Naming Rules, Updating & Multiple Assignment

AiCE Software Engineering Foundations — Python Programming

You now know what a variable is and the three core data types it can hold. This lesson covers the rules that govern how you're allowed to name a variable, the conventions that separate readable code from confusing code, how variables change over time, and a few techniques for assigning several variables at once.

1. Rules for Naming Variables

Python gives you flexibility in naming variables, but there are rules that must be followed, and conventions that should be followed. Breaking the rules causes errors; ignoring the conventions makes your code harder to read and maintain.

The Rules — Must Follow

These are enforced by Python. Violating any of them causes a SyntaxError or unexpected behavior.

1. Variable names cannot contain spaces — use underscores to separate words:

```
year_founded = 2025          # correct
year founded = 2025          # SyntaxError — space in name
```

2. Variable names cannot start with a number — they must start with a letter or an underscore:

```
cofounder1 = 'Adesina'        # correct
1cofounder = 'Adesina'        # SyntaxError — starts with
number
```

3. Variable names cannot be Python's reserved keywords — words Python has already assigned special meaning to:

```
# Reserved words you cannot use as variable names:
# if, else, elif, for, while, in, not, and, or, True, False,
# None, print, type, input, return, def, class, import, from
```

4. Variable names are case-sensitive — city, City, and CITY are three separate variables:

```
city = "Accra"
City = "Nairobi"
CITY = "Kigali"

print(city)      # Accra
print(City)      # Nairobi
print(CITY)      # Kigali
```

The Conventions — Should Follow

Python's style guide, called PEP 8, recommends these conventions. They aren't enforced by Python but are followed by virtually every professional Python developer:

Convention	Example	Used For
snake_case (lowercase, underscores)	year_founded, student_name	Variables and functions
UPPER_SNAKE_CASE (all caps)	MAX_STUDENTS, PI_VALUE	Constants (values that never change)
Descriptive names	student_score, not ss	Everything, always

> **Name Things So the Reader Never Has to Guess**
> *A variable named x tells you nothing. A variable named student_exam_score tells you exactly what it contains. Your code will be read far more times than it will be written — name things clearly enough that anyone, including your future self, understands immediately what a variable holds.*

2. Updating Variables

A variable is not a one-time assignment — you can change the value stored in it at any point in a program. The new value simply replaces the old one. This is why variables are called variables: their values can vary.

```
# Tracking the growth of a pan-African tech community
city = "Lagos"
print(city)

city = "Nairobi"
print(city)

city = "Accra"
print(city)
```

This produces Lagos, then Nairobi, then Accra — each new assignment replaces the previous value.

> **Python Doesn't Keep a History for You**
> *If you assign a value to a variable and then assign a new value without using the first one, the first value is simply lost — Python holds only the most recent value. That's exactly what Git is for: a searchable history of everything you've changed over time.*

3. Multiple Assignment — Assigning Several Variables at Once

Python lets you assign values to multiple variables on a single line — a clean, efficient way to initialize related variables together:

```
# Assigning multiple variables on one line
first_name, last_name, country = "Kwame", "Nkrumah", "Ghana"

print(first_name)
print(last_name)
```

```
print(country)
```

Python matches values to variables left to right: first_name gets "Kwame", last_name gets "Nkrumah", country gets "Ghana".

You can also assign the same value to several variables at once:

```
score_a = score_b = score_c = 0

print(score_a)
print(score_b)
print(score_c)
```

This is useful when initializing several counters or totals to zero at the start of a program.

4. A Real-World Variable Exercise — The AiCE Student Profile

Bringing everything together — all three data types, proper naming, and type() to verify our work:

```
# AiCE Student Profile
# Demonstrates strings, integers, floats, and type checking

# String variables
full_name = "Amara Diallo"
country_of_origin = "Senegal"
program = "Software Engineering Foundations"

# Integer variables
age = 22
year_of_enrollment = 2025

# Float variables
gpa = 3.85
```

```
# Display the profile
print("====================================")
print("         AiCE STUDENT PROFILE        ")
print("====================================")
print(full_name)
print(country_of_origin)
print(program)
print(age)
print(year_of_enrollment)
print(gpa)

# Verify data types
print("Data Types:")
print(type(full_name))
print(type(age))
print(type(gpa))
```

The output displays a complete student profile followed by the confirmed data types for the string, integer, and float variables. This is a real-world pattern — in production systems, student records, user accounts, and data entries are all built from structurally similar variables.

Key Takeaways

Python enforces a small set of naming rules — no spaces, no starting with a number, no reserved keywords, and case sensitivity — while PEP 8's conventions, like snake_case and descriptive names, are what separates readable code from a maintenance headache. Variables can be updated freely, with each new assignment replacing the last, and several variables can be assigned in a single line when it makes the code cleaner. With variables and data types firmly in place, the next lesson goes deep into one of Python's most frequently used data types: strings.