1. In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

Ans:- In Python, the main difference between a built-in function and a user-defined function lies in their origin and usage. (1)Built-in function: A built-in function is a function that comes pre-defined in Python's standard library. These functions are readily available for use without requiring any additional import or module installation. They cover a wide range of tasks and operations, making it convenient for developers to perform common tasks easily. Examples of built-in functions include print(), len(), sum(), abs(), etc For Example:-

In [1]:

```
# Using the built-in function print() to display a message
print("This is a built-in function example.")
```

```
This is a built-in function example.
```

(2)User-defined function: A user-defined function is a function created by the programmer to perform a specific task or operation. These functions are defined using the def keyword and can take parameters, execute a block of code, and return values if necessary. User-defined functions allow developers to encapsulate a set of instructions into a single unit, promoting code reusability and maintainability. For Example:-

In [2]:

```
# User-defined function to calculate the area of a rectangle
def calculate_area(length, width):
    area = length * width
    return area

# Using the user-defined function to calculate the area of a rectangle
length = 5
width = 3
rectangle_area = calculate_area(length, width)
print("The area of the rectangle is:", rectangle_area)
```

```
The area of the rectangle is: 15
```

2. How can you pass arguments to a function in Python? Explain the difference between positional arguments and keyword arguments.

Ans:- (1)Positional Arguments: Positional arguments are the most common way of passing arguments to a function. When you call a function and provide values, they are assigned to the function's parameters in the order they appear in the function definition. The position of the argument determines which parameter it corresponds to. For Example:-

In [3]:

```
def add(a, b):
    return a + b

result = add(3, 5)
print(result)
```

```
8
```

(2)Keyword Arguments:
Keyword arguments allow you to pass values to a function by explicitly specifying the parameter names. By using keyword arguments, you can pass the arguments in any order, regardless of the parameter order in the function definition.
For Example:-

In [4]:

```
def subtract(a, b):
    return a - b

result = subtract(b=2, a=7)
print(result)
```

```
5
```

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 serves the purpose of specifying the value that the function will return when it is called. When a return statement is encountered in a function, the function execution is immediately halted, and the value following the return keyword is sent back to the caller. This value can then be used or assigned to a variable as needed. For Example:-

In [5]:

```
def find_grade(score):
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

print(find_grade(85))
print(find_grade(95))
print(find_grade(60))
print(find_grade(45))
```

```
B
A
D
F
```

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:- In Python, a lambda function is a small, anonymous function that can have any number of arguments but can only have one expression. Lambda functions are defined using the lambda keyword, followed by the arguments (if any) separated by commas, a colon, and then the expression. The lambda function returns the value of the expression when called. For Example:-

In [7]:

```
people = [('Alice', 25), ('Bob', 30), ('Charlie', 20), ('David', 28)]
sorted_people = sorted(people, key=lambda x: x[1])
print(sorted_people)
```

```
[('Charlie', 20), ('Alice', 25), ('David', 28), ('Bob', 30)]
```

5. How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.

Ans:- (1) Local Scope: Local scope refers to the area within a function where variables are defined and can only be accessed within that function. These variables are known as local variables. Once the function execution is completed, the local variables are destroyed, and their values are no longer accessible. Local scope is created when a function is called and destroyed when the function returns. For Example:-

In [11]:

```
def my_function():
    x = 10  # x is a local variable
    print("Local variable x:", x)

my_function()
# The following line will raise a NameError since x is not defined in this scope
# print(x)
```

```
Local variable x: 10
```

(2)Global Scope:

Global scope refers to the area outside of any function or block in the code, where variables are defined and can be accessed throughout the program. These variables are known as global variables. Global variables retain their values until the program execution ends or until they are explicitly modified. For Example:-

In [12]:

```
y = 20  # y is a global variable

def my_function():
    global y  # Using 'global' keyword to indicate that we want to use the global y
    print("Global variable y:", y)

my_function()
print("Global variable y (outside function):", y)
```

```
Global variable y: 20
Global variable y (outside function): 20
```

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 in a function to return multiple values by packing them into a data structure such as a tuple, list, or dictionary. Here's how you can do it using different data structures:

In [13]:

```
## Using Tuple
def return_multiple_values():
    value1 = 42
    value2 = "Hello, world!"
    return value1, value2

result_tuple = return_multiple_values()
print(result_tuple)  # Output: (42, 'Hello, world!')
```

```
(42, 'Hello, world!')
```

In [14]:

```
## Using List
def return_multiple_values():
    value1 = 42
    value2 = "Hello, world!"
    return [value1, value2]

result_list = return_multiple_values()
print(result_list)  # Output: [42, 'Hello, world!']
```

```
[42, 'Hello, world!']
```

In [15]:

```
## Using Dictionary
def return_multiple_values():
    value1 = 42
    value2 = "Hello, world!"
    return {'value1': value1, 'value2': value2}

result_dict = return_multiple_values()
print(result_dict)  # Output: {'value1': 42, 'value2': 'Hello, world!'}
```

```
{'value1': 42, 'value2': 'Hello, world!'}
```

7. What is the difference between the "pass by value" and "pass by reference" concepts when it comes to function arguments in Python?

Ans:- (1)Pass by Value: In a pure pass-by-value language, a copy of the value of the variable is passed to the function. Any modifications made to the function parameter inside the function do not affect the original variable outside the function. This means that the function works with a local copy of the variable. For Example:-

In [1]:

```
def modify_value(num):
    num += 1
    print("Inside function:", num)

x = 5
modify_value(x)
print("Outside function:", x)
```

```
Inside function: 6
Outside function: 5
```

(2)Pass by Reference:

In a pure pass-by-reference language, the function parameter is just an alias or reference to the original variable. Any changes made to the parameter inside the function will directly affect the original variable outside the function. For Example:-

In [2]:

```
def modify_list(lst):
    lst.append(4)
    print("Inside function:", lst)

my_list = [1, 2, 3]
modify_list(my_list)
print("Outside function:", my_list)
```

```
Inside function: [1, 2, 3, 4]
Outside function: [1, 2, 3, 4]
```

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 (2x)

d. Square root

In [3]:

```
import math

def math_operations(x):
    try:
        x = float(x)  # Convert the input to a float in case an integer is provided
    except ValueError:
        return "Invalid input. Please provide a numeric value."

    log_x = math.log(x)  # Logarithmic function (base e)
    exp_x = math.exp(x)  # Exponential function (e^x)
    power_2_x = 2 ** x   # Power function with base 2 (2^x)
    sqrt_x = math.sqrt(x)  # Square root

    results = {
        "log_x": log_x,
        "exp_x": exp_x,
        "power_2_x": power_2_x,
        "sqrt_x": sqrt_x}

    return results

# Test the function
input_value = input("Enter an integer or decimal value: ")
results = math_operations(input_value)
print("Logarithmic function (log x):", results["log_x"])
print("Exponential function (exp(x)):", results["exp_x"])
print("Power function with base 2 (2^x):", results["power_2_x"])
print("Square root:", results["sqrt_x"])
```

```
Enter an integer or decimal value: 5
Logarithmic function (log x): 1.6094379124341003
Exponential function (exp(x)): 148.4131591025766
Power function with base 2 (2^x): 32.0
Square root: 2.23606797749979
```

9. Create a function that takes a full name as an argument and returns first name and last name.

In [4]:

```
def get_first_and_last_name(full_name):
    name_parts = full_name.split()

    # Extract the first name
    first_name = name_parts[0]

    # Extract the last name
    last_name = name_parts[-1]

    # Return the first name and last name as a tuple
    return first_name, last_name

# Test the function
full_name = "John Doe"
first_name, last_name = get_first_and_last_name(full_name)
print("First Name:", first_name)
print("Last Name:", last_name)
```

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

In [ ]:

```

```