**Built-in Functions in Python**

Built-in functions in Python are pre-defined functions that are always available for use without requiring imports. These functions perform common operations such as data type conversion, mathematical calculations, input/output handling, and more.

**Common Built-in Functions:**

1. **Data Type Conversion:**
   * int(), float(), str(), bool(), list(), tuple(), set(), dict()
   * Example:
   * x = int("10") # Converts string "10" to integer 10
   * y = list((1, 2, 3)) # Converts tuple to list
2. **Mathematical Functions:**
   * abs(x), pow(x, y), round(x, n), min(), max(), sum()
   * Example:
   * print(abs(-5)) # Output: 5
   * print(pow(2, 3)) # Output: 8 (2^3)
3. **Iterables & Sequences:**
   * len(), sorted(), reversed(), enumerate(), zip()
   * Example:
   * names = ["Alice", "Bob", "Charlie"]
   * print(len(names)) # Output: 3
   * print(sorted(names)) # Output: ['Alice', 'Bob', 'Charlie']
4. **Input & Output Functions:**
   * print(), input()
   * Example:
   * name = input("Enter your name: ")
   * print("Hello,", name)
5. **File Handling:**
   * open(), read(), write(), close()
   * Example:
   * with open("example.txt", "w") as file:
   * file.write("Hello, World!")
6. **Other Utility Functions:**
   * type(), id(), isinstance()
   * Example:
   * x = 100
   * print(type(x)) # Output: <class 'int'>

**Custom Data Types in Python**

Python allows users to create their own data types using **classes**. These custom data types help define objects with specific attributes and methods.

**Defining a Custom Data Type:**

A class is used to define a custom data type. An **object** is an instance of a class.

**Example: Creating a Custom Data Type**

class Car:

def \_\_init\_\_(self, brand, model, year):

self.brand = brand

self.model = model

self.year = year

def display\_info(self):

return f"{self.year} {self.brand} {self.model}"

# Creating an instance (object) of the Car class

car1 = Car("Toyota", "Camry", 2023)

print(car1.display\_info()) # Output: 2023 Toyota Camry

**Key Features of Custom Data Types:**

1. **Encapsulation** – Data is wrapped within objects.
2. **Abstraction** – Hides unnecessary details.
3. **Inheritance** – One class can inherit from another.
4. **Polymorphism** – Same method name can work for different data types.

**Example: Inheritance**

class ElectricCar(Car): # Inheriting from Car class

def \_\_init\_\_(self, brand, model, year, battery\_capacity):

super().\_\_init\_\_(brand, model, year)

self.battery\_capacity = battery\_capacity

def display\_info(self):

return f"{self.year} {self.brand} {self.model} with {self.battery\_capacity} kWh battery"

# Creating an ElectricCar object

tesla = ElectricCar("Tesla", "Model S", 2024, 100)

print(tesla.display\_info()) # Output: 2024 Tesla Model S with 100 kWh battery

**Conclusion**

* **Built-in functions** simplify tasks such as input handling, mathematical operations, and file handling.
* **Custom data types** allow the creation of complex structures using classes, enabling object-oriented programming (OOP) in Python.