1)

```
'''
Built-in functions are functions that are pre-defined in the Python
language and are always available for use.

'''
# Example of a built-in function: len()
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print("Length of the list:", length)

Length of the list: 5

'''
User-defined functions are functions that are created by the user to
perform
a specific task or a set of tasks.

'''
# Example of a user-defined function: add_numbers()
def add_numbers(a, b):
    return a + b

result = add_numbers(3, 5)
print("Sum of numbers:", result)

Sum of numbers: 8
```

2)

```
# Positional arguments are passed to a function based on their
position or order in the function call.

def add_numbers(a, b):
    return a + b

result = add_numbers(3, 5)
print("Sum of numbers:", result)

Sum of numbers: 8

'''
Keyword arguments are passed with the parameter names, and the order
can be different from the function definition.

'''
```

```
def greet(name, message):
    return f"Hello, {name}! {message}"

greeting = greet(name="Alice", message="How are you?")
print(greeting)
```

```
Hello, Alice! How are you?
```

3)

```
'''
The return statement in a function serves the purpose of ending the
function's
execution and returning a value to the caller. When a function
encounters a return
statement, it immediately exits, and the specified value is sent back
to the point
where the function was called. If no return statement is present or if
it doesn't
specify a value, the function returns None by default.

Yes, a function can have multiple return statements. However, only one
of them will be
executed during the function call. Once a return statement is
executed, the function exits,
and subsequent return statements are not processed.

'''
def classify_number(num):
    if num > 0:
        return "Positive"
    elif num < 0:
        return "Negative"
    else:
        return "Zero"

# Example usage:
result1 = classify_number(5)
result2 = classify_number(-3)
result3 = classify_number(0)

print(result1)  # Output: Positive
print(result2)  # Output: Negative
print(result3)  # Output: Zero
```

```
Positive
Negative
Zero
```

4)

```
'''
In Python, a lambda function is a small anonymous function defined
using the lambda keyword. It is also known as a "lambda expression."
'''

# Regular function
def add(a, b):
    return a + b

result_regular = add(3, 5)
print("Regular function result:", result_regular)

# Lambda function
lambda_add = lambda a, b: a + b
result_lambda = lambda_add(3, 5)
print("Lambda function result:", result_lambda)

'''
In this example, both the add function and the lambda_add lambda
function do the same thing—they add two numbers. The lambda function,
however, is more concise and doesn't require a formal function
definition.
'''
```

```
Regular function result: 8
Lambda function result: 8
```

```
pairs = [(1, 5), (3, 2), (7, 8), (2, 6)]

# Sort based on the second element of each tuple
sorted_pairs = sorted(pairs, key=lambda x: x[1])

print("Sorted pairs:", sorted_pairs)
```

```
Sorted pairs: [(3, 2), (1, 5), (2, 6), (7, 8)]
```

5)

```
'''
In Python, the concept of "scope" refers to the region in a program
where a variable
is accessible or can be modified. The scope of a variable is
determined by where it is
defined, and it can be categorized into two main types: local scope
and global scope.

Local Scope:
```

```
Variables defined inside a function have local scope.
They are only accessible within that function and are not visible to
the rest of the program.
Once the function completes execution, the local variables are
destroyed.
'''
def my_function():
    x = 10  # local variable
    print("Inside the function:", x)

my_function()
# print(x)  # This would result in an error since x is not defined in
this scope
```

```
Inside the function: 10
```

```
'''
Global Scope:

Variables defined outside of any function or block have global scope.
They are accessible throughout the entire program, including within
functions.
If a variable with the same name is defined both globally and locally,
the local
variable takes precedence within its scope.

'''
y = 20  # global variable

def another_function():
    print("Inside the function:", y)

another_function()
print("Outside the function:", y)
```

```
Inside the function: 20
Outside the function: 20
```

6)

```
def add_and_multiply(x, y):
    sum_result = x + y
    product_result = x * y
    return sum_result, product_result

# Example usage:
result_tuple = add_and_multiply(3, 5)
```

```python
# Unpack the tuple into individual variables
sum_result, product_result = result_tuple

print("Sum:", sum_result)
print("Product:", product_result)
```

```
Sum: 8
Product: 15
```

7)

```python
'''
In Python, the terms "pass by value" and "pass by reference" are often
used
to describe how arguments are passed to functions, but the terminology
can
be a bit misleading. Python uses a mechanism that is often called
"pass by
object reference" or "call by object reference." Understanding this
concept
can help clarify the behavior of function arguments in Python.
'''
```

8)

```python
import math

def math_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 = math.pow(2, x)

    # Square root
    sqrt_result = math.sqrt(x)

    return log_result, exp_result, power_result, sqrt_result

# Example usage:
number = 4.0
results = math_operations(number)

print(f"Logarithmic function (log {number}): {results[0]}")
```

```
print(f"Exponential function (exp {number}): {results[1]}")
print(f"Power function with base 2 (2^{number}): {results[2]}")
print(f"Square root of {number}: {results[3]}")
```

```
Logarithmic function (log 4.0): 1.3862943611198906
Exponential function (exp 4.0): 54.598150033144236
Power function with base 2 (2^4.0): 16.0
Square root of 4.0: 2.0
```

9)

```
def name(name):
    split = name.split()
    first_name = split[0]
    last_name = split[1]

    print("First_Name ",first_name, " Last_Name ", last_name)

name("Alisha Asthana")
```

```
First_Name  Alisha  Last_Name  Asthana
```