LAB MANUAL

NAME: SHOAIB ALI

ROLL NO: BIT-24S-003

GITHUB LINK: https://github.com/kingShoaibAli786

LAB NO: 1

INTRODUCTION TO PYTHON :

Python is a beginner-friendly programming language that's easy to learn and use. It’s popular because it has simple, readable code and can be used for many things, like building websites, analyzing data, or creating games. Python works on all major operating systems, like Windows and Mac, and it has many built-in tools (called libraries) to make your work easier.

TASK NO 1: Make 2-2 programs of each datatype.

1. NUMERIC TYPES

Integer (int)

```
[1]: a = 10
     b = 5
     sum = a + b
     print("The sum is:", sum)
```

```
The sum is: 15
```

```
[2]: a = 100
     b = 45
     sub = a - b
     print("The subtraction is:", sub)
```

```
The subtraction is: 55
```

---

*[1]:
```
a = 10
b = 5
sum = a + b
print("The sum is:", sum)
```

```
The sum is: 15
```

*[2]:
```
a = 100
b = 45
sub = a - b
print("The subtraction is:", sub)
```

```
The subtraction is: 55
```

Floating-point (float)

[3]:
```
num1 = 7.5
num2 = 2.5
sum_result = num1 + num2
product_result = num1 * num2
print("Sum of", num1, "and", num2, "is:", sum_result)
print("Product of", num1, "and", num2, "is:", product_result)
```

```
Sum of 7.5 and 2.5 is: 10.0
Product of 7.5 and 2.5 is: 18.75
```

[4]:
```
num1 = 9.0
num2 = 4.0
division_result = num1 / num2
print(f"The division of {num1} by {num2} is: {division_result:.2f}")
```

```
The division of 9.0 by 4.0 is: 2.25
```

Complex (complex)

•[5]:
```
complex_num = 3 + 4j
real_part = complex_num.real
imaginary_part = complex_num.imag
complex_sum = complex_num + (1 + 2j)
print("The complex number is:", complex_num)
print("Real part:", real_part)
print("Imaginary part:", imaginary_part)
print("Sum with another complex number:", complex_sum)
```

```
The complex number is: (3+4j)
Real part: 3.0
Imaginary part: 4.0
Sum with another complex number: (4+6j)
```

•[6]:
```
complex_num1 = 2 + 3j
complex_num2 = 4 + 5j
product = complex_num1 * complex_num2
print(f"The product of {complex_num1} and {complex_num2} is: {product}")
```

```
The product of (2+3j) and (4+5j) is: (-7+22j)
```

2. Sequence Types

String (str)

[7]:
```
message = "Hello, Python!"
print(message)
```

```
Hello, Python!
```

[8]:
```
sentence = "Python is awesome"
uppercase_sentence = sentence.upper()
print("Original string:", sentence)
print("Uppercase string:", uppercase_sentence)
```

```
Original string: Python is awesome
Uppercase string: PYTHON IS AWESOME
```

List (list)

```
[9]: fruits = ["apple", "banana", "cherry"]
print("Original list:", fruits)
first_fruit = fruits[0]
print("First fruit in the list:", first_fruit)
fruits.append("orange")
print("Updated list:", fruits)
```

```
Original list: ['apple', 'banana', 'cherry']
First fruit in the list: apple
Updated list: ['apple', 'banana', 'cherry', 'orange']
```

```
[10]: numbers = [10, 20, 30, 40, 50]
print("Original list:", numbers)
numbers.remove(30)
print("List after removing 30:", numbers)
index_of_40 = numbers.index(40)
print("Index of 40:", index_of_40)
```

```
Original list: [10, 20, 30, 40, 50]
List after removing 30: [10, 20, 40, 50]
Index of 40: 2
```

Tuple (tuple)

[11]:
```
tuple1 = (1, 2, 3, 4)
tuple2 = ("apple", "banana", "cherry")
print("First tuple:", tuple1)
print("Second tuple:", tuple2)
first_element = tuple1[0]
second_element = tuple2[1]
print("First element of the first tuple:", first_element)
print("Second element of the second tuple:", second_element)
```

```
First tuple: (1, 2, 3, 4)
Second tuple: ('apple', 'banana', 'cherry')
First element of the first tuple: 1
Second element of the second tuple: banana
```

[12]:
```
tuple1 = (10, 20, 30)
tuple2 = (40, 50)
concatenated_tuple = tuple1 + tuple2
repeated_tuple = tuple1 * 2
print("Concatenated tuple:", concatenated_tuple)
print("Repeated tuple:", repeated_tuple)
```

```
Concatenated tuple: (10, 20, 30, 40, 50)
Repeated tuple: (10, 20, 30, 10, 20, 30)
```

Range (range)

[21]:
```
for number in range(1, 6):
    print(number)
```

```
1
2
3
4
5
```

[22]:
```
for number in range(0, 11, 2):
    print(number)
```

```
0
2
4
6
8
10
```

3. SET TYPES

Set (set)

[23]:
```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
union_set = set1 | set2
intersection_set = set1 & set2
print("Set 1:", set1)
print("Set 2:", set2)
print("Union of sets:", union_set)
print("Intersection of sets:", intersection_set)
```

```
Set 1: {1, 2, 3, 4}
Set 2: {3, 4, 5, 6}
Union of sets: {1, 2, 3, 4, 5, 6}
Intersection of sets: {3, 4}
```

[25]:
```
fruits = {"apple", "banana", "cherry"}
fruits.add("orange")
fruits.remove("banana")
print("Updated set:", fruits)
```

```
Updated set: {'apple', 'orange', 'cherry'}
```

Frozen Set (frozenset)

[26]:
```
frozenset1 = frozenset([1, 2, 3, 4])
frozenset2 = frozenset([3, 4, 5, 6])
union_frozenset = frozenset1 | frozenset2
intersection_frozenset = frozenset1 & frozenset2
print("Frozenset 1:", frozenset1)
print("Frozenset 2:", frozenset2)
print("Union of frozensets:", union_frozenset)
print("Intersection of frozensets:", intersection_frozenset)
```

```
Frozenset 1: frozenset({1, 2, 3, 4})
Frozenset 2: frozenset({3, 4, 5, 6})
Union of frozensets: frozenset({1, 2, 3, 4, 5, 6})
Intersection of frozensets: frozenset({3, 4})
```

[27]:
```
fruits_frozenset = frozenset(["apple", "banana", "cherry"])
try:
    fruits_frozenset.add("orange")
except AttributeError as e:
    print("Error:", e)
print("Fruits frozenset:", fruits_frozenset)
```

```
Error: 'frozenset' object has no attribute 'add'
Fruits frozenset: frozenset({'apple', 'banana', 'cherry'})
```

3. MAPPING TYPE

Dictionary (dict)

[28]:
```
student = {
    "name": "Alice",
    "age": 20,
    "grade": "A"
}
print("Student Dictionary:", student)
name = student["name"]
age = student["age"]
print("Name:", name)
print("Age:", age)
student["subject"] = "Mathematics"
print("Updated Dictionary:", student)
```

```
Student Dictionary: {'name': 'Alice', 'age': 20, 'grade': 'A'}
Name: Alice
Age: 20
Updated Dictionary: {'name': 'Alice', 'age': 20, 'grade': 'A', 'subject': 'Mathematics'}
```

[29]:
```
person = {
    "first_name": "John",
    "last_name": "Doe",
    "age": 25,
    "city": "New York"
}
for key, value in person.items():
    print(key + ":", value)
```

```
first_name: John
last_name: Doe
age: 25
city: New York
```

4. Boolean Type

Boolean (bool)

*[30]:
```
is_raining = True
is_sunny = False
print("Is it raining?", is_raining)
print("Is it sunny?", is_sunny)
result_and = is_raining and is_sunny
print("Is it both raining and sunny?", result_and)
result_or = is_raining or is_sunny
print("Is it either raining or sunny?", result_or)
```

```
Is it raining? True
Is it sunny? False
Is it both raining and sunny? False
Is it either raining or sunny? True
```

[31]:
```
number = 10
is_greater_than_five = number > 5
print("Is the number greater than 5?", is_greater_than_five)
is_equal_to_ten = number == 10
print("Is the number equal to 10?", is_equal_to_ten)
```

```
Is the number greater than 5? True
Is the number equal to 10? True
```

TASK NO 2: Make up to 5 Shape programs using *.

[45]:
```
print("""
*
**
***
****
*****
""")
```

```
*
**
***
****
*****
```

[50]:
```
print("""
*********
 *******
  *****
   ***
    *


""")
```

```
*********
 *******
  *****
   ***
    *
```

[30]:
```
print("""
    *
   ***
  *****
*******
""")
```

```
    *
   ***
  *****
*******
```

[46]:
```
print("""
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
""")
```

```
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
```

[40]:

```
print("""
     *
    ***
   *****
*********
   *****
    ***
     *


""")
```

TASK NO 3: Make same shapes you have made in task 2, using * mutiple by number.

✅ 1: Right-Angled Triangle

```python
print("1: Right-angled triangle")
print("*" * 1)
print("*" * 2)
print("*" * 3)
print("*" * 4)
print("*" * 5)
print(" ")
```

🔶 Output:

```markdown
1: Right-angled triangle
*
**
***
****
*****
```

✅ 2: Inverted Right-Angled Triangle

python

```python
print(" ")
print("2:Inverted right- agled triangle")
print("*" * 5)
print("*" * 4)
print("*" * 3)
print("*" * 2)
print("*" * 1)
```

Output:

markdown

```
2:Inverted right- agled triangle
*****
****
***
**
*
```

✅ 3: Square Pattern

```python
print(" ")
print("3:Square pattern")
row = "*" * 5  # Create a row of 5 stars
print(row)
print(row)
print(row)
print(row)
print(row)
```

Output:

```markdown
3:Square pattern
*****
*****
*****
*****
*****
```

✅ 4: Pyramid Pattern

```python
print(" ")
print("4:Pyramid Pattern")
print(" " * 4 + "*" * 1)
print(" " * 3 + "*" * 3)
print(" " * 2 + "*" * 5)
print(" " * 1 + "*" * 7)
print(" " * 0 + "*" * 9)
```

Output:

```markdown
4:Pyramid Pattern
     *
    ***
   *****
  *******
*********
```

✅ 5: Diamond Pattern

```python
print(" ")
print("5:Diamond Pattern")
print(" " * 4 + "*" * 1)
print(" " * 3 + "*" * 3)
print(" " * 2 + "*" * 5)
print(" " * 1 + "*" * 7)
print(" " * 0 + "*" * 9)
print(" " * 1 + "*" * 7)
print(" " * 2 + "*" * 5)
print(" " * 3 + "*" * 3)
print(" " * 4 + "*" * 1)
```

```
5:Diamond Pattern
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```