The for Loop & Looping Over Sequences

AiCE Software Engineering Foundations — Python Programming

Repetition is the engine of automation. This lesson introduces loops — the mechanism that lets Python repeat an action dozens, hundreds, or millions of times without writing the instruction more than once — starting with the for loop and its most common use cases.

1. What Is a Loop?

A loop is a control structure that repeats a block of code multiple times. Instead of writing the same instruction ten times, you write it once and tell Python how many times to repeat it — or under what condition it should keep repeating.

Think about the roll call a teacher takes at the start of class. For each student on the list, the teacher calls the name, waits for a response, and marks attendance. The action — call name, record response — is identical for every student. A loop does exactly this: it iterates through a list and performs the same action for each item automatically.

> **The Power of Repetition**
> *Without loops, a program processing 1,000 student records would need 1,000 lines of code. With a loop, it needs fewer than 10. Python has two types of loops: the for loop and the while loop. Both repeat code, but they're used in different situations and work in fundamentally different ways.*

2. The for Loop

A for loop repeats a block of code a specific, known number of times. It's used when you know in advance how many iterations you need — either because you have a fixed number, or because you're iterating through a collection of items.

The most fundamental form uses Python's range() function to generate a sequence of numbers to iterate over. Its syntax is range(start, stop, step):

Parameter	Meaning	Default if Omitted
start	The number to begin from (inclusive)	0
stop	The number to stop before (exclusive) — never included	required
step	How much to increment each time	1

```
# Basic for loop — counting from 1 to 5
for i in range(1, 6):
    print(f"Count: {i}")

print("Loop complete.")
```

This produces:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Loop complete.
```

range(1, 6) generates 1, 2, 3, 4, 5 — starting at 1, stopping before 6. On each iteration, the variable i holds the current number. The indented print statement runs once per number in the range; the unindented 'Loop complete.' prints only once, after the loop finishes entirely.

> **The Loop Variable Name**
> *i is a convention, not a rule — you can name it anything: count, number, student_number, n. i is used so commonly because it's short for 'index.' When your loop variable has a more specific meaning, give it a more descriptive name.*

range() Variations

```
# range with only a stop value (starts at 0 by default)
print("Counting from 0:")
for i in range(5):
    print(i)

# range with start, stop, and step
print("Even numbers from 2 to 10:")
for i in range(2, 11, 2):
    print(i)

# Counting backwards
print("Countdown:")
for i in range(10, 0, -1):
    print(i)
print("Launch!")
```

range(5) starts at 0 and goes up to but not including 5, giving 0, 1, 2, 3, 4. range(2, 11, 2) starts at 2, stops before 11, stepping by 2, giving every even number in between. range(10, 0, -1) counts down from 10 to 1 using a negative step — range() works backwards just as naturally as forwards.

3. Looping Over a List

The for loop isn't limited to numbers — it can iterate over any sequence, including lists and strings, which is one of its most powerful applications:

```
# Iterating over a list of African countries
african_hubs = ["Lagos", "Nairobi", "Accra", "Kigali",
"Dakar"]

for city in african_hubs:
    print(f"AiCE has a presence in {city}.")

print(f"Total hubs: {len(african_hubs)}")

# Iterating over a list of scores
scores = [78, 85, 92, 67, 88, 74, 95]
```

```
total = 0

for score in scores:
    total += score

average = total / len(scores)
print(f"Total marks : {total}")
print(f"Average     : {average:.2f}")
```

In the first loop, city holds the current item from african_hubs on each pass. In the second, total accumulates the sum of every score using +=. This is called the accumulator pattern — initialize a variable to zero before the loop, then add to it inside the loop — one of the most commonly used patterns in all of programming. You'll use it constantly once lists are covered fully in an upcoming lesson.

Looping Over a String

A string is a sequence of characters, so a for loop can iterate over it character by character:

```
word = "AFRICA"
for letter in word:
    print(letter)
```

This prints A, F, R, I, C, A, one per line — each iteration gives you exactly one character. This technique is used in text analysis, encryption, and data processing.

Key Takeaways

A for loop repeats a block of code a known number of times, either using range(start, stop, step) or by iterating directly over a list or string. The accumulator pattern — a variable initialized before the loop and updated inside it — is how totals, sums, and averages get built across iterations. The next lesson covers the while loop, for situations where you don't know in advance how many times you'll need to repeat, along with break and continue for controlling a loop's flow mid-execution.