**ASSIGNMENT8**

**TOPIC NAME:FUNCTIONS**

Defining functions in Python

In Python, functions are defined using the def keyword. Here's a simple example of how to define and use functions in Python:

**Syntax:**

def function\_name(parameters):

# Function body

# Your code here

return value # (optional)

**Example 1: A simple function with no parameters and no return value**

def greet():

print("Hello, welcome to Python!")

# Calling the function

greet()

**Output:**

Hello, welcome to Python!

**Example 2: A function with parameters and a return value**

def add(a, b):

return a + b

# Calling the function with arguments

result = add(5, 3)

print("The sum is:", result)

**Output:**

The sum is: 8

**Example 3: A function with default parameters**

def greet(name="Guest"):

print(f"Hello, {name}!")

# Calling the function with and without arguments

greet("Alice") # Custom greeting

greet() # Default greeting

**Output:**

Hello, Alice!

Hello, Guest!

**Example 4: A function with multiple return values (using a tuple)**

def calculate\_area\_and\_perimeter(length, width):

area = length \* width

perimeter = 2 \* (length + width)

return area, perimeter

# Calling the function

area, perimeter = calculate\_area\_and\_perimeter(5, 3)

print("Area:", area)

print("Perimeter:", perimeter)

**Output:**

Area: 15

Perimeter: 16

.

Different types of functions: with/without parameters, with/without return values.

In Python, functions can be categorized based on whether they take parameters or return values. Here’s a breakdown of the different types of functions:

**1. Function without parameters and without return value**

* This type of function does not accept any arguments and does not return any value.

def greet():

print("Hello, welcome to Python!")

# Calling the function

greet()

**Output:**

Hello, welcome to Python!

**2. Function with parameters but without return value**

* This function accepts one or more parameters but does not return any value. It typically performs some operations using the passed parameters.

EXAMPLE:

def print\_message(message):

print("Message:", message)

# Calling the function

print\_message("This is a sample message.")

**Output:**

Message: This is a sample message.

**3. Function without parameters but with return value**

* This function does not accept any parameters but returns a value when called.

def get\_greeting():

return "Hello, world!"

# Calling the function

greeting = get\_greeting()

print(greeting)

**Output:**

Hello, world!

**4. Function with parameters and with return value**

* This function accepts parameters and also returns a value based on those parameters.

def add(a, b):

return a + b

# Calling the function

result = add(5, 3)

print("Sum:", result)

**Output:**

Sum: 8

**ANONYMOUS FUNCTIONS(LAMBDA FUNCTIONS)**

In Python, **anonymous functions**, also known as **lambda functions**, are small, one-line functions that are defined using the lambda keyword. These functions do not have a name, and they are often used for short, simple tasks.

**Syntax of a lambda function:**

lambda arguments: expression

* arguments: The parameters the function takes (can be multiple).
* expression: The expression the function evaluates and returns.

**Example 1: A simple lambda function with one argument**

# Lambda function that squares a number

square = lambda x: x \*\* 2

# Calling the lambda function

result = square(5)

print(result)

**Output:**

25

In this example, the lambda function takes one argument (x) and returns its square (x \*\* 2).

**Example 2: A lambda function with multiple arguments**

# Lambda function that adds two numbers

add = lambda x, y: x + y

# Calling the lambda function

result = add(3, 4)

print(result)

**Output:**

7

This lambda function takes two arguments (x and y) and returns their sum.

**Example 3: Using lambda functions with map(), filter(), and sorted()**

* **Using map()** to apply a lambda function to each item of an iterable (like a list):

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

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

print(squared\_numbers)

**Output:**

[1, 4, 9, 16, 25]

* **Using filter()** to filter out elements based on a condition:

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

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers)

**Output:**

[2, 4]

* **Using sorted()** with a lambda function as a key for custom sorting:

# Sorting a list of tuples by the second element

pairs = [(1, 3), (2, 2), (3, 1)]

sorted\_pairs = sorted(pairs, key=lambda x: x[1])

print(sorted\_pairs)

**Output:**

[(3, 1), (2, 2), (1, 3)]

**Characteristics of lambda functions:**

* They can have any number of arguments but only one expression.
* They are often used for short, simple operations that can be defined in one line.
* They are commonly used with functions like map(), filter(), and sorted() for concise and efficient code.