❖ 4.1 What is a Function in Python?

A function in Python is a **block of reusable code** that performs a specific task.

- Functions help **reduce code duplication** and increase **modularity**.
- Functions **accept inputs (parameters), process them, and return output**.
- Python provides **built-in functions** like print(), len(), and also allows **user-defined functions**.
- Functions in Python are defined using the def keyword.
- **Example:**

```
def greet():
    print("Hello, welcome to Python!")
greet()  # Calling the function
```

- **Output:**

```
Hello, welcome to Python!
```

◆ What is the Use of Python Built-in Functions?

Python **built-in functions** are **predefined functions** available for common tasks.

- They simplify programming by handling **calculations, conversions, and data operations**.
- They help **save time** as they eliminate the need to write complex logic from scratch.
- Examples include print(), len(), type(), sum(), and max().
- Built-in functions **do not require imports**, making them **efficient and easy to use**.
- **Example:**

```
numbers = [10, 20, 30, 40]
print(len(numbers))  # Output: 4
```

◆ Enlist and Explain 5 Type/Data Conversion and Math Functions

(a) Type/Data Conversion Functions in Python **(ask 2 function)**

1. **int(x):** Converts x to an integer. Example: int (4.7) → 4.
2. **float(x):** Converts x to a floating-point number. Example: float (10) → 10.0.
3. **str(x):** Converts x to a string. Example: str (25) → '25'.
4. **list(x):** Converts x to a list. Example: list("abc") → ['a', 'b', 'c'].
5. **tuple(x):** Converts x to a tuple. Example: tuple ([1,2,3]) → (1,2,3).

Example:

```
num = "100"
converted_num = int(num)  # Converts string to integer
print(converted_num)  # Output: 100
```

(b) Math Functions in Python **(ask 2 function and lambda function)**

1. **abs(x):** Returns the absolute value of x. Example: abs(-5) → 5.
2. **pow(x, y):** Returns x raised to the power of y. Example: pow(2, 3) → 8.
3. **round(x, n):** Rounds x to n decimal places. Example: round(3.14159, 2) → 3.14.
4. **max(x, y, ...):** Returns the largest value from the given numbers. Example: max(10, 20) → 20.
5. **min(x, y, ...):** Returns the smallest value from the given numbers. Example: min(5, 15) → 5.
6. **Lambda Function**: A **lambda function** is an **anonymous function** that is written in a single line using the lambda keyword. It is mainly used for short, simple operations.
    - **Example**: lambda x: x * 2 defines a function that multiplies a number by 2.
    - **Example Usage**:

```
double = lambda x: x * 2
print(double(5))  # Output: 10
```

Marathi Tip 💡

- *Python चे अंतर्निहित फंक्शन्स तुमचा वेळ वाचवतात आणि प्रोग्रॅमिंग सोपे करतात. लॅम्ब्डा फंक्शनचा वापर करून तुम्ही एकच ओळीत लॉजिक लिहू शकता!* 🚀

❖4.2 Python Functions – User Defined Functions

1) What is Meant by User-Defined Function?

A **user-defined function (UDF)** is a function that is created by the programmer to perform a specific task.

- It helps in **code reusability** by reducing redundancy.
- Functions make code **modular and easier to debug**.
- It is defined using the def keyword.
- A user-defined function can **accept parameters, process data, and return output**.
- **Example:**

```
def greet():
    print("Hello, Welcome to Python!")
greet()  # Calling the function
```

- **Output:**

```
Hello, Welcome to Python!
```

---

2) What is Function Definition?

A **function definition** is the process of creating a function in Python.

- It starts with the def keyword followed by the function name.
- Parentheses ( ) may contain parameters.
- A colon : is used after the function name.
- The function **body contains statements** that define what the function does.
- **Example:**

```
def add(a, b):  # Function definition
    return a + b  # Function body
```

---

3) What is Function Calling?

A **function call** is used to **execute** a function.

- The function is called by using its name followed by parentheses ( ).
- If the function has parameters, **values (arguments) must be passed** during the call.
- The return value (if any) can be stored in a variable.
- **Example:**

```
def square(n):
    return n * n


result = square(4)  # Function calling
print(result)  # Output: 16
```

---

4) What is Function Argument and Parameter Passing?

➤ **Parameters**

- Parameters are **variables defined in the function definition**.
- They act as placeholders for values that will be passed during function calls.

➤ **Arguments**

- Arguments are the **actual values** passed to the function when calling it.
- Arguments replace parameters during execution.

Example:

```
def greet(name):  # 'name' is a parameter
    print("Hello, " + name)


greet("Rahul")  # "Rahul" is an argument
```

Output:

```
Hello, Rahul
```

---

5) What is Return Statement?

The **return statement** is used to send a value back from a function.

- It **terminates the function** and returns a result.
- If no return is used, the function returns None.
- The returned value can be stored in a variable.

Example:

```
def multiply(x, y):
    return x * y


result = multiply(5, 3)
print(result)  # Output: 15
```

Q. Explain How to Use User-Defined Functions in Python with an Example (Most IMP)

➤ **Introduction**

In Python, **user-defined functions (UDFs)** are created using the def keyword. A function can take **parameters** (inputs), perform operations, and **return** a result.

Steps to Create and Use a User-Defined Function

1 **Use the def keyword** to define a function.

2 **Provide a function name** followed by parentheses ().

3 **Write function body** (indented statements inside the function).

4 **Call the function** by using its name followed by parentheses.

Syntax of a Function in Python

```
def function_name():
    # Function body
    statements
```

Example 1: Simple User-Defined Function

```
def greet():
    print("Hello! This is a User-Defined Function.")
greet()  # Function call
```

- **Output:**

```
Hello! This is a User-Defined Function.
```

Example 2: Function with Parameters

```
def square(x):
    print("Square =", x * x)
square(2)  # Calling function with an argument
```

- **Output:**

```
Square = 4
```

Example 3: Function with Multiple Parameters

```
def add(a, b):
    return a + b
result = add(5, 3)
print("Addition =", result)
```

- **Output:**

```
Addition = 8
```

❖Scope of Variables in Python (Most IMP)

- In Python, the **scope of a variable** defines the **visibility and accessibility** of a variable in different parts of a program. Variables can either be **local** (accessible only within a function) or **global** (accessible throughout the program).
- There are 2 types in scope of variable:

  1) Global Variable.

  2) Local Variable.

1. Global Variable

🔹 **Definition:**

A **global variable** is a variable that is declared outside any function and can be accessed from **anywhere in the program** (inside or outside functions).

🔹 **Example:**

```
x = 10  # Global variable
def show():
    print("Value of x inside function:", x)
show()
print("Value of x outside function:", x)
```

🔹 **Output:**

```
Value of x inside function: 10
Value of x outside function: 10
```

📌 *Here, x is a global variable and is accessible both inside and outside the function.*

2. Local Variable

- **Definition:**

  A **local variable** is a variable declared **inside a function**, and it can only be accessed **within that function**. It **cannot be used outside** the function.

- **Example:**

```
def show():
    y = 20  # Local variable
    print("Value of y inside function:", y)
show()
print("Value of y outside function:", y)  # This will give an error!
```

- **Output:**

```
Value of y inside function: 20
NameError: name 'y' is not defined
```

📌 *Here, y is a local variable and cannot be accessed outside the function.*

◆ **Key Differences Between Global and Local Variables**

Feature	Global Variable	Local Variable
Scope	Entire program	Only inside the function
Declared	Outside any function	Inside a function
Accessible	Both inside and outside functions	Only inside the function
Example	x = 10 (outside function)	y = 20 (inside function)

Marathi Tip 💡

- global वेरिएबल संपूर्ण प्रोग्राममध्ये वापरता येतो,
- पण local वेरिएबल फक्त त्या फंक्शनमध्येच वापरता येतो.
- त्यामुळे वेरिएबल योग्य ठिकाणी डिक्लेअर करणे महत्त्वाचे आहे!

❖ 4.3 Modules (IMP)

Explain Module

A **module** in Python is a file containing Python definitions, functions, variables, and classes. It is used to organize code into separate files for better readability and reusability.

- **Definition**: A module is a collection of Python code that can be reused in other Python programs.
- **Importing**: Modules can be imported using the import keyword.
- **Namespace**: Every module creates its own namespace to avoid conflicts between different modules.
- **Modularity**: Modules help in dividing large programs into smaller, manageable, and logically organized parts.

Uses of a Module

1. **Code Reusability**: Once a module is created, it can be reused in different programs without rewriting the code.
2. **Organization:** Modules help in organizing code logically into separate files, which makes the program easier to maintain.
3. **Namespace Management:** Using modules allows for better organization of global variables and functions within a specific namespace.
4. **Avoids Redundancy**: Instead of writing the same functions or classes again, you can import and reuse them from a module.
5. **Ease of Debugging**: When code is modularized into separate files, it is easier to identify and fix issues in specific sections.
6. **Improves Readability**: By splitting the code into smaller modules, the overall readability of the program improves.

How to Define a Module

- A module is simply a .py file that contains Python code (functions, classes, variables, etc.).
- To define a module, follow these steps:
    1. Create a new .py file (e.g., mymodule.py).
    2. Define functions, classes, or variables inside the file.
    3. Save the file.

- Example:

```
# mymodule.py

def greet(name):

    return f"Hello, {name}!"
```

◆Python Program Using Module

Below is an example of creating and using a Python module:

Step 1: Create the module greetings.py

```
# greetings.py

def greet(name):

    return f"Hello, {name}!"
```

Step 2: Create the main program and import the module

```
# main_program.py

import greetings  # Import the greetings module


name = input("Enter your name: ")

message = greetings.greet(name)  # Use the greet function from the greetings module

print(message)
```

Explanation:

- We first create a module greetings.py that has a function greet() to return a greeting message.
- In the main_program.py file, we import the greetings module using the import keyword and then call the greet() function from the module.

◆User-Defined Module to Display Program Name

Create a user-defined module program_name.py that asks the user for their program name and displays it.

Step 1: Define the module program_name.py

```
# program_name.py

def get_program_name():

    name = input("Enter your program name: ")

    return f"The program name is: {name}"
```

Step 2: Main program that uses this module

```
# main_program.py

import program_name  # Import the user-defined module

message = program_name.get_program_name()  # Call the function from the module

print(message)
```

Explanation:

- In the module program_name.py, the function get_program_name() asks the user for the program name and returns the formatted message.
- In the main program main_program.py, we import the program_name module and use the get_program_name() function to display the entered program name.

मराठी टिप:

- **पायथन मॉड्यूल्स** वापरणे आपल्या कोडला अधिक व्यवस्थित, पुनर्वापरता आणि सुसंगत बनवते.
- जेव्हा तुम्ही मोठ्या प्रोजेक्ट्सवर काम करत असता, तेव्हा मॉड्यूल्सचा वापर करून तुमच्या कोडला वेगवेगळ्या फायलीत विभागून ठेवणे सोपे आणि अधिक व्यवस्थापित होईल.
- यामुळे कोड डिबग करणे आणि त्याचे देखभाल करणे खूप सोपे होईल.
- तुम्ही चांगले मॉड्यूल तयार केल्यास, दुसऱ्या लोकांनाही ते वापरण्याची सुविधा मिळते, जे तुमच्या कामाचे मूल्य वाढवते.

---

1) What is Namespace?

- A **namespace** in Python refers to a container where names (such as variables, functions, classes, etc.) are mapped to objects. It helps in organizing the code and avoiding conflicts by ensuring that different parts of the program can use the same names without interfering with each other.
- Each namespace exists in a specific scope and can be local, global, or built-in, determining how Python looks for variables and functions.

2) What is Scoping?

- **Scoping** refers to the region of the program where a particular variable or function is accessible. The scope determines the visibility of a variable within different parts of the code.
- There are mainly **four types of scopes** in Python: Local, Enclosing, Global, and Built-in (often abbreviated as LEGB rule).

Uses of Namespace and Scoping

1. **Avoiding Name Conflicts**: Namespaces ensure that variables and functions in different parts of the program do not clash, even if they have the same name.
2. **Efficient Memory Management**: By creating isolated namespaces for variables, Python ensures that variables from different scopes do not unnecessarily consume memory.
3. **Modular Programming**: Helps in organizing code by defining variables and functions that are specific to certain sections of the code or modules.
4. **Improving Code Readability**: By maintaining separate namespaces for different parts of the program, code becomes easier to understand and maintain.

Example of Namespace and Scoping

```
# Global Namespace

x = 10  # Global variable

def func1():

    # Local Namespace inside func1

    x = 20  # Local variable in func1

    print("Inside func1, x =", x)  # Prints the local x

def func2():

    print("Inside func2, x =", x)  # Prints the global x

func1()  # Calls func1, prints local x

func2()  # Calls func2, prints global x
```

❖ 4.4 Python Packages

1) What is meant by Packages in Python?

1. A package in Python is a **collection of modules** grouped in a directory with an __init__.py file.
2. It helps in **organizing large codebases** by structuring related modules together.
3. Packages **promote code reusability** by allowing users to import only required modules.
4. It follows a hierarchy: **Package → Modules → Functions/Classes**.

---

2) Writing Python Packages

1. Create a folder with the **package name**.
2. Inside the folder, create an __init__.py file (**this makes it a package**).
3. Add multiple .py files (**modules**) inside the package folder.
4. Import and use the modules in a Python script using import package.module.

📌 **Example:**

 # Step 1: Create a package directory "mypackage"

 # Step 2: Inside it, create a file `__init__.py` (empty or with initialization code)

 # Step 3: Create a module inside the package: file "greet.py"

```
# greet.py
def hello():
    print("Hello from mypackage!")
```

 # Step 4: Using the package

```
import mypackage.greet as gt
gt.hello()
```

Output:

```
Hello from mypackage!
```

---

3) Using Standard Packages in Python (i, ii only read kara iii, iv, v most imp)

(i) Math Package

- **Definition:** The math module provides mathematical functions like trigonometry, logarithms, and factorials.
- **Uses:**
  - Performing advanced mathematical calculations.
  - Finding square roots, power, and logarithms.
  - Working with trigonometric functions like sin(), cos(), etc.
  - Calculating factorial, GCD, etc.

📌 **Example:**

```
import math

print(math.sqrt(25))  # Output: 5.0

print(math.factorial(5))  # Output: 120

print(math.sin(math.radians(30)))  # Output: 0.5
```

(ii) SciPy Package

- **Definition:** SciPy is used for scientific computing, including optimization, integration, and signal processing.
- **Uses:**
  - Solving linear algebra problems.
  - Integration and differentiation of functions.
  - Performing statistical analysis.
  - Signal/image processing.

📌 **Example:**

```
from scipy import stats

data = [1, 2, 3, 4, 5, 5, 6, 8, 9]

print(stats.mode(data))  # Output: ModeResult(mode=array([5]), count=array([2]))
```

(iii) NumPy Package

- **Definition:** NumPy provides support for multi-dimensional arrays and numerical computations.
- **Uses:**
  - Working with large numerical datasets efficiently.
  - Performing matrix operations.
  - Generating random numbers.
  - Mathematical operations on arrays.

📌 **Example:**

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr * 2)  # Output: [ 2  4  6  8 10]
```

(iv) Matplotlib Package

- **Definition:** Matplotlib is used for data visualization like plots, graphs, and charts.
- **Uses:**
  - Creating line graphs, bar charts, and histograms.
  - Visualizing data trends over time.
  - Customizing plot appearance.
  - Generating multiple subplots.

📌 **Example:**

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y, marker="o")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Graph")
plt.show()
```

(v) Pandas Package

- **Definition:** Pandas is used for data manipulation and analysis.
- **Uses:**
  - Handling large datasets efficiently.
  - Performing data cleaning and transformation.
  - Reading and writing data from files (CSV, Excel).
  - Performing operations like sorting, filtering, and grouping.

📌 **Example:**

```
import pandas as pd
data = {"Name": ["Alice", "Bob", "Charlie"], "Age": [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

Output:

```
      Name  Age
0  Alice   25
1    Bob   30
2 Charlie   35
```

Q . What is a User-Defined Package? and Write a program illustrating use of user defined package in python. (Most IMP)

1. A **user-defined package** is a collection of custom modules created by the user.
2. It helps in **modularizing code**, making it more reusable.
3. It follows the same structure as built-in packages.
4. **To use a user-defined package,** simply import it like built-in ones.

Program: Creating and Using a User-Defined Package

📌 **Step 1: Create a package structure**

```
mypackage/
|— __init__.py
|— greetings.py
|— math_ops.py
```

📌 **Step 2: Write module files**

greetings.py

```
def say_hello():
    return "Hello, Welcome to Python Packages!"
```

math_ops.py

```
def add(a, b):
    return a + b
```

📌 **Step 3: Use the package**

```
import mypackage.greetings as greet
import mypackage.math_ops as mops

print(greet.say_hello())  # Output: Hello, Welcome to Python Packages!
print(mops.add(5, 3))  # Output: 8
```

Marathi Tip:

- **पॅकेजेस वापरल्याने कोडचे पुनर्वापर (Reusability) व व्यवस्थापन (Modularity) सोपे होते.**
- **Python मधील Math, NumPy, SciPy, Pandas यासारखी पॅकेजेस डेटा सायन्स आणि संगणकीय गणनांमध्ये महत्त्वाची भूमिका बजावतात.**
- **स्वतःचे पॅकेज तयार करून मोठ्या प्रमाणावर कोडिंग करणे सोपे होते.**

❖ Summer 2022

1. Explain Local and Global variable. **(2marks)**
2. Explain how to use user defined function in python with example. **(4marks)**
3. Write a program for importing module for addition and subtraction of two numbers. **(4marks)**
4. Write a program illustrating use of user defined package in python**(4marks)**
5. Explain package Numpy with example. **(6marks)**

❖ Winter 2022

1. Write use of lambda function in python. **(2marks)**
2. What is local and global variables? Explain with appropriate example. **(4marks)**
3. What is command line argument? Write python code to add b) two numbers given as input from command line arguments and print its sum. **(4marks)**
4. Example module. How to define module. **(6marks)**

❖ Summer 2023

1. Describe any two data conversion function. **(2marks)**
2. Describe mkdir( ) function. **(2marks)**
3. Explain any four Python's Built-in Function with example. **(4marks)**
4. Explain Module and its use in Python. **(4marks)**
5. Write a Python program to calculate sum of digit of given number using function. **(6marks)**

❖ Winter 2023

1. State use of namespace in python **(2marks)**

2. Write python program using module, show how to write and use module by importing it. **(4marks)**

3. Explain Numpy package in detail. **(4marks)**

4. Write a program illustrating use of user defined package in python. **(6marks)**

❖ Summer 2024

1. Write use of matplotlib package in python. **(2marks)**

2. Describe following Standard Packages
   i) Numpy
   ii) Pandas. **(4marks)**

3. Write a python program to create a user defined module that will ask your program name and display the name of the program. **(4marks)**

4. Explain following functions with example:
   i) The open( ) function
   ii) The write( ) function. **(4marks)**

❖ Winter 2024

1. Write a program illustrating use of user defined package in Python. **(4marks)**

2. Explain how to use user defined function in Python with example . **(4marks)**

3. How to write , import and alias modules. **(6marks)**