In [ ]:
```
1. Difference between a built-in function and a user-defined function
```

In [11]:
```
# Built-in Function: Provided by Python and available without any import (e.g., len()).

# Built-in function example
print(len("Hello"))  # Output: 5
```
```
5
```

In [13]:
```
# User-defined Function: Created by the programmer to perform specific tasks.

# User-defined function example
def greet(name):
    return f"Hello, {name}!"
print(greet("Alice"))  # Output: Hello, Alice!
```
```
Hello, Alice!
```

In [ ]:
```
2. Passing arguments to a function: Positional vs. Keyword Arguments
```

In [17]:
```
# Positional Arguments: Passed in the same order as defined in the function.

def subtract(a, b):
    return a - b
print(subtract(10, 5))  # Output: 5
```
```
5
```

In [19]:
```
# Keyword Arguments: Specify the argument name explicitly.

def subtract(a, b):
    return a - b
print(subtract(b=5, a=10))  # Output: 5
```
```
5
```

In [ ]:
```
3. Purpose of the return statement
```

In [21]:
```
# The return statement allows a function to send a value back to the caller. A function can have
# multiple return statements but only one is executed per call.

def check_number(num):
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    return "Zero"
print(check_number(10))  # Output: Positive
```
```
Positive
```

In [ ]:
```
4. Lambda Functions
```

In [23]:
```
# Lambda Functions: Anonymous, single-expression functions.

square = lambda x: x ** 2
print(square(5))  # Output: 25
```
```
25
```

In [25]:
```
# Use Case: Useful for short operations like sorting.

data = [(1, 'Alice'), (2, 'Bob'), (3, 'Charlie')]
data.sort(key=lambda x: x[1])
print(data)  # Output: [(1, 'Alice'), (2, 'Bob'), (3, 'Charlie')]
```
```
[(1, 'Alice'), (2, 'Bob'), (3, 'Charlie')]
```

In [ ]:
```
5. Scope in Python
```

In [27]:
```
# Local Scope: Variables defined within a function.

def func():
    x = 10  # Local variable
    print(x)
func()
```
```
10
```

In [29]:
```
# Global Scope: Variables defined outside all functions, accessible globally.

x = 10  # Global variable
def func():
    print(x)
func()
```
```
10
```

In [ ]:
```
6. Returning Multiple Values
```

In [31]:
```
# A function can return multiple values using tuples.

def calculations(a, b):
    return a + b, a - b, a * b
add, sub, mul = calculations(10, 5)
print(add, sub, mul)  # Output: 15 5 50
```
```
15 5 50
```

In [ ]:
```
7. Pass by Value vs. Pass by Reference
```

In [37]:
```
# Pass by Value: The function receives a copy of the variable's value (not applicable in Python).
# Pass by Reference: The function receives a reference to the variable, allowing modifications.

def modify_list(lst):
    lst.append(10)
nums = [1, 2, 3]
modify_list(nums)
print(nums)  # Output: [1, 2, 3, 10]
```
```
[1, 2, 3, 10]
```

In [ ]:
```
8. Function for Mathematical Operations
```

In [35]:
```
import math

def math_operations(x):
    return {
        "log": math.log(x),
        "exp": math.exp(x),
        "power_base_2": 2 ** x,
        "square_root": math.sqrt(x),
    }

result = math_operations(2)
print(result)
```
```
{'log': 0.6931471805599453, 'exp': 7.38905609893065, 'power_base_2': 4, 'square_root': 1.4142135623730951}
```

In [ ]:
```
9. Function to Extract First and Last Name
```

In [39]:
```
def split_name(full_name):
    names = full_name.split()
    return names[0], names[-1]
```

```python
first_name, last_name = split_name("John Doe")
print(f"First Name: {first_name}, Last Name: {last_name}")
```

```
First Name: John, Last Name: Doe
```