Govt. Polytechnic Hamirpur

Information Technology Department

LAB MANUAL

for

Programming in Python

ITPC206 **Programming in Python Lab**

L T P
- - 4

Internal Assessment Marks : 40 Marks
External Assessment Marks: 60 Marks
Total : 100 Marks

After completing this course the students will be able to practically demonstrate the following experiments related to python programming.

S.No	**Experiment**
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

Practical 1: Install and Configure Python 3 and IDLE on Windows/Linux Platforms

Objective:

To install and configure Python 3 and IDLE on Windows/Linux platforms.

Procedure:

1. **For Windows**:
   - Download the latest version of Python from the official website: https://www.python.org/downloads/.
   - Run the installer and check the box "Add Python to PATH" before clicking "Install Now".
   - After installation, verify by opening the Command Prompt and typing `python --version`.
   - To launch IDLE, search for IDLE from the start menu and open it.
2. **For Linux**:
   - Open the terminal.
   - Use the command `sudo apt-get install python3` for Ubuntu or `sudo yum install python3` for Fedora.
   - Verify the installation with `python3 --version`.
   - IDLE can be installed by running `sudo apt-get install idle3` or `sudo yum install idle3`.

Practical 2: Practice Various Arithmetic Operators on Python Interactive Shell

Objective:

To practice using arithmetic operators in the Python interactive shell.

Procedure:

1. Open the Python shell by typing `python` or `python3` in the terminal.
2. Try using different arithmetic operators:

Code:

```
a = 10
b = 3
print(a + b)   # Addition
print(a - b)   # Subtraction
print(a * b)   # Multiplication
print(a / b)   # Division
print(a // b)  # Floor division
print(a % b)   # Modulus
print(a ** b)  # Exponentiation
```

Output:

```
13
7
30
3.3333333333333335
3
1
1000
```

Practical 3: Write a Program to Calculate the Area and Circumference of a Circle

Objective:

To write a program that calculates the area and circumference of a circle. The input should be entered by the user, and the output should be formatted to two decimal places.

Code:

```
import math

# Input radius from user
radius = float(input("Enter the radius of the circle: "))

# Calculating area and circumference
area = math.pi * radius**2
circumference = 2 * math.pi * radius

# Display the results
print(f"Area: {area:.2f}")
print(f"Circumference: {circumference:.2f}")
```

Sample Output:

```
Enter the radius of the circle: 5
Area: 78.54
Circumference: 31.42
```

Practical 4: Create Variables of Different Data Types and Verify Using `type()` Function

Objective:

To create variables of different data types and verify their types using the `type()` function.

Code:

```
# Different types of variables
a = 10                # Integer
b = 3.14              # Float
c = "Hello"           # String
d = [1, 2, 3]         # List
e = (4, 5, 6)         # Tuple
f = {7, 8, 9}         # Set
g = {'name': 'Alice', 'age': 25}  # Dictionary

# Print types
print(type(a))  # <class 'int'>
print(type(b))  # <class 'float'>
print(type(c))  # <class 'str'>
print(type(d))  # <class 'list'>
print(type(e))  # <class 'tuple'>
print(type(f))  # <class 'set'>
print(type(g))  # <class 'dict'>
```

Sample Output:

```
<class 'int'>
<class 'float'>
```

```
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'set'>
<class 'dict'>
```

Practical 5: Understand Variable Copies and References Using `id()` Function

Objective:

To demonstrate variable copies and references using `id()` function.

Code:

```
# Variable assignment and id check
x = 10
y = x
z = 10

print(id(x))  # Unique identifier for x
print(id(y))  # Same as x, because y references x
print(id(z))  # Same as x, because z is a copy with the same value
```

Sample Output:

```
140501923292784
140501923292784
140501923292784
```

Practical 6: Slicing, Concatenation, Repetition, and Membership Testing on Lists, Tuples, and Strings

Objective:

To practice slicing, concatenation, repetition, and membership testing on lists, tuples, and strings.

Code:

```
# Slicing and concatenation on strings
string = "Hello, Python!"
print(string[0:5])  # Slicing (Hello)
print(string + " World!")  # Concatenation

# Repetition on strings
print("Python " * 3)  # Repetition

# Membership testing in strings
print("Python" in string)  # True

# Slicing, concatenation, and repetition on lists
my_list = [1, 2, 3]
print(my_list[0:2])  # Slicing
print(my_list + [4, 5])  # Concatenation
print(my_list * 2)  # Repetition

# Membership testing in lists
print(3 in my_list)  # True
```

Sample Output:

```
Hello
Hello, Python! World!
Python Python Python
True
[1, 2]
[1, 2, 3, 4, 5]
[1, 2, 3, 1, 2, 3]
True
```

Practical 7: Methods of Lists, Tuples, Strings, Ranges, Sets, and Dictionaries

Objective:

To practice various methods of lists, tuples, strings, ranges, sets, and dictionaries.

Code:

```
# Methods of lists
my_list = [1, 2, 3, 4]
my_list.append(5)  # Add element
print(my_list)
my_list.remove(3)  # Remove element
print(my_list)

# Methods of tuples
my_tuple = (1, 2, 3, 4)
print(len(my_tuple))  # Length
print(my_tuple.count(2))  # Count occurrences

# Methods of strings
my_str = "Hello, Python!"
print(my_str.lower())  # Convert to lowercase
print(my_str.replace("Python", "World"))  # Replace substring

# Methods of ranges
my_range = range(1, 6)
print(list(my_range))  # Convert range to list

# Methods of sets
my_set = {1, 2, 3, 4}
my_set.add(5)  # Add element
print(my_set)

# Methods of dictionaries
my_dict = {'name': 'Alice', 'age': 25}
print(my_dict.keys())  # Keys
print(my_dict.values())  # Values
```

Sample Output:

```
[1, 2, 3, 4, 5]
[1, 2, 4, 5]
4
1
hello, python!
Hello, World!
[1, 2, 3, 4, 5]
```

```
{1, 2, 3, 4, 5}
dict_keys(['name', 'age'])
dict_values(['Alice', 25])
```

Practical 8: `map()`, `filter()`, `all()`, and `any()` Functions on Lists

Objective:

To practice the `map()`, `filter()`, `all()`, and `any()` functions on lists.

Code:

```
# map() example
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)

# filter() example
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)

# all() example
print(all([True, True, False]))  # False

# any() example
print(any([False, False, True]))  # True
```

Sample Output:

```
[1, 4, 9, 16]
[2, 4]
False
True
```

Practical 9: Create a List of Squares of the First 10 Natural Numbers Using List Comprehension

Objective:

To create a list of squares of the first 10 natural numbers using list comprehension.

Code:

```
squares = [x**2 for x in range(1, 11)]
print(squares)
```

Sample Output:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Practical 10: Demonstrate Dictionary Comprehension

Objective:

To demonstrate dictionary comprehension.

Code:

```
my_dict = {x: x**2 for x in range(1, 6)}
print(my_dict)
```

Sample Output:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Practical 11: Demonstrate the Working of `if` Statement and Its Variants

Objective:

To demonstrate the working of `if` statements and its variants.

Code:

```
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

Sample Output:

```
x is greater than 5
```

Practical 12: Compute the Factorial of a Given Number Using a While Loop

Objective:

To compute the factorial of a given number using a `while` loop.

Code:

```
n = int(input("Enter a number: "))
fact = 1
while n > 0:
    fact *= n
    n -= 1
print("Factorial:", fact)
```

Sample Output:

```
Enter a number: 6
Factorial: 720
```

Practical 13: Navigate Through the Elements of a List Using a `for` Statement

Objective:

To demonstrate navigation through a list using a `for` loop.

Code:

```
my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)
```

Sample Output:

```
1
2
3
4
5
```

Practical 14: Demonstrate Exception Handling Mechanism in Python

Objective:

To demonstrate exception handling in Python.

Code:

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Invalid input. Please enter a valid number.")
else:
    print(f"Result: {result}")
```

Sample Output:

```
Enter a number: 5
Result: 2.0
```

Practical 15: Write a Function to Compute the Greatest of Two Numbers

Objective:

To write a function that computes the greatest of two numbers.

Code:

```
def greatest(a, b):
    return a if a > b else b

num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
print(f"The greatest number is: {greatest(num1, num2)}")
```

Sample Output:

```
Enter first number: 10
Enter second number: 5
The greatest number is: 10
```

Practical 16: Demonstrate the Use of `continue`, `break`, and `pass` Statements

Objective:

To demonstrate the use of `continue`, `break`, and `pass` statements.

Code:

```
# continue statement
for i in range(1, 6):
    if i == 3:
        continue
    print(i)

# break statement
for i in range(1, 6):
    if i == 3:
        break
    print(i)

# pass statement
for i in range(1, 6):
    if i == 3:
        pass
    print(i)
```

Sample Output:

```
1
2
4
5
1
2
1
2
3
4
5
```

Practical 17: Demonstrate Named Arguments, Variable Arguments, and Keyword Arguments in Python Functions

Objective:

To demonstrate the use of named arguments, variable arguments, and keyword arguments in Python functions.

Code:

```
# Function with named arguments
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")
```

```
# Calling the function with named arguments
greet(name="Alice", age=30)

# Function with variable arguments (*args)
def add_numbers(*args):
    return sum(args)

# Calling the function with a variable number of arguments
print("Sum of numbers:", add_numbers(1, 2, 3, 4, 5))

# Function with keyword arguments (**kwargs)
def print_person_details(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Calling the function with keyword arguments
print_person_details(name="Bob", age=25, city="New York")
```

Explanation:

- **Named arguments** allow you to pass values to specific parameters by specifying the name.
- **Variable arguments (`*args`)** allow passing a variable number of arguments as a tuple.
- **Keyword arguments (`**kwargs`)** allow passing named arguments dynamically and are captured as a dictionary.

Sample Output:

```
Hello, Alice! You are 30 years old.
Sum of numbers: 15
name: Bob
age: 25
city: New York
```

Practical 18: Demonstrate Lambda Functions in Python

Objective:

To demonstrate the use of lambda functions in Python.

Code:

```
# Basic lambda function to add two numbers
add = lambda x, y: x + y
print("Sum of 10 and 20:", add(10, 20))

# Lambda function used inside a function like `map()`
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print("Squared numbers:", squared)

# Lambda function used inside a function like `filter()`
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print("Even numbers:", even_numbers)

# Lambda function with sorting
pairs = [(1, 2), (3, 1), (5, 4), (7, 2)]
sorted_pairs = sorted(pairs, key=lambda x: x[1])
print("Sorted pairs by second element:", sorted_pairs)
```

Explanation:

- A **lambda function** is an anonymous function defined using the `lambda` keyword.
- **Lambda functions** are often used in cases where a small function is needed temporarily.
- The `map()` function applies the lambda function to each element in a sequence.
- The `filter()` function uses the lambda function to filter elements from a sequence.
- The `sorted()` function can also accept a lambda to define custom sorting criteria.

Sample Output:

```
Sum of 10 and 20: 30
Squared numbers: [1, 4, 9, 16, 25]
Even numbers: [2, 4]
Sorted pairs by second element: [(3, 1), (1, 2), (7, 2), (5, 4)]
```

Practical 19: Copy the Contents of One File into Another Using Python File Handling Functions and `with` Statement

Objective:

To demonstrate how to copy the contents of one file into another using Python file handling functions and the `with` statement.

Code:

```
# Create a file and write some content into it
with open("source.txt", "w") as source_file:
    source_file.write("Hello, this is a sample text file.\nIt contains some 
text.")

# Copy the contents of source.txt into destination.txt
with open("source.txt", "r") as source_file, open("destination.txt", "w") as 
dest_file:
    content = source_file.read()
    dest_file.write(content)

# Check if the content is copied by reading the destination file
with open("destination.txt", "r") as dest_file:
    copied_content = dest_file.read()
    print("Copied content:\n", copied_content)
```

Explanation:

- The **`with` statement** automatically takes care of closing the files after the operation is complete, even if an error occurs.
- The file content is read using the `read()` method and written to another file using the `write()` method.

Sample Output:

```
Copied content:
 Hello, this is a sample text file.
It contains some text.
```

Notes:

1. Ensure that `source.txt` exists with content before running the code.
2. If `destination.txt` does not exist, it will be created automatically.