Ans. 1.

Built-in Functions: Built-in functions are functions that are already defined in the Python language. These functions are readily available for use without requiring any additional code or imports. Python provides a wide range of built-in functions that perform common tasks.

Example, len() is a built-in function that takes a string as an argument and returns the length of the string.

User-defined functions are functions that are created by the Python programmer. These functions are defined and implemented by the user to perform specific tasks based on their requirements. User-defined functions help modularize code, improve reusability, and make the code easier to read and maintain.

def calculate\_area(length, width):

area = length \* width

return area

# Calling the function

rectangle\_area = calculate\_area(5, 3)

print(rectangle\_area) # Output: 15

calculate\_area() is a user-defined function that takes two arguments, length and width, and returns the calculated area of a rectangle. The function is defined by the user and can be reused throughout the code.

Ans. 2.

Positional Arguments: Positional arguments are passed to a function based on their position or order. The order of the arguments matters, and the function receives the values in the same order as they are passed.

def greet(name, age):

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

# Calling the function with positional arguments

greet("Alice", 25)

Keyword Arguments: Keyword arguments are passed to a function based on their names. Each argument is explicitly assigned a value using the argument name and the assignment operator (=). The order of the arguments doesn't matter when using keyword arguments.

def greet(name, age):

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

# Calling the function with keyword arguments

greet(name="Bob", age=30)

Ans. 3.

The return statement in a function serves the purpose of specifying the value that the function should return when it is called. It allows a function to compute a result or perform certain operations and provide that result back to the caller.

A function can have multiple return statements, but only one return statement is executed during the function's execution. Once a return statement is encountered, the function immediately exits, and the specified value (if any) is returned.

Ans. 4.

Lambda functions, also known as anonymous functions, are a way to create small, one-line functions in Python. They are defined using the lambda keyword and are typically used for simple and concise operations.

Here are some key characteristics of lambda functions:

They can take any number of arguments but can only have one expression.

They do not require a return statement, as the expression is automatically returned.

They are defined inline and do not require a separate function name.

# Example: Sorting a list of tuples based on the second element

my\_list = [(1, 5), (3, 1), (2, 3), (4, 2)]

# Sort the list based on the second element of each tuple

my\_list.sort(key=lambda x: x[1])

print(my\_list) # Output: [(3, 1), (4, 2), (2, 3), (1, 5)]

Ans. 5.

Local Scope: Local scope refers to the visibility and accessibility of variables within a specific block of code, typically within a function. Variables defined inside a function are considered local variables and can only be accessed within that function. They are not accessible outside the function or in other functions.

def my\_function():

x = 10 # Local variable

print(x) # Accessible within the function

my\_function()

print(x) # Error: x is not defined outside the function

Global Scope: Global scope refers to the visibility and accessibility of variables at the top level of a module or throughout the entire program. Variables defined outside any function or at the module level are considered global variables and can be accessed from anywhere in the code, including inside functions.

x = 10 # Global variable

def my\_function():

print(x) # Accessible within the function

my\_function()

print(x) # Accessible outside the function

Ans. 6.

In Python, the return statement in a function can be used to return multiple values by returning them as a tuple, list, or any other iterable object.

def get\_user\_info():

name = "Alice"

age = 25

email = "alice@example.com"

return name, age, email

# Calling the function and unpacking the returned values

user\_name, user\_age, user\_email = get\_user\_info()

# Printing the returned values

print(user\_name) # Output: Alice

print(user\_age) # Output: 25

print(user\_email) # Output: [alice@example.com](mailto:alice@example.com)

n this example, the get\_user\_info() function returns multiple values: name, age, and email. The values are separated by commas in the return statement, which automatically creates a tuple containing those values.

When calling the function, we assign the returned values to three variables: user\_name, user\_age, and user\_email. This process is known as unpacking. Each variable will receive the corresponding value from the tuple returned by the function.

Ans. 7.

Pass by Value: In a "pass by value" approach, when a function is called, the values of the arguments are copied into the function's parameters. Any changes made to the parameters within the function do not affect the original values of the arguments outside the function.

Pass by Reference: In a "pass by reference" approach, when a function is called, the references or memory addresses of the arguments are passed into the function's parameters. Any modifications made to the parameters within the function directly affect the original values of the arguments outside the function.