Creating a Simple Function

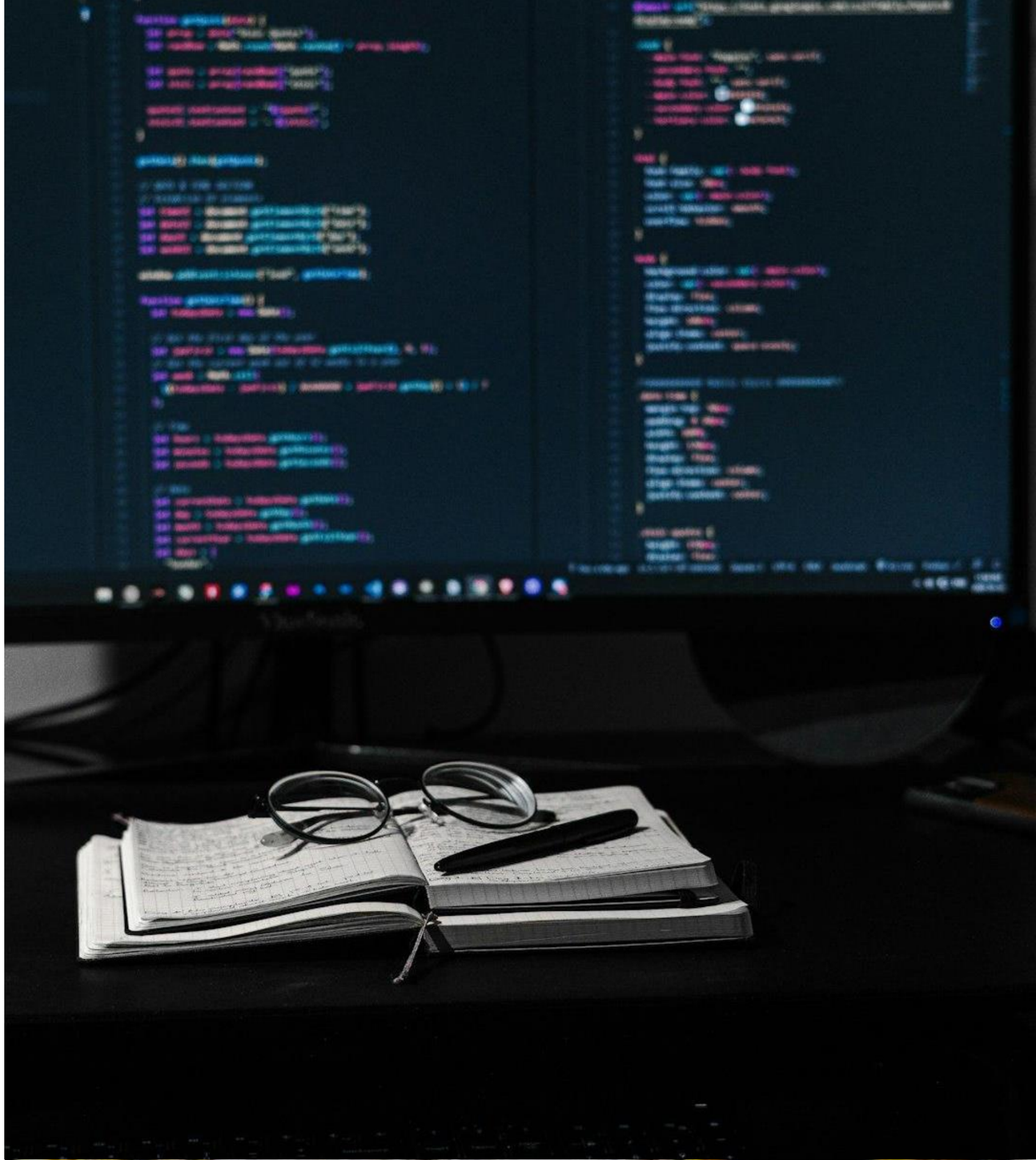
- A function is a block of code that runs when called.
- Objective: Understand how to create a basic function in Python.

Example:

```python
def greet():
    print("Hello from a function")

greet()  # Calling the function
```

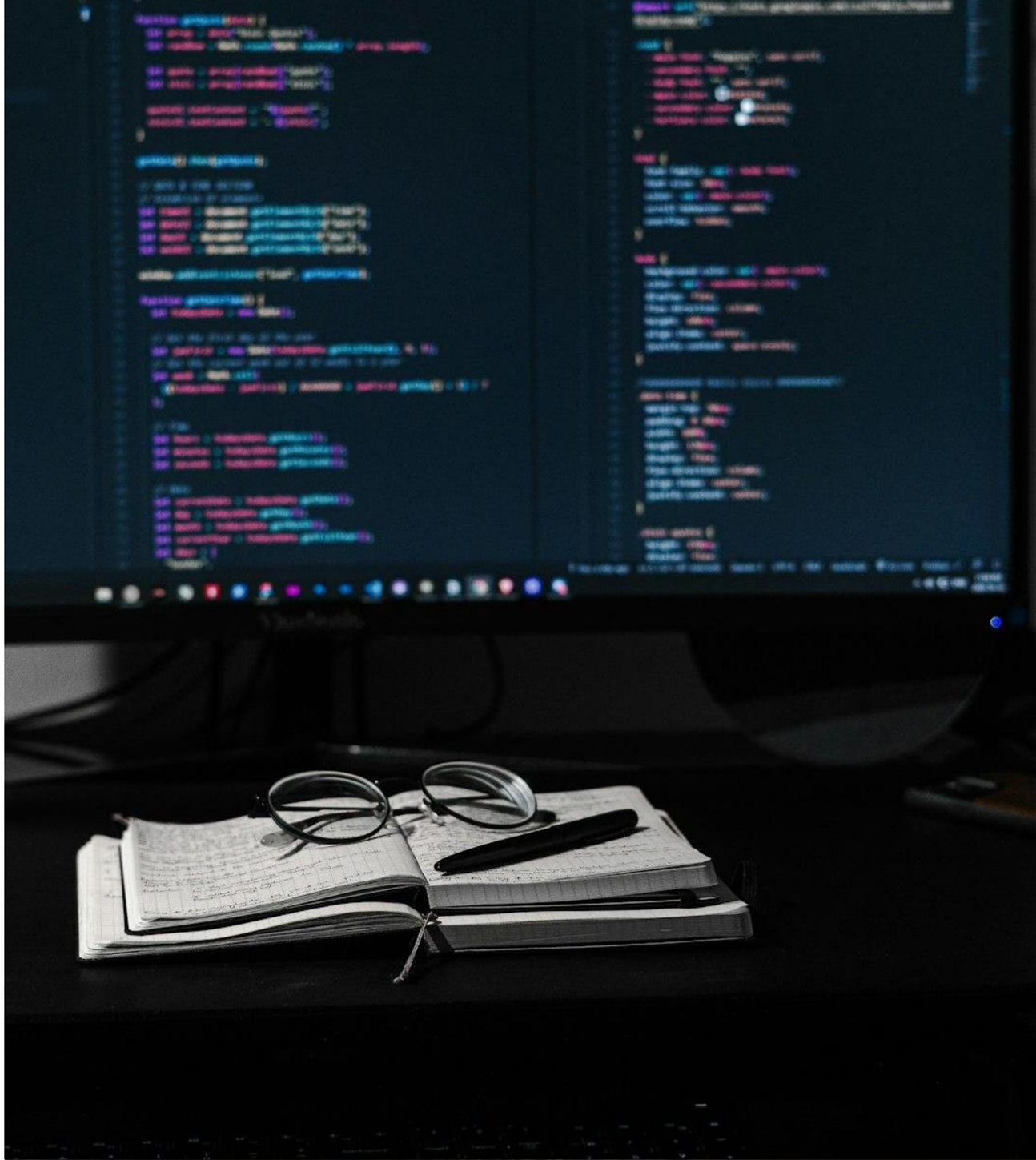
Function with Arguments

- Arguments allow values to be passed into a function.
- Objective: Learn how to pass and use arguments.
- Example:
- ```python
- def greet_name(name):
-     print("Hello, " + name)

- greet_name("Alice") # Passing the argument "Alice"
- ```



Function with Multiple Arguments

- Functions can accept multiple arguments.
- Objective: Use more than one argument to make functions flexible.
- Example:
- ```python
- def full_name(first_name, last_name):
-     print(first_name + " " + last_name)

- full_name("John", "Doe") # Passing two arguments
- ```



Arbitrary Arguments (*args)

- Use *args to handle unknown numbers of positional arguments.
- Objective: Enable functions to accept variable-length arguments.
- Example:
- ```python
- def greet_all(*names):
-     for name in names:
-         print("Hello, " + name)

- greet_all("Alice", "Bob", "Charlie")
- ```

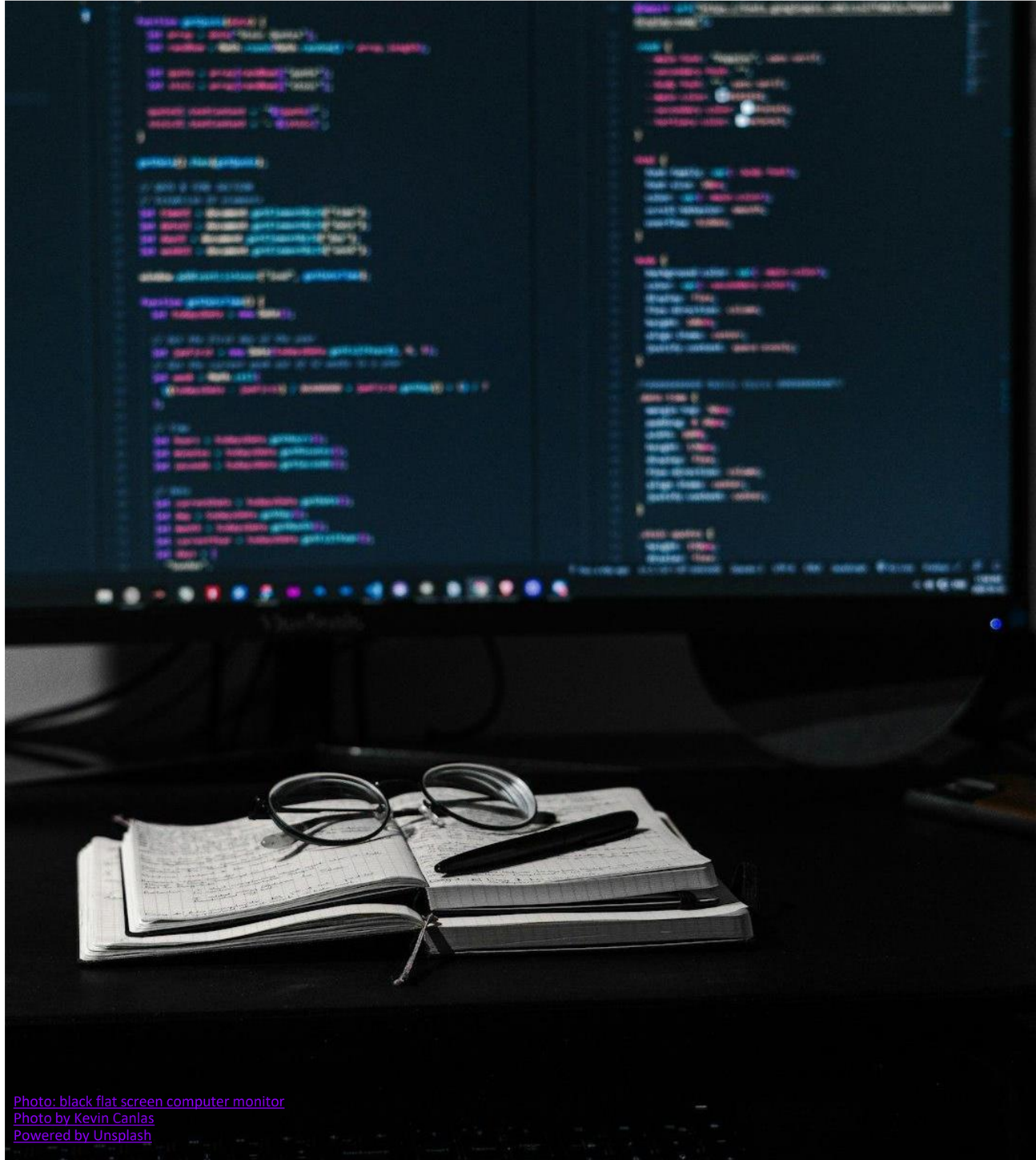
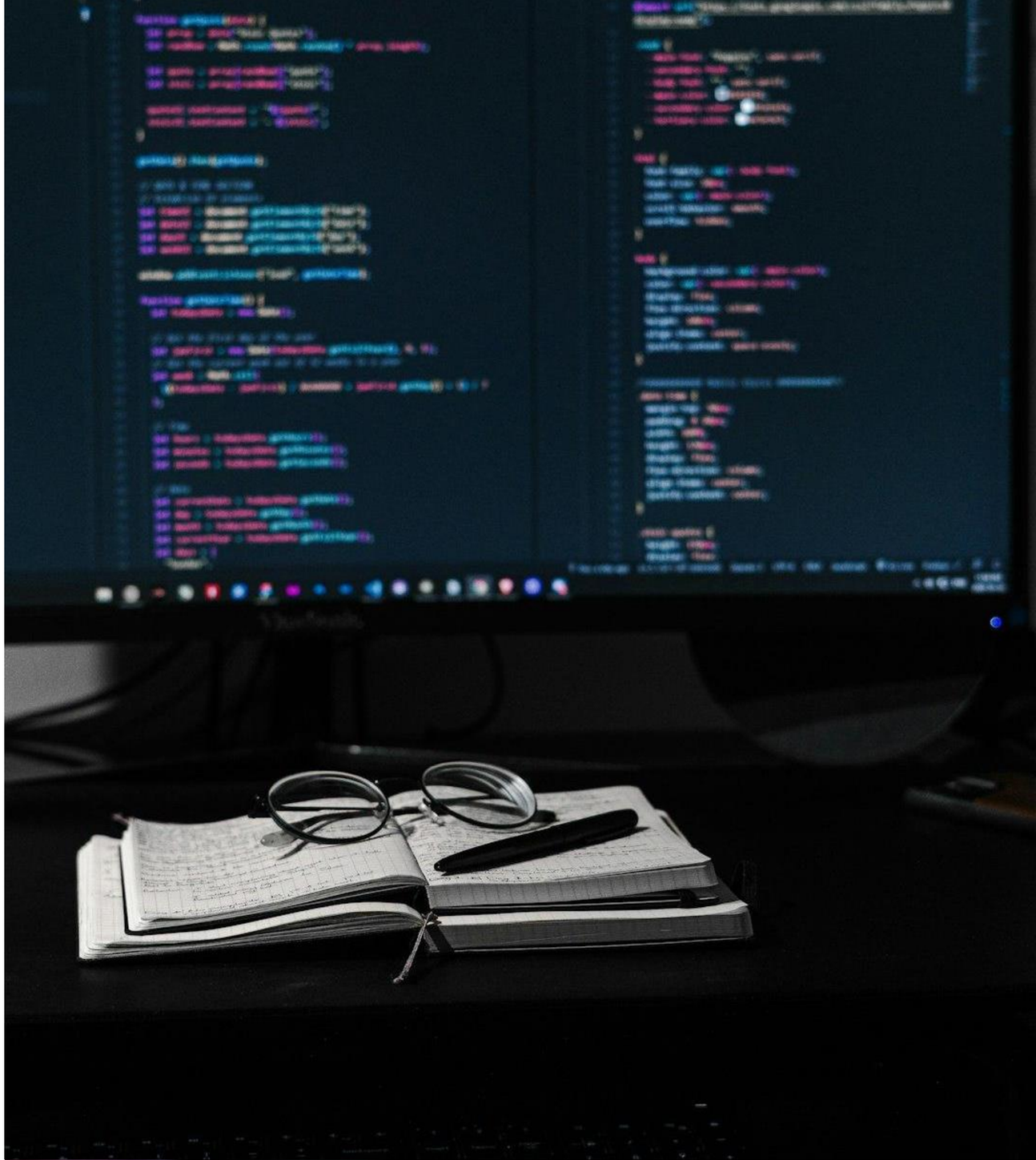


Photo: black flat screen computer monitor
Photo by Kevin Canlas
Powered by Unsplash

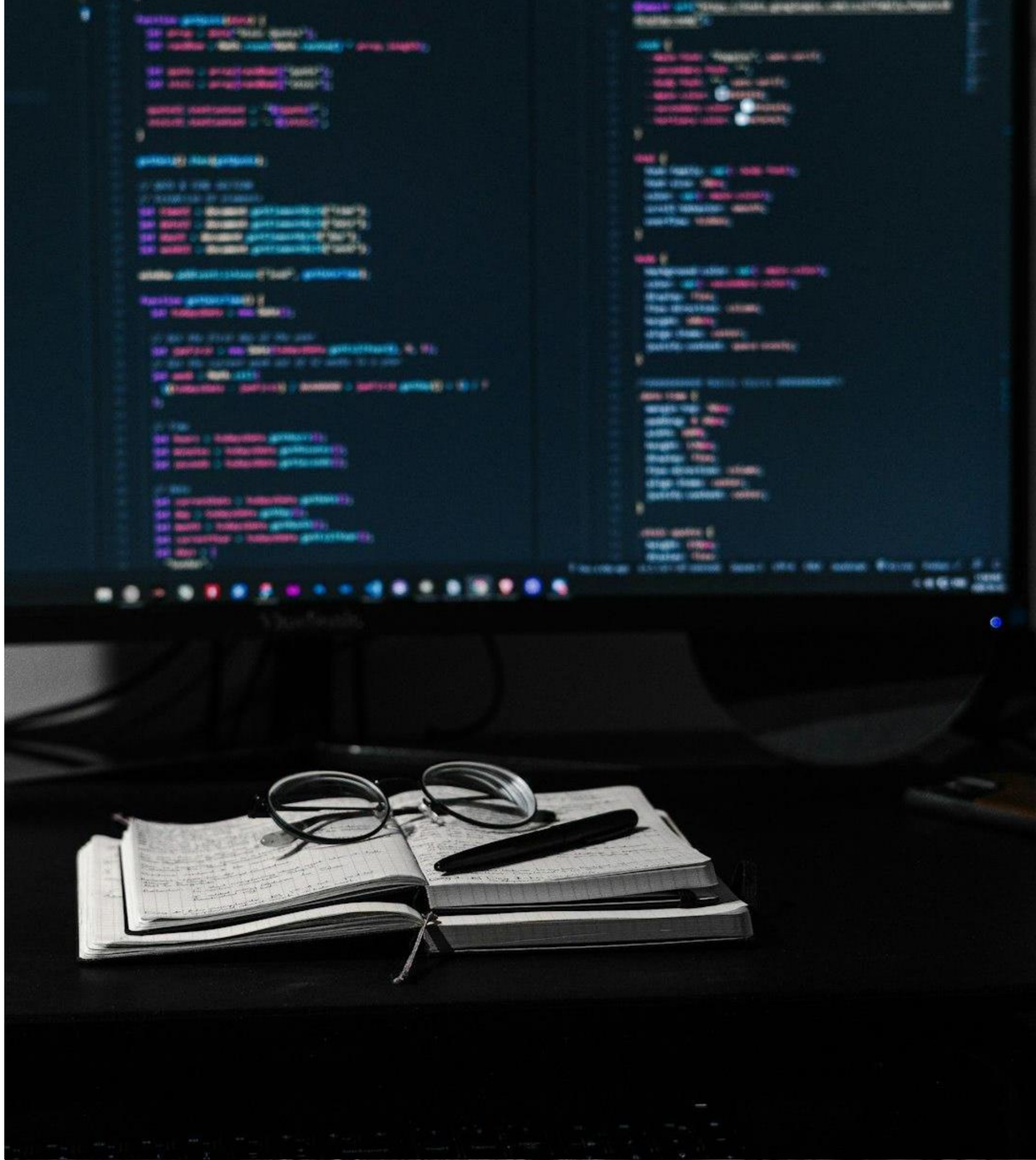
Keyword Arguments

- Arguments can be passed with names for clarity.
- Objective: Understand how keyword arguments work.
- Example:
- ```python
- def introduce(name, age, country):
- print(f"My name is {name}, I am {age} years old, and I am from {country}.")

- introduce(age=30, name="John", country="USA")
- ```



Arbitrary Keyword Arguments (**kwargs)

- Use **kwargs to handle unknown numbers of keyword arguments.
- Objective: Enable functions to accept flexible keyword arguments.
- Example:
- ```python
- def person_info(**kwargs):
-     for key, value in kwargs.items():
-         print(f"{key}: {value}")

- person_info(name="Alice", age=25, city="New York")
- ```



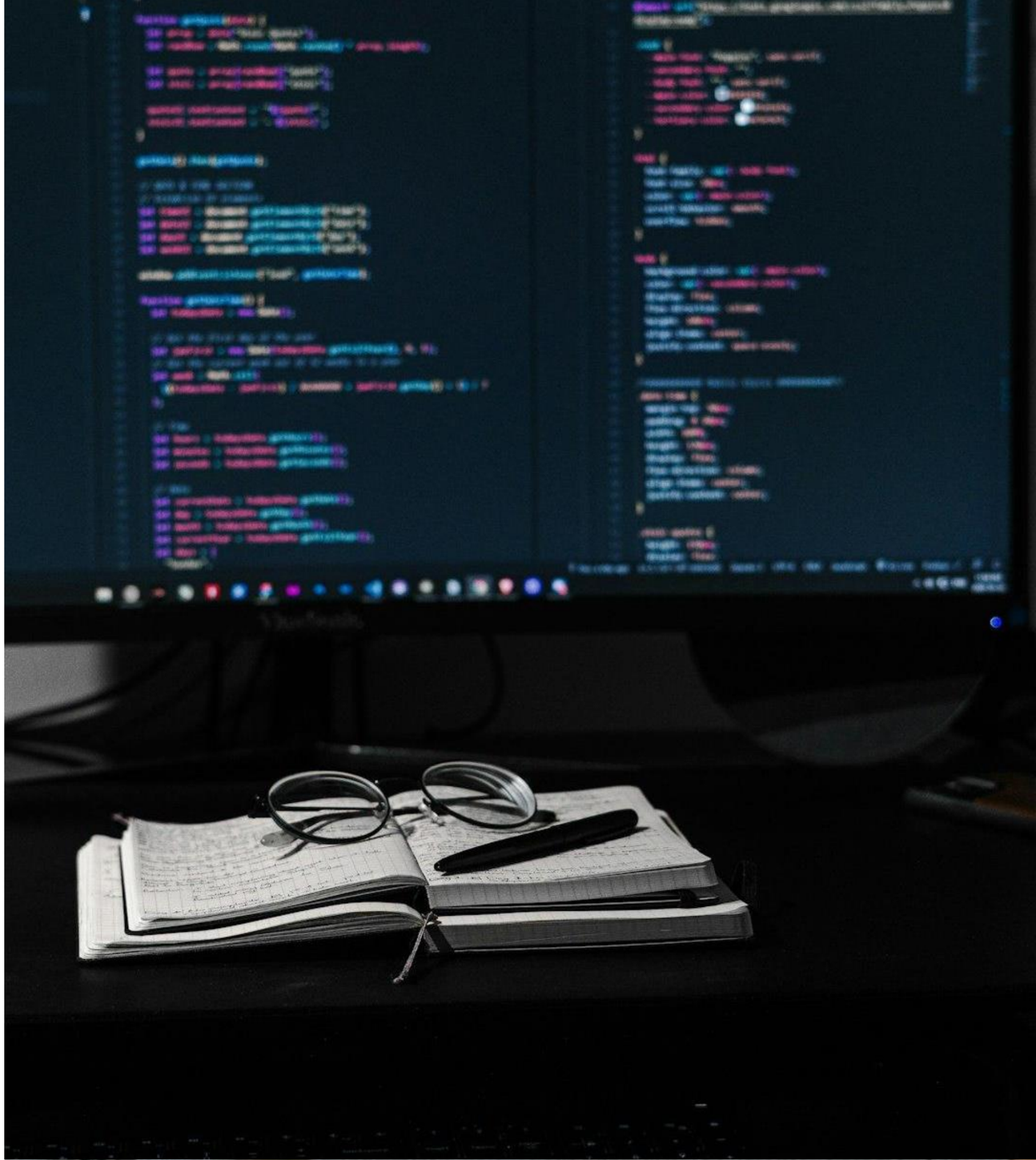
Default Parameter Value

- Assign default values to function parameters.
- Objective: Avoid passing values explicitly if defaults suffice.
- Example:
- ```python
- def greet(country="USA"):
-     print(f"I am from {country}")

- greet() # Uses default value
- greet("India") # Uses passed value
- ```

Photo: black flat screen tv turned on near black laptop computer
Photo by Boitumelo
Powered by Unsplash

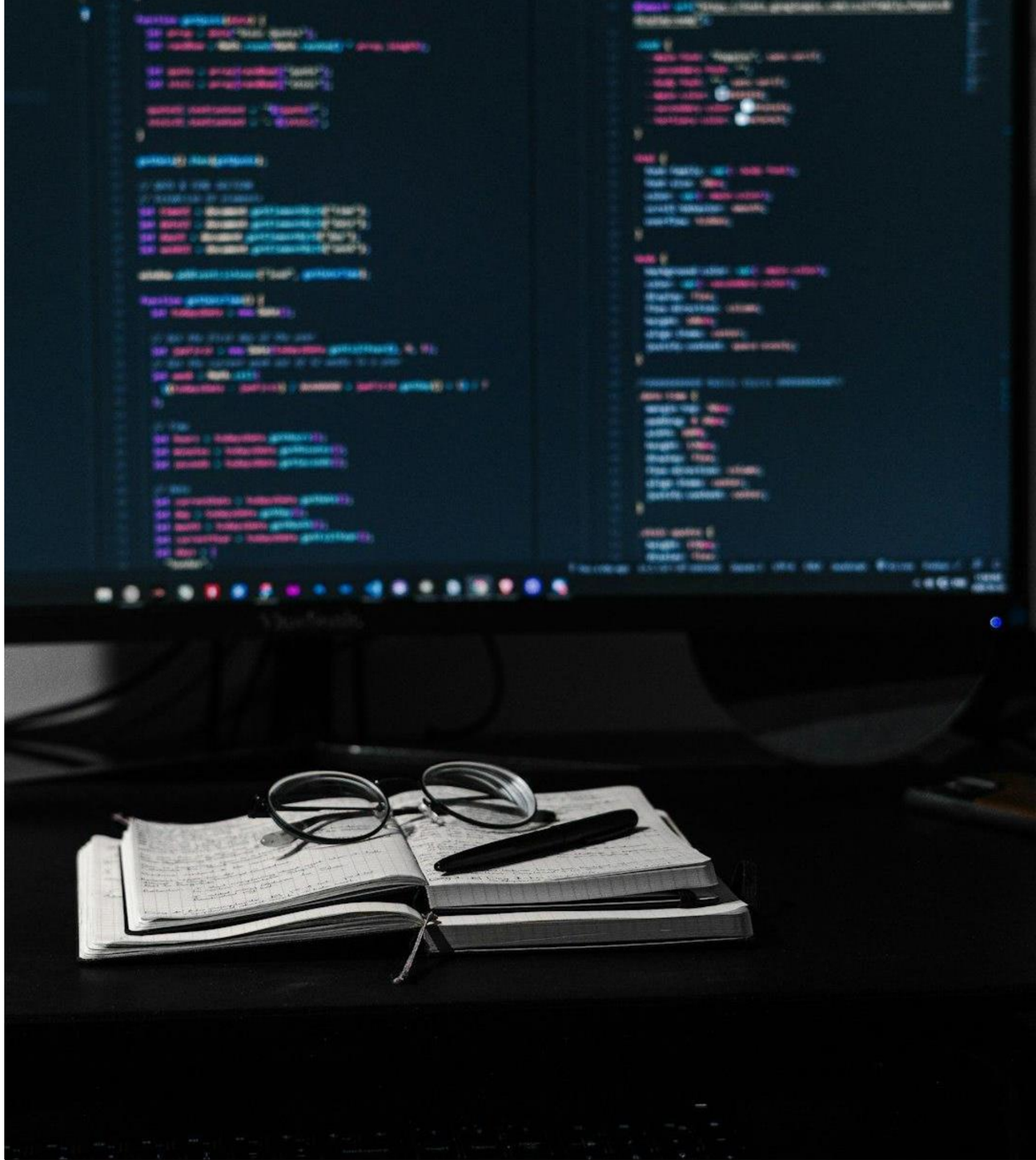
Passing a List as an Argument

- Functions can accept lists as arguments.
- Objective: Learn how to handle iterable data types in functions.
- Example:
- ```python
- def print_foods(food_list):
-     for food in food_list:
-         print(food)

- foods = ["Pizza", "Burger", "Pasta"]
- print_foods(foods)
- ```



Return Values

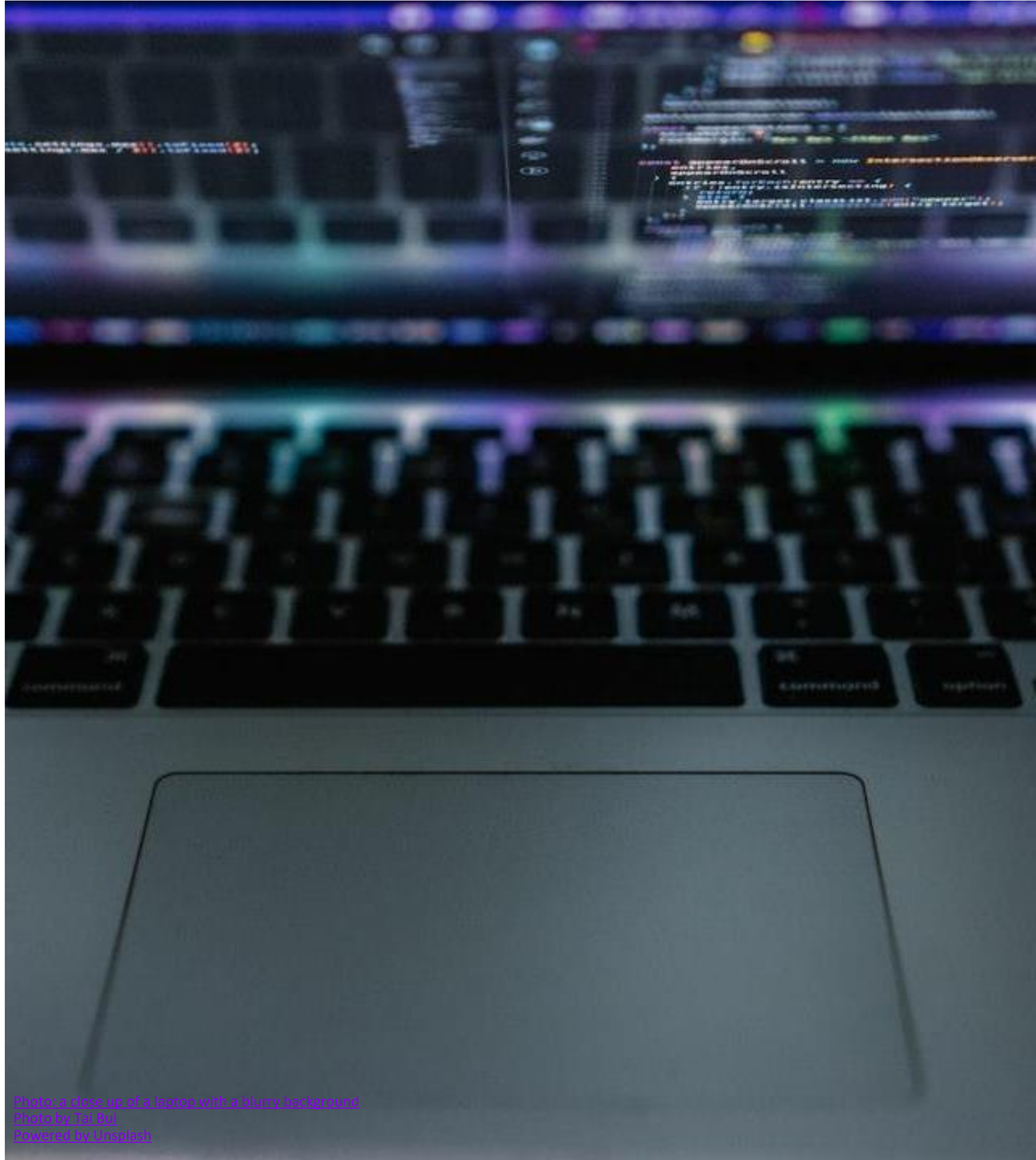
- Functions can return values to the caller.
- Objective: Use the return keyword for computation or data retrieval.
- Example:
- ```python
- def multiply(x, y):
-     return x * y

- result = multiply(3, 4)
- print(result) # Output: 12
- ```



The pass Statement

- Create placeholder functions using pass.
- Objective: Reserve function names for future use.
- Example:
- ```python
- def my_function():
-     pass # Placeholder for future implementation

- my_function()
- ```





Positional-Only Arguments

- Use / to enforce positional-only arguments.
- Objective: Control how arguments are passed to a function.
- Example:
- ```python
- def greet_name(name, /):
-     print(f"Hello, {name}")

- greet_name("Alice") # Valid
- # greet_name(name="Alice") # Error
- ```

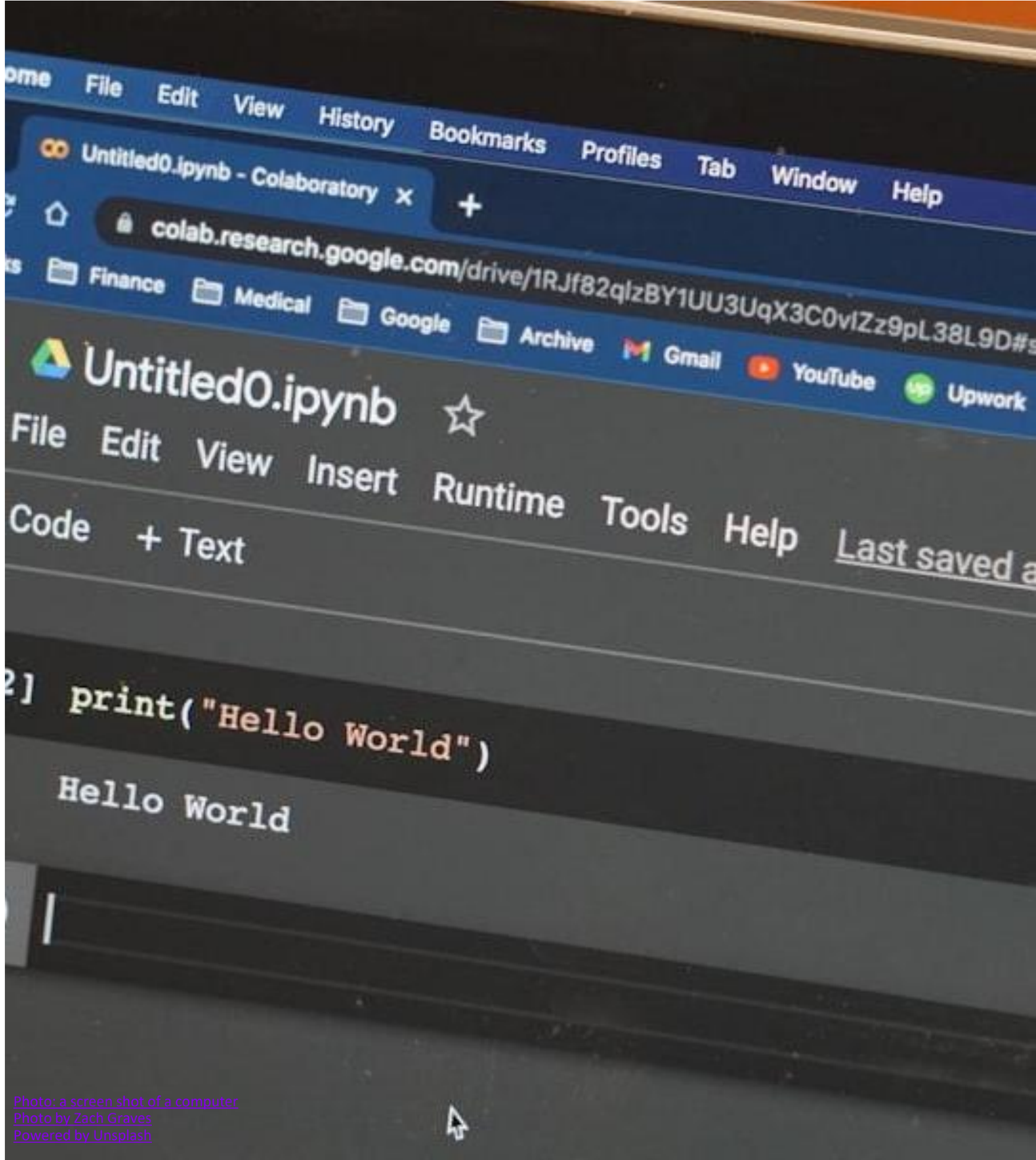
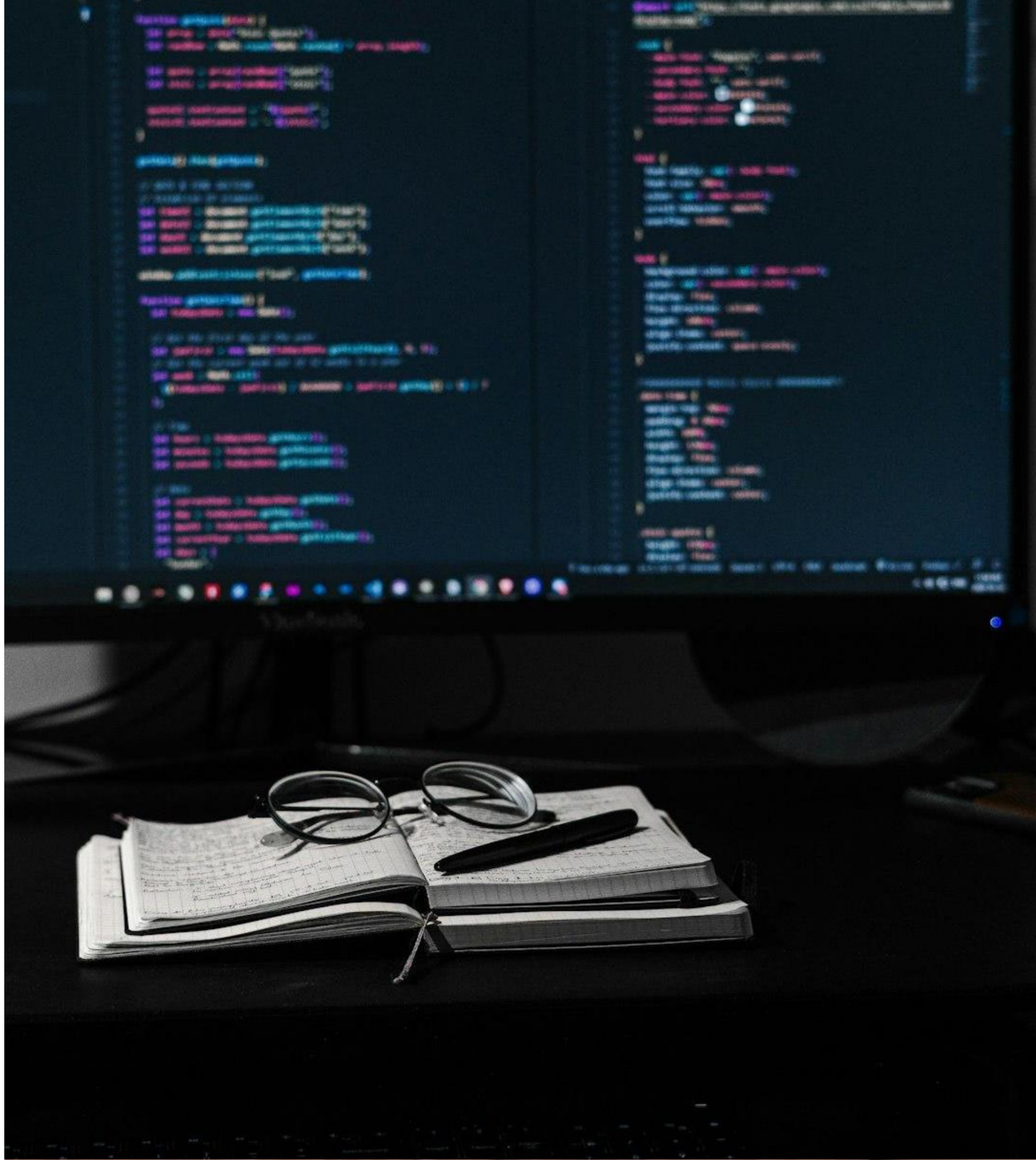


Photo: a screen shot of a computer
Photo by Zach Graves
Powered by Unsplash

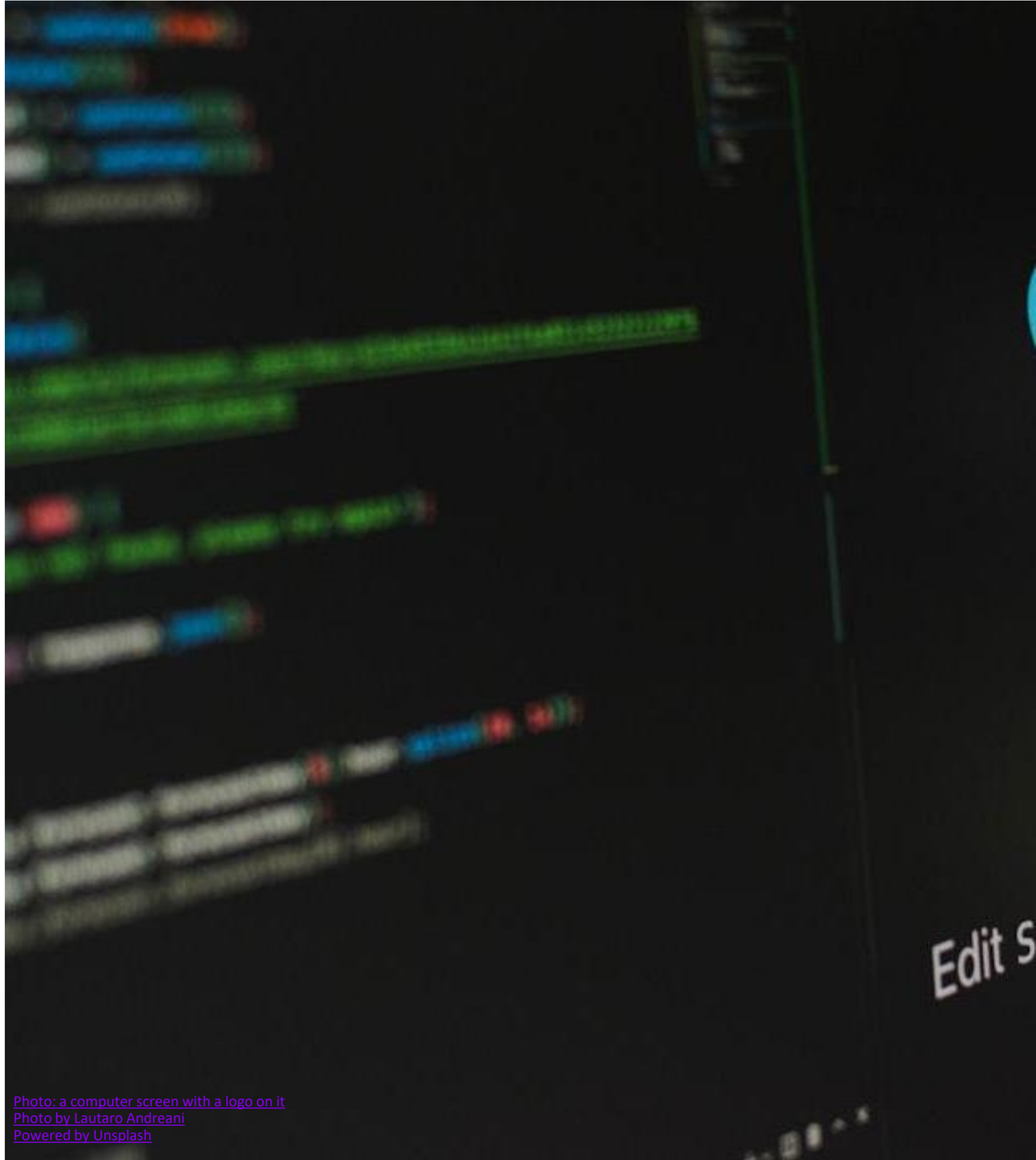
Keyword-Only Arguments

- Use * to enforce keyword-only arguments.
- Objective: Avoid positional arguments for specific parameters.
- Example:
- ```python
- def greet(*, name):
-     print(f"Hello, {name}")

- greet(name="Alice") # Valid
- # greet("Alice") # Error
- ```



Combine Positional-Only and Keyword-Only Arguments

- Combine / and * for both argument types.
- Objective: Create advanced, flexible function signatures.
- Example:
- ```python
- def func(a, b, /, *, c, d):
-     print(f"a={a}, b={b}, c={c}, d={d}")

- func(1, 2, c=3, d=4)
- ```



Recursion Example

- Recursive functions call themselves for repeated computation.
- Objective: Solve problems like summation or factorial.
- Example:
- ```python
- def tri_recursion(k):
-     if k > 0:
-         result = k + tri_recursion(k - 1)
-         print(result)
-     else:
-         result = 0
-     return result

- tri_recursion(6)
- ```