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

Ans

In Python, the main difference between a built-in function and a user-defined function lies in their origin and implementation:

1. Built-in function:

Built-in functions are part of the Python programming language's standard library. They are pre-defined and readily available for use without requiring any additional import or definition. These functions provide a wide range of functionalities and are optimized for performance and efficiency.

Example of a built-in function:

```python

# The 'len' function is a built-in function to get the length of a sequence.

string\_length = len("Hello, world!")

print(string\_length)

# Output: 13

```

2. User-defined function:

User-defined functions, as the name suggests, are functions created by the Python programmer. These functions allow you to encapsulate a block of code with specific logic or computations, making the code modular and easier to manage. User-defined functions are defined by the programmer using the `def` keyword.

Example of a user-defined function:

```python

# This is a user-defined function to calculate the area of a rectangle.

def rectangle\_area(length, width):

area = length \* width

return area

# Calling the function and passing arguments to calculate the area of a rectangle.

length = 5

width = 10

result = rectangle\_area(length, width)

print("Area of the rectangle:", result)

# Output: 50

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

Ans

In Python, you can pass arguments to a function when calling it. Arguments are values that are supplied to the function so that it can perform its operations or computations. When defining a function, you can specify parameters to accept these arguments.

Here's how you can pass arguments to a function in Python:

1. Positional arguments:

Positional arguments are the most basic way of passing arguments to a function. They are matched based on their position in the function call. The order and number of arguments passed must match the order and number of parameters defined in the function definition.

Example of positional arguments:

```python

def greet(name, age):

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

# Calling the 'greet' function with positional arguments.

greet("Alice", 30)

# Output: Hello, Alice. You are 30 years old.

greet("Bob", 25)

# Output: Hello, Bob. You are 25 years old.

```

2. Keyword arguments:

Keyword arguments are passed to a function using the names of the parameters. This way, the order of the arguments doesn't matter as long as they are associated with the correct parameter names.

Example of keyword arguments:

```python

def greet(name, age):

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

# Calling the 'greet' function with keyword arguments.

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

# Output: Hello, Alice. You are 30 years old.

greet(age=25, name="Bob")

# Output: Hello, Bob. You are 25 years old.

```

Difference between positional arguments and keyword arguments:

1. Positional arguments rely on the order of the arguments passed, while keyword arguments rely on the parameter names to match the values.

2. When using positional arguments, the order of the arguments matters. The first argument passed corresponds to the first parameter in the function definition, the second argument to the second parameter, and so on.

3. With keyword arguments, you can explicitly mention the parameter names, making the order of the arguments irrelevant. This provides more clarity and flexibility, especially when dealing with functions that have many parameters.

4. You can use a mix of positional and keyword arguments in a function call, but all positional arguments must be placed before any keyword arguments.

Example of mixing positional and keyword arguments:

```python

def calculate(a, b, c):

result = a \* b + c

return result

# Using a mix of positional and keyword arguments.

result1 = calculate(2, 3, 4) # Positional arguments

result2 = calculate(a=2, b=3, c=4) # Keyword arguments

result3 = calculate(2, c=4, b=3)

# Mix of both

```

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

Ans

The purpose of the `return` statement in a function is to specify what value or values the function should output or return to the caller. When a `return` statement is encountered in a function, it immediately stops the execution of the function and sends the specified value(s) back to the caller.

A function can indeed have multiple `return` statements, and the one that gets executed depends on the flow of the function and any conditional statements present. As soon as a `return` statement is executed, the function exits, and the result is returned to the caller.

Example of a function with multiple return statements:

```python

def divide\_numbers(a, b):

if b == 0:

# Avoid division by zero, return an error message.

return "Error: Division by zero is not allowed."

result = a / b

if result.is\_integer():

# If the result is an integer, return it as an integer.

return int(result)

else:

# Otherwise, return the result as a floating-point number.

return result

# Calling the 'divide\_numbers' function with different inputs.

result1 = divide\_numbers(10, 2)

print(result1) # Output: 5

result2 = divide\_numbers(7, 3)

print(result2) # Output: 2.3333333333333335

result3 = divide\_numbers(8, 0)

print(result3) # Output: Error: Division by zero is not allowed.

```

1. 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 in Python are small, anonymous, and inline functions defined using the `lambda` keyword. They are also known as anonymous functions because they don't have a name like regular functions defined using the `def` keyword. Lambda functions can take any number of arguments, but they can only have one expression, which is evaluated and returned.

Lambda functions are typically used when a small function is needed for a short period, and it is more convenient to define the function inline without assigning it a name. They are often used in situations where a function is needed as an argument to another function, like in higher-order functions (e.g., `map()`, `filter()`, `sorted()`, etc.).

Difference between lambda functions and regular functions:

1. Syntax: Lambda functions have a more concise syntax with the `lambda` keyword, while regular functions are defined using the `def` keyword, along with a function name and block of code.

2. Name: Lambda functions do not have a name, whereas regular functions are explicitly given a name when defined.

3. Number of expressions: Lambda functions can have only one expression, while regular functions can contain multiple statements and expressions.

4. Use cases: Lambda functions are typically used for short, simple operations, whereas regular functions are used for more complex tasks or when reusability is needed.

Example of using a lambda function:

```python

# Using a lambda function to add two numbers.

add = lambda x, y: x + y

result = add(5, 3)

print(result)

# Output: 8

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

Ans

In Python, "scope" refers to the region or context in which a variable or name can be accessed. It defines the area of a program where a variable or function is visible and can be referenced. Understanding scope is crucial for managing variables and avoiding conflicts in larger programs.

Python has two main types of scope: local scope and global scope.

1. Local Scope:

- When a variable is defined inside a function, it is said to have a local scope.

- Variables with local scope are only accessible within the function in which they are defined.

- Once the function execution completes, the local variables are destroyed, and their values are lost.

- Attempting to access a local variable outside of the function that defines it will result in a NameError.

2. Global Scope:

- When a variable is defined at the top level of a script or module (outside any function), it has a global scope.

- Global variables can be accessed from anywhere within the script or module, including inside functions.

- Global variables persist throughout the entire execution of the program.

- If a variable is modified inside a function with a global scope, the changes will affect its value outside the function as well.

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

ans. def multiple\_values():

value1 = 42

value2 = "Hello, world!"

return value1, value2

result1, result2 = multiple\_values()

print(result1) # Output: 42

print(result2) # Output: Hello, world!

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

Ans

Mutable objects (pass by reference-like behavior):

If you pass a mutable object like a list to a function and modify it within the function, the changes will be reflected outside the function as well.

Immutable objects (pass by value-like behavior):

If you pass an immutable object like an integer to a function and modify it within the function, the original value won't change outside the function. Instead, a new object will be created and assigned to the local variable within the function.

1. Create a function that can intake integer or decimal value and do following operations:
   1. Logarithmic function (log x)
   2. Exponential function (exp(x))
   3. Power function with base 2 (2x)
   4. Square root

Ans

import math

def perform\_operations(value):

logarithmic\_result = math.log(value)

exponential\_result = math.exp(value)

power\_result = math.pow(2, value)

square\_root\_result = math.sqrt(value)

results = {

'Logarithmic': logarithmic\_result,

'Exponential': exponential\_result,

'Power (2^x)': power\_result,

'Square Root': square\_root\_result

}

return results

input\_value = float(input("Enter a number: "))

operations\_results = perform\_operations(input\_value)

for operation, result in operations\_results.items():

print(f"{operation} of {input\_value} is {result}")

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

Ans

def extract\_names(full\_name):

names = full\_name.split()

first\_name = names[0]

last\_name = names[-1]

return first\_name, last\_name

full\_name = input("Enter your full name: ")

first\_name, last\_name = extract\_names(full\_name)

print("First Name:", first\_name)

print("Last Name:", last\_name)