

Python Variables:

1. What are Variables?

A variable is a name that stores data (value) in memory.

You can use it to store, update, and retrieve information.

In Python, variables are created automatically when you assign a value .

2. Definition

A variable is simply a reference (name) that points to a value stored in memory.

. Example

```
country = "Pakistan" # String
age = 20             # Integer
price = 99.50        # Float
is_active = True     # Boolean
```

4. Syntax

variable_name = value

Example:

```
name = "Ali"
marks = 85
```

5. Rules for Identifiers:

1. Must start with a letter or an underscore.
2. Cannot start with a number.
3. Can contain only letters, digits, and underscores (no spaces/special symbols).
4. Cannot use Python keywords (e.g., if, for, class).
5. Case-sensitive → Name and name are different.
6. Use meaningful names.

✗ Wrong: 2age, first name, my-var

✓ Correct: age2, _first_name, user_id

8. Variable Naming Conventions

Snake Case (recommended in Python):

Words separated underscore

student_name, total_marks

Camel Case:

first letter of 1st word is small, 1st letter of 2nd word is capital

studentName, totalMarks

Pascal Case (commonly used for Classes):

First letter of both words is capital

StudentName, TotalMarks

Constants (always uppercase):

MAX_VALUE = 100

9. Best Practices

- ☒ Use clear, meaningful names (user_age instead of ua).
- ☒ Use snake_case for variables.
- ☒ Use uppercase for constants.
- ☒ Initialize with None if value will be given later. **✗** Don't use reserved keywords as names.
- ✗** Avoid chaining mutable objects (like lists).

10. Common Errors

NameError → using variable before assigning.

print(x) # Error if x not defined

SyntaxError → invalid characters in name.

my-name = 5 # **✗**

Keyword Error → using reserved words.

for = 10 # **✗**

11. Conclusion

A variable is just a name pointing to a value.

Python creates it automatically when assigned.

Follow naming rules and conventions for readability.

Use snake_case as standard practice.

Be careful with mutable objects in chained assignments.

Python Data Types & Operators — Notes

1. Data Types

a) Integer (int)

Definition:

Whole numbers (positive, negative, or zero) without decimal point.

Example:

```
age = 25
temperature = -10
```

b) String (str)

Definition: A sequence of characters enclosed in single (') or double (") quotes.

Example:

```
name = "Ayesha"
```

```
message = 'Hello, Python!'
```

c) Float (float)

Definition: Numbers with decimal point (fractional values).

Example:

```
price = 99.99
pi = 3.1416
```

d) Boolean (bool)

Definition:

Represents truth values — either True or False.

Example:

```
is_active = True
is_passed = False
```

e) None (NoneType)

Definition:

Represents the absence of a value or a null value.

Example:

```
result = None
```

☒ Key Points

int, float, str, bool, and NoneType are basic built-in types.

Python is dynamically typed → no need to declare data type, Python detects automatically.

Example:

```
x = 10    # int
x = "ten" # str (changed type dynamically)
```

2. Types of Operators in Python

a) Arithmetic Operators

Used for mathematical operations.

Examples: +, -, *, /, %, //, **

```
a = 10
b = 3

print(a + b) # 13
print(a - b) # 7
print(a * b) # 30
print(a / b) # 3.33
print(a // b) # 3 (floor division)
print(a % b) # 1 (remainder/modulus)
print(a ** b) # exponentiation

print(a ** b) # 1000 (power)
```

b) Relational (Comparison) Operators

Used to compare values, returns True or False.

Examples: == , != , > , < , >= , <=

```
x = 5
y = 10
print(x == y) # False #equal
print(x != y) # True #is not equal
print(x < y) # True #GREATER
```

c) Assignment Operators

Used to assign values to variables.

Examples: = , += , -= , *= , /= , %=

```
x = 5
x += 3 # x = x + 3 → 8
x *= 2 # x = x * 2 → 16
```

d) Logical Operators

Used with Boolean values.

Examples: and , or , not

```
x = True
y = False
print(x and y) # False #both condition must be true
print(x or y) # True] #at least one condition is true
print(not x) # False #it reverse the result
```

3. Type Conversion vs Type Casting

Type Conversion (Automatic / Implicit)

Python converts one type to another automatically when needed.

```
x = 10    # int
y = 2.5   # float
z = x + y  # int + float → float
print(z)  # 12.5
```

Type Casting (Manual / Explicit)

Programmer converts type using functions like `int()`, `float()`, `str()`, `bool()`.

```
x = int("2") # string → int
y = 4.25
print(x + y) # 6.25
]
```

🔥 Recap

Data Types: int, float, str, bool, None

Operators: Arithmetic, Relational, Assignment, Logical

Type Conversion: Automatic

Type Casting: Manual using functions

INPUT():

`input()` is a function used to take data (text/number) from the user.

Whatever the user types → Python reads it as a string (text) by default.

Example 1: Basic input

```
name = input("Enter your name: ")
print("Hello,", name)
```

👉 If user types Alice, output will be:

```
Enter your name: Alice
Hello, Alice
```

Example 2: Numbers with input

Since `input()` always gives data as string, you must convert it into a number if needed.

```
age = int(input("Enter your age: "))
```

```
print("You are", age, "years old")
```

👉 If user types 20, output:

```
Enter your age: 20
```

```
You are 20 years old
```

Example 3: Float (decimal numbers)

```
marks = float(input("Enter your marks: "))
```

```
print("Your marks are:", marks)
```

KEY POINTS:

- `input()` = asks the user to type something.
- Always returns text (string).
- If you want numbers → convert with `int()` or `float()`.