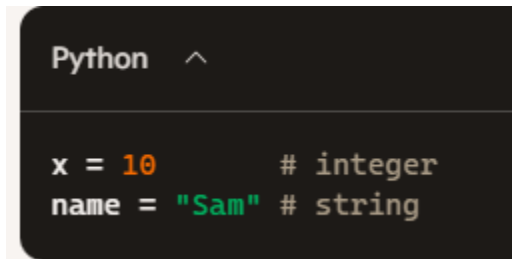


Variables in Python

Variables are containers for storing data values.

- **Declaration:** No need to declare type explicitly.



```
Python ^  
  
x = 10      # integer  
name = "Sam" # string
```

Rules:

- Must start with a letter or underscore.
- Cannot start with a number.
- Case-sensitive (Name \neq name).

Keywords in Python

Keywords are reserved words that have special meaning.

As of Python 3.11, there are **35 official keywords** in the language². These are reserved words that have special meaning and cannot be used as variable names, function names, or identifiers.



Examples of Python Keywords

Here are some commonly used ones:

- **Control flow:** if, else, elif, for, while, break, continue, pass
- **Functionality:** def, return, lambda, yield
- **Error handling:** try, except, finally, raise, assert
- **Boolean & logic:** True, False, None, and, or, not, is, in
- **Class & scope:** class, global, nonlocal, del, from, import, as
- **Concurrency:** async, await

You can list all keywords using:

```
Python ^  
  
import keyword  
print(keyword.kwlist)
```

Data Types in Python

Python has several built-in data types:

Category	Data Types	Example
Numeric	int, float, complex	10, 3.14, 2+3j
Sequence	str, list, tuple	"hello", [1,2], (3,4)
Set	set, frozenset	{1,2,3}
Mapping	dict	{"a": 1, "b": 2}
Boolean	bool	True, False
Binary	bytes, bytearray, memoryview	b"hello"
None Type	NoneType	None

Type Conversion in Python

Python supports **implicit** and **explicit** type conversion.

Implicit Conversion

Python automatically converts types when needed:

Python ^

```
x = 10      # int
y = 2.5     # float
z = x + y   # float (Python converts x to float)
```

👉 Explicit Conversion (Type Casting)

You manually convert types using built-in functions:

Python ^

```
int("5")      # Converts string to int
float("3.14")  # Converts string to float
str(100)       # Converts int to string
list("abc")    # Converts string to list
```

Operators

+ 1. Arithmetic Operators

Used for basic mathematical operations.

```
a = 10
b = 3

print(a + b)  # 13 (Addition)
print(a - b)  # 7  (Subtraction)
print(a * b)  # 30 (Multiplication)
print(a / b)  # 3.33 (Division)
print(a % b)  # 1  (Modulus: remainder)
print(a ** b) # 1000 (Exponentiation: 10^3)
print(a // b) # 3  (Floor Division: drops decimal)
```



2. Comparison Operators

Used to compare values; returns True or False.

```
x = 5
y = 8

print(x == y) # False
print(x != y) # True
print(x > y)  # False
print(x < y)  # True
print(x >= y) # False
print(x <= y) # True
```



3. Logical Operators

Used to combine conditional statements.

Python ^

```
a = True
b = False

print(a and b) # False
print(a or b)  # True
print(not a)   # False
```



4. Assignment Operators

Used to assign values to variables.

Python ^

```
x = 10
x += 5 # x = x + 5 → x becomes 15
x *= 2 # x = x * 2 → x becomes 30
print(x)
```



5. Bitwise Operators

Operate on binary numbers.

Python ^

```
a = 5      # 0101 in binary
b = 3      # 0011 in binary

print(a & b) # 1 (AND)
print(a | b) # 7 (OR)
print(a ^ b) # 6 (XOR)
print(~a)    # -6 (NOT)
print(a << 1) # 10 (Left shift)
print(a >> 1) # 2 (Right shift)
```

For Bitwise NOT(~)



Step-by-Step: How $1011 = -5$ in 2's Complement



Step 1: Understand the context

We're working with **4-bit binary numbers**. In 2's complement, the **leftmost bit** is the **sign bit**:

- 0 means positive
- 1 means negative

So 1011 starts with 1 → it's a **negative number**.



Step 2: Invert the 2's complement to find the original positive value

To find the decimal value of a 2's complement negative number:

1. Invert all bits of 1011 :

Code ^

$1011 \rightarrow 0100$

2. Add 1 to the inverted bits:

Code ^

$0100 + 1 = 0101$

3. Convert to decimal:

Code ^

$0101 = 5$

So, 1011 is the 2's complement representation of -5 .

6. Identity & Membership Operators

Identity (`is`, `is not`)

Python ^

```
x = [1, 2]
y = x
z = [1, 2]

print(x is y)      # True (same object)
print(x is z)      # False (same content, different object)
```

Membership (`in`, `not in`)

Python ^

```
nums = [1, 2, 3]
print(2 in nums)    # True
print(4 not in nums) # True
```

Comments in Python

Purpose: Comments are used to explain code, make it readable, or temporarily disable parts of it. Python ignores comments during execution.

✓ Single-line Comment

Use # at the beginning of the line:

```
# This is a single-line comment
x = 5 # This sets x to 5
```

✓ Multi-line Comment

Use triple quotes ''' or ''':

```
'''  
This is a multi-line comment.  
It can span several lines.  
'''
```

📥 Input Function: input()

Purpose: Takes user input from the keyboard. Always returns a string.

✓ Basic Usage

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

✓ Type Conversion

Convert input to other types using int(), float(), etc.:

```
age = int(input("Enter your age: "))  
print("You will be", age + 1, "next year.")
```

📤 Output Function: print()

Purpose: Displays text, variables, and results on the screen.

✓ Basic Usage

```
print("Welcome to Python!")
```


✓ Printing Multiple Items

Python ^

```
name = "Sam"  
age = 25  
print("Name:", name, "Age:", age)
```

✓ Formatted Output

Use f-strings for cleaner formatting:

Python ^

```
print(f"Hello {name}, you are {age} years old.")
```