**Name:** Flores, Christian T.

**Course & Section:** CS3C

* **Defining a Function**
* is a set of instructions or a chunk of code that can be executed again and again by simply calling its name without having to type the whole code each time. It allows you to organize your code better and make it easier to read and understand. A function can take input parameters, perform a specific task or calculation, and return a value or do a specific action. It can be used for almost any kind of task or calculation, making programming much simpler and more efficient. It's like having a small robot that can perform repetitive tasks for you, saving you time and effort.
* **Reasons of Using Functions**
* Functions make it easier to separate your code into smaller, more manageable sections and also help to increase the readability and maintainability of the code. They allow you to reuse code and perform calculations or operations in a single place, making it more efficient. Another benefit of functions is that they allow you to test and debug specific parts of the code without having to worry about the rest of it. This can save a lot of time and effort when dealing with larger projects.
* **Types of Functions in Python**
* **Simple Function:** A simple function takes in input parameters, performs a calculation or operation, and returns a value. Simple functions do not have any side effects, meaning that they only perform the calculation or operation that they are intended to do and do not alter any external state.
* **Decorator Function:** A decorator function is a function that wraps another function to add additional behavior to it. Decorator functions are commonly used to add features such as caching, logging, or security to other functions.
* **Generator Function:** A generator function is a function that generates a sequence of values on demand. A generator function uses the yield keyword instead of return to provide a value and it can be suspended and resumed with the next() function.
* **Method Function:** A method function is a function that is called as a part of an object. Method functions are commonly used in object-oriented programming to provide functionality for a specific object.
* **Advantages of User – Defined Function**
* User-defined functions are building blocks that offer numerous advantages in Python programming. By creating a function,
* You can write a specific task's code once and call it whenever needed
* Saving time and reducing errors.
* Functions also improve code readability by separating complex logic into smaller, clear sections with descriptive names.
* This modular approach makes your code easier to understand, maintain, and debug.
* Functions can be individually tested, promoting well-organized testing practices. Additionally, they create namespaces to avoid naming conflicts between variables in different parts of your code.
* Overall, user-defined functions are essential for writing clean, reusable, and efficient Python programs.
* **Rules in Declaring a Function in Python**
* **Def keyword:** You start by writing the def keyword, which indicates you're defining a function.
* **Function name:** Choose a meaningful name that reflects the function's purpose. The name follows the same rules as variable naming (letters, numbers, and underscores, must start with a letter).
* **Parameters (optional):** Functions can optionally take inputs, called parameters. These are listed within parentheses after the function name. Each parameter name follows variable naming rules. If there are multiple parameters, they are separated by commas.
* **Colon:** A colon (':') follows the parameters (or the function name if there are no parameters).
* **Docstring (optional):** A docstring is a concise explanation of the function's purpose and how to use it. It's added within triple quotes (''' or """) directly below the colon. While optional, it's good practice to include a docstring for better code readability and maintainability.
* **Function body:** Indented lines after the colon define the function's code block. This is where you write the statements the function will execute. The indentation level (usually 4 spaces) is crucial in Python as it defines code blocks.
* **Return statement (optional):** A function can optionally return a value using the return keyword followed by the value to be returned. This value becomes the output of the function call. If no return statement is specified, the function implicitly returns None.
* **Python Function Syntax**
* **def** function\_name(parameter1, parameter2, ...):

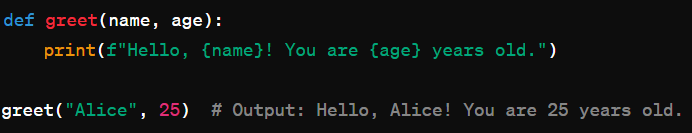
# Function body - code block

# Perform actions or computations

# Optionally, return a value using the return statement

return value

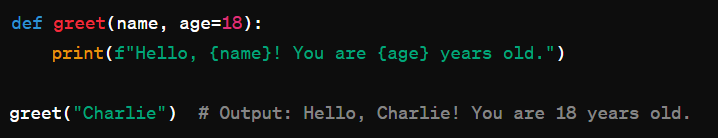
* **Function Argument and Parameter**
* **Positional Arguments:** These are arguments passed to a function in the order they are defined in the function signature.

****

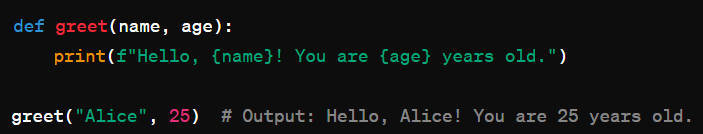
* **Keyword Arguments:** These are arguments explicitly identified by their parameter names when calling the function.



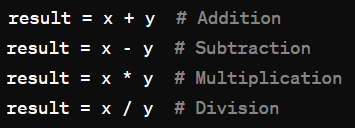
* **Default Arguments:** These are arguments with default values specified in the function signature.



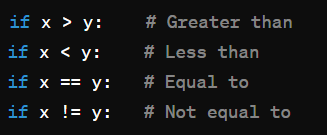
* **Arbitrary Number of Arguments:** Functions can accept a variable number of arguments using \*args or \*\*kwargs.

****

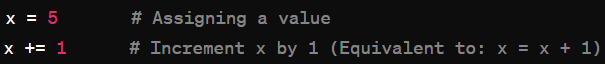
* **Arithmetic Operators:** Used for arithmetic operations like addition, subtraction, multiplication, division, etc.

****

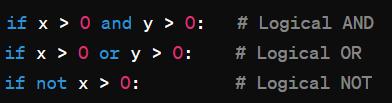
* **Comparison Operators:** Used to compare two values and return a Boolean result (True or False).



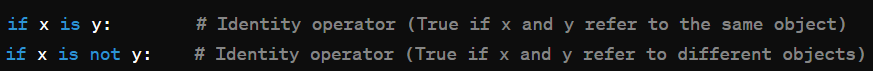
* **Assignment Operators:** Used to assign values to variables.



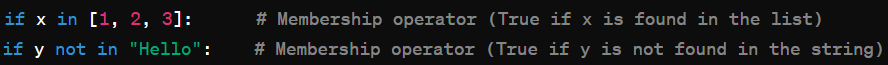
* **Logical Operators:** Used to combine multiple conditions and return a Boolean result.



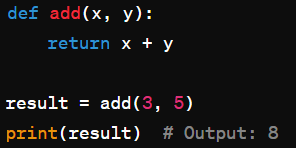
* **Identity Operators:** Used to compare the memory locations of two objects.



* **Membership Operators:** Used to test whether a value or variable is found in a sequence (e.g., lists, tuples, or strings).



* **The Return Statement**
* the return statement is used within a function to specify the value that the function should return when called. It allows a function to pass data back to the code that called it.



* In this example, the add function takes two arguments (x and y), adds them together, and returns the result using the return statement.