17 June 2024

Topics done

- Exception Handling
- File Handling
- Module-
- Random module

```
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):
    try:
        if value < 0:
            raise NegativeValueError(value)
        return "Value is positive."
    except NegativeValueError as e:
        return str(e)
    finally:
        print("Execution of check_positive function complete.")

print(check_positive(10))   # Output: Value is positive.
print(check_positive(-5))
```

```
Execution of check_positive function complete.
Value is positive.
Execution of check_positive function complete.
Negative value error: -5
```

1. 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 InsufficientFundsError(Exception):
    pass

def atm_withdrawal(balance, withdrawal_amount):
    try:
        # Check if balance is sufficient for withdrawal
        if withdrawal_amount > balance:
            raise InsufficientFundsError("Insufficient funds in your account")

        balance -= withdrawal_amount
        return balance

    except ValueError as value:
        print(f"Error: {value}")
    except InsufficientFundsError as b:
        print(f"Error: {b}")
    except Exception as exec:
        print(f"Error: {exec}")
balance=167890
withdrawal_amount=600

try:
    new_balance = atm_withdrawal(balance, withdrawal_amount)
    if new_balance is not None:
        print(f"Done! Remaining balance is {new_balance:.2f}")
except:
    print("An error occurred during withdrawal")
```

```
Done! Remaining balance is 167290.00
```

2. 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 user_login(username, password):
    try:
        if username == "" or password == "":
            raise ValueError("Username and password cannot be empty")

        valid_username = "username"
        valid_password = "password"

        # Check if username and password match the fixed credentials
        if username == valid_username and password == valid_password:
            print("Login successful!")
        else:
            raise ("Incorrect username or password")

    except ValueError as ve:
        print(f"Error: {ve}")
    except Exception as e:
        print(f"Error: {e}")

# Input username and password
username = input("Enter username: ")
password = input("Enter password: ")

user_login(username, password)
```

```
Enter username: username
Enter password: password
Login successful!
```

3. 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.

```
class CartError(Exception):
    pass

class Item:
    def __init__(self, item_id, name, price, stock):
        self.item_id = item_id
        self.name = name
        self.price = price
        self.stock = stock

def add_item(cart, inventory, item_id, quantity):
    try:
        # Find the item in the inventory
        item = find_item(inventory, item_id)

        # Check if item ID is valid
        if item is None:
            raise CartError(f"Invalid item ID: {item_id}")

        # Check if item is in stock
        if quantity > item.stock:
            raise CartError(f"Sorry, there is no enough stock  ")

    except ValueError as ve:
        print(f"Error: {ve}")
    except CartError as ce:
        print(f"Error: {ce}")
    except Exception as e:
        print(f"Error: {e}")

def find_item(inventory, item_id):

    for item in inventory:
        if item.item_id == item_id:
            return item
    return None

# Example usage:
inventory = [
    Item(item_id=1, name="Laptop", price=1000.0, stock=5),
    Item(item_id=2, name="Mouse", price=20.0, stock=10),
]

cart = []

# Test adding items to the cart
add_item(cart, inventory, 1, 2)   # Adding 2 laptops (valid scenario)
add_item(cart, inventory, 2, 15)  # Attempting to add 15 mice (more than available stock)
add_item(cart, inventory, 4, 1)   # Attempting to add an invalid item ID
add_item(cart, inventory, 3, -3)  # Attempting to add a negative quantity
```

```
Error: Sorry, there is no enough stock
Error: Invalid item ID: 4
Error: Invalid item ID: 3
```

4. Temperature Conversion: Implement a function that converts temperatures between Fahrenheit and Celsius. Raise an exception if the input is not a number and handle this error gracefully.

```
class temp(Exception):
  pass

  def __init__(self,celcius,fahrenheit):
    self.celcius=celcius
    self.fahrenheit=fahrenheit

def  convert_temperature(temp):
    try:
      temp=float(temp)
      if temp in fahrenheit:
       celcius=5/9*(temp-2)
       return celcius
      elif temp in celcius:
       fahrenheit=9/5*(temp+2)
       return fahrenheit
      else:
       raise "invalid units"
    except ValueError as v:
      print(f"Error:{v}")
    except Exception as ex:
      print(f"Error:{ex}")

      temp=45.8
      convert_temperature(45.8)
      print(f"{temp}F is in {celcius}C")
```

5. Student Grades: Write a function that calculates the average grade of a list of student grades. Handle scenarios where the list might be empty, the grades might not be valid numbers, or some grades might be missing

```
def calculate_average_grade(grades):
    try:
        if not grades:
            raise ValueError("List of grades is empty")

        valid_grades = []
        for grade in grades:
            if ispresent(grade, (int, float)):
                valid_grades.append(grade)
            else:
                raise ValueError(f"Invalid grade: {grade}")

        if not valid_grades:
            raise ValueError("No valid grades")

        average_grade = sum(valid_grades) / len(valid_grades)
        return average_grade

    except ValueError as ve:
        print(f"Error: {ve}")
    except ZeroDivisionError:
        print("Error: Division by zero occurred")
    except Exception as e:
        print(f"Error occurred: {e}")

# Example usage:
try:
    grades1 = [85, 90, None, 75, 'A', 95]
    average_grade1 = calculate_average_grade(grades1)
    print(f"Average grade: {average_grade1}")

    grades2 = []  # Empty list
    average_grade2 = calculate_average_grade(grades2)
    print(f"Average grade: {average_grade2}")  #

    grades3 = [80, 85, 90]  # List of valid grades
    average_grade3 = calculate_average_grade(grades3)
    print(f"Average grade: {average_grade3}")

except Exception as e:
    print(f"Error: {e}")
```

```
Error occurred: name 'ispresent' is not defined
Average grade: None
Error: List of grades is empty
Average grade: None
```

```
Error occurred: name 'ispresent' is not defined
Average grade: None
```

File Handling

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

```
with open('/content/sample_data/setting.txt','r') as file:
  data="Hey! This is a python file having some code realted to exception and file handling"
  data=file.read()
```

2. Your application generates a report. Write a function to save the string "Monthly Report: June" to a file named `report.txt`.

```
def write_file(file_path, content):
    try:
        with open(file_path, 'w') as file:
            file.write(content)
```