In [3]:
```
Q.1: In Python, what is the difference between a built-in function and a user-defined function? Provide an
     example of each.
ans : Built-in functions are predefined in the Python language, while user-defined functions are created by
      #the programmer to add custom functionality to their programs.'''
```

In [1]:
```
Q.2: How can you pass arguments to a function in Python? Explain the difference between positional
     arguments and keyword arguments.
Ans: There are two ways to pass arguments to a function in Python:
     positional arguments and keyword arguments.
     Positional arguments are arguments that are passed to a function based on their position in the argument list.
     For example, in the following function call, the argument 1 is passed to the x parameter and the argument 2 is
     passed to the y parameter:
     def add_numbers(x, y):
     return x + y

    add_numbers(1, 2)
    Keyword arguments are arguments that are passed to a function by specifying the parameter name and its corresponding value.
    For example, in the following function call, the argument x is passed to the x parameter and the argument y is passed to the
    y parameter, even though they are passed in the opposite order than they are defined in the function:
    add_numbers(y=2, x=1)
```

In [ ]:
```
Q.3:. What is the purpose of the return statement in a function? Can a function have multiple return
      statements? Explain with an example.
ANS : The return statement in a function is used to terminate the execution of the function and return a value to the caller.
      It allows a function to pass back a result to the code that called it?

      a function can have multiple return statements. However, only one of them will be executed, and it depends on the condition or
      path taken within the function. Once a return statement is executed, the function exits, and no further code within the function is executed.
```

In [ ]:
```
Q.4 : What are lambda functions in Python? How are they different from regular functions? Provide an
      example where a lambda function can be useful.
ANS:  Lambda functions are similar to user-defined functions but without a name. They are  commonly referred to as anonymous functions.
      Lambda functions are efficient whenever you want to create a function that will only contain simple expressions
      that is, expressions that are usually a single line of a statement.
      Lambda functions are often used in scenarios where a small, temporary function is needed, for example, when using functions like map(),
      filter(), or sorted().

#  syntax is : lambda arguments: expression
example
# Regular function to add two numbers
def add_numbers(x, y):
    return x + y

# Equivalent lambda function
add_lambda = lambda x, y: x + y

# Using both functions
result1 = add_numbers(3, 5)
result2 = add_lambda(3, 5)

print(result1)
print(result2)
```

In [ ]:
```
Q.5: How does the concept of "scope" apply to functions in Python? Explain the difference between local
    scope and global scope.
    In Python, the concept of "scope" refers to the region of the code where a variable is accessible or where it
    can be modified. Python has two main types of scopes: local scope and global scope.

Local Scope:

        1. Variables defined within a function have local scope. They are only accessible within that specific function.
        2.  Once the function finishes executing, the local variables are destroyed, and their values are no longer accessible.
def example_function():
    local_variable = 10
    print(local_variable)

example_function()


Global Scope:

        Variables defined outside any function or in the global scope are accessible throughout the entire program.
        Global variables can be used both inside functions and outside functions.
        If a variable with the same name exists both globally and locally within a function, the local variable takes precedence within that function.


global_variable = 20

def example_function():
    local_variable = 10
    print(f"Local variable: {local_variable}, Global variable: {global_variable}")

example_function()
# Output: Local variable: 10, Global variable: 20

print(f"Global variable outside the function: {global_variable}")
# Output: Global variable outside the function: 20
```

In [ ]:
```
Q.6 : . How can you use the "return" statement in a Python function to return multiple values?
Ans: In Python, you can use the return statement to return multiple values from a function by separating them with commas.
     The values will be packed into a tuple, and this tuple can be unpacked when the function is called.
     example:

def multiple_values():
    value1 = 10
    value2 = "Hello"
    value3 = [1, 2, 3]

    # Using the return statement to return multiple values
    return value1, value2, value3

# Calling the function and unpacking the returned tuple
result1, result2, result3 = multiple_values()

print(result1)  # Output: 10
print(result2)  # Output: Hello
print(result3)  # Output: [1, 2, 3]
```

In [ ]:
```
Q.7 : What is the difference between the "pass by value" and "pass by reference" concepts when it
      comes to function arguments in Python?
ANS: Pass by Value:

In pass by value, a copy of the actual value of the variable is passed to the function.
Changes made to the parameter inside the function do not affect the original variable.

def pass_by_value_example(x):
    x = 20

original_value = 10
pass_by_value_example(original_value)

print(original_value)  # Output: 10


Pass by Reference (or Object Reference in Python):

In pass by reference, a reference to the memory location of the variable is passed to the function.
Changes made to the parameter inside the function affect the original variable.

def pass_by_reference_example(lst):
    lst.append(4)

original_list = [1, 2, 3]
pass_by_reference_example(original_list)

print(original_list)  # Output: [1, 2, 3, 4]
```

In [ ]:
```
Q.8:  Create a function that can intake integer or decimal value and do following operations:
a. Logarithmic function (log x)
b. Exponential function (exp(x))
c. Power function with base 2 (2^x)
d. Square root

import math

def mathematical_operations(x):
    # Logarithmic function (log x)
    log_result = math.log(x)

    # Exponential function (exp(x))
    exp_result = math.exp(x)

    # Power function with base 2 (2^x)
    power_result = 2 ** x

    # Square root
    sqrt_result = math.sqrt(x)

    return log_result, exp_result, power_result, sqrt_result

# Example usage:
input_value = 4.0

logarithmic, exponential, power, square_root = mathematical_operations(input_value)

print(f"Logarithmic function: {logarithmic}")
print(f"Exponential function: {exponential}")
print(f"Power function with base 2: {power}")
print(f"Square root: {square_root}")
```