**1)Arithmatic operaters**

* + **Addition (+)**: Adds num1 and num2.
  + **Subtraction (-)**: Subtracts num2 from num1.
  + **Multiplication (\*)**: Multiplies num1 by num2.
  + **Division (/)**: Divides num1 by num2 and returns a float.
  + **Modulus (%)**: Returns the remainder of num1 divided by num2.
  + **Exponentiation (\*\*)**: Raises num1 to the power of num2.
  + **Floor Division (//)**: Performs division but returns the largest integer less than or equal to the result.

1. **Outputs:**
   * The code prints the results of each operation to the console.

**2)Comparison operators**

* + The program uses conditional statements (if-else) to compare the two numbers:
    - **Greater than (>)**: If x > y, it prints that x is greater than y.
    - **Less than (<)**: If x < y, it prints that y is greater than x.
    - **Equal to (==)**: If neither x > y nor x < y is true, it concludes that x is equal to y and prints this.

1. **Outputs:**
   * The code outputs one of three possibilities:
     + x is greater than y
     + y is greater than x
     + x is equal to y

**Code Strengths**

* **Simplicity and Readability**: The code is straightforward and easy to follow, making it ideal for beginners to understand comparison operators in Python.
* **Basic Functionality**: It efficiently handles three possible scenarios (greater than, less than, equal to) with clear conditional logic.
* **Immediate Feedback**: The user receives immediate feedback on the relationship **Error Handling**:
  + **Invalid Input**: Currently, the code does not handle invalid input (such as non-integer input), which could raise a ValueError. Adding a try-except block would make the code more robust.

#### 3)Logical Operator Behavior:

1. **and Operator**:
   * **Function**: The and operator checks multiple conditions and returns the first falsy value it encounters. If all values are truthy, it returns the last value.
   * **Behavior in the code**: Since the input values are expected to be non-zero (which are considered truthy), the result will be the last input value (c).

Example:  
If a = 10, b = 20, and c = 30, the output will be 30 because all values are truthy.

1. **or Operator**:
   * **Function**: The or operator returns the first truthy value it finds. If all values are falsy, it returns the last value.
   * **Behavior in the code**: Since the input values are non-zero (truthy), the result will be the first input value (a).

Example:  
If a = 10, b = 20, and c = 30, the output will be 10 because a is the first truthy value.

1. **not Operator**:
   * **Function**: The not operator returns True if the value is falsy and False if the value is truthy.
   * **Behavior in the code**: Since all input values are non-zero (truthy), the not operator will return False for each of the values (a, b, c).

**4) string manipulation**

**Key Operations:**

1. **Reverse String ([::-1])**: Reverses the string from the last character to the first.
2. **Slice String ([3:])**: Extracts the substring starting from the 4th character (index 3) till the end.
3. **Uppercase Conversion (upper())**: Converts the entire string to uppercase.
4. **Lowercase Conversion (lower())**: Converts the entire string to lowercase.

**Approach:**

* **String Slicing**: Used to reverse or extract parts of the string.
* **Built-in Methods**: upper() and lower() are standard string methods for case transformation.

**Use Cases:**

* Reverse checking (e.g., palindromes).
* Extracting specific parts of a string.
* Normalizing text for comparisons or data consistency.

**5)substring search**

**Key Operations:**

1. **Input Sentence (s)**: Captures a sentence from user input.
2. **Input Substring (sub)**: Captures the substring (word) to search for within the sentence.
3. **Substring Check (if sub in s)**: Checks if the substring exists in the sentence.
4. **Output Result**: Prints whether the substring was found or not.

**Approach:**

* **Membership Operator**: Utilizes the in keyword to check for the presence of the substring in the sentence, which is efficient and straightforward.
* **Conditional Statements**: Employs an if statement to control the flow based on the presence of the substring.

**Use Cases:**

* Simple text search functionality for applications like search bars or keyword checks in documents.
* Can be expanded for more complex search functionalities, such as case-insensitive **searches or finding multiple occurrences.**

**6)String formatting**

**Key Operations:**

1. **Input Name (name)**: Captures the user's name as a string.
2. **Input Age (age)**: Captures the user's age and converts it to an integer.
3. **Formatted Output**: Uses format() to create a personalized greeting that includes the user's name and age.

**Approach:**

* **Input Handling**: Utilizes input() to gather user data for both name and age.
* **String Formatting**: Employs the format() method for embedding variables into strings, enabling clean and readable output.

**Use Cases:**

* Basic user interaction scenarios, such as welcome messages or user profile displays.
* Can be adapted for applications that require user feedback based on input.

This code demonstrates how to collect user input and provide formatted output, creating a personalized greeting that incorporates the user's name and age effectively.

**7)List operation**

**Key Operations:**

1. **Input Conversion to List**: Converts a space-separated string of numbers into a list of integers using map(int, input().split()).
2. **Sum Calculation (sum(list1))**: Computes the total of all numbers in the list.
3. **Maximum Value (max(list1))**: Finds the largest number in the list.
4. **Minimum Value (min(list1))**: Finds the smallest number in the list.

**Approach:**

* **Input Handling**: The input() function captures user input, which is then split into separate strings and converted to integers using map().
* **List Functions**: Utilizes Python's built-in functions (sum(), max(), min()) for efficient calculations on the list.

**Use Cases:**

* Quick calculations for basic statistics (sum, max, min) from user-provided data.
* Suitable for scenarios requiring aggregation or analysis of numerical input.

**8)List manipulation**

**Key Operations:**

1. **Append Item (append("grapes"))**: Adds "grapes" to the end of the list.
2. **Print List**: Displays the current state of the list after the append operation.
3. **Remove Item (remove(list2[1]))**: Removes the item at index 1 ("apple") from the list.
4. **Print List**: Displays the updated state of the list after the removal operation.

**Approach:**

* **List Methods**: Utilizes append() to add items and remove() to delete specific elements based on their index.
* **Indexing**: Accesses elements by their index to facilitate removal.

**Use Cases:**

* Dynamic modification of lists by adding or removing elements.
* Useful in scenarios requiring management of collections of items.

**9)Sorting lists**

**Key Operations:**

1. **Input Conversion to List**: Captures a space-separated string of numbers and converts it into a list of integers using map(int, input().split()).
2. **Sort List (sort())**: Sorts the list in ascending order.
3. **Print Ascending List**: Displays the sorted list after ascending order sorting.
4. **Sort List in Descending Order (sort(reverse=True))**: Sorts the list in descending order.
5. **Print Descending List**: Displays the sorted list after descending order sorting.

**Approach:**

* **Input Handling**: Uses input() to gather user input, which is then split and mapped to integers.
* **List Methods**: Utilizes the sort() method to modify the list in place, enabling sorting in both ascending and descending orders.

**Use Cases:**

* Quick sorting of numerical data for analysis or display purposes.
* Useful in scenarios requiring ordered data for reporting or decision-making.

**10)List slicing**

**Key Operations:**

1. **Original List**: Defines a list list1 containing the integers from 1 to 10.
2. **Slice First Five Elements (list1[0:5])**: Extracts and prints the first five numbers from the list.
3. **Slice Last Five Elements (list1[5:10])**: Extracts and prints the last five numbers from the list.
4. **Slice Middle Elements (list1[2:7])**: Extracts and prints a subset of elements from index 2 to index 6 (the 3rd to 7th numbers).

**Approach:**

* **List Slicing**: Utilizes Python’s slicing feature to obtain specific parts of the list based on start and end indices.

**Use Cases:**

* Extracting portions of data for analysis or display.
* Useful for scenarios where segmented access to list data is required.

**11)Bonus**

**Key Operations:**

1. **Loop for Student Input**: A loop runs three times to collect the names and scores of three students.
   * **Name Input**: Prompts for the student's name.
   * **Scores Collection**: A nested loop collects three scores for each student.
2. **Store Student Data**: Each student's name and their list of scores are stored in student\_data as a nested list.
3. **Average Score Calculation**: For each student, calculates the average score by summing the scores and dividing by the number of subjects.
4. **Print Average Score**: Displays each student's name alongside their calculated average score.

**Approach:**

* **Nested Loops**: Utilizes a nested loop structure for input collection, allowing for grouped data entry (students and their scores).
* **List Operations**: Employs lists to store scores and access them for calculations.