# Practical No: 1

## Aim: Getting Started with Python and IDLE in Interactive and Batch Modes

## Introduction

Python is a high-level programming language known for its simplicity and readability, making it an excellent choice for beginners. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. IDLE is the default IDE (Integrated Development Environment) for Python, which comes bundled with the Python installation. It provides a user-friendly interface for writing and executing Python code.

Python can be executed in two primary modes:

1. **Interactive Mode**: This mode allows users to run Python commands one at a time and see the results immediately.
2. **Batch Mode**: In this mode, users can write Python code in a script and execute the script as a whole.

## Tools Required

 Python Interpreter (Python 3.x version recommended).  IDLE IDE (comes with Python installation).

**Procedure**

## Using Python in Interactive Mode:

Interactive mode allows for immediate feedback. It is helpful for testing small pieces of code or debugging.

#### Steps:

1. **Open IDLE**: Launch the IDLE application. You will see a Python shell, which provides an interactive environment.
2. **Enter Python Code**: Type Python commands directly in the shell. For example:

>>> print("Hello, World!") Hello, World!

1. **Execute Code**: Press the **Enter** key to run the command. The output will be displayed immediately after the code is executed.

**Examples**:

**Simple Arithmetic**:

>>> 2 + 3

5

## Using Python in Batch Mode:

In batch mode, Python code is written in a script (a .py file) and executed as a whole.

#### Steps:

1. **Create a Python Script**: Open IDLE and create a new file by selecting **File** → **New File**.
2. **Write Python Code**: Write your Python code in the file. For example:

# This is a simple script to print a message print("Welcome to Python Programming!")

1. **Save the File**: Save the file with a .py extension (e.g., hello.py).
2. **Run the Script**: To run the script, press **F5** or go to **Run** → **Run Module** in IDLE.

#### Example of Batch Mode Script:

# This is a script to perform basic arithmetic

a = 10

b = 20

sum = a + b

print("The sum of", a, "and", b, "is:", sum)

After saving the script, you can run it, and the output will be displayed in the IDLE shell:

The sum of 10 and 20 is: 30

## Switching Between Interactive and Batch Mode:

In **Interactive Mode**, you are directly executing code line by line, which is ideal for debugging or testing small snippets of code.  In **Batch Mode**, the entire script is executed at once, which is useful for running larger, more complex programs.

# Practical No: 2

## Aim: Understanding String Methods in Python

## Introduction

In Python, strings are sequences of characters enclosed in single or double quotes. Python provides several built-in methods to work with strings. These methods allow for case conversion, counting occurrences of substrings, and replacing portions of a string. In this practical, we will explore the following string methods:

1. **lower()**: Converts all characters in the string to lowercase.
2. **count()**: Counts the occurrences of a substring in a string.
3. **replace()**: Replaces occurrences of a substring with another substring.

## Procedure

### Using the lower() Method

The lower() method is used to convert all characters in the string to lowercase.

#### Example:

text = "HELLO World" result = text.lower() print(result)

**Explanation**: The original string HELLO World will be converted to lowercase, resulting in hello world.

**Expected Output**:

hello world

### Using the count() Method

The count() method counts the number of times a specified substring appears in the string.

#### Example:

text = "apple banana apple grape" result = text.count("apple") print(result)

**Explanation**: The substring "apple" appears **2 times** in the string apple banana apple grape.

**Expected Output**:

2

### Using the replace() Method

The replace() method is used to replace a specified substring with another substring.

#### Example:

text = "apple banana apple grape"

result = text.replace("apple", "orange") print(result)

**Explanation**: All occurrences of the substring "apple" will be replaced with "orange", resulting in orange banana orange grape.

**Expected Output**:

orange banana orange grape

# Practical No: 3

## Aim: Program to Determine if a Number is Prime

## Introduction

A **prime number** is a number greater than 1 that has no divisors other than 1 and itself. In other words, it is only divisible by 1 and itself. For example, 2, 3, 5, 7, and 11 are prime numbers. On the other hand, numbers like 4, 6, 8, and 9 are not prime, as they have divisors other than 1 and themselves.

The goal of this practical is to write a program that checks if the input number is prime or not and displays the appropriate message.

## Procedure

**Step 1**: Accept user input (the favorite number).

**Step 2**: Check if the number is prime. A prime number has the following properties:

It must be greater than 1.

It must have no divisors other than 1 and itself.

**Step 3**: Display the result ("prime" or "not prime").

## Code Implementation

# Function to check if a number is prime def is\_prime(number):

if number <= 1: # Numbers less than or equal to 1 are not prime return False

for i in range(2, int(number \*\* 0.5) + 1): # Check for factors from 2 to sqrt(number) if number % i == 0:

return False # If divisible by any number, it's not prime return True # If no factors found, it's prime

# Main program

favorite\_number = int(input("What is your favorite number? ")) # Take user input

if is\_prime(favorite\_number): print(f"{favorite\_number} is prime")

else:

print(f"{favorite\_number} is not prime")

**Explanation of the Code**:

#### is\_prime() Function:

 This function takes a number as input and checks if it is prime.

 If the number is less than or equal to 1, it returns False (since numbers less than or equal to 1 are not prime).

 For numbers greater than 1, it checks divisibility from 2 to the square root of the number (rounded up). If any number divides evenly into the given number, it returns False, meaning the number is not prime.

 If no divisor is found, the function returns True, indicating that the number is prime.

#### User Input:

 The program asks the user for their favorite number using input().  The input is converted to an integer using int().

#### Prime Check:

 The is\_prime() function is called with the user's input, and based on the return value, it prints whether the number is prime or not.

## Example Outputs:

1. **For the input 24**:

What is your favorite number? 24

24 is not prime

1. **For the input 31**:

What is your favorite number? 31

31 is prime

# Practical No: 4

# Aim: Find All Numbers which are Multiples of 17, but Not the Multiples of 5, Between 2000 and 2500

## Introduction

This practical involves identifying numbers that satisfy two conditions:

1. The number is a multiple of 17.
2. The number is **not** a multiple of 5.

To solve this, we will use a loop to iterate through the numbers between 2000 and 2500 and check for these conditions. If both conditions are satisfied, the number will be printed.

## Code Implementation

**# Program to find numbers that are multiples of 17 but not multiples of 5 between 2000 and 2500**

**for num in range(2000, 2501):  # Loop through numbers from 2000 to 2500 inclusive**

**if num % 17 == 0 and num % 5 != 0:  # Check if num is a multiple of 17 and not a multiple of 5**

**print(num, end=" ")**

**Explanation of the Code**:

1. **range(2000, 2501)**: This generates a range of numbers starting from 2000 to 2500 (inclusive of 2500).
2. **Condition num % 17 == 0**: This checks if the number is divisible by 17 (i.e., it is a multiple of 17).
3. **Condition num % 5 != 0**: This ensures that the number is **not** divisible by 5 (i.e., it is not a multiple of 5).
4. **print(num)**: If both conditions are satisfied, the number is printed.

## Output:

2017 2051 2085 2153 2187 2221 2255 2323 2357 2391 2425 2459 2493

# Practical No: 5

# Aim: Swap Two Integer Numbers Using a Temporary Variable and Tuple Assignment

## Introduction

Swapping two numbers means exchanging the values of the two numbers. There are various ways to do this in Python:

1. **Using a temporary variable**: A third variable is used to hold the value of one number while the second number is assigned to the first.
2. **Using tuple assignment**: Python allows swapping values using a simple tuple unpacking method, which does not require a temporary variable.

## Code Implementation

**Method 1: Using a Temporary Variable**

# Method 1: Using a temporary variable to swap numbers

a = 5

b = 10

# Swapping using a temporary variable temp = a

a = b

b = temp

print("After swapping (using temp):") print("a =", a)

print("b =", b)

**Method 2: Using Tuple Assignment**

# Method 2: Using tuple assignment to swap numbers

a = 5

b = 10

# Swapping using tuple assignment a, b = b, a

print("After swapping (using tuple assignment):") print("a =", a)

print("b =", b)

## Explanation of the Code:

#### Method 1 (Using a Temporary Variable):

 The value of a is stored in a temporary variable temp.

 Then the value of b is assigned to a, and the value of temp (which is the original value of a) is assigned to b.

#### Method 2 (Using Tuple Assignment):

In Python, the tuple unpacking method allows us to swap the values of a and b directly using the statement a, b = b, a. This approach is more concise and eliminates the need for an additional temporary variable.

**Example Output**:

After swapping (using temp):

a = 10

b = 5

After swapping (using tuple assignment):

a = 10

b = 5

# Practical No: 6

## Aim: Python and IDLE: Interactive, Batch Modes, and Finding Largest of N Numbers.

## Introduction

In this practical, we will:

1. Define a function largest() that takes a list of numbers as input and returns the largest number.
2. Take n numbers as input from the user.
3. Pass the numbers to the largest() function and print the largest number.

## Code Implementation

# Function to find the largest number def largest(numbers):

return max(numbers)

# Main program

n = int(input("How many numbers do you want to enter? ")) # Ask the user for the number of inputs numbers = [] # List to store the numbers

# Loop to take n numbers as input for i in range(n):

num = float(input(f"Enter number {i+1}: ")) # Take user input numbers.append(num)

# Find the largest number using the largest() function largest\_number = largest(numbers)

# Print the largest number

print(f"The largest number is: {largest\_number}")

**Explanation of the Code**:

#### largest() function:

 This function takes a list of numbers and returns the largest number using Python’s built-in max() function.

#### Taking User Input:

 First, the user is asked how many numbers they want to enter.

 Then, a for loop is used to collect n numbers from the user. These numbers are stored in the list numbers.

#### Finding the Largest Number:

 The list of numbers is passed to the largest() function, and the largest number is found.

#### Output:

Finally, the largest number is printed to the screen.

**Example Output**:

How many numbers do you want to enter? 5 Enter number 1: 23

Enter number 2: 56

Enter number 3: 12

Enter number 4: 45

Enter number 5: 89

The largest number is: 89.0

# Practical No: 7

## Aim: Write a Function myReverse() to Reverse a String

## Introduction

Reversing a string involves changing the order of the characters in the string so that the last character becomes the first, the second-to-last becomes the second, and so on. This practical will demonstrate how to implement a custom function to reverse a string in Python.

## Code Implementation

# Function to reverse a string def myReverse(input\_string):

return input\_string[::-1] # Reverse the string using slicing

# Main program

string = input("Enter a string: ") # Take user input for the string reversed\_string = myReverse(string) # Call the myReverse function

# Print the reversed string print("Reversed string:", reversed\_string)

**Explanation of the Code**:

#### myReverse() Function:

 The function myReverse() takes an input string and reverses it using Python’s slicing technique [::-1], which means:  Start from the end of the string (by using a negative step) and work back to the beginning.

 This is a simple and efficient way to reverse a string in Python.

#### Taking User Input:

 The user is asked to input a string using input(). This input string is passed to the myReverse() function.

#### Output:

The function returns the reversed string, which is then printed.

**Example Output**:

Enter a string: Python Reversed string: nohtyP

# Practical No: 8

## Aim: Check if a Given String is a Palindrome

## Introduction

A **palindrome** is a word, phrase, number, or other sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization). In this practical, we will check if a string is a palindrome by comparing the string with its reverse.

## Code Implementation

# Function to check if a string is palindrome def is\_palindrome(input\_string):

# Remove spaces and convert to lowercase for comparison cleaned\_string = input\_string.replace(" ", "").lower()

return cleaned\_string == cleaned\_string[::-1] # Compare string with its reverse

# Main program

string = input("Enter a string: ") # Take user input for the string

# Check if the string is a palindrome if is\_palindrome(string):

print("The string is a palindrome.") else:

print("The string is not a palindrome.")

**Explanation of the Code**:

#### is\_palindrome() Function:

This function takes a string and removes spaces and converts it to lowercase to ensure the comparison is case-insensitive and ignores spaces.

It then compares the cleaned string with its reverse (using slicing [::-1]).

 If the string is the same as its reverse, it is a palindrome.

#### Taking User Input:

 The user is prompted to enter a string.

#### Checking Palindrome:

 The string is passed to the is\_palindrome() function, which returns True if the string is a palindrome and False

otherwise.

#### Output:

 Based on the return value from is\_palindrome(), the program prints whether the string is a palindrome or not.

## Example Output:

1. **For the input madam**:

Enter a string: madam

The string is a palindrome.

1. **For the input hello**:

Enter a string: hello

The string is not a palindrome.

Enter temperature in Celsius: 25

25.0 Celsius is equal to 77.0 Fahrenheit.