1. Loop Control

for loop:

- A for loop is typically used when you know in advance how many times the loop will run.
- It iterates over a sequence (like a list, tuple, range, string, etc.).
- The loop stops when the sequence is exhausted.

Example:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)
```

while loop:

- A while loop is used when the number of iterations is not known in advance and depends on a condition being met.
- It continues until a given condition becomes false.

Example:

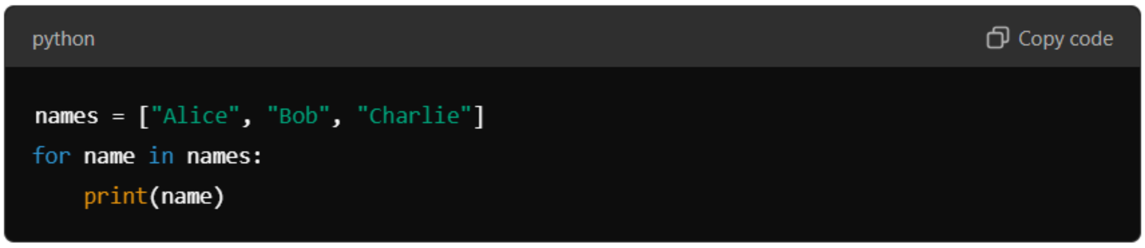
```
i = 1
while i <= 5:
    print(i)
    i += 1
```

2. Use Case

for loop:

- Best suited for iterating over a **sequence** (like lists, tuples, dictionaries, ranges, etc.).
- Often used when the **number of iterations is predetermined**

Example: Iterating over a list of names.

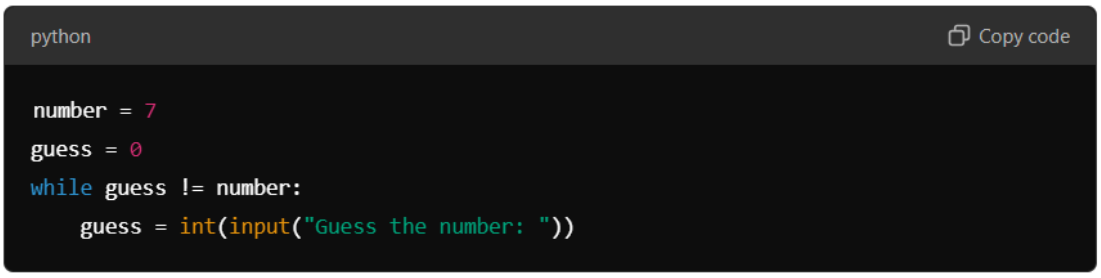


```python
names = ["Alice", "Bob", "Charlie"]
for name in names:
    print(name)
```

while loop:

- Used when the loop continues until a **certain condition** is met.
- Best for situations where the **termination condition is not strictly tied to a sequence**, and the loop might need to exit based on a dynamic or unknown condition.

Example: A loop that runs until the user guesses the correct number.



```python
number = 7
guess = 0
while guess != number:
    guess = int(input("Guess the number: "))
```

3. Termination Condition

- **for loop**:

    - The loop automatically stops after it has iterated through the entire sequence.
    - No explicit condition required to stop the loop, as it relies on the sequence’s length.

Example:

```python
for i in range(5):
    print(i)  # Automatically stops after 5 iterations
```

while loop:

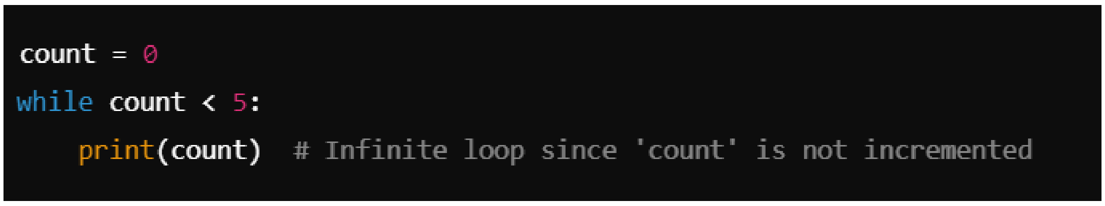
- The loop continues based on a condition that must be explicitly checked inside the loop.
- If the condition never becomes False, the loop will run indefinitely.

```python
count = 0
while count < 5:
    print(count)
    count += 1  # Stops when count reaches 5
```

4. Infinite Loop Risk

- **for loop**:
    - Very low risk of getting stuck in an infinite loop since it relies on iterating over a predefined sequence.
- **while loop**:

    - Higher risk of running into an infinite loop if the loop’s condition is not managed properly



```
count = 0
while count < 5:
    print(count)  # Infinite loop since 'count' is not incremented
```

5. Readability and Preference

- **for loop**:
    - Often preferred when you are working with collections (like lists, tuples, dictionaries) as it directly iterates over them, making the code more concise and readable.
- **while loop**:
    - Preferred when you need more control over the loop execution and when the number of iterations is not predetermined, as in event-driven programs (like waiting for user input, specific conditions, etc.).