Replacing, Cleaning & Splitting Text

AiCE Software Engineering Foundations — Python Programming

Real-world text is rarely clean. It arrives with stray spaces, inconsistent formatting, and characters you didn't ask for. This lesson covers three of the most practically important string methods: .replace() for substituting text, the strip methods for removing unwanted whitespace, and .split() for breaking a string into usable parts.

1. .replace() — Substituting Text

.replace() finds every occurrence of a specified substring and replaces it with another substring. It takes two arguments: the text to find, and the text to replace it with.

```
message = "Hello, Africa! Africa is rising. Africa leads."

# Replace every occurrence of "Africa" with "The Continent"
new_message = message.replace("Africa", "The Continent")
print(new_message)

# Replace spaces with underscores (common in file naming)
folder_name = "African Tech Summit 2026"
safe_name = folder_name.replace(" ", "_")
print(safe_name)

# Replace a character with nothing (effectively deleting it)
phone = "+234-801-234-5678"
clean_phone = phone.replace("-", "")
print(clean_phone)
```

This produces:

```
Hello, The Continent! The Continent is rising. The Continent
leads.
African_Tech_Summit_2026
+2348012345678
```

.replace() replaced every occurrence of 'Africa' in the first example — not just the first one. In the second, spaces became underscores to create a file-system-safe name. In the third, dashes were replaced with nothing, effectively deleting them — a common technique for cleaning phone numbers and other formatted data.

> **.replace() Is Case-Sensitive**
> *.replace('africa', 'The Continent') would not find 'Africa' with a capital A. The search text must match exactly, including case. Use .lower() or .upper() first if you need case-insensitive replacement.*

2. .strip(), .lstrip(), and .rstrip() — Removing Whitespace

Whitespace — spaces, tabs, and newline characters — often sneaks into strings unintentionally, especially from user input or files. The strip methods clean it up:

Method	What It Removes
.strip()	Whitespace from both the beginning and end of a string
.lstrip()	Whitespace from the left (beginning) only
.rstrip()	Whitespace from the right (end) only

```
# Simulating user input with accidental spaces
username = "   adesina_olagunju   "

print(f"Original: '{username}'")
print(f"Stripped: '{username.strip()}'")
```

This produces:

```
Original: '   adesina_olagunju   '
Stripped: 'adesina_olagunju'
```

> **Always Strip User Input**
> *In real applications, .strip() is applied to user input before storing or comparing it almost universally. Someone who types their email as ' kofi@ghana.com ' with accidental spaces should not be treated as having a different email from 'kofi@ghana.com'. Cleaning data before using it is not optional — it's professional.*

3. .split() — Breaking a String Into a List

.split() divides a string into parts based on a separator and returns the parts as a list. By default, it splits on whitespace, but you can specify a custom separator:

```
# Splitting on whitespace (default)
sentence = "Africa is the cradle of civilization"
words = sentence.split()
print(words)

# Splitting on a comma
african_countries = "Nigeria,Ghana,Kenya,Senegal,Rwanda"
countries_list = african_countries.split(",")
print(countries_list)

# Splitting on a slash (common in date strings)
date = "25/06/2026"
date_parts = date.split("/")
day = date_parts[0]
month = date_parts[1]
year = date_parts[2]
print(f"Day: {day}, Month: {month}, Year: {year}")
```

This produces:

```
['Africa', 'is', 'the', 'cradle', 'of', 'civilization']
['Nigeria', 'Ghana', 'Kenya', 'Senegal', 'Rwanda']
Day: 25, Month: 06, Year: 2026
```

The square brackets in the output indicate a list — Python's way of storing multiple values in sequence, covered thoroughly later in this program. For now, notice you can access individual parts using square bracket indexing: [0] gives the first item, [1] the second, [2] the third.

> **.split() Converts Text Into Usable Parts**
> *.split() is one of Python's most powerful string tools — it turns a single formatted string into a list of parts you can work with individually, which is exactly what happened with the date string above: one string became three usable pieces.*

Key Takeaways

.replace() swaps every occurrence of one substring for another and is case-sensitive; the strip methods remove unwanted whitespace from one or both ends of a string, a habit worth applying to virtually all user input; and .split() breaks a formatted string into a list of parts using any separator you choose. The next lesson covers searching within strings, accessing individual characters by position, and chaining several methods together in a single line.