Closes the file.

Random Module

The random module in Python provides functions to generate random numbers and perform random selections.

Key Functions of the Random Module

1. Generating Random Numbers

- random.random(): Returns a random float in the range [0.0, 1.0).
- random.randint(a, b): Returns a random integer between a and b, inclusive.
- random.uniform(a, b): Returns a random float between a and b.

2. Random Choices

- random.choice(seq): Returns a random element from the non-empty sequence seq.
- random.choices(population, weights=None, k=1): Returns a list of k elements chosen from population with optional weights.
- random.sample(population, k): Returns a k-length list of unique elements chosen from population.

3. Shuffling and Randomization

- random.shuffle(lst): Shuffles the elements of list lst in place.
- random.sample(population, k): Returns a k-length list of unique elements chosen from population.

4. Seed Control

- random.seed(a=None): Initializes the random number generator with a seed value a. Ensures reproducibility of results when the same seed is used

Math Module

The math module in Python provides access to mathematical functions that perform mathematical operations on numerical data. It includes functions for basic arithmetic, trigonometry, logarithms, exponentiation, and more complex operations.

Key Functions of the Math Module

1. Basic Arithmetic Operations

- math.sqrt(x): Returns the square root of x.
- math.pow(x, y): Returns x raised to the power of y.
- math.factorial(x): Returns the factorial of x.

2. Trigonometric Functions

- math.sin(x), math.cos(x), math.tan(x): Returns the sine, cosine, and tangent of x (in radians).

- math.radians(x), math.degrees(x): Convert angles from degrees to radians and vice versa.

3. Logarithmic and Exponential Functions

- math.log(x, base): Returns the logarithm of x to the given base (default is natural logarithm).
- math.exp(x): Returns e raised to the power of x.

4. Constants

- math.pi: Mathematical constant π (pi).
- math.e: Mathematical constant e (base of natural logarithm).

some question for practice:- ATM Withdrawal: Write a function that simulates an ATM withdrawal. The function should check if the account balance is sufficient for the withdrawal amount and raise an exception if not. Handle scenarios where the input withdrawal amount is not a number or is negative.

```
class NegativeValueError(Exception):
    def __init__(self, value):
        self.value = value
        self.message = f"Negative value error: {value}"
        super().__init__(self.message)

def check_positive(value):
    balance = 60000
    balance -= value
    try:
        if balance < 0:
            raise NegativeValueError(value)
        return "Value is positive."
    except NegativeValueError as e:
        return balance
    finally:
        print("Execution of check_positive function complete.")
check_positive(70000)
```

```
Execution of check_positive function complete.
-10000
```

User Login System: Create a function that simulates a user login system. It should raise an exception if the username or password is incorrect and handle cases where the input values are empty strings.

```
def login(user,password):
  try:
    if (type(user) == str and type(password) == str):
      print("entered")
    else:
      print("invalid")
  except ValueError:
    print("user name or password is incorrect.")
  except TypeError:
    print("user name or password is incorrect.")
  else:
    print("login!!")
login("ABC",123)
```

```
invalid
login!!
```

Online Shopping Cart: Write a function that adds items to an online shopping cart. Handle scenarios where the item is out of stock, the item ID is invalid, or the quantity requested is more than the available stock.

```
def add_items(item,value,res):
  cart = []
  try:
    if item in res and res[item]>=value:
      cart.append(item)
    elif res[item] < value:
      raise ValueError("item is out of stock !!")
    else:
      raise KeyError("item is not found")
  except KeyError as e:
    print(e)
  except ValueError as f:
    print(f)
  else:
    print("item is added succesfully")
res = {"bag":67,"copy":78,"dress":45}
add_items(None,906,res)
```

```
None
```

```
def process_input(value):
    try:
        result = int(value)
    except ValueError:
        return "Invalid input! Please enter a number."
    except TypeError:
        return "Invalid type! Please enter a valid input."
    else:
        return f"Valid input: {result}"
    finally:
        print("Execution of process_input function complete.")

print(process_input("10"))  # Output: Valid input: 10
print(process_input("abc"))  # Output: Invalid input! Please enter a number.
print(process_input(None))   # Output: Invalid type! Please enter a valid input.
```

Your program needs to load user settings from a file named settings.txt. Write a function to read the entire content of this file.

```
file = open('data.txt', 'w')
file.write("your program needs to load user setting")
file.close()
```

```
with open('data.txt','r') as f:
  c = f.read()
print(c)
```

```
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError as e:
        return f"FileNotFoundError: {e}"
    except IOError as e:
        return f"IOError: {e}"

# Usage
file_content = read_file("data.txt")
print(file_content)
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

```
def write_file(file_path, content):
    try:
        with open(file_path, 'w') as file:
            file.write(content)
            return "Write successful"
    except IOError as e:
        return f"IOError: {e}"

# Usage
result = write_file("data.txt", "Hello, World!")
print(result)
```

```
Write successful
```

Start coding or generate with AI.