Assignment - 2

1. Write a Python program to take the name and age of a user as input and display it in the format: "Hello <name>, you are <age> years old."

```
[27] #LAB Asignment - 2

     #1 WAP to take the name and age of a user as input and display it in the format: "Hello <name>, you are <age> years old."

     name = input("Enter your name: ")
     age = int(input("Enter your age: "))

     print("Hello",name,", you are",age ,"years old");
```

```
Enter your name: John
Enter your age: 30
Hello John , you are 30 years old
```

2. Examine the default data type returned by the input() function. Analyze how converting this data type affects subsequent operations, and suggest a method for conversion.

```
[8]  #2 Examine the default data type returned by the input() function.
     #Analyze how converting this data type affects subsequent operations, and suggest a method for conversion.

     a = input("Enter the number: ")
     print(type(a))
     c = a + 'hello'
     print(c)
     a = int(a)
     d = 5 + a
     print(type(a))
     print(d)
```

```
Enter the number: 3
<class 'str'>
3hello
<class 'int'>
8
```

3. Write a Python program that asks the user to input two numbers and then prints their sum, difference, product, and quotient.

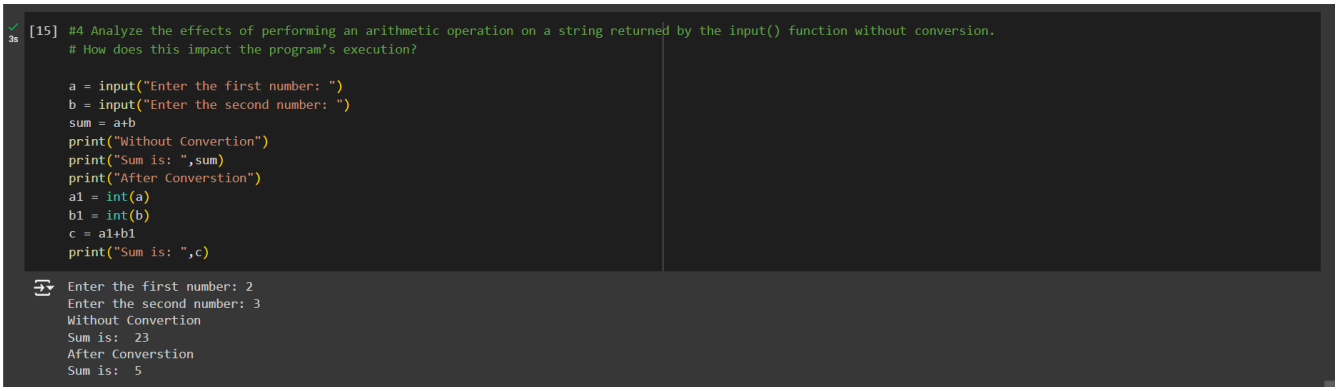
```
[9]  #3 Write a Python program that asks the user to input two numbers and then prints their sum, difference, product, and quotient.
     a = int(input("Enter the first number: "))
     b = int(input("Enter the second number: "))

     sum = a+b
     diff = a-b
     product = a*b
     quotient = a/b

     print("Sum is: ",sum)
     print("Difference is: ",diff)
     print("Product is: ",product)
     print("Quotient is: ",quotient)
```

```
Enter the first number: 4
Enter the second number: 2
Sum is:  6
Difference is:  2
Product is:  8
Quotient is:  2.0
```

4. Analyze the effects of performing an arithmetic operation on a string returned by the input() function without conversion. How does this impact the program's execution?



```
[15] #4 Analyze the effects of performing an arithmetic operation on a string returned by the input() function without conversion.
     # How does this impact the program's execution?

     a = input("Enter the first number: ")
     b = input("Enter the second number: ")
     sum = a+b
     print("Without Convertion")
     print("Sum is: ",sum)
     print("After Converstion")
     a1 = int(a)
     b1 = int(b)
     c = a1+b1
     print("Sum is: ",c)

Enter the first number: 2
Enter the second number: 3
Without Convertion
Sum is:  23
After Converstion
Sum is:  5
```

5. Identify the syntax error in the following code:

i. Python
ii. Copy code
iii. print("Hello World"

A.CASE-SENSITIVE ERROR

6. Explain what a syntax error is and why it prevents the execution of a program.

A syntax error happens when you don't follow the rules of the programming language. For example, if you forget to close a bracket or misspell something, the computer won't understand what to do. This stops the program from running because the computer can't figure out the instructions properly until you fix the mistake.

7. What is a logical error in Python? Provide an example where a program runs without any errors but produces an incorrect result.

A logical error in Python is when the code runs without any syntax or runtime errors, but the output is incorrect because the logic used in the program is flawed. In this case, the program does exactly what you've told it to do, but it's not what you intended, which leads to wrong results.

```
[2] num1 = 10
    num2 = 20
    average = num1 + num2 / 2
    print("Average:", average)

Average: 20.0
```

8. What is a runtime error? Write a Python code that raises a "division by zero" error and explain why it happens.

A runtime error is an error that occurs while the program is running, after the code has been successfully compiled or interpreted. Unlike syntax errors, runtime errors happen during the execution of the program when something unexpected occurs, like trying to perform an illegal operation.

```
[3] numerator = 10
    denominator = 0
    result = numerator / denominator
    print("Result:", result)

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-34f475f5a478> in <cell line: 3>()
      1 numerator = 10
      2 denominator = 0
----> 3 result = numerator / denominator
      4 print("Result:", result)

ZeroDivisionError: division by zero
```

9. Modify the following code to prevent a runtime error when dividing by zero:
    i. python
    ii. Copy code
    iii. numerator = 10
    iv. denominator = 0
    v. result = numerator / denominator

```
[4] numerator = 10
    denominator = 0

    if denominator != 0:
        result = numerator / denominator
        print("Result:", result)
    else:
        print("Error: Division by zero is not allowed.")

Error: Division by zero is not allowed.
```

10. How do you write comments in Python? Provide examples of a full-line comment, an inline comment, and a multi-line comment.

In Python, comments are written using the # symbol.

1. **Full-line Comment:**

```
[5]  # This is a full-line comment
     print("Hello, world!")  # This is not part of the comment

     Hello, world!
```

2. **Inline Comment:**

```
[ ]  x = 10  # This is an inline comment explaining the value of x
```

3. **Multi-line Comment:**

```
[6]  # This is a multi-line comment
     # explaining that the code below
     # adds two numbers and prints the result
     x = 5
     y = 10
     result = x + y
     print(result)

     15
```

11. What are the different types of numeric data in Python? Provide an example for each.
    a. **Integer (int):**

```
[ ]  x = 10  # Positive integer
     y = -5  # Negative integer
     z = 0   # Zero
```

    b. **Float (float):**

```
[ ]  a = 10.5   # Decimal number (float)
     b = -2.75  # Negative float
     c = 1.2e3  # Scientific notation (1.2 * 10^3), equivalent to 1200.0
```

    c. **Complex (complex):**

```
     p = 3 + 4j  # Complex number with real part 3 and imaginary part 4
     q = -2 - 5j  # Complex number with negative real and imaginary parts
```

Example:

```python
# Integer (int) examples
x = 10
y = -5
z = 0

print("Integer examples:")
print("x =", x)
print("y =", y)
print("z =", z)

# Float (float) examples
a = 10.5
b = -2.75
c = 1.2e3

print("\nFloat examples:")
print("a =", a)
print("b =", b)
print("c =", c)

# Complex (complex) examples
p = 3 + 4j
q = -2 - 5j

print("\nComplex number examples:")
print("p =", p)
print("q =", q)

# Performing operations on numeric types
int_sum = x + y
print("\nSum of integers (x + y):", int_sum)

float_mult = a * b
print("Multiplication of floats (a * b):", float_mult)

complex_add = p + q
print("Addition of complex numbers (p + q):", complex_add)
```

```
Integer examples:
x = 10
y = -5
z = 0

Float examples:
a = 10.5
b = -2.75
c = 1200.0

Complex number examples:
p = (3+4j)
q = (-2-5j)

Sum of integers (x + y): 5
Multiplication of floats (a * b): -28.875
Addition of complex numbers (p + q): (1-1j)
```

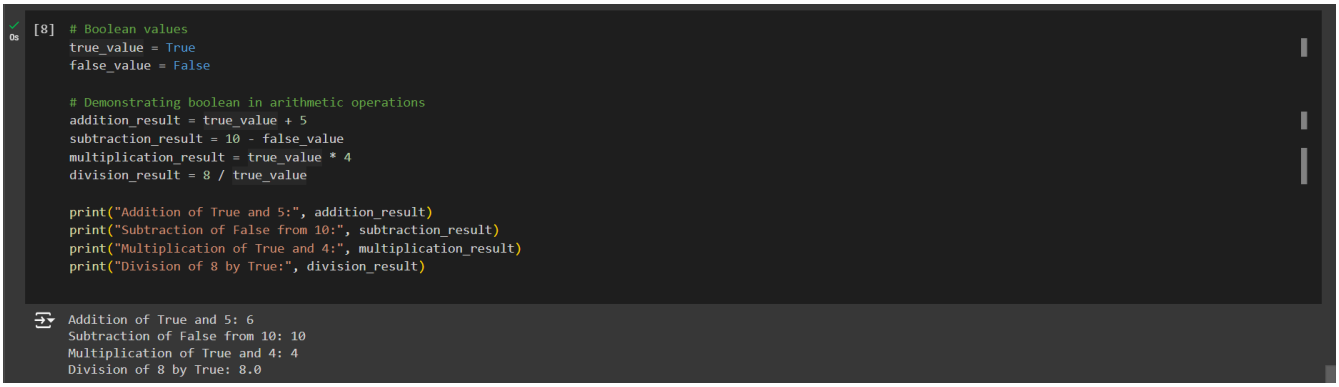
12. What is the boolean data type in Python? How is it related to integers? Write a Python code that demonstrates the use of True and False in an arithmetic operation.

The boolean data type in Python has two possible values: True and False. It is used to represent truth values and is often the result of comparisons or logical operations.

Relationship to integers:

In python booleans are the subclass of integers, we use the numeric values to represent the true/false.

- True is equivalent to 1
- False is equivalent to 0



```
[8]  # Boolean values
     true_value = True
     false_value = False

     # Demonstrating boolean in arithmetic operations
     addition_result = true_value + 5
     subtraction_result = 10 - false_value
     multiplication_result = true_value * 4
     division_result = 8 / true_value

     print("Addition of True and 5:", addition_result)
     print("Subtraction of False from 10:", subtraction_result)
     print("Multiplication of True and 4:", multiplication_result)
     print("Division of 8 by True:", division_result)

     Addition of True and 5: 6
     Subtraction of False from 10: 10
     Multiplication of True and 4: 4
     Division of 8 by True: 8.0
```

13. Explain the difference between simple data types (like int, float) and sequence data types (like lists, tuples) in Python.

Simple data types, also known as primitive data types, are the basic building blocks of data in Python. They include:

1. **int**: Represents integer values.
    a. e.g., 5, -3, 42.
2. **float**: Represents floating-point numbers (decimal values),
    a. e.g., 3.14, -0.001
3. **bool**: Represents boolean values,
    a. True or False.
4. **str**: Represents strings, which are sequences of characters,
    a. e.g., "hello", "Python".

These data types are atomic, meaning they cannot be broken down into simpler components.

Sequence Data Types

Sequence data types are collections of items that are ordered and indexed. They include:

1. **list**: An ordered, mutable collection of items,
    a. e.g., [1, 2, 3], ["apple", "banana", "cherry"].
2. **tuple**: An ordered, immutable collection of items,
    a. e.g., (1, 2, 3), ("apple", "banana", "cherry").
3. **str**: While also a simple data type, strings are technically sequences of characters,
    a. e.g., "hello", "Python".

Major Differences:

- **Mutability**
  - Simple Data Types: Generally immutable except for the list and dict.
  - Sequence Data Types: Lists are mutable; tuples are immutable.
- **Indexing & Slicing**
  - Simple Data Types: Generally do not support indexing or slicing except for strings.
  - Sequence Data Types: Allow indexing and slicing for element access and modification.
- **Homogeneity**
  - Simple Data Types: Represent a single value of some kind.
  - Sequence Data Types: Some data types can hold multiple items of different types. Some examples include: A list holding integers, strings and even other lists.

14. What is a string in Python? Write a Python program that takes a user's full name as input and prints the initials.
    - In Python, a string is a sequence of characters enclosed within single quotes (' '), double quotes (" "), or triple quotes (''' ''' or """ """).
    - Strings are used to represent text and can include letters, numbers, symbols, and whitespace.
    - They are immutable, meaning once created, their content cannot be changed.

15. What are the differences between lists and tuples in Python? Provide examples where you create a list and a tuple to store a collection of student names.

List	Tuples
• Lists are mutable, meaning their contents can be changed (elements can be added, removed, or modified).	• Tuples are immutable, meaning once they are created, their contents cannot be ltered.
• Lists are created using square brackets [].	• Tuples are created using parentheses ().
• They are slower compared to tuples	• Tuples are generally faster than lists because they are immutable and simpler in memory.
• Lists are used when you need a collection of items that may change.	• Tuples are used when the collection is fixed and should not be modified.

Creating a list:

Creating a Tuple:

16. What is a dictionary in Python? Write a Python program that creates a dictionary to store the names and phone numbers of three people and then prints the phone number of one of them.

A dictionary in Python is a collection of key-value pairs. Each key is unique and is used to access its corresponding value. Dictionaries are defined using curly braces {} with keys and values separated by a colon ':'

Example:

```
[4] # Create a dictionary to store names and phone numbers
    phone_book = {
        "Alice": "123-456-7890",
        "Bob": "987-654-3210",
        "Charlie": "555-666-7777"
    }
    print("Bob's phone number is:", phone_book["Bob"])

Bob's phone number is: 987-654-3210
```

17. Write a Python program to declare two variables x and y. Initialize x with a value of 50 and y with x + 20. Print the value of y using a print statement.

```
[5] x = 50
    y = x + 20

    print("The value of y is:", y)

The value of y is: 70
```

18. What will be the output of the following code snippet? Explain why:

    python
    Copy code
    a = 15
    b = a * 2
    print(b)

```
[6] a = 15
    b = a * 2
    print(b)

30
```

- a = 15 , creates a variable a and assigns it the value 15. Now, a holds the value 15.
- b = a*2, This line creates another variable b and assigns it the value of a multiplies by 20. Since a is 15, b becomes 15*2 = 30.
- print(b),This line prints the value of b. The output will be: 30.

19. Create a Python script that assigns values of different numeric types to variables: an integer, a float, and a complex number. Print each variable with its type.

```
[7] integer_var = 10
    float_var = 15.75
    complex_var = 3 + 4j

    print("Integer value:", integer_var, "Type:", type(integer_var))
    print("Float value:", float_var, "Type:", type(float_var))
    print("Complex number value:", complex_var, "Type:", type(complex_var))

Integer value: 10 Type: <class 'int'>
Float value: 15.75 Type: <class 'float'>
Complex number value: (3+4j) Type: <class 'complex'>
```

20. Write a Python program that demonstrates the usage of integer, float, and complex numbers in arithmetic operations. Print the results.

```
[1]  integer_var = 10
     float_var = 5.5
     complex_var = 2 + 3j

     int_add_float = integer_var + float_var
     float_mul_complex = float_var * complex_var
     complex_add_int = complex_var + integer_var

     print("Integer + Float (10 + 5.5):", int_add_float)
     print("Float * Complex (5.5 * (2 + 3j)):", float_mul_complex)
     print("Complex + Integer ((2 + 3j) + 10):", complex_add_int)
```

```
Integer + Float (10 + 5.5): 15.5
Float * Complex (5.5 * (2 + 3j)): (11+16.5j)
Complex + Integer ((2 + 3j) + 10): (12+3j)
```

21. Write a Python program that assigns boolean values True and False to two variables. Use these variables in an arithmetic operation and print the result.

```
[5]  a = True
     b = False

     result = a + b
     subsrtact = a-b
     multiply = a*b
     print(result)
     print(subsrtact)
     print(multiply)
```

```
1
1
0
```

22. Create a function that accepts a boolean value and prints "True value" if the input is True and "False value" if the input is False.

```
def check_boolean(value):
    if value == True:
        print("True value")
    else:
        print("False value")

is_open = True
is_closed = False

check_boolean(is_open)
check_boolean(is_closed)
```

```
True value
False value
```

23. Write a Python program that takes a string input from the user and prints the string in uppercase and lowercase.

```
[7]  name = input("Enter your name: ")

     upper = name.upper()
     lower = name.lower()

     print("Uppercase:", upper)
     print("Lowercase:", lower)
```

```
Enter your name: Nandu
Uppercase: NANDU
Lowercase: nandu
```

24. Create a list of five different items, including at least one string and one number. Add a new item to the list and print the updated list. Demonstrate accessing an element by index.

```
[8] lst = ['nandu',15,'#']
    print(lst)
    lst.append("hello")
    print(lst)

    print(lst[3])
```

```
['nandu', 15, '#']
['nandu', 15, '#', 'hello']
hello
```

25. Create a tuple with four items of different data types. Print the tuple and access the second item.

```
[12] tuple_ex = ('nandu',15,'3.24',True)
     print(tuple_ex)
     print(tuple_ex[1])
```

```
('nandu', 15, '3.24', True)
15
```

26. Write a Python program that creates both a list and a tuple containing the same set of items. Modify the list by adding and removing items. Attempt to modify the tuple and explain the result.

```
my_list = [10, 20, 30, 40]
my_tuple = (10, 20, 30, 40)

my_list.append(50)
my_list.remove(20)
print("Modified list:", my_list)
my_tuple[1] = 25
print("Tuple:", my_tuple)
```

```
Modified list: [10, 30, 40, 50]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-6690f4bddfc6> in <cell line: 9>()
      7 print("Modified list:", my_list)
      8
----> 9 my_tuple[1] = 25
     10
     11 print("Tuple:", my_tuple)

TypeError: 'tuple' object does not support item assignment
```

Tuples are immutable object, once we create it, we cannot change the value in it.

27. Create a list of integers and a tuple of integers. Print both and compare their performance in terms of mutability and iteration.

```
[17] import time
     num_items = 10

     my_list = list(range(num_items))
     my_tuple = tuple(range(num_items))
     print("List:", my_list)
     print("Tuple:", my_tuple)
     print()

     print("Mutability Test")
     try:
         if num_items > 0:
             my_list[0] = -1
             print("Modified List:", my_list)
     except Exception as e:
         print(f"Error modifying list: {e}")

     try:
         if num_items > 0:
             my_tuple[0] = -1  # This will raise an error
     except Exception as e:
         print(f"Error modifying tuple: {e}")

     print()
     print("Itreation Test")
     start_time = time.time()
     for item in my_list:
         pass
     list_iteration_time = time.time() - start_time

     start_time = time.time()
     for item in my_tuple:
         pass
     tuple_iteration_time = time.time() - start_time

     print("List iteration time:", list_iteration_time)
     print("Tuple iteration time:", tuple_iteration_time)
```

```
List: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Tuple: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

Mutability Test
Modified List: [-1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Error modifying tuple: 'tuple' object does not support item assignment

Itreation Test
List iteration time: 8.130073547363281e-05
Tuple iteration time: 5.8650970458984375e-05
```

28. Create a set containing five unique items, including a mix of strings and numbers. Print the set and demonstrate adding a new item to the set.

```
[18] my_set = {1, 2, 3, 'apple', 'banana'}
     print("Original Set:", my_set)

     my_set.add('cherry')
     print("Updated Set:", my_set)
```

```
Original Set: {'banana', 1, 'apple', 2, 3}
Updated Set: {'banana', 1, 'apple', 2, 3, 'cherry'}
```

29. Write a Python program that shows how sets handle duplicate values. Add duplicate items to a set and print the result.

```
[19] my_set = {'cat', 'dog', 'fish'}
     my_set.add('cat')
     my_set.add('dog')
     my_set.add('bird')
     my_set.add(42)
     my_set.add(42)

     print("Set after adding duplicates:", my_set)
```

```
Set after adding duplicates: {42, 'bird', 'fish', 'cat', 'dog'}
```

30. Write a Python program to create a dictionary that stores names of three people and their ages. Print the dictionary, access an age using a key, and add a new key-value pair.

```
[20] people_ages = {
        'John': 28,
        'Emma': 32,
        'Liam': 45
    }

    print("Dictionary:", people_ages)

    name = 'Emma'
    age = people_ages[name]
    print(f"{name}'s age:", age)

    people_ages['Olivia'] = 29

    print("Updated Dictionary:", people_ages)
```

```
Dictionary: {'John': 28, 'Emma': 32, 'Liam': 45}
Emma's age: 32
Updated Dictionary: {'John': 28, 'Emma': 32, 'Liam': 45, 'Olivia': 29}
```

31. Modify the dictionary by changing one person's age and removing another person. Print the final dictionary. Include code to check if a specific key exists.

```
[21] people_ages = {
        'John': 28,
        'Emma': 32,
        'Liam': 45
    }

    print("Original Dictionary:", people_ages)

    people_ages['John'] = 30
    del people_ages['Liam']

    key_to_check = 'Emma'
    if key_to_check in people_ages:
        print(f"{key_to_check} is in the dictionary.")
    else:
        print(f"{key_to_check} is not in the dictionary.")

    print("Final Dictionary:", people_ages)
```

```
Original Dictionary: {'John': 28, 'Emma': 32, 'Liam': 45}
Emma is in the dictionary.
Final Dictionary: {'John': 30, 'Emma': 32}
```

32. Write a Python program that uses the len() function to find the length of a string, list, and dictionary. Print the results.

```
string_ex = "Hello, world!"
list_ex = [1, 2, 3, 4, 5]
dict_EX = {'apple': 1, 'banana': 2, 'cherry': 3}

string_length = len(string_ex)
print("Length of the string:", string_length)

list_length = len(list_ex)
print("Length of the list:", list_length)

dict_length = len(dict_EX)
print("Length of the dictionary:", dict_length)
```

```
Length of the string: 13
Length of the list: 5
Length of the dictionary: 3
```

33. Create a function named multiply_numbers that takes two numbers as arguments and returns their product. Call the function with different arguments and print the results.

```
[23] def multiply_numbers(a, b):
         return a * b

     result1 = multiply_numbers(0, 5)
     result2 = multiply_numbers(-4, 6)
     result3 = multiply_numbers(-3, -7)

     print("Result of 0 * 5:", result1)
     print("Result of -4 * 6:", result2)
     print("Result of -3 * -7:", result3)
```

```
Result of 0 * 5: 0
Result of -4 * 6: -24
Result of -3 * -7: 21
```

34. Write a Python program that creates a dictionary where the keys are names of fruits and the values are lists containing the color and average price of the fruit. Print the dictionary, then access and print the color and price of a specific fruit.

```
[1]  fruits = {
         'Apple': ['Red', 1.2],
         'Banana': ['Yellow', 0.5],
         'Cherry': ['Red', 2.0],
         'Grape': ['violet', 2.5]
     }

     print("Fruit Info :", fruits)

     fruit = 'Banana'
     color, price = fruits[fruit]
     print(f"The color of {fruit} is {color} and the average price is ${price}.")
```

```
Fruit Info : {'Apple': ['Red', 1.2], 'Banana': ['Yellow', 0.5], 'Cherry': ['Red', 2.0], 'Grape': ['violet', 2.5]}
The color of Banana is Yellow and the average price is $0.5.
```

35. Identify and correct the syntax error in the following code snippet:

    python
    Copy code
    my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'
    print(my_dict['city'])

my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'

print(my_dict['city'])

36. Write a Python program that handles a ZeroDivisionError by using a try-except block. The program should attempt to divide two numbers and catch the division by zero error, printing a friendly message instead.

```
[2]  def division_zero(a, b):
         try:
             result = a / b
             print(f"The result of {a} divided by {b} is {result}.")
         except ZeroDivisionError:
             print("Error: Cannot divide by zero.")

     division_zero(10, 2)
     division_zero(5, 0)
```

```
The result of 10 divided by 2 is 5.0.
Error: Cannot divide by zero.
```