Python Variables: Store, Name and Work With Data

Learning Objectives

By the end of this guide, you will understand:

- What variables are and why they exist
- How Python variable assignment works
- How variable names reference values
- How to create, update, and reuse variables
- Python naming rules and conventions
- Multiple assignment
- Assigning different data types
- Reassigning variables
- Using variables in expressions
- Common variable mistakes
- How variables behave in real applications
- The difference between a variable name and the object it refers to

---

1. Why Do We Need Variables?

In the previous guide, you learned how to display information:

```
print("Hafsa")
print("Software Engineer")
print("Python")
```

This works.

But imagine building a real application.

A user changes their name.

A product changes its price.

A shopping cart changes its quantity.

A dashboard receives new statistics.

A website has thousands of users.

You cannot hard-code every possible value.

You need a way to **store information and use it later**.

That is where variables come in.

---

2. What Is a Variable?

A variable is a name that allows your program to refer to a value.

For example:

```
name = "Hafsa"
```

Now `name` can be used whenever you need that value.

```
print(name)
```

Output:

```
Hafsa
```

Instead of repeatedly writing:

```
print("Hafsa")
```

you can write:

```
print(name)
```

The program becomes easier to modify.

---

3. The Basic Assignment Syntax

The general pattern is:

```
variable_name = value
```

Example:

```
name = "Hafsa"
age = 25
language = "Python"
```

Think of the code as:

```
name     → "Hafsa"
age      → 25
language → "Python"
```

You can then use these names:

```
print(name)
print(age)
```

```
print(language)
```

Output:

```
Hafsa
25
Python
```

---

4. What Does `=` Mean?

One of the most important things to understand is that:

```
=
```

means **assignment** in Python.

It does not mean "is equal to" in the mathematical sense.

For example:

```
age = 25
```

means:

> Assign the value `25` to the name `age`.

Later, you can change it:

```
age = 26
```

Now `age` refers to `26`.

---

5. Creating Your First Variables

Let's create a small developer profile.

```
name = "Hafsa"
role = "Software Engineer"
language = "Python"
projects = 5
```

Now display the information:

```
print(name)
print(role)
print(language)
print(projects)
```

Output:

```
Hafsa
Software Engineer
Python
5
```

The important part is that the values are no longer directly embedded in every `print()` statement.

---

6. Why Variables Make Programs Flexible

Without variables:

```
print("Hafsa")
print("Hafsa is learning Python")
print("Hafsa has built 5 projects")
```

If the user's name changes, you have to find every occurrence.

With variables:

```
name = "Hafsa"

print(name)
print(name, "is learning Python")
print(name, "has built 5 projects")
```

Change one value:

```
name = "Sara"
```

The rest of the program can continue using `name`.

This is one of the fundamental ideas behind programming.

---

7. Variables Can Store Different Types of Data

Python variables can refer to different kinds of values.

String

```
name = "Hafsa"
```

Integer

```
age = 25
```

Float

```
rating = 4.8
```

Boolean

```
is_active = True
```

You can then print them:

```
print(name)
print(age)
print(rating)
print(is_active)
```

Output:

```
Hafsa
25
4.8
True
```

You will study Python's data types in depth in the next stage of the course.

---

8. Variables and Expressions

Variables become especially useful when combined with expressions.

Instead of:

```
print(100 + 50)
```

you can write:

```
price = 100
shipping = 50

total = price + shipping

print(total)
```

Output:

```
150
```

Now the program represents an actual calculation.

The values can change:

```
price = 200
shipping = 75
```

The same logic still works.

---

9. Variables Can Be Used Inside Calculations

Example:

```
a = 10
b = 20

result = a + b

print(result)
```

Output:

```
30
```

Another example:

```
width = 10
height = 5

area = width * height

print(area)
```

Output:

```
50
```

This is the beginning of building programs that process data.

---

10. Reassigning a Variable

A variable can be assigned a new value.

```
score = 50

print(score)

score = 80

print(score)
```

Output:

```
50
80
```

The second assignment replaces the value previously associated with `score`.

---

11. Variables Change Over Time

Consider a shopping cart:

```
quantity = 1

print(quantity)

quantity = 2

print(quantity)

quantity = 3

print(quantity)
```

Output:

```
1
2
3
```

This is important because real applications constantly deal with changing state.

Examples include:

- cart quantity
- account balance
- notification count
- game score
- current page
- login status
- number of likes
- inventory quantity

Variables allow programs to represent changing information.

---

12. A haas.dev Example

Imagine a simplified resource system.

```
resource_title = "Python Variables"
category = "Python Development"
pages = 25
is_free = True

print(resource_title)
print(category)
print(pages)
print(is_free)
```

Output:

```
Python Variables
Python Development
25
True
```

A real haas.dev application would likely retrieve this information from structured data rather than hard-coding it.

But the basic programming concept is the same:

> Give meaningful names to pieces of information so the program can work with them.

---

13. A Product Example

Imagine an online store.

```
product_name = "Laptop"
price = 120000
quantity = 2

total = price * quantity

print(product_name)
print(total)
```

Output:

```
Laptop
240000
```

Now change:

```
quantity = 3
```

The same calculation produces:

```
360000
```

The program is becoming dynamic.

---

14. Variable Naming

Good variable names make code easier to understand.

Compare:

```
x = "Hafsa"
y = 25
z = "Python"
```

with:

```
name = "Hafsa"
age = 25
language = "Python"
```

Both can work.

The second version communicates much more information.

Prefer names that explain what the value represents.

---

15. Python Variable Naming Rules

Variable names can contain:

- letters
- numbers
- underscores

Examples:

```
name = "Hafsa"
user_name = "Hafsa"
age2 = 25
project_count = 5
```

But there are rules.

A variable name cannot start with a number.

Incorrect:

```
2name = "Hafsa"
```

Correct:

```
name2 = "Hafsa"
```

---

16. Spaces Are Not Allowed in Variable Names

This is incorrect:

```
user name = "Hafsa"
```

Python interprets the space as part of its syntax.

Use an underscore:

```
user_name = "Hafsa"
```

This style is called **snake_case**.

Python code commonly uses snake_case for variable names.

---

17. Variable Names Are Case Sensitive

These are different names:

```
name = "Hafsa"
Name = "Ali"
NAME = "Sara"
```

Python treats them separately.

Therefore:

```
name
Name
NAME
```

are not interchangeable.

This is another reason to use consistent naming conventions.

---

18. Don't Use Python Keywords as Variable Names

Python has reserved keywords used by the language.

Examples include:

```
if
else
for
while
class
def
return
import
True
False
None
```

You should not use these as ordinary variable names.

For example:

```
class = "Python"
```

is invalid.

Choose another name:

```
course = "Python"
```

---

19. Good Naming vs Bad Naming

Avoid unclear names when a meaningful name is possible.

Less readable:

```
x = 120000
y = 2
z = x * y
```

Better:

```
price = 120000
quantity = 2
total = price * quantity
```

The second version is easier to understand without additional explanation.

---

20. Naming Conventions

Python commonly uses **snake_case**.

Examples:

```
first_name = "Hafsa"
total_price = 120000
user_count = 150
is_logged_in = True
```

For Boolean variables, names beginning with words such as:

```
is_
has_
can_
should_
```

can make the meaning clearer.

Examples:

```
is_active = True
has_account = False
can_edit = True
```

---

21. Variables Don't Need Type Declarations

In some programming languages, you may need to explicitly declare the type.

Python generally does not require this.

You can write:

```
age = 25
```

rather than something like:

```
integer age = 25
```

Python determines the type of the value at runtime.

You can later inspect the type using:

```
type(age)
```

For example:

```
age = 25

print(type(age))
```

Output will indicate that `age` is an integer.

You will learn `type()` and Python's type system more deeply in the next guide.

---

22. Variables Can Change Type

Python allows a name to refer to values of different types at different times.

For example:

```
value = 10

print(value)

value = "Python"

print(value)
```

Output:

```
10
Python
```

This flexibility is one aspect of Python's dynamic typing.

However, changing the meaning of a variable unnecessarily can make code confusing.

Prefer clear and consistent variable usage.

---

23. Multiple Assignment

Python allows multiple variables to be assigned in one statement.

```
name, age = "Hafsa", 25
```

This is equivalent to:

```
name = "Hafsa"
age = 25
```

You can then write:

```
print(name)
print(age)
```

Output:

```
Hafsa
25
```

---

24. Assigning the Same Value to Multiple Variables

You can also write:

```
a = b = c = 0
```

Now all three names refer to the value `0`.

```
print(a)
print(b)
print(c)
```

Output:

```
0
0
0
```

This can be useful in some situations, but use it carefully when mutable objects are involved. That topic will become important later.

---

25. Swapping Values

Python makes swapping values simple.

```
a = 10
b = 20

a, b = b, a

print(a)
print(b)
```

Output:

```
20
10
```

You don't need a temporary variable for this common operation.

This is one of Python's convenient features.

---

26. Variables and Memory

A common beginner explanation says:

> A variable is a box containing a value.

This is useful initially, but it is not the complete Python model.

A more accurate mental model is:

> A variable name refers to an object.

For example:

```
name = "Hafsa"
```

Conceptually:

```
name ———→ "Hafsa"
```

The name `name` refers to a string object.

Another assignment:

```
age = 25
```

can be visualized as:

```
age ———→ 25
```

This model becomes very important when you later learn about:

- objects
- mutability
- references
- copying
- lists
- dictionaries
- function arguments

---

27. Assignment Does Not Mean “Copy Everything”

Consider:

```
a = 10
b = a
```

Conceptually:

```
a ——→ 10
b ——→ 10
```

Both names refer to the same immutable integer object/value representation.

If you later write:

```
b = 20
```

you are changing what `b` refers to.

```
a ──→ 10

b ──→ 20
```

`a` remains `10`.

This distinction becomes much more important with mutable data structures.

---

28. Variables Can Reference the Results of Expressions

You can assign an expression directly:

```
price = 100
tax = 20

final_price = price + tax
```

Python evaluates:

```
price + tax
```

and assigns the resulting value to:

```
final_price
```

So:

```
print(final_price)
```

produces:

```
120
```

This pattern is extremely common in real programs.

---

29. Building a Simple Calculator

Variables allow us to create a basic calculator.

```
first_number = 20
second_number = 5
```

```
addition = first_number + second_number
subtraction = first_number - second_number
multiplication = first_number * second_number
division = first_number / second_number

print(addition)
print(subtraction)
print(multiplication)
print(division)
```

Output:

```
25
15
100
4.0
```

The calculator is still static because the numbers are hard-coded.

Later, `input()` will allow users to provide their own values.

---

30. Building an Order Calculator

Let's make the previous product example more realistic.

```
product_name = "Laptop"
price = 120000
quantity = 2

subtotal = price * quantity

print("Product:", product_name)
print("Price:", price)
print("Quantity:", quantity)
print("Subtotal:", subtotal)
```

Output:

```
Product: Laptop
Price: 120000
Quantity: 2
Subtotal: 240000
```

Now the logic is reusable.

Change:

```
quantity = 3
```

and the subtotal updates automatically.

---

31. Building a haas.dev Resource Record

Consider a simple resource object represented using separate variables:

```
title = "Python Variables"
category = "Python Development"
pages = 25
is_free = True

print("Title:", title)
print("Category:", category)
print("Pages:", pages)
print("Free:", is_free)
```

Output:

```
Title: Python Variables
Category: Python Development
Pages: 25
Free: True
```

Later, Python dictionaries and data structures will let us represent this kind of record much more naturally.

---

32. Variables Make Code Maintainable

Consider:

```
print(120000 * 2)
print("Laptop")
print(120000)
print(2)
```

A reader has to figure out what each number means.

Compare:

```
product_name = "Laptop"
price = 120000
quantity = 2
```

```
total = price * quantity

print(product_name)
print(price)
print(quantity)
print(total)
```

The second version communicates intent.

Good code should make the developer's thinking visible.

---

33. Constants and Naming Conventions

Python does not enforce constants in the same way some languages do.

However, developers commonly use uppercase names to indicate a value should not normally change.

Example:

```
TAX_RATE = 0.10
```

The uppercase naming convention communicates:

> Treat this as a constant.

Python will not prevent you from changing it:

```
TAX_RATE = 0.20
```

The convention is primarily for developers.

---

34. Common Mistake: Using a Variable Before Creating It

This will cause an error:

```
print(name)
```

if `name` has not been defined earlier.

Correct:

```
name = "Hafsa"
print(name)
```

Your program must have a valid name/value relationship before you try to use the variable.

---

35. Common Mistake: Misspelling a Variable

Consider:

```
user_name = "Hafsa"

print(username)
```

These are different names:

```
user_name
username
```

Python will not automatically know that you intended them to be the same.

Consistent naming is important.

---

36. Common Mistake: Starting With a Number

Incorrect:

```
1st_place = "Gold"
```

Correct:

```
first_place = "Gold"
```

A variable name cannot begin with a number.

---

37. Common Mistake: Using Spaces

Incorrect:

```
first name = "Hafsa"
```

Correct:

```
first_name = "Hafsa"
```

Use underscores when multiple words are needed.

---

38. Common Mistake: Confusing `=` and `==`

This distinction will become very important.

```
age = 25
```

means assignment.

Later you will learn:

```
age == 25
```

which checks whether two values are equal.

For now, remember:

```
=   → assignment
==  → equality comparison
```

The second operator belongs to the comparison concepts you will learn later.

---

39. Variable Lifecycle

A useful way to think about variables is:

```
Create
  ↓
Assign a value
  ↓
Use the variable
  ↓
Possibly update it
  ↓
Use the new value
```

Example:

```
score = 0

print(score)

score = 10

print(score)

score = 25

print(score)
```

Output:

```
0
10
25
```

This simple pattern is the foundation of stateful programs.

---

40. Variables in Real Software

Variables appear everywhere.

Web application

```
username = "Hafsa"
is_logged_in = True
```

E-commerce

```
cart_items = 4
total_price = 8500
```

Banking

```
balance = 50000
```

Social media

```
followers = 1200
likes = 350
```

Analytics

```
page_views = 15000
conversion_rate = 4.5
```

haas.dev

```
resource_count = 100
category = "Python Development"
```

The syntax is simple, but the same idea scales into large systems.

---

41. Mini Project: Developer Profile

Create a small profile using variables.

```
name = "Hafsa"
role = "Software Engineer"
language = "Python"
projects = 5
is_learning = True

print("=== Developer Profile ===")
```

```
print("Name:", name)
print("Role:", role)
print("Language:", language)
print("Projects:", projects)
print("Currently Learning:", is_learning)
```

This is your first step from static output toward dynamic programs.

---

42. Mini Project: Product Invoice

```
product = "Keyboard"
price = 5000
quantity = 2

subtotal = price * quantity

print("=== Invoice ===")
print("Product:", product)
print("Price:", price)
print("Quantity:", quantity)
print("Subtotal:", subtotal)
```

Challenge yourself to modify the values without changing the calculation logic.

---

43. Mini Project: haas.dev Resource

```
title = "Python Variables"
category = "Python Development"
pages = 25
is_free = True

print("=== Resource ===")
print("Title:", title)
print("Category:", category)
print("Pages:", pages)
print("Free:", is_free)
```

Now change the values and observe how the same program can represent a completely different resource.

---

44. Practice Exercises

Easy

Exercise 1

Create variables for:

- your name
- your age
- your city
- your programming language

Print all of them.

Exercise 2

Create:

```
price = 500
quantity = 3
```

Calculate and print the total.

Exercise 3

Create a variable called `favorite_language` and assign it a programming language.

Medium

Exercise 4

Create variables for:

- product name
- product price
- quantity
- total

Calculate the total automatically.

Exercise 5

Create a developer profile containing at least six variables.

Exercise 6

Create a haas.dev resource record containing:

- title
- category
- page count
- free/paid status

Hard

Exercise 7

Create a simple monthly project tracker:

```
Projects Completed: 4
Projects In Progress: 2
Projects Planned: 3
```

Store each value in a variable and calculate the total number of projects.

Exercise 8

Create a simple shopping bill containing three products.

Use variables for each product's price and quantity, then calculate the grand total.

---

45. Five Minute Challenge

Build a **Developer Stats Dashboard**.

Your program should store information such as:

```
Name
Role
Projects Completed
Current Project
Programming Language
Years Learning
Is Available
```

Then display it in a readable format.

Requirements:

- use at least 7 variables
- use meaningful variable names
- include strings
- include numbers
- include a Boolean
- calculate at least one value
- do not repeat the same hard-coded value unnecessarily

---

46. Mini Quiz

Question 1

What does this do?

```
name = "Hafsa"
```

Answer: It assigns the string `"Hafsa"` to the variable name `name`.

Question 2

Which is a valid variable name?

A. `2name`

B. `user name`

C. `user_name`

D. `class`

Answer: C

Question 3

What does `=` mean in:

```
score = 100
```

Answer: Assignment.

Question 4

What happens here?

```
score = 10
score = 20
```

Answer: The name `score` is reassigned to `20`.

Question 5

Which naming style is commonly used for Python variables?

A. camelCase

B. PascalCase

C. snake_case

D. kebab-case

Answer: C

---

47. Interview Questions

What is a variable in Python?

A variable is a name used to refer to a value/object so the program can work with that information.

Does Python require variable type declarations?

No. Python uses dynamic typing, so you normally assign a value directly and Python determines its type.

Can a Python variable change its value?

Yes.

```
score = 10
score = 20
```

Can a variable change the type of value it refers to?

Yes.

```
value = 10
value = "Python"
```

Although possible, changing a variable's meaning unnecessarily can reduce code clarity.

What is snake_case?

A naming convention where words are separated using underscores:

```
total_price = 5000
```

What is the difference between a variable and its value?

The variable is the name used to refer to an object/value. The value is the actual data being referenced.

What does multiple assignment mean?

It allows multiple variables to be assigned in one statement:

```
name, age = "Hafsa", 25
```

---

48. Cheat Sheet

Create a variable

```
name = "Hafsa"
```

Number

```
age = 25
```

Decimal

```
rating = 4.8
```

Boolean

```
is_active = True
```

Use a variable

```
print(name)
```

Update a variable

```
age = 26
```

Calculate with variables

```
total = price * quantity
```

Multiple assignment

```
name, age = "Hafsa", 25
```

Swap values

```
a, b = b, a
```

Python naming style

```
first_name = "Hafsa"
total_price = 5000
is_logged_in = True
```

Constant convention

```
MAX_USERS = 100
```

---

49. Key Takeaways

Remember these core ideas:

1. Variables allow programs to work with information.

2. Assignment uses `=`.
3. A variable name refers to a value/object.
4. Variables can be reassigned.
5. Python variables do not normally require explicit type declarations.
6. Variable names should be meaningful.
7. Python commonly uses `snake_case`.
8. Variable names cannot start with numbers.
9. Spaces are not allowed in variable names.
10. Python is case-sensitive.
11. Variables can be used in expressions and calculations.
12. Multiple assignment is supported.
13. Good naming improves readability and maintainability.
14. Variables are the foundation for working with dynamic data.

The most important transition is:

```
Static program
     ↓
Hard-coded values
     ↓
Variables
     ↓
Dynamic data
     ↓
Useful programs
```

---

50. What Comes Next?

You can now store information using variables.

But an important question remains:

> How does Python know whether `25`, `"25"`, `25.5`, and `True` are different kinds of values?

That leads to one of the most important Python fundamentals:

Data Types.

Next, you will learn how Python represents different kinds of data, why types matter, how Python determines a value's type, and how choosing the correct type affects what your program can do.

Visit haas.dev for more resources and guides.

haas.dev

https://dev-roast-app.vercel.app