Python
for everybody

7. Exception Handling (try-except)

Problem Example

```
x = float(input("x = "))
y = float(input("y = "))
opr = input("operation : ")
if opr == "*":
    print(x * y)
elif opr == "+":
    print(x + y)
elif opr == "-":
    print(x - y)
elif opr == "/":
    print(x / y)
```

Possible errors:

- y = 0 when doing division ZeroDivisionError
- Entering "five" instead of a number ValueError

Solution : We need exception handling.

Exception Handling (Try-Except)

The purpose of try and except is that you know that some sequence of instruction(s) may have a problem and you want to add some statements to be executed if an error occurs.
These extra statements (the except block) are ignored if there is no error.

Syntax:

```
try:
    # Code that may raise an exception
except SomeException:
    # Code to handle the exception
```

Try–Except Example

Example:

```
try:
    x = float(input("x = "))
    y = float(input("y = "))
    opr = input("operation : ")

    if opr == "*":
        print(x * y)
    elif opr == "+":
        print(x + y)
    elif opr == "-":
        print(x - y)
    elif opr == "/":
        print(x / y)

except ZeroDivisionError:
    print("Impossible to divide by zero!")
```

Catching Any Error

Example:

```
try:
    x = float(input("x = "))
    y = float(input("y = "))
    opr = input("operation : ")
    if opr == "*":
        print(x * y)
    elif opr == "+":
        print(x + y)
    elif opr == "-":
        print(x - y)
    elif opr == "/":
        print(x / y)
except ZeroDivisionError:
    print("Impossible to divide by zero!")
except:
    print("You entered invalid inputs.")
```

8. Object Oriented Programming

8. Object Oriented Programming

8.1. Introduction to OOP

What is OOP?

So far, we have written code like a **Recipe** (Procedural Programming): step-by-step instructions.

Object Oriented Programming (OOP) is a paradigm based on the concept of "objects", which can contain:

- **Data:** (Attributes)
- **Behavior:** (Methods)

Goal: OOP aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming.

General Concepts:

To understand OOP, we must distinguish between the **Class** and the **Object**.

1. **Class (The Blueprint/Factory):** It is the template or definition. It describes what the object will look like. *(e.g., The architectural plan of a house, or the mold in a factory).*
2. **Object (The Instance/Product):** It is the concrete realization of the class. *(e.g., The actual house built from the plan).*
3. **Attribute:** The data stored inside an object (e.g., color, height, name).
4. **Method:** The behavior or actions the object can perform (e.g., drive, speak, calculate).

8. Object Oriented Programming

8.2. Classes VS Instances

Creating a Class & Instance

Example 1: The Basic Syntax

```
# 1. Define the Class (The Factory)
class Car:
    pass  # Empty class for now

# 2. Create Instances (The Products)
car1 = Car()
car2 = Car()

print(car1)
# Output: <__main__.Car object at 0x...>
# This proves car1 is an instance of Car
```

The Constructor (__init__)

How do we give data to the object *when* we create it? We use the **Constructor**.

- In Python, it is the **`__init__`** method.
- It runs automatically when an object is instantiated.
- **`self`** refers to the **current instance** being created.

```
class Car:
    def __init__(self, model, color):
        self.model = model  # Attribute
        self.color = color  # Attribute

# Creating objects with specific data
c1 = Car("BMW", "Black")
c2 = Car("Ferrari", "Red")

print(c1.model)  # BMW
print(c2.model)  # Ferrari
```

8. Object Oriented Programming

8.3. Attributes and Methods

Attributes and Methods

Adding Behavior:

```python
class Human:
    def __init__(self, name):
        self.name = name  # Instance Attribute

    # This is a Method
    def speak(self):
        return "Hi! My name is " + self.name

ahmed = Human("Ahmed")
print(ahmed.speak())
# Output: Hi! My name is Ahmed
```

Example: School System

```python
class Student:
    def __init__(self, name, grades):
        self.name = name
        self.grades = grades

    def get_average(self):
        if len(self.grades) == 0:
            return 0
        return sum(self.grades) / len(self.grades)

# Usage
s1 = Student("Ali", [15, 18, 14])
print(f"{s1.name} has average: {s1.get_average()}")
# Output: Ali has average: 15.66
```