Functions and OOP: Control Structures

September 11, 2025

Functions:

A **function** is a block of code that is organized, reusable, and performs a single, related action. This helps to break down large programs into smaller, manageable parts.

- **Types of Functions:**
  - **Built-in Functions:** Functions that are part of Python's core, like print(), len(), sum(), and input().
  - **User-defined Functions:** Functions you create yourself to perform specific tasks in your code.
- **Types of Arguments:**
  - **Positional Arguments:** Arguments passed to a function based on their position or order.

```
def describe_pet(animal, name):
    print(f"I have a {animal} named {name}.")
describe_pet("dog", "Buddy")
```

  - **Keyword Arguments:** Arguments specified by name, allowing you to pass them in any order.

```
describe_pet(name="Buddy", animal="dog")
```

  - **Default Arguments:** A parameter is assigned a default value in the function definition, which is used if no argument is provided for it.

```
def greet(name, message="Hello"):
    print(f"{message}, {name}!")
greet("Alice")          # Uses default: "Hello, Alice!"
greet("Bob", "Hi")      # Overrides default: "Hi, Bob!"
```

- **Arbitrary Arguments (*args and **kwargs):**
  - *args (non-keyworded arguments): Allows a function to accept any number of positional arguments. They are packed into a **tuple**.
  - **kwargs (keyworded arguments): Allows a function to accept any number of keyword arguments. They are packed into a **dictionary**.

```
def show_info(*args, **kwargs):
   print("Positional arguments:", args)
   print("Keyword arguments:", kwargs)
show_info(1, "apple", age=25, city="New York")
# Output:
# Positional arguments: (1, 'apple')
# Keyword arguments: {'age': 25, 'city': 'New York'}
```

- **Variable Scope: The global Keyword:** The global keyword is used to modify a global variable from within a function.

```
x = 10  # A global variable
def modify_x():
   global x
   x = 20
modify_x()
print(x) # Output: 20
```

Overview of Object-Oriented Programming (OOP)

Objective: To provide a brief introduction to the core principles of Object-Oriented Programming (OOP) in Python by creating and using a simple class.

Theory: Object-Oriented Programming (OOP) is a paradigm that organizes code around **objects** rather than functions and logic. A **class** is a blueprint for creating objects, defining attributes (data) and methods (behavior). An **object** is an instance of a class. The four pillars of OOP are Encapsulation, Inheritance, Polymorphism, and Abstraction.