Q1. In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

Ans: built-in function: Built-in functions are pre-defined functions provided by Python's standard library. These functions are readily available for use without the need for explicit definition or import. Examples: Common examples of built-in functions include print(), len(), range(), abs(), type(), and many others.

user-defined function: User-defined functions are functions created by the programmer to perform specific tasks. These functions are defined using the def keyword followed by the function name and parentheses.

```
print("Hello, World!")  # Using the built-in print() function
```

```
Hello, World!
```

```
# Using the user-defined function
def greet(name):
    return f"Hello, {name}!"
print(greet("Harshal"))
```

```
Hello, Harshal!
```

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

Ans: Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
"""
Ans.2
Positional Arguments: Positional arguments are arguments that are
passed to a function
in a specific order. The position of each argument
in the function call determines which parameter it corresponds to.
"""
#Example:
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

# Calling the function with positional arguments
greet("Harshal", 23)
```

```
Hello, Harshal! You are 23 years old.
```

```
"""
Ans.2
Keyword Arguments: Keyword arguments are passed to a function by
explicitly specifying the parameter name
and the corresponding value. This allows you to pass arguments in any
```

```
order.
"""
#Example:

def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

greet(name="Bob", age=25)
greet(age=26, name="Carol")
```

```
Hello, Bob! You are 25 years old.
Hello, Carol! You are 26 years old.
```

Q3. What is the purpose of the return statement in a function? Can a function have multiple return statements? Explain with an example.

Ans:Yes, a function can indeed have multiple return statements. When a return statement is encountered, the function immediately exits and returns the specified value. Having multiple return statements allows you to conditionally return different values based on certain conditions within the function.

```
def compare_numbers(a, b):
    if a > b:
        return f"{a} is greater than {b}"
    elif a < b:
        return f"{a} is less than {b}"
    else:
        return f"{a} is equal to {b}"

#  examples
print(compare_numbers(5, 3))   # Output: "5 is greater than 3"
print(compare_numbers(3, 5))   # Output: "3 is less than 5"
print(compare_numbers(4, 4))   # Output: "4 is equal to 4"
```

```
5 is greater than 3
3 is less than 5
4 is equal to 4
```

Q4. 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, also known as anonymous functions or lambda expressions, are a way of creating small, unnamed functions in Python. They are defined using the lambda keyword, which takes any number of arguments and a single expression. Lambda functions are typically used for short functions where defining a separate function using def would be overkill.

```
numbers = [1, 2, 3, 4, 5]

# Using map() with a lambda function to square each number
squared_numbers = map(lambda x: x ** 2, numbers)
```

```python
print(list(squared_numbers))
```

```
[1, 4, 9, 16, 25]
```

Q5. How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.

Ans:In Python, the concept of "scope" refers to the visibility and accessibility of variables within different parts of a program. The scope of a variable determines where in the code that variable can be accessed and modified. Difference between is Variables with global scope are available from all other scopes within the JavaScript code. Variables with local scope are available only within a specific local context and are created by keywords, such as var , let , and const .

Q.6. How can you use the "return" statement in a Python function to return multiple values?

Ans:It is possible for a Python function to return multiple values. To To accomplish this, we can combine the desired return values together into a list or a tuple, which we then return. Alternately, we can simply list the return values in the return statement, separated them by commas.

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:When you pass function arguments by reference, those arguments are only references to existing values. In contrast, when you pass arguments by value, those arguments become independent copies of the original values.

Q8. 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

```python
## creating the required function
import math

def math_operations(x):
    results = {
        'logarithmic': math.log(x),
        'exponential': math.exp(x),
        'power_of_2': 2 ** x,
        'square_root': math.sqrt(x)
    }
    return results

# Example of the code and the  usage:
number = 4
```

```
operations_result = math_operations(number)

print(f"Logarithmic function (log {number}): 
{operations_result['logarithmic']}")
print(f"Exponential function (exp({number})): 
{operations_result['exponential']}")
print(f"Power function with base 2 (2^{number}): 
{operations_result['power_of_2']}")
print(f"Square root of {number}: {operations_result['square_root']}")
```

```
Logarithmic function (log 4): 1.3862943611198906
Exponential function (exp(4)): 54.598150033144236
Power function with base 2 (2^4): 16
Square root of 4: 2.0
```

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

```
# creating the function

def extract_first_last_name(full_name):
    # Split the full name into components based on spaces
    name_parts = full_name.split()

    # Extract the first name and last name
    f_name = name_parts[0]
    l_name = name_parts[-1]

    return f_name, l_name

# Example of the above code and the usage:
full_name = "Harshal kate"
f_name, l_name = extract_first_last_name(full_name)
print("First Name:", f_name)
print("Last Name:",l_name)
```

```
First Name: Harshal
Last Name: kate
```