F-Strings, Multi-Line Text & String Techniques

AiCE Software Engineering Foundations — Python Programming

Concatenation works, but it becomes unwieldy fast once several variables are involved. This lesson introduces f-strings — the modern, professional way Python developers build text — along with multi-line strings, quotation mark conflicts, and a couple of small but genuinely useful string techniques.

1. F-Strings — The Modern and Professional Way

Python 3.6 introduced f-strings (formatted string literals), and they changed the way Python developers write output. An f-string lets you embed variables directly inside a string, exactly where you want them, using curly braces {}.

To create an f-string, place the letter f immediately before the opening quotation mark:

```
name = "Adesina"
age = 26

print(f"My name is {name} and I am {age} years old.")
```

This produces: My name is Adesina and I am 26 years old.

Python sees the f before the quote, scans the string for curly braces, finds {name} and {age}, fetches the values of those variables, and inserts them at those exact positions. Notice that age — an integer — was embedded directly, with no str() needed. F-strings handle type conversion automatically, which is one of their biggest advantages over concatenation.

F-Strings vs. Concatenation — A Direct Comparison

```
# Concatenation
print("Name: " + name + ", Age: " + str(age))

# F-string
print(f"Name: {name}, Age: {age}")
```

Both produce identical output, but the f-string reads almost like a sentence with blanks filled in — no quote or plus-sign management required. In professional Python code, f-strings are the standard.

2. F-Strings With Expressions

F-strings aren't limited to plain variables — you can place any valid Python expression inside the curly braces, and Python evaluates it and inserts the result:

```
price_naira = 15000
exchange_rate = 0.012      # approximate USD rate
student_count = 1200
modules = 4

print(f"Course price in USD: ${price_naira *
exchange_rate:.2f}")
print(f"Average students per module: {student_count /
modules}")
```

This produces:

```
Course price in USD: $180.00
Average students per module: 300.0
```

The expression price_naira * exchange_rate is calculated inside the curly braces before being embedded. The :.2f after it is a format specifier, telling Python to display the float with exactly two decimal places — the standard format for currency. You don't need to memorize format specifiers yet; the key takeaway is that f-strings can hold entire calculations, not just variable names.

3. Multi-Line Strings with Triple Quotes

Sometimes you need a string spanning multiple lines — a paragraph, a message template, an address block. Python's triple-quote syntax handles this cleanly:

```
mission_statement = """
The African Institute for Computing Excellence (AiCE)
exists to produce world-class software engineers
from every corner of the African continent.
"""

print(mission_statement)
```

Triple-quoted strings preserve every line break and space exactly as typed. F-strings work with triple quotes too, by combining the f prefix with triple quotes:

```
platform = "AiCE"
announcement = f"""
Welcome to {platform}!
"""
```

4. Strings and Quotation Mark Conflicts

A common beginner problem is accidentally ending a string early because it contains the same quote type used to open it:

```
# This causes a SyntaxError
message = "She said "Welcome to AiCE" and smiled."
```

Python sees the second double quote as the end of the string and gets confused by what follows. Three clean solutions:

Situation	Recommended Solution
String contains double quotes	Use single quotes on the outside, or escape with \
String contains an apostrophe	Use double quotes on the outside
String contains both quote types	Use triple quotes
String spans multiple lines	Use triple quotes

```
message = 'She said "Welcome to AiCE" and smiled.'
# or, escaping the quote:
message = "She said \"Welcome to AiCE\" and smiled."
```

The backslash (\) before a quotation mark tells Python: 'this quote is part of the string content, not the end of the string.' This is called escaping a character.

5. String Repetition with *

Just as + joins strings, the * operator repeats a string a specified number of times:

```
separator = "-" * 45
star_line = "*" * 45

print(star_line)
print("Welcome to AiCE Python Course")
print(star_line)
```

The "-" * 45 expression creates a string of 45 dash characters — a simple, effective technique for building visual separators and borders in terminal output.

6. Checking the Type of a String

type() always returns <class 'str'> for a string, no matter what the text looks like:

```
name = "Adesina Olagunju"
number_as_string = "2026"
empty_string = ""

print(type(name))
print(type(number_as_string))
print(type(empty_string))
```

All three print <class 'str'>. Notice that "2026" — a number inside quotation marks — is a string, not an integer, confirming that quotes make something a string regardless of what the content looks like. Also notice that "" — an empty string — is still a valid string with a length of zero; it exists, it's just empty, which is not the same as having no string at all.

Key Takeaways

F-strings — written with an f before the opening quote and {} around each variable or expression — are the professional standard for building text in Python, handling type conversion automatically and reading far more clearly than concatenation. Triple quotes handle multi-line text and let you sidestep quotation mark conflicts entirely, while * repeats a string for quick visual formatting. With strings fully covered, the next lesson goes deeper still, into the built-in string methods that let you transform, search, and inspect text directly.