**Question 1**:

**Built-in Function**: Built-in functions are functions that are already defined within Python's core library, and they are readily available for use without the need for additional definition.

Example: print(), len(), max(), min(), sum()

**User-defined Function**: User-defined functions are functions that are defined by the user to perform a specific task or set of tasks. These functions allow users to encapsulate a set of instructions into a single reusable block of code

Example:

def greet(name): # greet() is a user defined function  
 print(f"Hello, {name}!") # print() is a built-in function  
  
greet("Suzad")

**Question 2**:

Passing arguments can be done in 2 ways – positional arguments and keyword arguments

**Positional argument**: Positional arguments are passed to a function based on their position or order in the function's parameter list.

def greet(name, age):  
 print(f"Hello, {name}! You are {age} years old.")  
  
greet("Alice", 25)

**Keyword argument**: Keyword arguments are passed to a function by specifying the parameter name followed by the argument value.

def greet(name, age):  
 print(f"Hello, {name}! You are {age} years old.")  
  
greet(age=25, name="Alice")

**Question 3**:

The return statement in a function serves the purpose of exiting the function and returning a value (or values) back to the caller. It allows a function to compute a result and pass it back to the code that called the function.

A function can have multiple return statements. When a return statement is encountered, the function immediately exits and returns the specified value. Having multiple return statements allows a function to conditionally return different values based on certain conditions. Even multiple value can be return as an object.

def calculate\_stats(numbers):  
 total = sum(numbers)  
 average = total / len(numbers)  
 maximum = max(numbers)  
 minimum = min(numbers)  
 return {  
 "total": total,  
 "average": average,  
 "maximum": maximum,  
 "minimum": minimum  
 }

def check\_even\_odd(number):  
 if number % 2 == 0:  
 return "Even"  
 else:  
 return "Odd"

def math\_operations(x):  
 logarithm = math.log(x)  
 exponential = math.exp(x)  
 power\_of\_2 = 2 \*\* x  
 square\_root = math.sqrt(x)  
 return logarithm, exponential, power\_of\_2, square\_root

**Question 4**:

Lambda functions, also known as anonymous functions, are small, inline functions defined using the lambda keyword in Python. They are used when you need a simple function for a short period of time, often for operations like filtering, mapping, or sorting data. Lambda functions can take any number of arguments, but they can only have one expression.

Lambda functions are written in a single line and do not require a return statement. They are typically used for simple operations. Lambda functions are often used for short-lived or one-time operations, while regular functions are used for more complex or reusable tasks. Lambda functions are used to define simple operations inline without the need for defining a separate named function. They make the code more concise and readable, especially when the operations are short and straightforward.

**Question 5**:

"scope" refers to the region of a program where a variable can be accessed. There are two main types of scope in Python: local scope and global scope.

**Local Scope**:

Local scope refers to the region within a function where variables are defined. Variables defined within a function are said to have local scope, meaning they can only be accessed within that specific function. Once the function execution is complete, these variables are destroyed, and their memory space is reclaimed. Local variables cannot be accessed from outside the function.

**Global Scope**:

Global scope refers to the region of the program outside of any function where variables are defined. Variables defined outside of all functions have global scope, meaning they can be accessed from anywhere within the program, including inside functions. Global variables persist throughout the entire execution of the program.

**Question 6**:

def calculate\_stats(numbers):  
 total = sum(numbers)  
 average = total / len(numbers)  
 maximum = max(numbers)  
 minimum = min(numbers)  
 return {  
 "total": total,  
 "average": average,  
 "maximum": maximum,  
 "minimum": minimum  
 }

return as an object.

import math  
  
def math\_operations(x):  
 logarithm = math.log(x)  
 exponential = math.exp(x)  
 power\_of\_2 = 2 \*\* x  
 square\_root = math.sqrt(x)  
 return logarithm, exponential, power\_of\_2, square\_root

return multiple value putting comma

**Question 7**:

**Pass by value**: In pass by value, a copy of the value of the actual parameter (the argument) is passed to the function. This means that changes made to the parameter within the function do not affect the original variable outside the function.

def modify\_value(x):  
 x = x + 1  
 print("Inside function:", x)  
  
value = 10  
modify\_value(value)  
print("Outside function:", value) # Output: 10

**Pass by Reference**: In pass by reference, a reference (memory address) to the actual parameter is passed to the function. This means that changes made to the parameter within the function affect the original variable outside the function.

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) # Output: [1, 2, 3, 4]