جامـــــعـة المــــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قـســــــــم الانــظـــمــــة الــطـبـيـة الــذكـــيــة

Lecture: (8)

(Exception handling)

Subject: Object oriented programming II
Class: Second
Lecturer: Dr. Maytham N. Meqdad

Exception handling

"Exception handling" is a concept that refers to how errors and exceptions that can occur during computer program execution are managed. It allows you to control the flow of a program when different errors occur, helping to prevent program crashes and directing execution appropriately.

In the Python programming language, it provides a straightforward and powerful way to handle exceptions. You can use the `try` and `except` statements to deal with exceptions. Here's how it works:

1. You place the code that may potentially raise an exception inside a `try` block.
2. If an exception occurs within the `try` block, it is caught, and the code inside the appropriate `except` block, which corresponds to the type of exception that occurred, is executed.
3. You can also use the `else` and `finally` blocks to add more control over exception handling.

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("The result is:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Please enter a valid integer.")
except Exception as e:
    print("An unexpected exception occurred:", e)
```

- In this example, the `try` block is used to execute code that might raise an exception, and if an exception occurs, it is caught and the code inside the appropriate `except` block is executed. This helps avoid program crashes and directs the flow of execution appropriately when errors occur.

- This program will demonstrate how to use exception handling to manage errors when taking patient information.

```
class Patient:
    def __init__(self, name, age):
        self.name = name
        self.age = age

def get_patient_information():
    try:
        name = input("Enter the patient's name: ")
        age = int(input("Enter the patient's age: "))
        if age < 0:
            raise ValueError("Age cannot be negative.")
        return Patient(name, age)
    except ValueError as ve:
        print("Error:", ve)
        return None

def main():
    patient = get_patient_information()
    if patient:
        print("Patient information:")
        print(f"Name: {patient.name}")
        print(f"Age: {patient.age}")
    else:
        print("Failed to retrieve patient information.")

if __name__ == "__main":
    main()
```

- In this example:

1. We define a `Patient` class to represent patient information.
2. The `get_patient_information()` function collects patient data, including their name and age. It uses exception handling to check for invalid age input (negative age) and raises a `ValueError` if necessary.
3. The `main()` function calls `get_patient_information()` to gather patient data and then displays the patient's information if the input is valid.
4. We run the `main()` function when the script is executed.

This is a simple example to get you started. In a real-world medical program, you would need to expand on this foundation to incorporate more features and handle a wider range of exceptions, such as data validation, treatment plans, and medical records.

```
class Patient:
    def __init__(self, name, age, diagnosis):
        self.name = name
        self.age = age
        self.diagnosis = diagnosis

def get_patient_information():
    while True:
        try:
            name = input("Enter the patient's name: ")
            age = int(input("Enter the patient's age: "))
            diagnosis = input("Enter the patient's diagnosis: ")
            if age < 0:
                raise ValueError("Age cannot be negative.")
            return Patient(name, age, diagnosis)
        except ValueError as ve:
            print("Error:", ve)
            continue

def main():
    print("Medical Record System")
    patient = get_patient_information()
    print("Patient information:")
    print(f"Name: {patient.name}")
    print(f"Age: {patient.age}")
    print(f"Diagnosis: {patient.diagnosis}")

if __name__ == "__main__":
    main()
```

Note : is used to ensure that the `main()` function is only called when the program is executed as the main file, not when it's imported as a module into another file.

The purpose of this condition is to control the execution of the program when it's run as the main script. When you run a Python file directly, the code inside the `if __name__ == "__main__":` block is executed, and thus, the `main()` function is called.

On the other hand, when you use this file as a module in another file, like when you import it using `import`, the code within the `if __name__ == "__main__":` block won't be automatically executed. This separation allows you to organize your code so that a part of it is only executed when the file is run as the main program.

-
-
-
-
-
-

- This program that demonstrates exception handling for handling division by zero:

```
def divide_numbers(dividend, divisor):
    try:
        result = dividend / divisor
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

def main():
    print("Division Program")
    try:
        dividend = float(input("Enter the dividend: "))
        divisor = float(input("Enter the divisor: "))
        result = divide_numbers(dividend, divisor)
        print(f"Result: {result}")
    except ValueError:
        print("Error: Please enter valid numeric inputs.")

if __name__ == "__main__":
    main()
```

In this program:

1. We define a function `divide_numbers(dividend, divisor)` that takes two numbers and attempts to perform division. It includes a try-except block to catch the `ZeroDivisionError` that may occur if the divisor is zero.
2. In the `main()` function, we prompt the user to enter the dividend and divisor, and we call the `divide_numbers` function to perform the division. We also include a try-except block to handle potential `ValueError` when converting user input to floating-point numbers.
3. When the program is run, it checks for division by zero and handles it gracefully, as well as invalid input values.

You can run this program, and it will allow you to perform division while handling exceptions for division by zero and invalid inputs.

```
def calculate_average(numbers):
    try:
        total = sum(numbers)
        average = total / len(numbers)
        return average
    except ZeroDivisionError:
        return "Error: Division by zero (empty list)."
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
    print("Average Calculator")
    try:
        filename = input("Enter the name of the file with numbers: ")
        with open(filename, 'r') as file:
            numbers = [float(line) for line in file if line.strip()]
            average = calculate_average(numbers)
            if isinstance(average, float):
                print(f"Average of numbers: {average:.2f}")
            else:
                print(average)
    except FileNotFoundError:
        print("Error: File not found.")
    except PermissionError:
        print("Error: Permission denied.")
    except ValueError:
        print("Error: Invalid number format in the file.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```

```
def perform_operation():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        operation = input("Enter the operation (+, -, *, /): ")

        if operation == '+':
            result = num1 + num2
        elif operation == '-':
            result = num1 - num2
        elif operation == '*':
            result = num1 * num2
        elif operation == '/':
            if num2 == 0:
                raise ZeroDivisionError("Division by zero is not allowed.")
            result = num1 / num2
        else:
            raise ValueError("Invalid operation.")

        return result
    except ValueError as ve:
        return f"Error: {ve}"
    except ZeroDivisionError as zde:
        return f"Error: {zde}"
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
    print("Basic Calculator")
    try:
        result = perform_operation()
        if isinstance(result, float):
            print(f"Result: {result}")
        else:
            print(result)
    except KeyboardInterrupt:
        print("Program terminated by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```

```
def calculate_urea():
    try:
        creatinine = float(input("Enter creatinine level (mg/dL): "))
        bun = float(input("Enter blood urea nitrogen (BUN) level (mg/dL): "))
        urea = (bun / creatinine) * 100
        return urea
    except ValueError:
        return "Error: Invalid input. Please enter numeric values."
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
    print("Urea Measurement Calculator")
    try:
        urea_level = calculate_urea()
        if isinstance(urea_level, float):
            print(f"Urea measurement in the blood: {urea_level:.2f} mg/dL")
        else:
            print(urea_level)
    except KeyboardInterrupt:
        print("Program terminated by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main(
```

```
import math

def calculate_triangle_area():
    try:
        a = float(input("Enter the length of side a: "))
        b = float(input("Enter the length of side b: "))
        c = float(input("Enter the length of side c: "))

        # Check if the sides can form a valid triangle
        if a + b <= c or a + c <= b or b + c <= a:
            raise ValueError("Invalid sides. Cannot form a triangle.")

        # Calculate the semi-perimeter
        s = (a + b + c) / 2

        # Calculate the area using Heron's formula
        area = math.sqrt(s * (s - a) * (s - b) * (s - c))

        return area
    except ValueError as ve:
        return f"Error: {ve}"
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
```

```
    print("Triangle Area Calculator")
    try:
        area = calculate_triangle_area()
        if isinstance(area, float):
            print(f"The area of the triangle is: {area:.2f} square units")
        else:
            print(area)
    except KeyboardInterrupt:
        print("Program terminated by user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```

```
import math

def calculate_square_root():
    try:
        num = float(input("Enter a non-negative number: "))
        if num < 0:
            raise ValueError("Square root is not defined for negative 
numbers.")
        square_root = math.sqrt(num)
        return square_root
    except ValueError as ve:
        return f"Error: {ve}"
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
    print("Square Root Calculator")
    try:
        square_root = calculate_square_root()
        if isinstance(square_root, float):
            print(f"The square root is: {square_root:.4f}")
        else:
            print(square_root)
    except ValueError:
        print("Error: Please enter a non-negative number.")
    except KeyboardInterrupt:
        print("Program terminated by the user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```

```
import math

def calculate_circle_area():
    try:
        radius = float(input("Enter the radius of the circle: "))
        if radius < 0:
```

```
            raise ValueError("Radius cannot be negative.")
        area = math.pi * (radius ** 2)
        return area
    except ValueError as ve:
        return f"Error: {ve}"
    except Exception as e:
        return f"An unexpected error occurred: {e}"

def main():
    print("Circle Area Calculator")
    try:
        area = calculate_circle_area()
        if isinstance(area, float):
            print(f"The area of the circle is: {area:.4f} square units")
        else:
            print(area)
    except ValueError:
        print("Error: Please enter a non-negative radius.")
    except KeyboardInterrupt:
        print("Program terminated by the user.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
```