**Python**

**Escape sequence in python:**

That type of series of characters that have their own functionalities when used inside a string or a character. Some of them are:

* \n: Newline (line break)
* \t: Horizontal tab
* \\: Backslash
* \': Single quote
* \": Double quote
* \r: Carriage return
* \b: Backspace
* \f: Formfeed
* \v: Vertical tab
* \a: Bell/alert
* \0: Null character

Example are:

* print("Hello\nWorld") Output: Hello

World

* print("My name is Bhaskar:\tSubedi") Output: My name is Bhaskar: Subedi
* print("Escape backslash: \\") Output: Escape backslash: \

**Comments:**

There are two types of comments: Single line and multiline comment. In python, simply using # in front of line can make it single line comment. But there is no concept of multiline comment. To make multiline comment, select all line to comment and press ctrl+/ to produce # in multiline.

**Operators in Python**

Python supports a wide range of operators that can be categorized as:

* Arithmetic Operators
* Comparison Operators
* Logical Operators
* Assignment Operators
* Bitwise Operators
* Shift Operators
* Equality Operators
* Ternary Operator
* Membership Operators
* Identity Operators

**Arithmetic operators:**

Arithmetic operators are used to perform basic mathematical operations. These operators can be used with numbers, variables, or expressions. There were 2 type of operators in string performance that are concatenation and repetition operator.'

Concatenation operator adds 2 strings as: a= “Bhaskar” b= “Subedi” print(a+b) output: Bhaskar Subedi

Repetition operator multiply the string. It needs one string and one number variable

Here are the most common arithmetic operators:

* Addition(+):result = 5 + 3 output: 8
* Subtraction(-) : result = 5 - 3 output: 2
* Multiplication(\*):result = 5 \* 3 output: 15
* Division(/):result = 5 / 2 output: 2.5
* Floor division(//):result = 5 // 2 output: 2
* Modulus(%): result= 5%2 output: 1
* Exponential(\*\*): result= 5\*\*2 output: 25

**Comparison operators:**

Comparison operators are used to compare two values or expressions. The result of a comparison is a Boolean value: True or False. These comparison operators are commonly used in conditional statements such as if, while, and in loops to make decisions based on the comparisons. Here are the common comparison operators:

* Equal to(==): Check if 2 values are equal. result = 5 == 3 output: False
* Not equal to (!=): Checks if two values are not equal. result = 5 != 3 output: True
* **Greater than (>):** Checks if the left operand is greater than the right. result = 5 > 3 output: True
* **Less than (<):** Checks if the left operand is less than the right. result = 5 < 3 output: False
* **Greater than or equal to (>=):** Checks if the left operand is greater than or equal to the right. result = 5 >= 3 output: True
* **Less than or equal to (<=):** Checks if the left operand is less than or equal to the right.

result = 5 <= 3 output: False

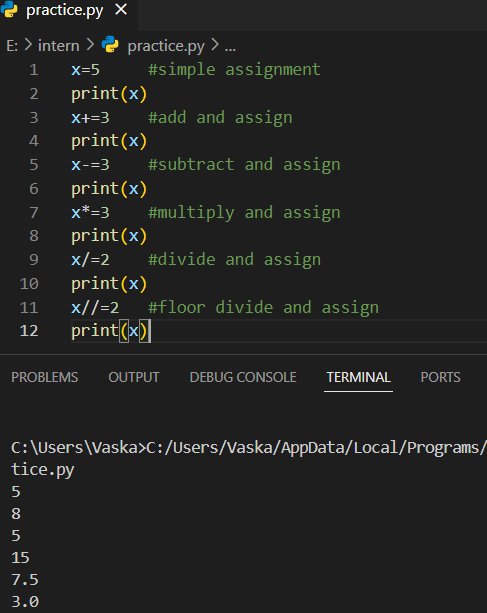
**Logical operators:**

Logical operators are used to combine conditional statements or expressions. These operators return a Boolean value (True or False) based on the logic of the expressions. These logical operators are often used in conditional expressions such as if statements to control the flow of a program based on multiple conditions. The three main logical operators are:

* AND (and): Returns True if both conditions are True. result = (5 > 3) and (2 < 4) output: True
* OR (or): Returns True if at least one of the conditions is True. result = (5 > 3) or (2 > 4) output: True
* NOT (not): Reverses the result of the condition. If a condition is True, it returns False, and vice versa. result = not (5 > 3) output: False

**Assignment Operators:**

Assignment operators are used to assign values to variables. Besides the simple assignment operator (=), there are several compound assignment operators that perform an operation and then assign the result to a variable. Here’s a list of common assignment operators in the figure of code are:



**Shift operators:**

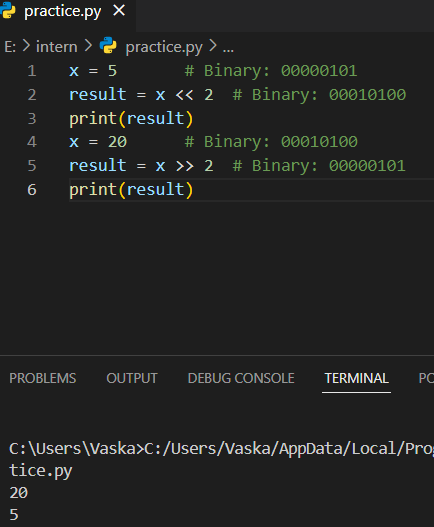
**Shift operators** are used to shift the bits of a number left or right. These operators work at the binary level, where numbers are represented as a sequence of bits (0s and 1s). There are two types of shift operators:

**Left Shift (<<)**

The left shift operator shifts the bits of the number to the left by a specified number of positions. Each left shift operation effectively multiplies the number by 2 for every position shifted.

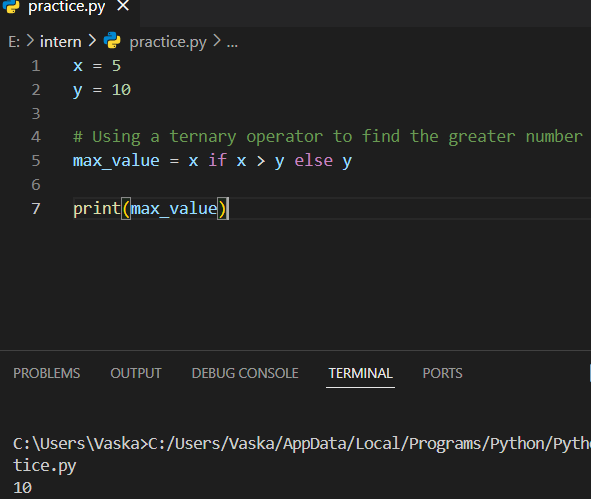
**Right Shift (>>)**

The right shift operator shifts the bits of the number to the right by a specified number of positions. Each right shift operation effectively divides the number by 2 for every position shifted, discarding the remainder.



**Ternary Operator (Conditional Expressions):**

**Ternary operator** is a way to write conditional expressions in a single-line format. It allows us to return a value based on a condition, similar to an if-else statement, but in a compact form.



**Bitwise Operator:**

The operation which is performed in bit level is called bitwise operator.

There are three bitwise operators:

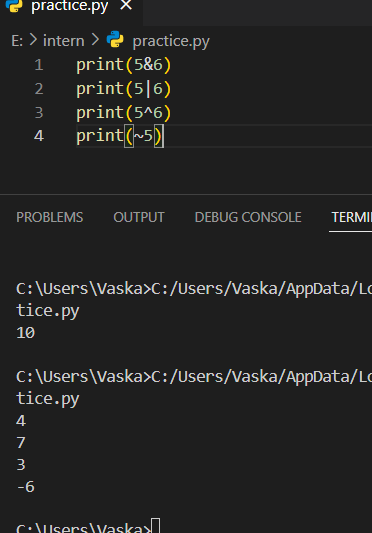
* Bitwise and: &
* Bitwise or: |
* Bitwise X-OR: ^
* Bitwise not: ~

Bitwise operator must be operated in int or bool and throws error when performed in any other data type.

* Bitwise and: If both bits are 1 then result is 1 else 0.
* Bitwise or: If any one bit is 1 the result is 1 else 0.
* Bitwise X-OR: If bits are different result is 1 else 0.
* Bitwise Not: If 1 then result is 0 else 1.

To do bitwise complement, we need to remember two things:

* If MSB is 1, it is negative. If MSB is 0, it is positive.
* Positive number will be represented directly in memory whereas negative number will be represented in 2’s complement form.



**Python Inputs and Outputs**

In Python, input and output operations are essential for interacting with users or handling data.

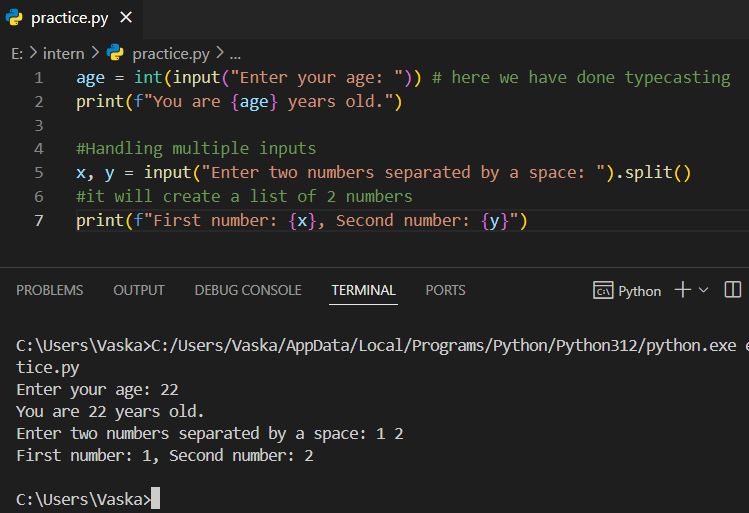
**Python Inputs:**

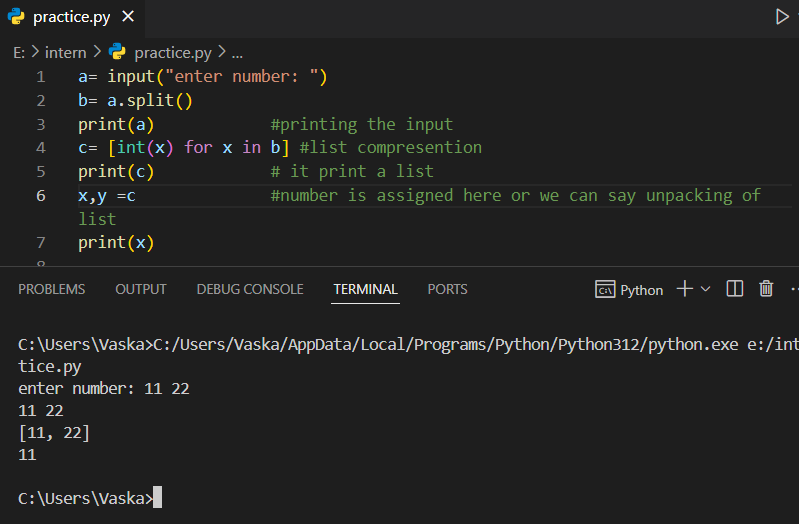
The input() function in Python allows the user to tak data while the program is running. The input() function always returns the user's input as a **string**. If we need it as an integer, float, or another type, you need to convert it by typecasting.

Example: A= input(“enter a number”)

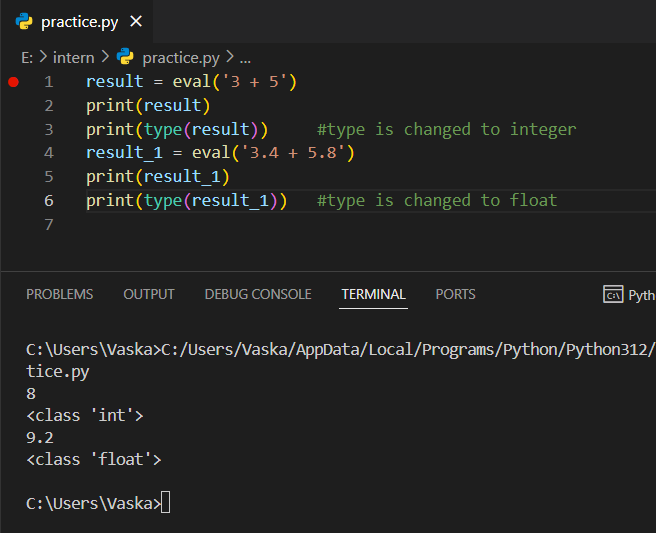
In python2 there were 2 inputs : Raw\_input() and input(). Raw\_input() used to return a input as string by default. Now in python3 we use raw\_input as input() only. And input() in python2 converts the input as its type.

We can take multiple inputs from the user in a single line using the split() method, which splits the input string into a list of items.





**Eval()** : The eval() function in Python is used to evaluate a string as a Python expression and return the result. It's a powerful tool, but it should be used with caution due to potential security risks. It converts the type as necessary from the default type. Typecasting is automatic here.

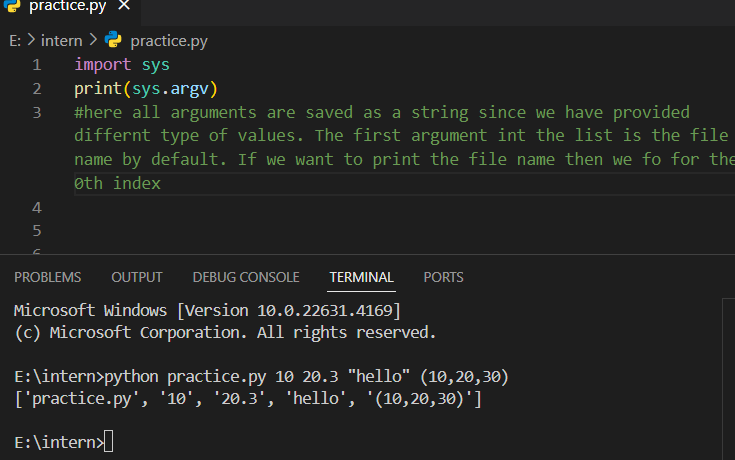


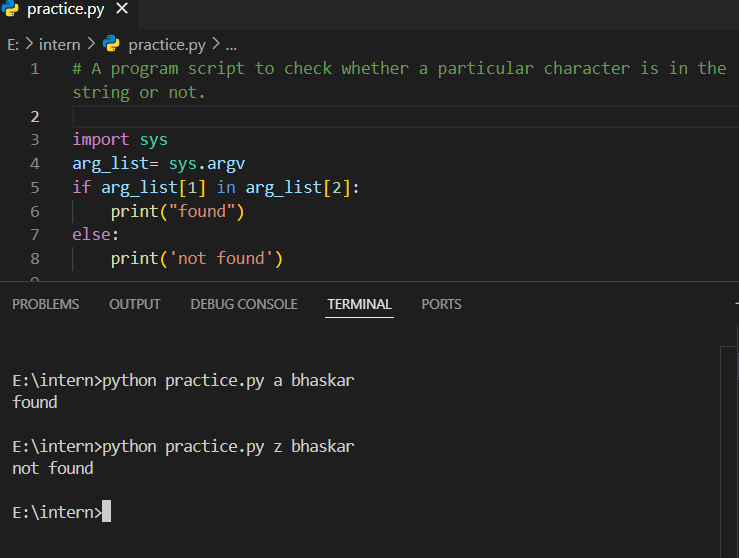
**Command line argument**

Command-line arguments allow us to pass information to a Python script when we run it from the terminal or command prompt. This feature enables scripts to be more flexible and dynamic by allowing input values to be provided at runtime.

We can access command-line arguments using the sys module, specifically sys.argv, which is a list that contains the command-line arguments passed to the script.

Syntax: python\_filename.py arg1 arg2 arg3





**Argpase**

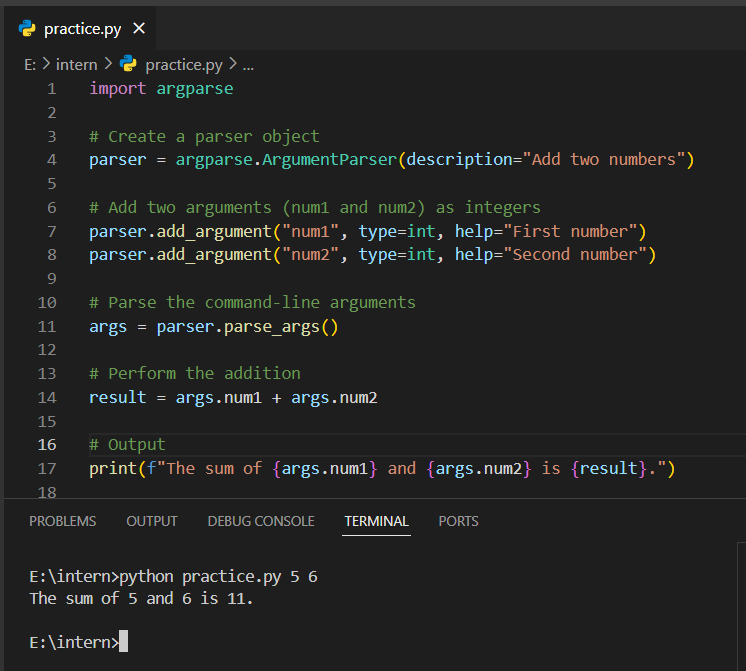
The argparse module in Python is used for handling command-line arguments more robustly and efficiently. It allows us to define arguments that our script expects and automatically handles help messages, argument parsing, type checking, and default values.

**Why Use argparse?**

* **More control**: We can specify required/optional arguments, types, and default values.
* **Automatic help**: It generates helpful messages for users on how to use the script.
* **Error handling**: It provides built-in error handling for incorrect usage or arguments.

Here is one example:

* argparse.ArgumentParser() creates the parser for handling command-line arguments.
* parser.add\_argument() is used to define what arguments the script expects (in this case, two integers: num1 and num2).
* parser.parse\_args() parses the command-line arguments and stores them in args.
* The parsed values can be accessed using args.num1 and args.num2.

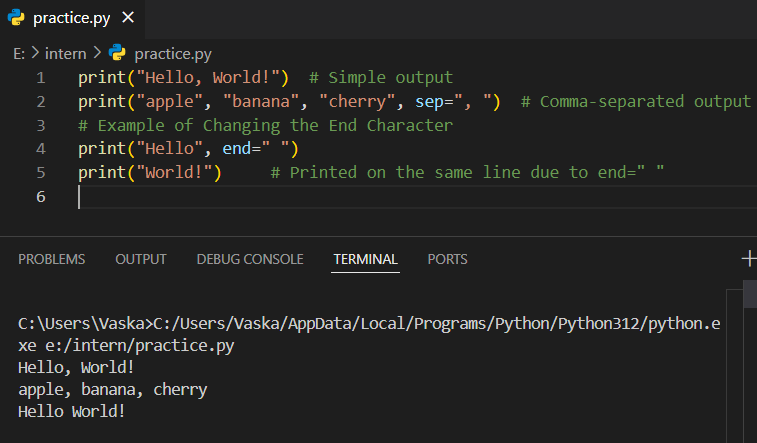


**Python Output Statement**

Output statements are used to display data or messages to the user or to other parts of the program. The primary way to output data in Python is using the print() function.

Eg: print("Hello, World!") output: Hello, World!

We can customize the behavior of print() by changing the separator (sep) between values or the end character (end).



**Formatted Output:**

Using f-strings (formatted string literals) : In Python 3.6, f-strings allow expressions to be embedded inside string literals using curly braces {}.

Using str.format():This method uses placeholders ({}) to insert values into the string.

Using % Formatting: This is an older method that uses the % operator to insert values into a string.

