Contents

[1. Keywords 2](#_Toc167792183)

[2. Basic Data Types 2](#_Toc167792184)

[3. Variable and Input 5](#_Toc167792185)

[4. Logic and Comparison Operators 12](#_Toc167792186)

[5. Conditional Statement 14](#_Toc167792187)

[6. Loops 14](#_Toc167792188)

# Keywords

Keywords are predefined, reserved words used in programming that have their special meanings. Keywords are part of a syntax. They are reserved for specific purpose in the language. Currently there are 36 keywords in python. Python Keywords cannot be used as identifiers (name given to a variable or a function). If we use keyword as an identifier then python will throw syntax error.

Here, is a list of some python keywords.

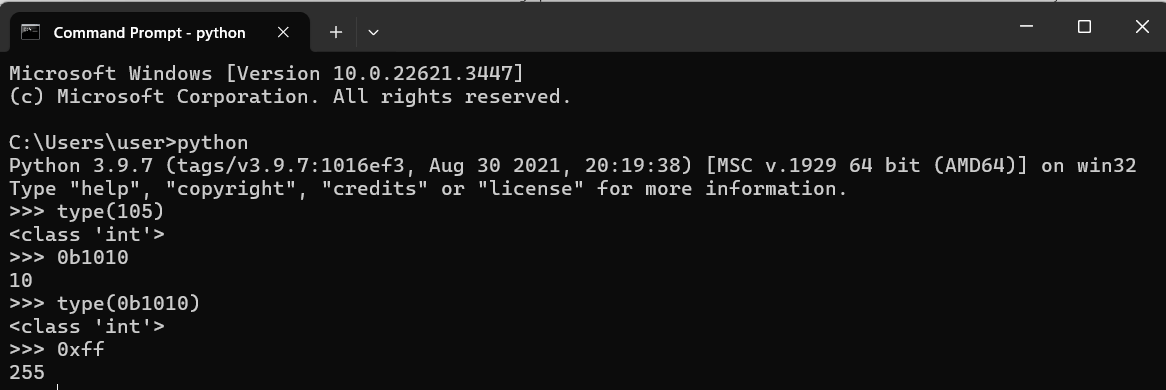
|  |  |
| --- | --- |
| **Keywords** | **Remarks** |
| if, elif, else, for, while, break, continue, pass | Used in control flow structures |
| def, class | Used in function and class definition |
| try, except, finally, raise | Used in exception handling |
| global, nonlocal | Used for namespace and scope |
| in | Used in Iterators |
| True, False, and, or, not | Used for Boolean and logical operations |
| del, return, as | Used in object creation and management |
| import, from | Used in import and module Management |

# Basic Data Types

Data types are the fundamental building blocks of programming language. Data type is a classification that specifies which type of value a variable can hold and what type of operation can be performed on the variable. Python has a dynamic type casting, which means that we don’t need to explicitly specify the data type of a variable when we declare it. Python infers the data type based on the value assigned to the variable. Python data types are classes and variables are the object of these classes.

Basic data types in python are:

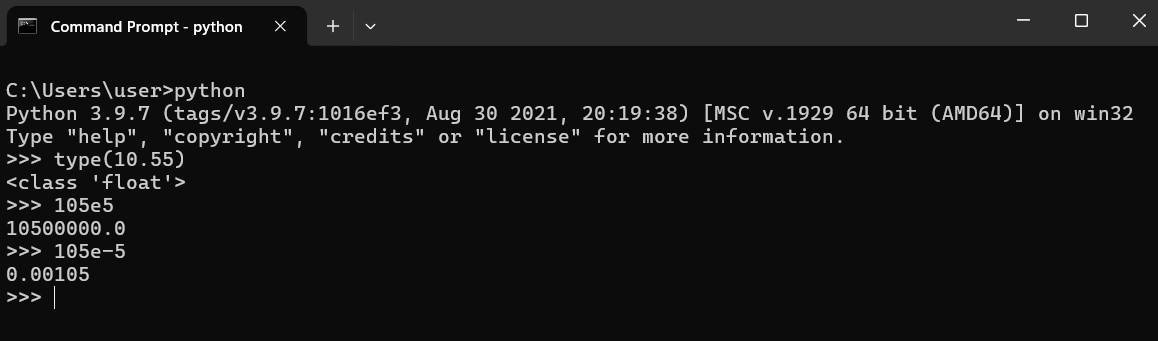
1. Integers: They are represented by int class. It contains positive and negative whole numbers without fractions. In python there is no limit to how long an integer value can be. In python a non-decimal integers like binary, octal and hexadecimal can be represented by adding a prefix 0B or 0b, 0O or 0o, 0X or 0x respectively.



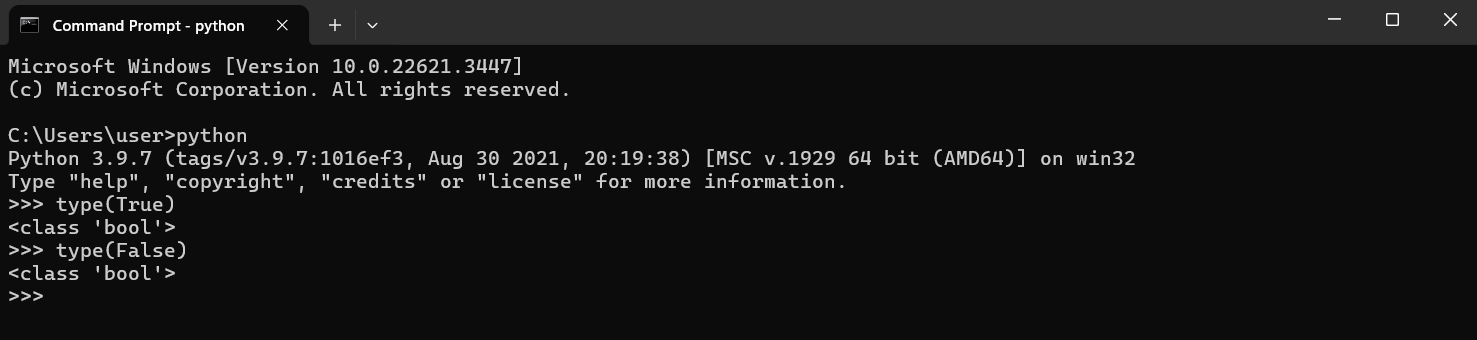
*\*Note: type() is a function to determine the type of data type in python*

Here, we have used a python interpreter in command on windows operating system. When we use type() to determine the type of a number 105 the interpreter prints the type as int. Everything in python is class so the interpreter prints the output as class ‘int’ since the value belong to integer class. 0b represents a binary number. So now 1010 is a binary number and the interpreter outputs the decimal value for 1010 binary number. Similarly, 0xff represent a hexadecimal number ff and the interpreter outputs the corresponding decimal value.

1. Floating-point: Floating point values are represented by float class. It contains real numbers with decimal point. Optionally, e or E is added at the end of a floating-point number to specify the scientific notation.

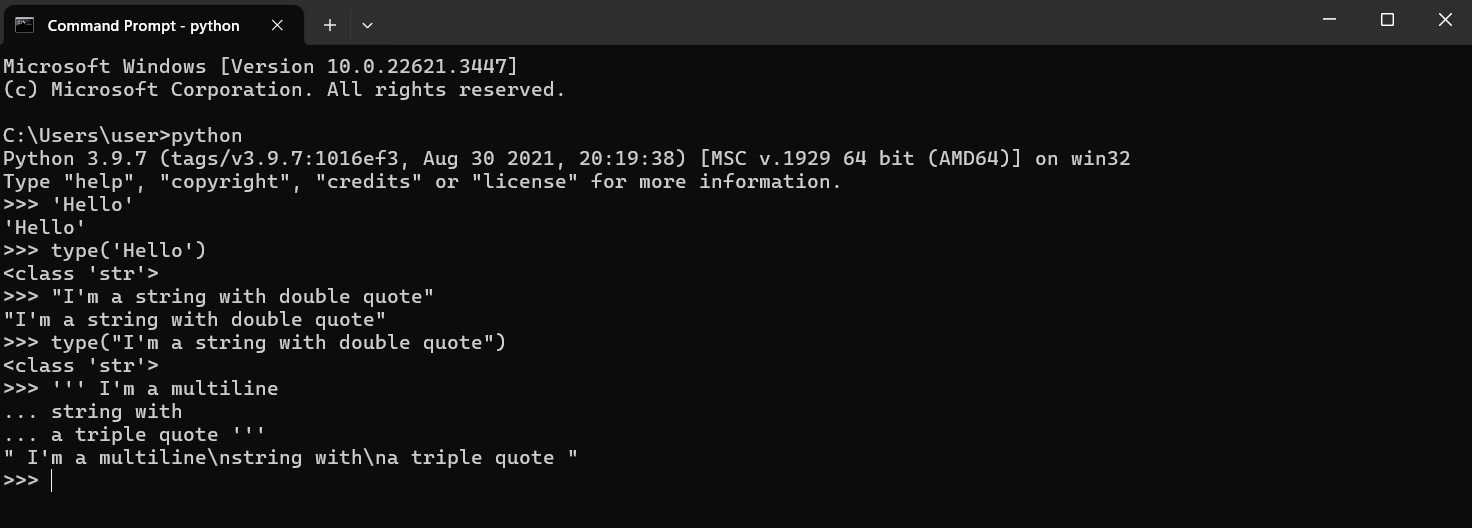


1. Boolean: Boolean data types are represented by bool class. They have two values true and false. Non- Boolean objects can be evaluated in the Boolean context as well and determined to be true or false.



*\*Note: True has ‘T’ capital. Interpreter throws an error if true or false is used.*

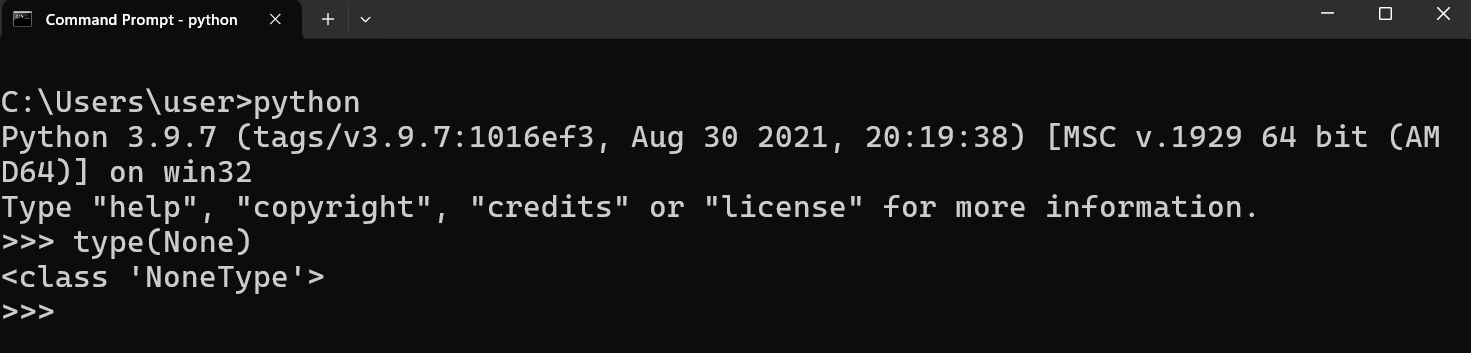
1. String: A string data type is represented by str class. String is a collection of one or more character put in a single quote ('Hello'), double quote ("I'm a string with double quote") and triple quote (''' I'm a multiline string with a triple quote ''').



Here, A string 'Hello' was created using single quote. "I'm a string with double quote" is a string created using double quote, the string begins with " and ends with " and everything inside the double quote is string data. ''' I'm a multiline string with a triple quote ''' is created using triple quote, the string begins with ''' and ends with '''. Triple quote can be used to create a multi-line string as shown in the above terminal.

*\*note """ Triple quote string """ is also a valid string as we can use """ """ to create a string.*

1. None: It is represented by NoneType object. None is a special constant representing the absence of a value or a null value.



*\*Note: None has ‘N’ capital. Interpreter throws an error if none is used.*

# Variable and Input

Variables are used to stores data such as integers, floats, strings in python. In python there is no need of declaring the variables. Variable are created when first value is assigned to it. Python is dynamically typed i.e. typed is inferred based on the value assign to it.

Rules for naming a variable in Python:

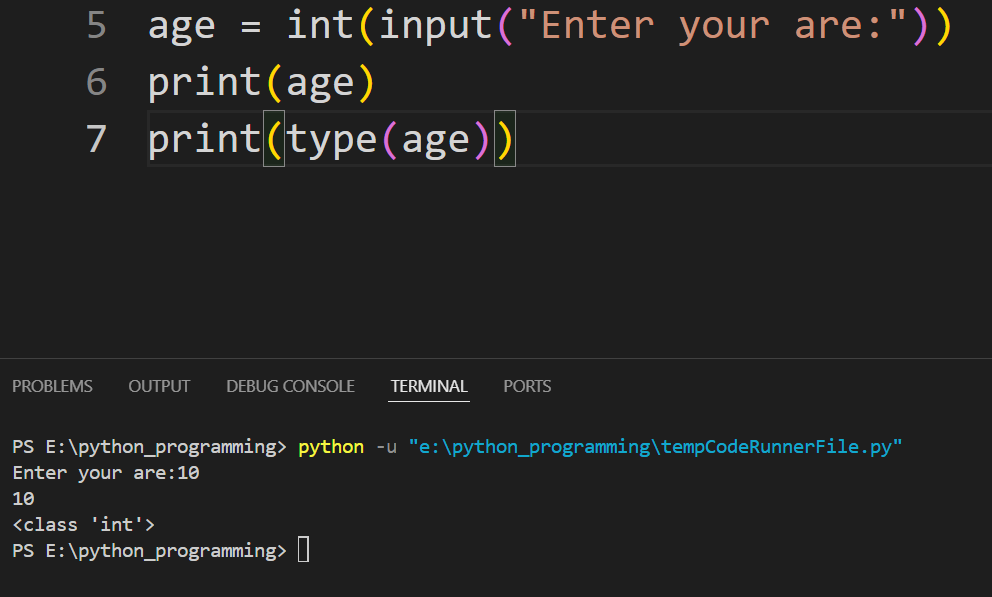
* Variable name can contain letters, digits and underscore (\_).
* Variable name cannot start with digit.
* Variable name cannot be keyword.

Here, x holds an integer value 10, y holds a float value 10.55, and z holds a string value "Hello".

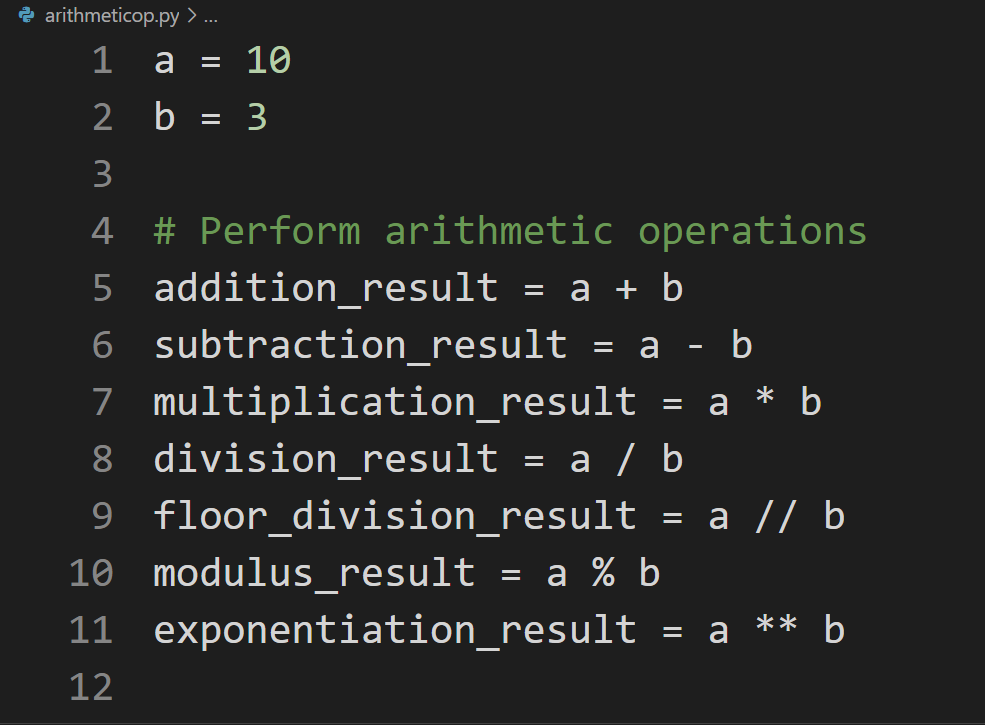
x is integer variable; y is float variable and z is a string variable. We now can perform respective operations based on the datatype of the variable.

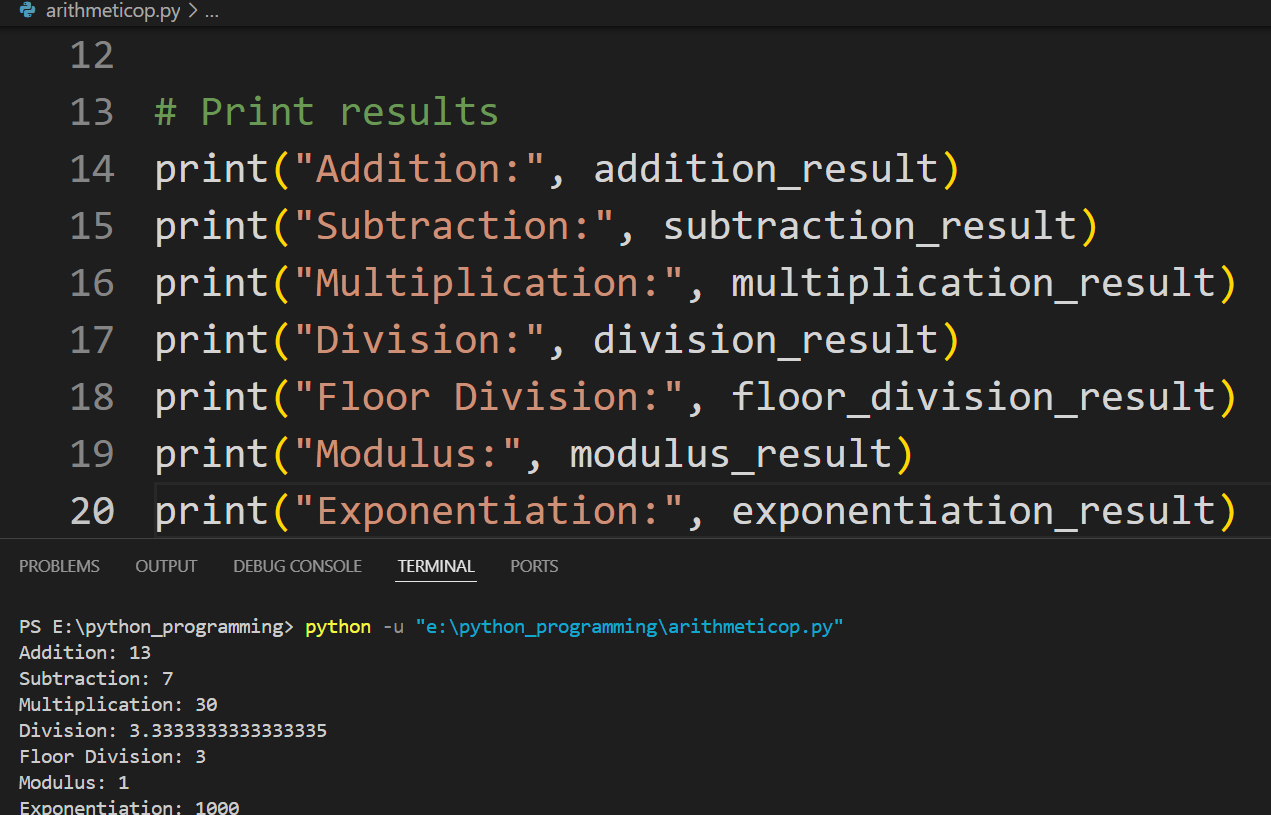
In python we can take user input using input() function. This function reads a line from a keyboard and it returns a string.

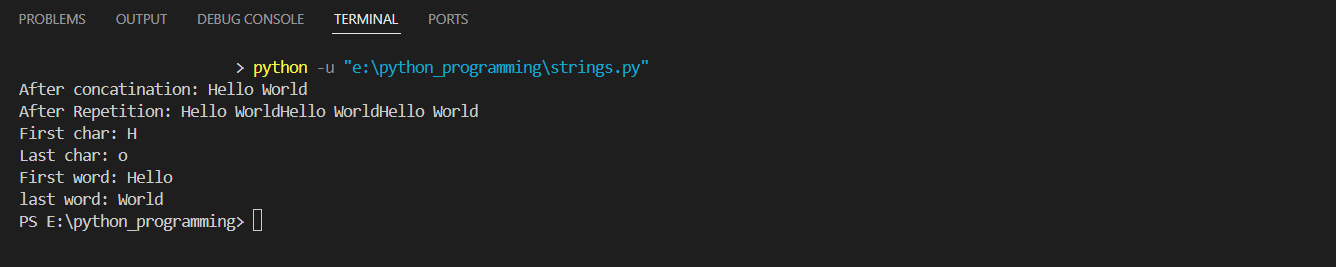
If we want to input to be of other type then we have to convert it accordingly. If we want to convert it to integer then we can do it using int() method. Here, int() is a function that is used to convert a value to an integer.



Basic Arithmetic operators in python:

* Addition (+), Subtraction (-), Multiplication (\*), Division (/), Floor Division (//), Modulus (%), Exponentiation (\*\*)



Working with strings in python:

The provided code demonstrates several fundamental string operations in Python. Initially, it concatenates two strings, str1 ("Hello") and str2 ("World"), with a space in between, resulting in result1 ("Hello World"). It then repeats this concatenated string three times to form result2 ("Hello WorldHello WorldHello World").

Indexing allows you to access individual characters in a string based on their position. Python uses zero-based indexing, meaning the first character of a string is at index 0. str1[0]: Accesses the first character of str1, which is "H". str1[-1]: Uses negative indexing to access the last character of str1, which is "o". Negative indexing counts from the end of the string, with -1 being the last character, -2 being the second last, and so on. Slicing allows you to extract a substring from a string by specifying a start and end index. The syntax is string[start:end], where the start index is inclusive and the end index is exclusive.

result1[:5] : Slices the string from the beginning up to, but not including, index 5. This extracts the first five characters of result1, which are "Hello".

result1[6:] : Slices the string from index 6 to the end of the string. This extracts the substring starting from the 7th character to the end, which is "World".

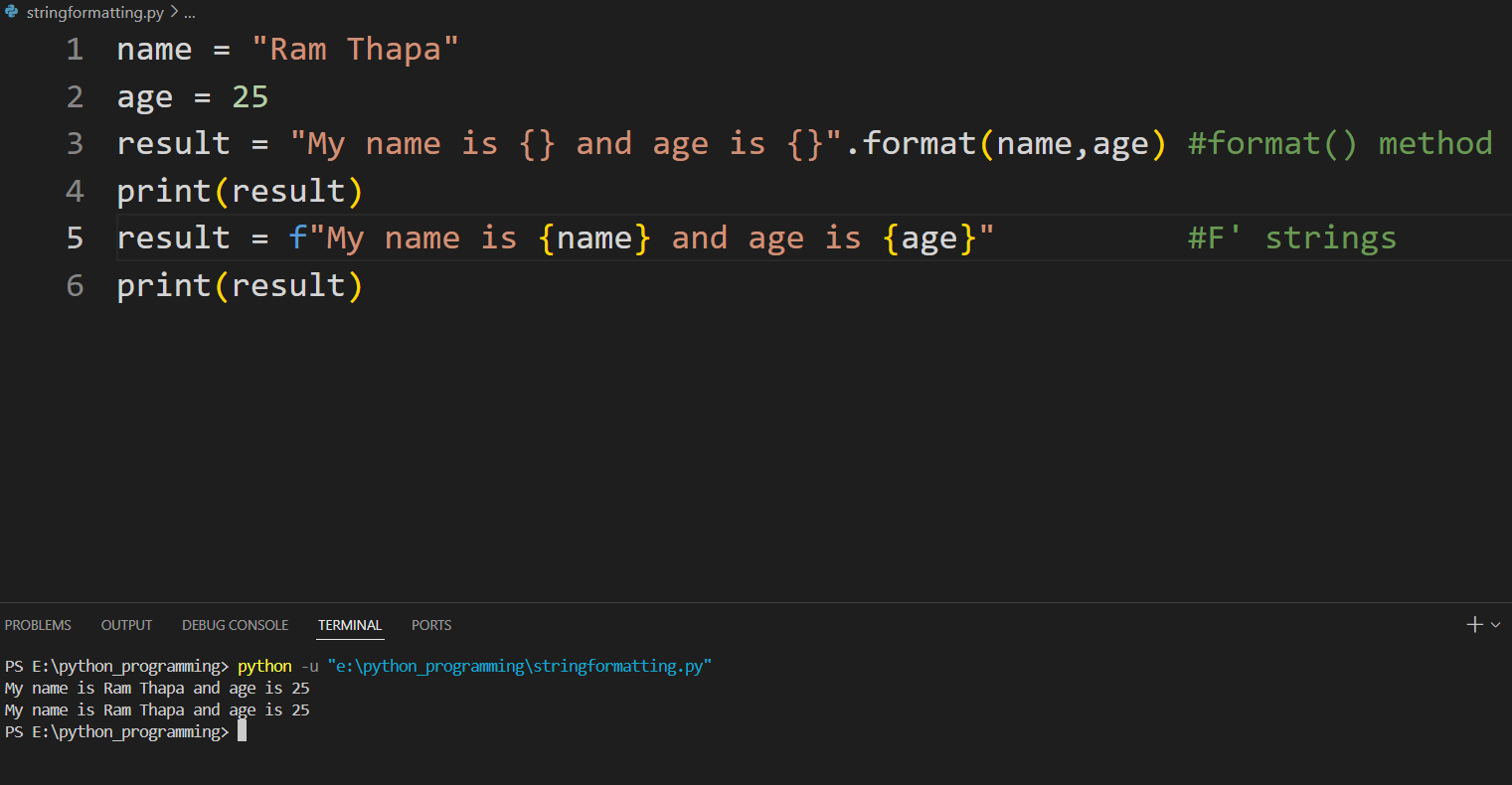
Working with string functions:

The upper() method converts all characters in a string to uppercase, while lower() does the opposite by converting all characters to lowercase. The split() method splits a string into a list of substrings based on a specified separator (in this case, ,). strip() removes leading and trailing whitespace characters from a string. Finally, replace() replaces occurrences of a specified substring within a string with another substring.

For instance, str1.upper() converts the string "Hello" to uppercase, yielding "HELLO", while str1.lower() does the opposite, resulting in "hello". str3.split(",") splits the string "Hello,namaste,nepal" into a list of substrings based on the comma separator, resulting in ["Hello", "namaste", "nepal"]. str4.strip() removes leading and trailing whitespace characters from the string " Hello ", resulting in "Hello". Lastly, str5.replace("Nepol", "Nepal") replaces the substring "Nepol" with "Nepal" in the string "Hello Nepol", producing "Hello Nepal".

String Formatting:

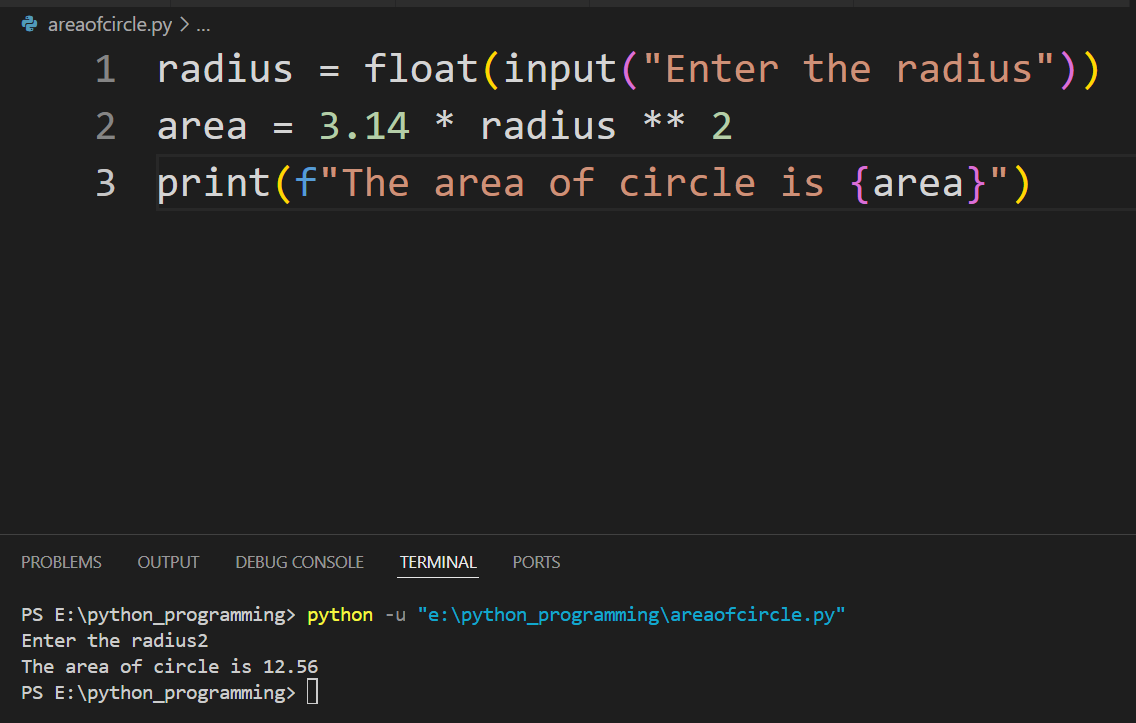
We will discuss two types of formatting.

1. format() method
2. F-Strings (Python 3.6+)

The first method discussed is the format() method. With this approach, you construct a string template containing placeholders ({}) to indicate where variables or expressions should be inserted. Then, you call the format() method on the template string, passing the variables or expressions as arguments. Inside the placeholders, you can specify the order of variables or use numbered placeholders for explicit positioning.

f-strings provide a more concise and intuitive way to format strings by allowing you to directly embed variables and expressions within curly braces {} within the string literal. This is achieved by prefixing the string literal with an 'f' or 'F'. With f-strings, there's no need to explicitly call a method like format(); instead, the variables or expressions within the curly braces are automatically evaluated and replaced with their values at runtime. This results in more readable and maintainable code, especially when dealing with complex string formatting scenarios.

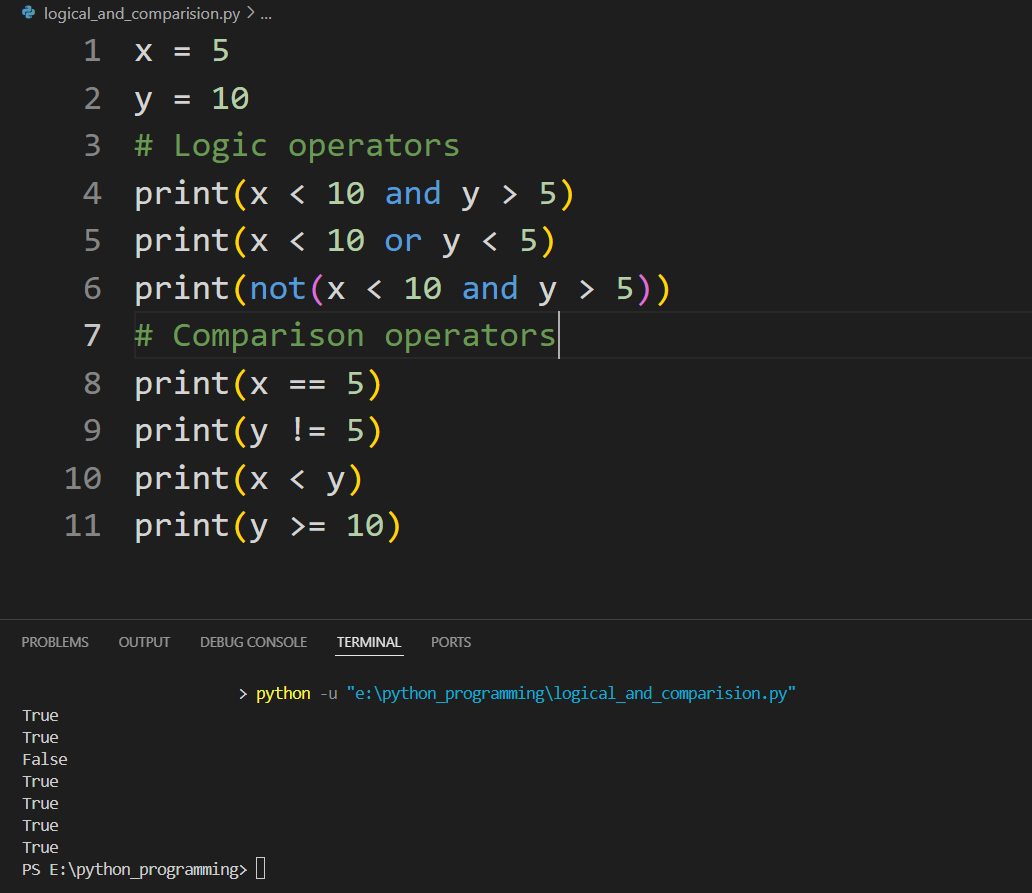
In the first example, result = "My name is {} and age is {}".format(name, age), the format() method replaces the curly braces {} in the template string with the values of name and age, resulting in "My name is Ram Thapa and age is 25". In the second example, result = f"My name is {name} and age is {age}", f-strings allow you to directly embed variables and expressions within curly braces {} within the string literal, making the code more concise and readable. Both techniques offer powerful ways to format strings in Python, catering to different preferences and requirements.

Q. WAP in python to input the radius of circle and output “The area of circle is ………”

It first prompts the user to input the radius of the circle using the input() function, which returns a string. The float() function is then used to convert the input string to a floating-point number, ensuring that the radius can be a decimal value. Next, it calculates the area of the circle using the formula area = 3.14 \* radius \*\* 2, where radius is the user-provided value. Finally, the print() function displays the result using an f-string, where the curly braces {} are replaced with the value of the area variable, resulting in a message like "The area of the circle is 78.5" if the radius input by the user was 5. This code effectively demonstrates how to receive user input, perform a calculation, and display the result in a formatted message in Python.

# Logic and Comparison Operators

Logic operators evaluate logical expressions and return Boolean values (True or False), while comparison operators compare two values and return a Boolean result based on the comparison.

* Logical Operators (and , or, not)
* Comparison Operators (==, !=, <, >, <=, >=)

Here, several logic and comparison operators are applied to variables x and y. For logic operators, the and operator evaluates to True for the first print statement because both conditions, x < 10 (True) and y > 5 (True), are satisfied. Similarly, the or operator in the second print statement evaluates to True because at least one condition is True (x < 10). The not operator in the third print statement negates the result of the condition x < 10 and y > 5, which is True, resulting in False. Regarding comparison operators, x == 5 is True because x is equal to 5, y != 5 is True because y is not equal to 5, x < y is True because 5 is less than 10, and y >= 10 is True because 10 is greater than or equal to 10. These results demonstrate how logic and comparison operators function in Python, enabling developers to create conditional expressions and make decisions based on the evaluated conditions.

In this example, the code compares two string variables name1 and name2, which contain "Ram" and "Hari" respectively. The comparison operators check if name1 is equal to "Ram" (name1 == "Ram") and if name2 is not equal to "Ram" (name2 != "Ram"). Additionally, the less than operator (name1 < name2) compares the strings lexicographically, resulting in True because "Ram" comes before "Hari" alphabetically. For logic operators, startswith() verifies if both names start with the respective letters "R" and "H", returning True. endswith() checks if at least one name ends with the substring "am", yielding True for name1. Finally, the not operator negates the expression name1 == "Ram" and name2 == "Ram", indicating True because both names are not equal to "Ram".

# Conditional Statement

Conditional statements are fundamental for controlling the flow of a program and implementing decision-making logic. Python supports several types of conditional statements:

* if Statement: The if statement evaluates a condition and executes the block of code within it if the condition is true.
* if-else Statement: The if-else statement provides an alternative block of code to execute if the condition in the if statement is false.
* if-elif-else Statement: The if-elif-else statement is used to check multiple conditions. The first block of code with a true condition is executed, and the rest are skipped.

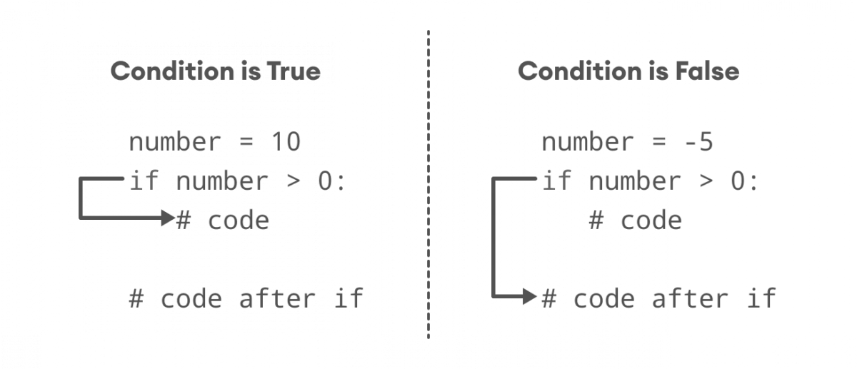
**if Statement**

Syntax:

if condition:

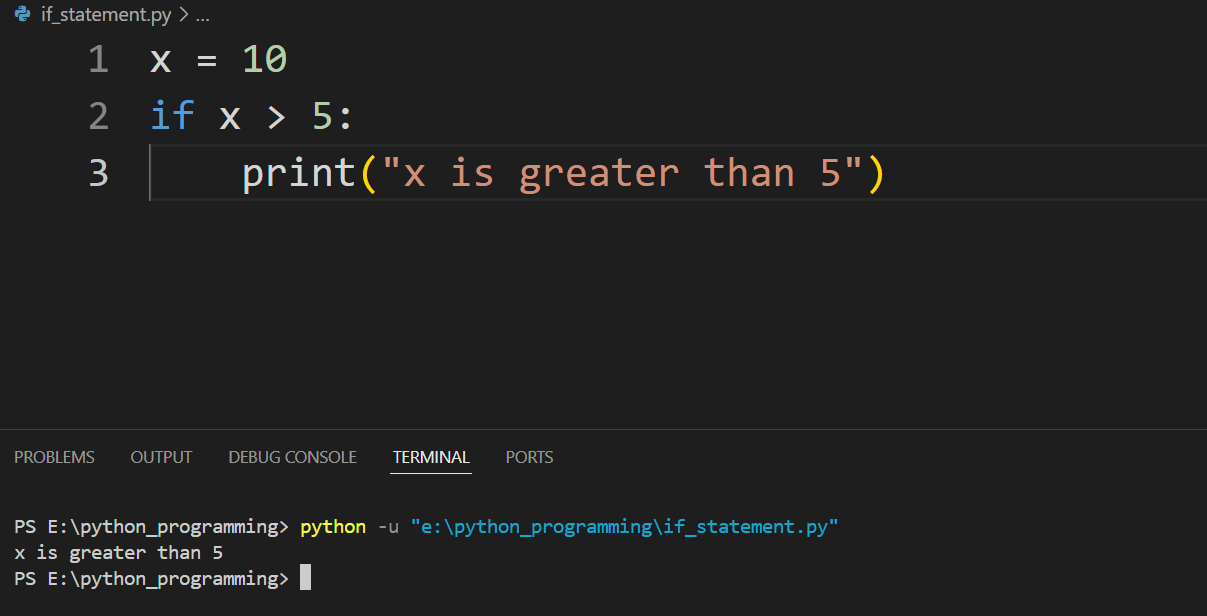
# body of if statement

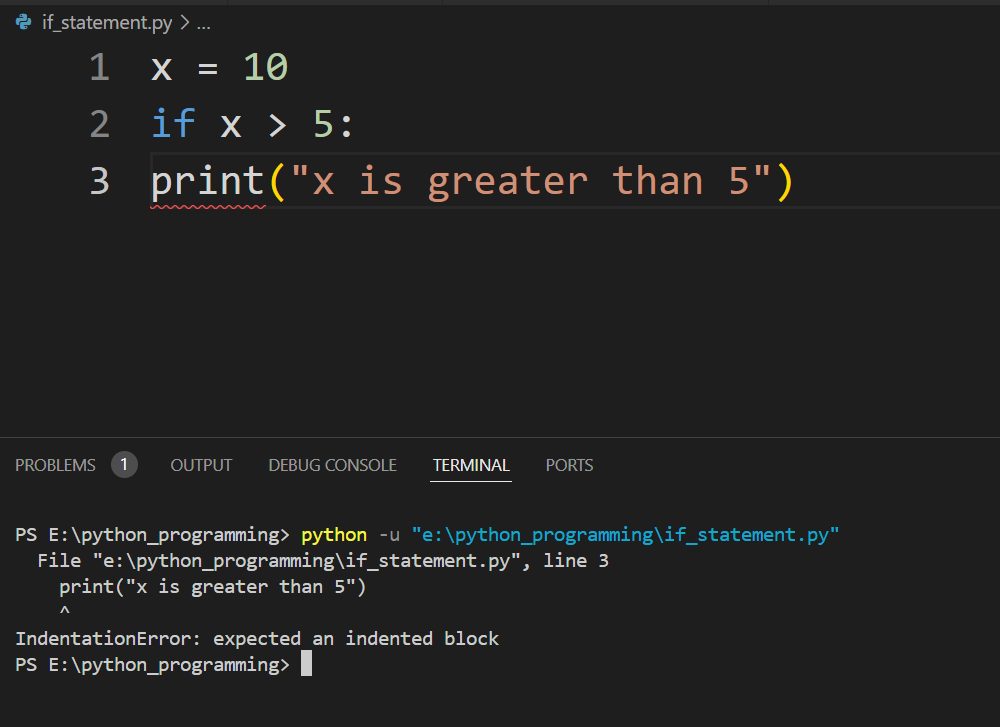
Here, if the condition of the if statement is:

* True - the body of the if statement executes.
* False - the body of the if statement is skipped from execution.

**Indentation: Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.**

Here is a basic if statement in Python. It starts by assigning the value 10 to the variable x. The if statement then checks if x is greater than 5 (x > 5). Since this condition is true, the indented block of code within the if statement is executed.

****

****In Python, the code block inside an if statement must be indented. The lack of indentation here will result in an IndentationError.

**if-else Statement:**

Syntax:

if condition:

# body of if statement

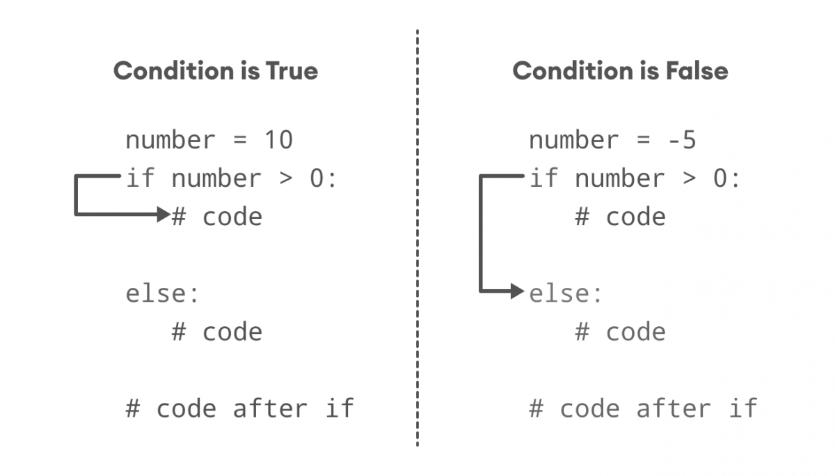
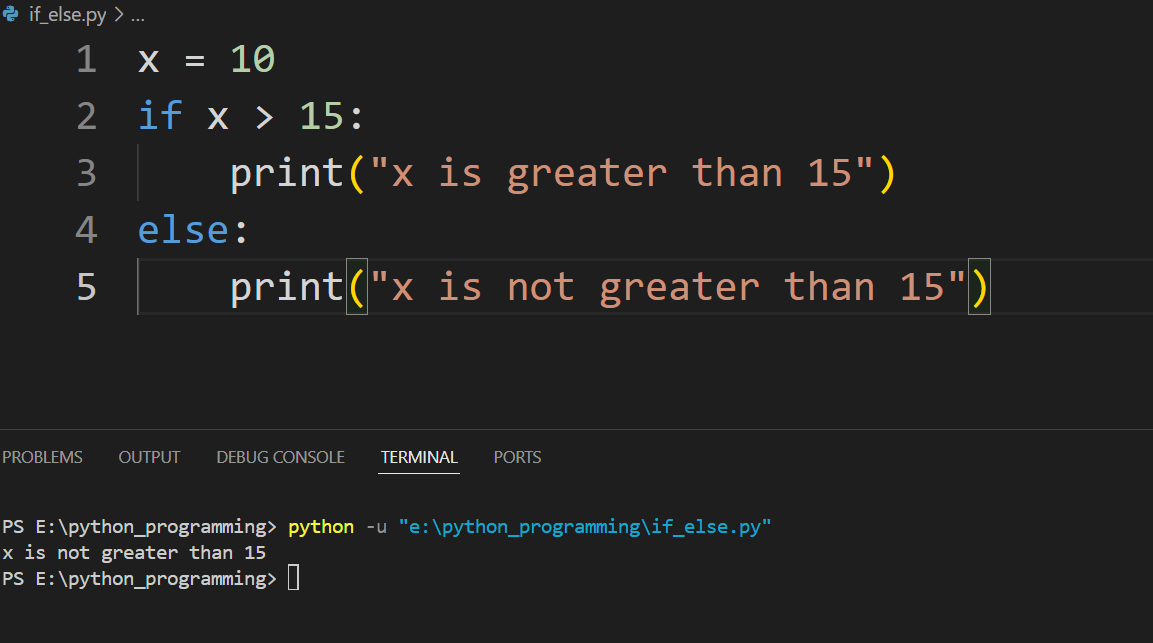
else:

# body of else statement

# 

Here, if the condition inside the if statement evaluates to

* True - the body of if executes, and the body of else is skipped.
* False - the body of else executes, and the body of if is skipped

Here, if-else statement to check whether the variable x is greater than 15. Initially, the variable x is assigned the value 10. The if statement evaluates the condition x > 15. Since the condition is false, so the code inside the if block (print("x is greater than 15")) is skipped. Instead, the else block is executed, printing "x is not greater than 15" to the console.

**if-elif-else Statement:**

Syntax:

if condition1:

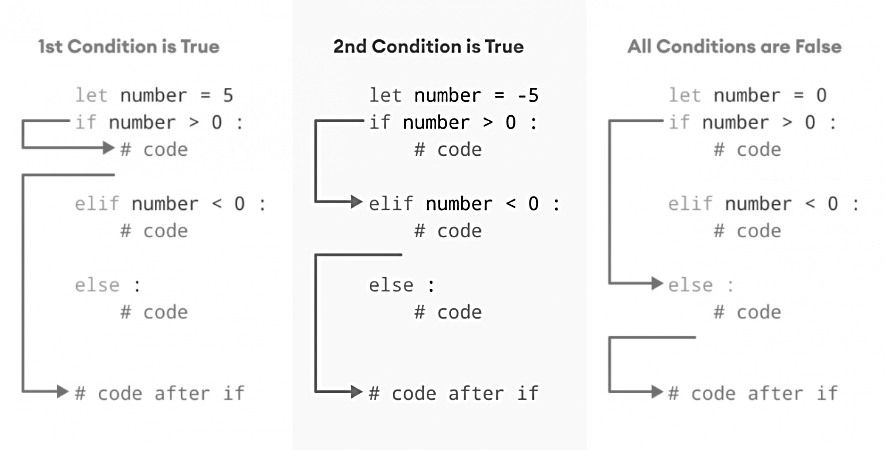
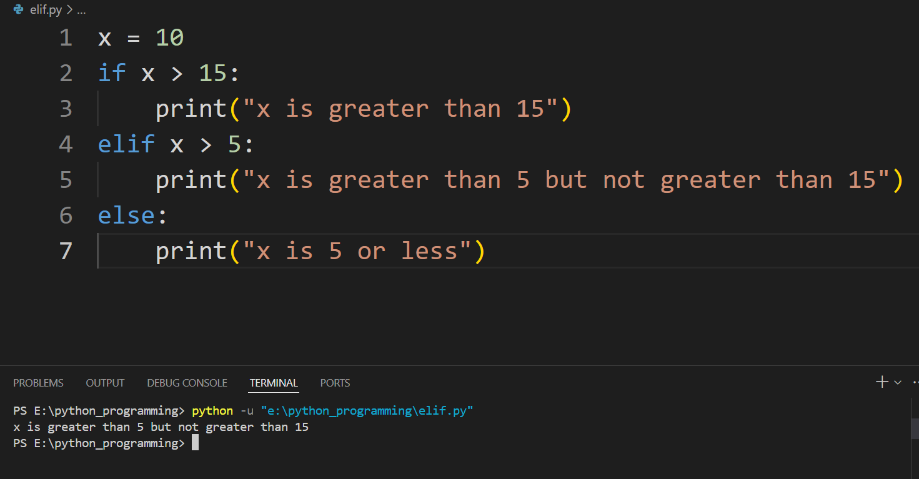
# code block 1

elif condition2:

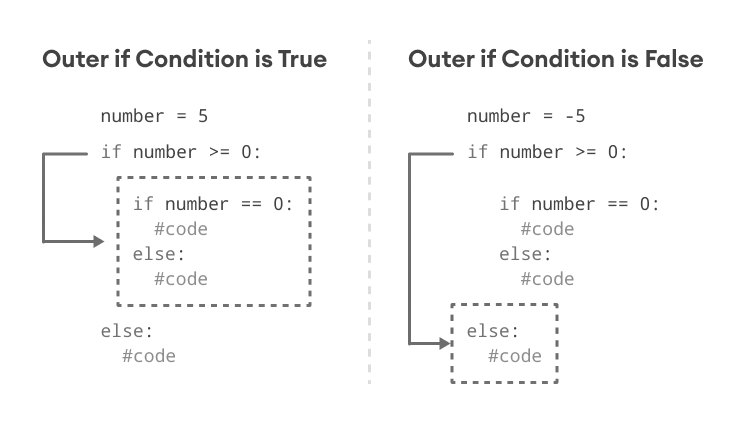
# code block 2

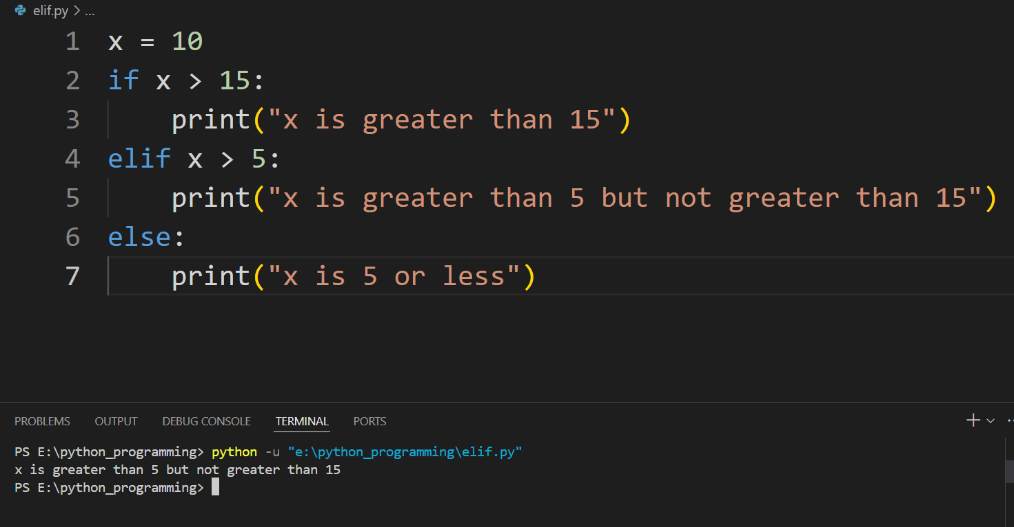
else:

# code block 3

Here the code is using an if-elif-else statement to evaluate multiple conditions for the variable x, which is assigned the value 10. The if statement first checks if x is greater than 15. Since this condition is false, so the code moves to the elif statement. The elif statement checks if x is greater than 5 this condition is true, and the code inside the elif block (print("x is greater than 5 but not greater than 15")) is executed. The else block is ignored because an earlier condition was met.

**Nested if Statements:**

It is possible to use conditional statements inside one another to check multiple conditions at different levels. This is known as nested statements.



# Loops

Python supports two main types of loops.

* for loop: for loops iterate over a sequence (such as a list, tuple, string, or range) and execute a block of code for each item in the sequence.
* while loop: while loops continue to execute a block of code as long as a specified condition is true.

Flowchart of Python for Loop:

