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

   Built-in Function:

   A built-in function is a function that is already defined and available in Python's standard library. These functions are included in Python by default, and you can directly use them without the need for any additional setup.

   Example of a built-in function:

   The *len()* function is a built-in function in Python that is used to determine the length of a sequence, such as a string, list, or tuple.

   User-defined Function:

   A user-defined function is a function that is created by the programmer to perform specific tasks or operations as per their requirements. This type of functions is defined using the *def* keyword.

   Example of a user-defined function:

   def add(a,b): # a user-defined function that calculate the addition of two numbers

   return a+b

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

   you can pass arguments to a function by specifying them within the parentheses when defining the function and providing values for those arguments when calling the function.

   Positional arguments are passed to a function based on their positions, i.e., the order in which they are defined in the function's parameter list. When calling the function, you need to provide the arguments in the same order as they appear in the function's definition.

   But, keyword arguments are passed to a function using the name of the parameter as a keyword and the corresponding value. When calling the function, you explicitly mention the argument names along with their values, disregarding their order.

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

   The return statement in a function is used to specify the value that the function should send back as the result when it is called.

   A function can indeed have multiple return statements. However, only one return statement is executed in a function call. When a return statement is executed, the function's execution ends, and the returned value (if any) is passed back to the caller. Any subsequent return statements after the first one will not be executed.

   Example of a function with multiple return statements:

```
def get_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"

# Call the function with different scores
score1 = 85
grade1 = get_grade(score1)
print(f"Score: {score1}, Grade: {grade1}")  # Output: Score: 85, Grade: B

score2 = 62
grade2 = get_grade(score2)
print(f"Score: {score2}, Grade: {grade2}")
```

4. **What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.**

   Lambda functions are small and concise functions in Python that can have only one expression. They are defined using the *lambda* keyword, regular functions defined with the *def* keyword, lambda functions don't have a name.

   Lambda functions are different from regular functions in the following ways:

   - Lambda functions are anonymous, meaning they have no name. They are usually used where a small, throwaway function is needed without the overhead of defining a full function using def.
   - Lambda functions can have only one expression, which is automatically returned as the result of the function.
   - Lambda functions are often used for simple operations and can be defined in a single line.

```
# List of tuples containing names and corresponding ages
people = [("Alice", 25), ("Bob", 20), ("Charlie", 30), ("David", 22)]
# Sort the list based on the second element (age) using a lambda function
sorted_people = sorted(people, key=lambda person: person[1])
print(sorted_people)
# Output: [('Bob', 20), ('David', 22), ('Alice', 25), ('Charlie', 30)]
```

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 program where a variable is accessible. The scope determines where the variable can be used, modified, or accessed during the execution of the code. Functions in Python have their own scope, and variables defined inside a function have local scope, while variables defined outside of any function have global scope.

   Local Scope:

   Variables defined inside a function are considered to have local scope. This means they are only accessible within that specific function and cannot be accessed outside of it. Once the function execution completes, the local variables are destroyed, and their values are not retained. Local variables are useful when you need temporary storage for computations or intermediate results within a function.

   Global Scope:

   Variables defined outside of any function (at the top level of the program) are considered to have global scope. These variables are accessible from any part of the code, including functions. Global variables persist throughout the entire program's execution, and their values can be modified from any function that has access to them.

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

   In Python, you can use the return statement in a function to return multiple values by simply separating them with commas. When a function uses the return statement with multiple values, they are packed into a tuple, and that tuple is returned as a single object. The caller can then unpack the tuple to access the individual values.

```
def get_min_max(numbers):
    # Find the minimum and maximum values from a list of numbers
    minimum = min(numbers)
    maximum = max(numbers)
    return minimum, maximum  # Multiple values returned as a tuple

# Call the function and unpack the returned tuple
my_numbers = [10, 5, 20, 8, 15]
min_value, max_value = get_min_max(my_numbers)

print("Minimum:", min_value)  # Output: Minimum: 5
print("Maximum:", max_value)  # Output: Maximum: 20
```

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

   In "pass by value," a copy of the value of the variable is passed to the function. Any modifications made to the parameter within the function do not affect the original variable outside the function. It essentially means that the function works with a local copy of the value, and changes to that copy don't affect the original value.

   In "pass by reference," a reference to the memory location of the original variable is passed to the function. Any changes made to the parameter within the function directly affect the original variable outside the function. This means that you are working with the same variable inside and outside the function. Python uses "pass by object reference," where references to objects are passed to functions by value. This means that you can modify mutable objects inside the function and affect the original object outside the function.

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 perform_operations(x):
    # a) Logarithmic function (log x)
    log_result = math.log(x)

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

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

```
    # d) Square root
    sqrt_result = math.sqrt(x)

    return log_result, exp_result, power_result, sqrt_result
```

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

```
def extract_first_last_name(full_name):
    # Split the full name into a list of words
    name_parts = full_name.split()

    # Assume the first part is the first name and the last part is the last name
    first_name = name_parts[0]
    last_name = name_parts[-1]

    return first_name, last_name
```