Python Basic Assignment – 10th June

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 provided as part of the Python programming language. These functions are available by default and do not require any additional imports or setup. They are pre-defined and can be used directly in your code without any further implementation.

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

length = len(my\_list)

print(length) # Output: 5

* **User-defined Function:** A user-defined function is a function created by the programmer to perform specific tasks based on the requirements of the program. These functions are defined using the def keyword followed by the function name, a set of parameters (if any), and a block of code inside the function's body.

def square(x):

return x \*\* 2

result = square(5)

print(result) # Output: 25

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

In Python, you can pass arguments to a function when you call it. Arguments are the values that you provide to the function, and they can be of any data type, such as integers, strings, lists, or even other functions. You pass these values inside the parentheses () when calling the function.

def add\_numbers(a, b):

return a + b

result = add\_numbers(3, 5)

print(result) # Output: 8

Now, the following will be the difference between positional arguments and keyword arguments:

* **Positional Arguments**: Positional arguments are the most common type of arguments. They are passed to a function in the order in which the function parameters are defined. The first argument corresponds to the first parameter, the second argument to the second parameter, and so on. The order of positional arguments is crucial, as it determines which argument is assigned to which parameter inside the function.

def greet(name, age):

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

greet("Alice", 30)

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

greet(30, "Alice")

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

* **Keyword Arguments:** Keyword arguments are passed with the parameter names explicitly specified, along with their corresponding values. When using keyword arguments, the order of the arguments does not matter. Using keyword arguments makes the code more readable and allows you to skip providing values for optional parameters with default values.

def greet(name, age):

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

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

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

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

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

1. **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 serves the purpose of specifying the value that the function will produce as its output. When a function is called and reaches a return statement, it immediately stops executing, and the value specified after return is returned as the result of the function call. This allows the function to provide a result that can be used in other parts of the program or assigned to a variable.

Yes, a function can have multiple return statements, but only one of them will be executed during the function call. As soon as a return statement is encountered, the function terminates, and the value after that particular return statement is returned.

def divide(a, b):

if b == 0:

return "Cannot divide by zero!"

else:

result = a / b

return result

# Example 1: Successful division

result1 = divide(10, 2)

print(result1) # Output: 5.0

# Example 2: Division by zero

result2 = divide(10, 0)

print(result2) # Output: Cannot divide by zero!

In this example, the divide() function takes two arguments a and b and checks if b is equal to zero. If b is zero, the function immediately returns the message "Cannot divide by zero!". If b is not zero, the function calculates the division result a / b and returns the result.

1. **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 in Python are small, anonymous functions that can have any number of arguments but can only have one expression. They are defined using the lambda keyword, followed by the list of arguments (if any), a colon :, and the expression to be evaluated. The result of the expression is automatically returned.

Lambda functions are also known as "anonymous" functions because they don't have a name like regular functions defined using def. Instead, they are typically used where a small function is needed for a short period and does not require a separate name.

square = lambda x: x \*\* 2

result = square(5)

print(result) # Output: 25

Difference between Lambda Functions and Regular Functions:

* Syntax: Lambda functions are defined using the lambda keyword and don't have a name, while regular functions are defined using the def keyword and have a name.
* Complexity: Lambda functions are meant to be simple and short, typically used for one-line expressions. Regular functions can be more complex and can include multiple statements and logic.
* Return: Lambda functions automatically return the result of the expression without needing an explicit return statement. In contrast, regular functions require a return statement to return a value.
* Scope: Lambda functions are often used in limited scopes and are not recommended for large or complex tasks. Regular functions have broader use and can be used for more extensive and reusable tasks.

numbers = [1, 2, 3, 4, 5]

squared\_numbers = map(lambda x: x \*\* 2, numbers)

print(list(squared\_numbers)) # Output: [1, 4, 9, 16, 25]

Here, the lambda function lambda x: x \*\* 2 calculates the square of each element in the numbers list using map(). The map() function applies the lambda function to each element in the list and returns an iterator, which is converted back to a list using list() to display the result.

1. **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 in which a variable or name is recognized and can be accessed. The scope determines the visibility and lifetime of variables within a program.

* **Local Scope:** Variables defined inside a function have local scope. These variables can only be accessed within the function where they are defined. They are not visible outside of the function. Local scope is temporary and exists only during the execution of the function. When the function finishes executing, the local variables are destroyed, and their values are no longer accessible.
* **Global Scope**: Variables defined outside of any function have global scope. These variables can be accessed from any part of the program, including inside functions. Global variables have a longer lifetime and exist throughout the entire execution of the program. Their values persist even after the function finishes executing.

In summary, the concept of scope in Python determines the visibility and accessibility of variables within functions. Local scope limits the accessibility of variables to the function in which they are defined, while global scope allows variables to be accessed from any part of the program.

1. **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 packing them into a data structure, such as a tuple, list, or dictionary. This allows you to return multiple values as a single object, and the caller of the function can unpack these values as needed.

def get\_user\_info():

name = "Alice"

age = 30

email = "alice@example.com"

return name, age, email

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

print(user\_name) # Output: Alice

print(user\_age) # Output: 30

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

In this example, the get\_user\_info() function returns three values: name, age, and email. These values are packed into a tuple, and the function returns this tuple. When calling the function, we can unpack the returned tuple into separate variables (user\_name, user\_age, and user\_email) to access each value individually.

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

In Python, the concepts of "pass by value" and "pass by reference" refer to how function arguments are passed and how they affect the original variables outside the function.

* **Pass by Value:** In "pass by value," a copy of the value of the variable is passed to the function. Any changes made to the parameter inside the function do not affect the original variable outside the function.

def modify\_number(num):

num += 10

return num

x = 5

result = modify\_number(x)

print(x) # Output: 5

print(result) # Output: 15

* **Pass by Reference:** In "pass by reference," a reference to the original variable is passed to the function. This means that any changes made to the parameter inside the function directly affect the original variable outside the function.

def modify\_list(lst):

lst.append(4)

return lst

my\_list = [1, 2, 3]

result\_list = modify\_list(my\_list)

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

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

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 (2 x )**
     4. **Square root**

import math

def math\_operations(x):

log\_result = math.log(x)

exp\_result = math.exp(x)

power\_result = 2 \*\* x

sqrt\_result = math.sqrt(x)

return log\_result, exp\_result, power\_result, sqrt\_result

number = 4.5

log\_res, exp\_res, power\_res, sqrt\_res = math\_operations(number)

print(f"Logarithmic function (log {number}): {log\_res}")

print(f"Exponential function (exp {number}): {exp\_res}")

print(f"Power function with base 2 (2^{number}): {power\_res}")

print(f"Square root of {number}: {sqrt\_res}")

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

def get\_first\_and\_last\_name(full\_name):

names\_list = full\_name.split()

first\_name = names\_list[0]

last\_name = names\_list[-1]

return first\_name, last\_name

full\_name = "John Doe"

first\_name, last\_name = get\_first\_and\_last\_name(full\_name)

print(f"First Name: {first\_name}")

print(f"Last Name: {last\_name}")