**5. Looping (For, While)**

**Introduction to for and while loops.**

Loops:

**Loops** are essential in programming for automating repetitive tasks. They allow a block of code to execute multiple times without needing to rewrite it

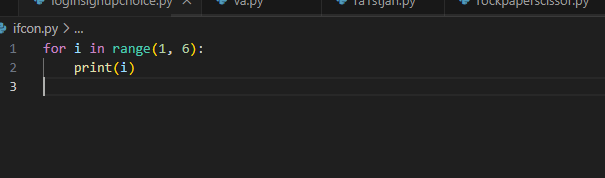
1. For Loop

A for loop is used to iterate over a sequence, such as a list, tuple, dictionary, string, or range of numbers. It automatically stops when the sequence ends.

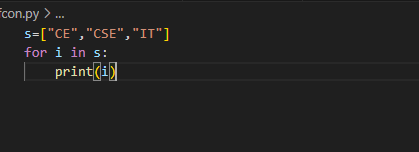
#### **Key Characteristics:**

* Ideal for iterating over a known sequence.
* Simplifies tasks like traversing lists, processing strings, and generating ranges.

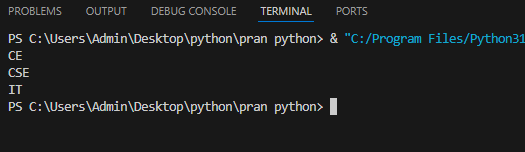
Example:



Looping through strings



Output:



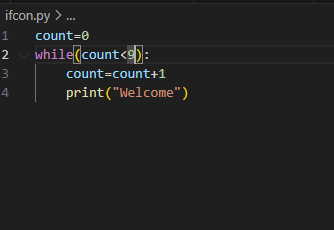
### **2. While Loop**

A while loop continues execution as long as the specified condition is True. It is useful when the number of iterations is not known beforehand.

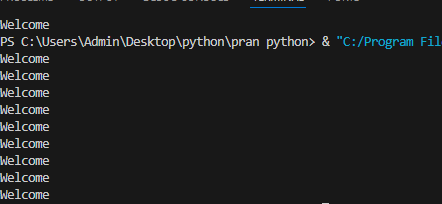
#### **Key Characteristics:**

* Ideal for situations where the loop depends on a condition rather than a sequence.
* Can lead to infinite loops if the condition is never False.

Example:



Output:



How do loops work in Python

In Python, loops are used to execute a block of code repeatedly. There are two main types of loops:

1. For loop:

* A for loop is used to iterate over a sequence (like a list, tuple, string, or range) or other iterable objects.
* It executes the block of code once for each item in the sequence.

Example:

Python

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

Output:

apple

banana

cherry

2. While loop:

* A while loop is used to execute a block of code repeatedly as long as a condition is true.
* The loop continues until the condition becomes false.

Example:

i = 1

while i < 6:

print(i)

i += 1

Output:

1

2

3

4

5

Key Concepts:

* Iteration: Each execution of the loop body is called an iteration.
* Loop variable: In a for loop, the loop variable takes on the value of each item in the sequence.
* Condition: In a while loop, the condition is evaluated before each iteration. If the condition is true, the loop body is executed. If the condition is false, the loop terminates.
* Indentation: In Python, indentation is used to define the block of code that is executed within a loop.
* Break statement: The break statement can be used to exit a loop prematurely.
* Continue statement: The continue statement can be used to skip the current iteration and proceed to the next one.
* Nested loops: Loops can be nested inside other loops.

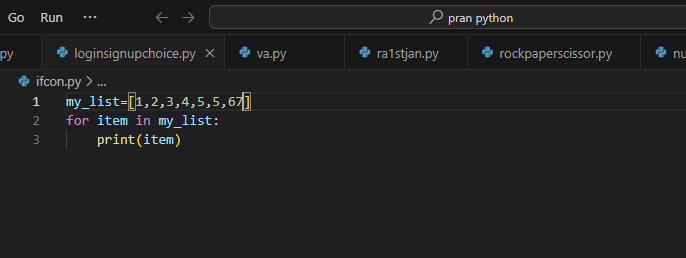
Using loops with collections (lists, tuples, etc.)

Python, loops (such as for loops) are commonly used with collections like lists, tuples, sets, and dictionaries to iterate through their elements. Here are examples demonstrating how to use loops with these collections:

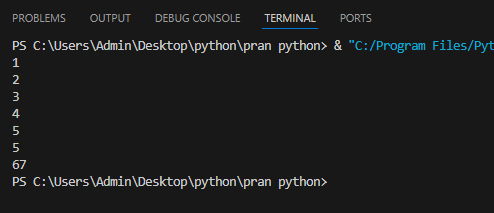
1. **Using a** for **loop with a list**

Lists are ordered collections, and you can use a loop to access each element:

**Example:**



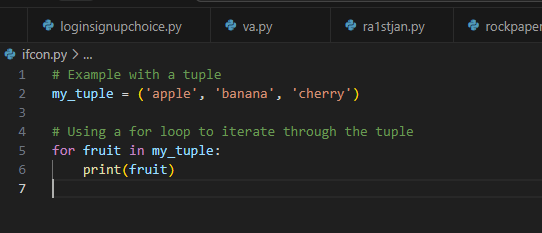
**Output:**



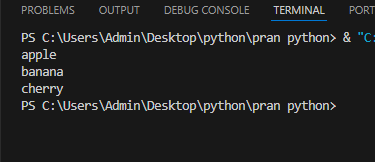
### 2. **Using a** for **loop with a tuple**

Tuples are similar to lists but are immutable. You can loop through a tuple in the same way:

Example:



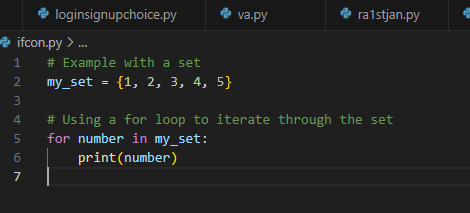
Output:



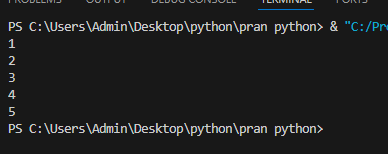
### 3. **Using a** for **loop with a set**

Sets are unordered collections, but you can still loop through them:

Example:



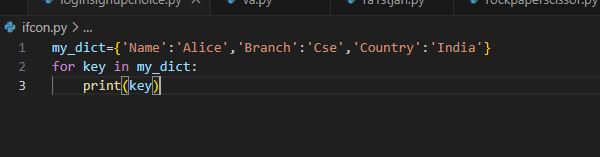
Output:



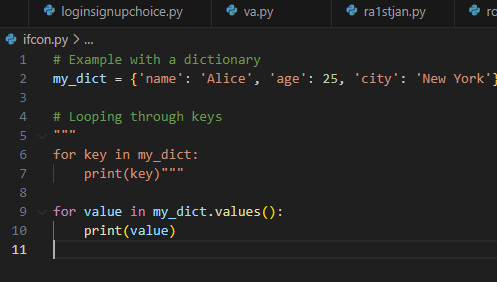
### 4 . **Using a** for **loop with a dictionary**

Dictionaries store key-value pairs. You can loop through them in different ways: keys, values, or both.

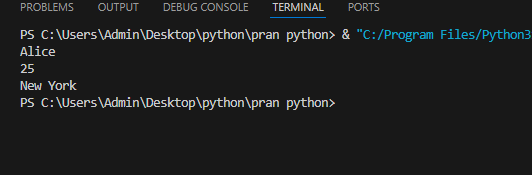
* Loop through keys:

Loop through values:

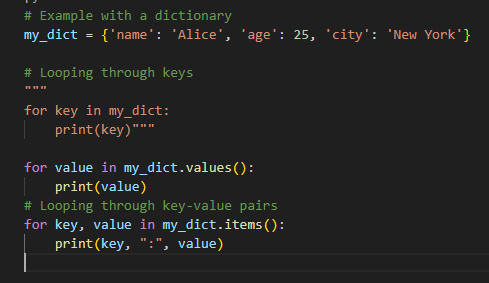
Example:



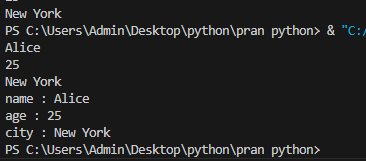
Output:



* Loop through key-value pairs:



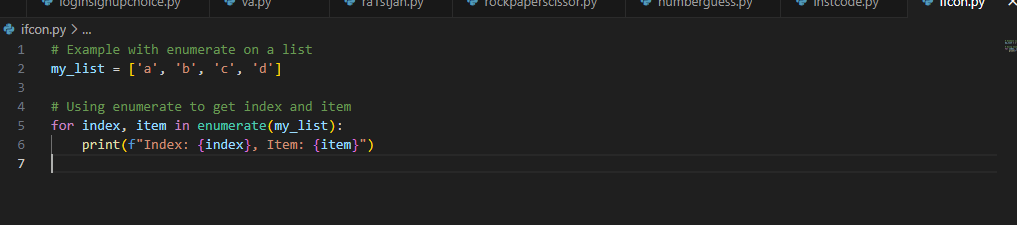
Output:



### 5. **Using a** for **loop with an index (enumerate)**

If you need both the index and the item from a collection (such as a list or tuple), you can use enumerate()

Example:



Output:

