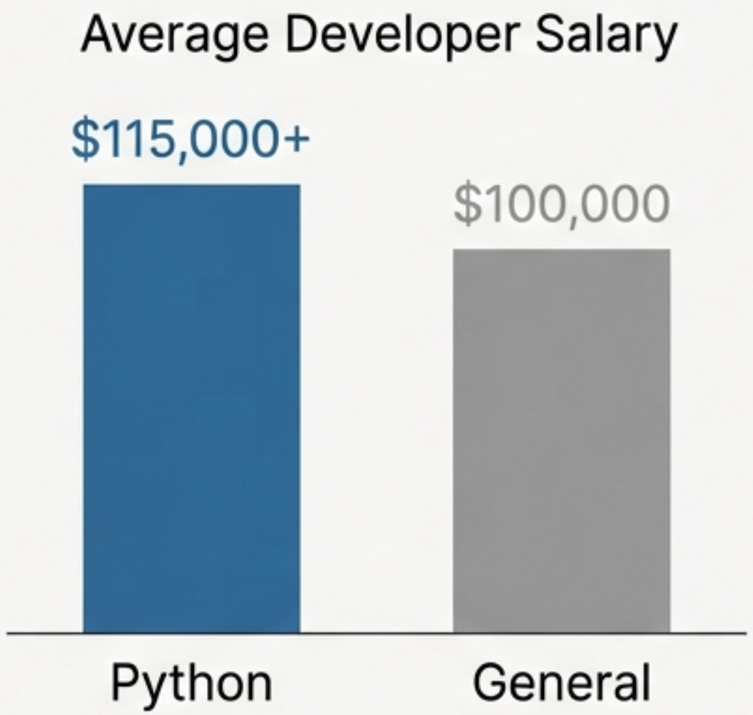
Python Mastery

From Zero to Hero: A Comprehensive Guide

Designed for beginners, engineered for professionals. Learn the syntax, the logic, and the "Clean Code" habits of top software engineers.

Why Python? The Language of Giants



Google Spotify Dropbox Facebook

Trusted by Industry Leaders

The "Hello World" Test

C Code (The Hard Way)

```
#include <stdio.h>
int main() {
   printf("Hello, World!");
   return 0;
}
```

Python Code (The Smart Way)

```
print('Hello World')
```

Solve complex problems in less time **with fewer lines of code.**

The Toolkit: Setting Up Your Environment

The Engine

Interpreter (CPython)

The default implementation written in C. Translates Python code into machine instructions.

The Workshop

VS Code + Python Extension

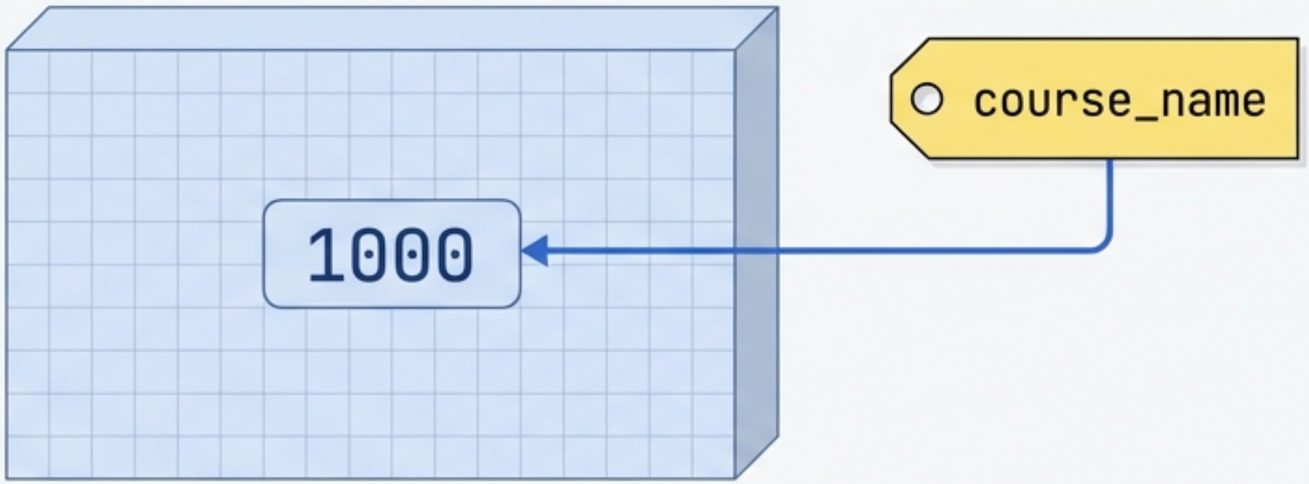
The industry-standard editor. Enhanced with Microsoft's Python extension for IntelliSense and debugging.

The Safety Net

Linting & Formatting

Analyzes code for errors before execution. Use 'Autopep8' to format on save.

Variables: Labels, Not Boxes



A variable is a reference pointing to data in memory.

Amateur Code

```
cn = 'Python' # Mystical
c = 1         # Ambiguous
```

Pro Code

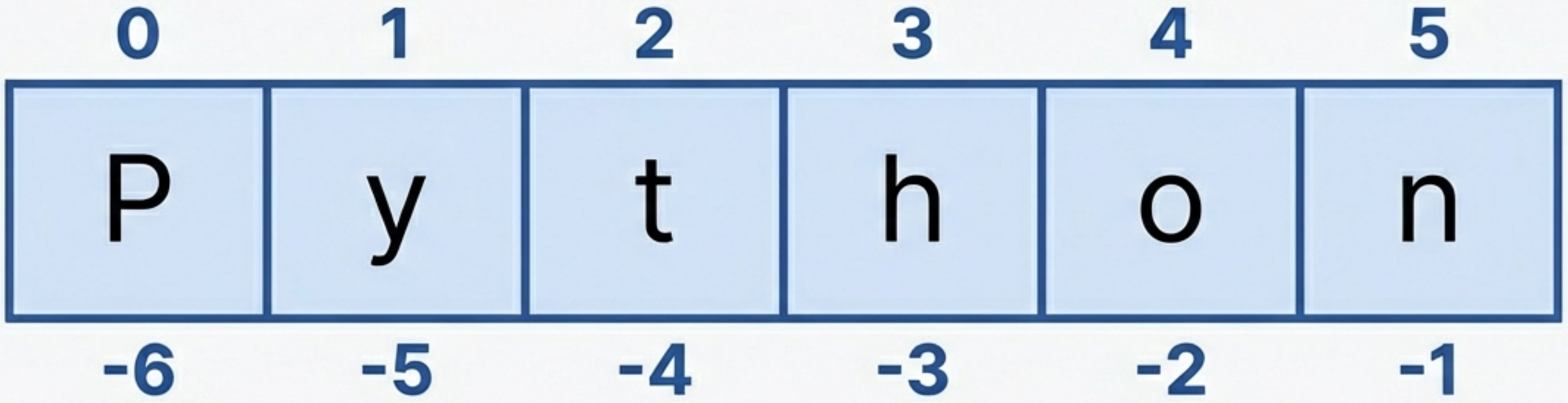
```
course_name = 'Python' # Descriptive
student_count = 1      # Meaningful
```

PEP 8 Rules

1. Use snake_case (lowercase_with_underscores)
2. Put spaces around operators (x = 1)
3. Names must tell a story

Anatomy of a String

Zero-Indexing and Slicing



Access

```
course[0]  ⟶ 'P'
course[-1] ⟶ 'n'
```

Slicing

```
course[0:3] ⟶ 'Pyt'
```

Start index inclusive,
end index exclusive.

Defaults

```
course[1:] ⟶ 'ython'
```

(To the end)

String Power Tools

Formatted Strings (f-strings)

Concatenation (The Old Way)

```
full = first + ' ' + last
```

f-strings (The Pro Way)

```
full = f'{first} {last}'
```

Expressions inside {} are evaluated at runtime.

Essential Methods

`.upper() / .lower()`	Converts case.
`.strip()`	Removes whitespace (vital for input).
`.replace('p', 'j')`	Swaps characters.
`'Pro' in course`	Returns Boolean True/False.

Numbers & Math

Data Types, Nuances, and Syntax

Integer
Whole numbers (e.g., 10)

Float
Decimals (e.g., 4.99)

Complex
a + bj (e.g., 1 + 2j)

Arithmetic Nuances

Standard Division - Float

`10 / 3 = 3.3333`

Float result

Floor Division - Integer

`10 // 3 = 3`

Integer result

Modulus - Remainder

`10 % 3 = 1`

Integer result

Professional Syntax

The Math Module

```
import math
print(math.ceil(2.2)) # Output: 3
```

Accessing mathematical functions.

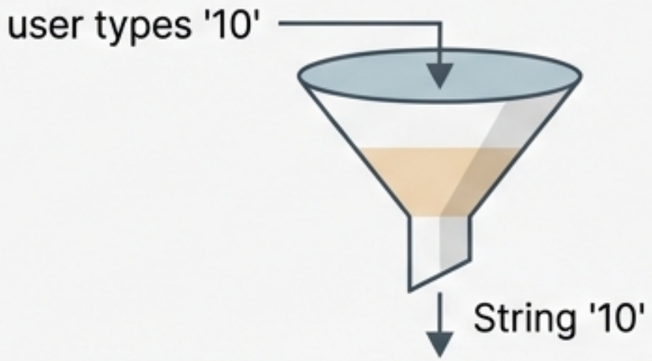
Augmented Assignment

```
x = 10
x += 3 # Equivalent to x = x + 3
```

Concise value updates.

Type Conversion & The Input Trap

The Trap




```
x = input('Enter x') # x is '10'
y = x + 1 # Error! Cannot add String + Int
```

The Fix & Logic

```
x = int(input('Enter x'))
y = x + 1 # Success
```

Python Boolean Interpretation

Falsy Values	0, '' (empty string), None	Evaluates to False
Truthy Values	Everything else (including 'False')	Evaluates to True

Comparison & Logic

The Syntax

= is Assignment → x = 10

== is Comparison → x == 10 → True/False

> (Greater)	**<=** (Less/Equal)	**!=** (Not Equal)

Lexicographical Comparison

JetBrains Mono

```
'bag' > 'apple'
```

→ True

‘b’ comes after ‘a’

Roboto

JetBrains Mono

```
'bag' == 'BAG'
```

→ False

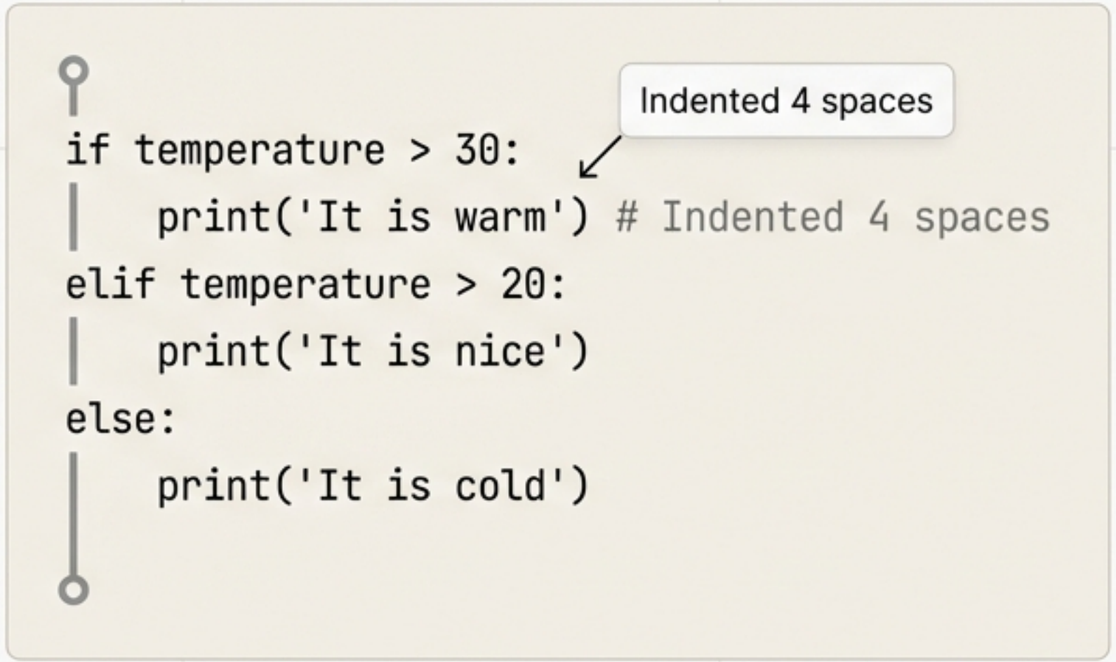
Case sensitive. ‘b’ (98) != ‘B’ (66)

Roboto

Python compares strings character by character based on their numeric (ASCII) value.

Conditional Statements

The Structure



The Ternary Operator (Pro Tip)

Writing cleaner one-liners.

Old Way:

```
if age >= 18:
    msg = 'Eligible'
else:
    msg = 'Not Eligible'
```

Pro Way:

```
msg = 'Eligible' if age >=
18 else 'Not Eligible'
```

Logical Operators

Operators

AND
Both True

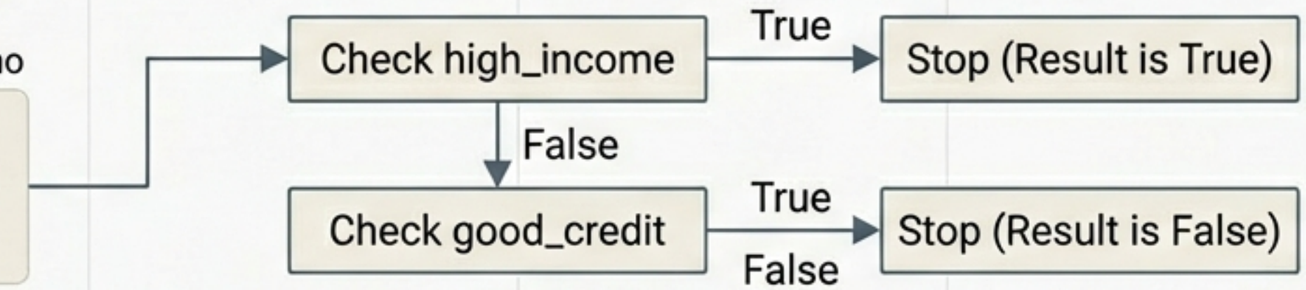
OR
At least one True

NOT
Inverses

Short Circuit Evaluation

JetBrains Mono

```
if high_income or
   good_credit:
```



Python optimizes by stopping evaluation as soon as the result is determined.

Chaining Comparison

Mathematical Syntax in Python

Bad:

JetBrains Mono

```
if age >= 18 and age < 65:
```

Good:

JetBrains Mono

```
if 18 <= age < 65:
```

Loops: The Engine of Repetition

For Loops

Iterating over sequences.

JetBrains Mono

```
for number in range(3):
    print(number)
```

```
0
1
2
```

range(start, end) e.g., range(0, 3) → 0, 1, 2

start, inclusive end, exclusive

While Loops

Condition-based iteration.

JetBrains Mono

```
while command != 'quit':
    command = input('> ')
```

Warning:

Ensure a termination condition to avoid Infinite Loops.

Advanced Loop Controls

Keywords

break
Terminates the loop immediately.

continue
Skips current iteration, moves to next.

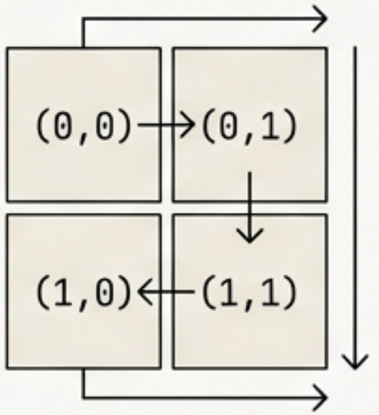
The For-Else Block

```
for name in names:
    if name.startswith('J'):
        print('Found')
        break
else:
    print('Not Found')
```

Executes ONLY if the loop completes without a break.

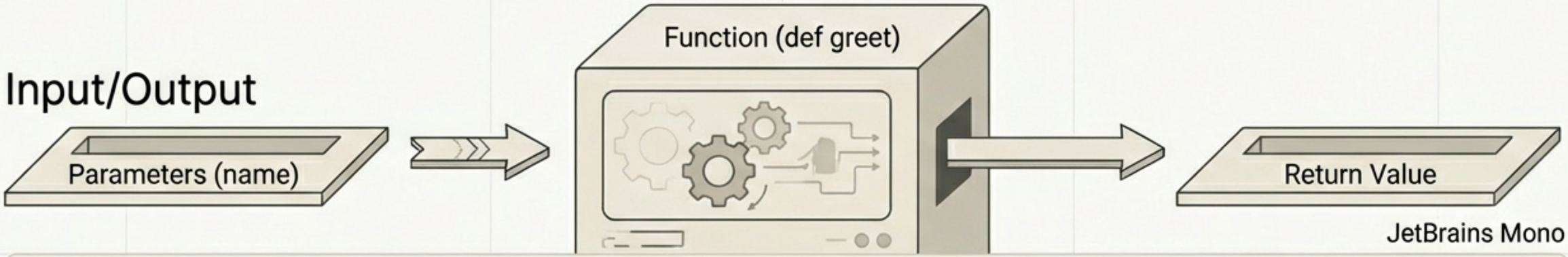
Nested Loops

```
for x in range(2):
    for y in range(2):
        print(f'({x}, {y})')
```



Functions: The Building Blocks

Input/Output



JetBrains Mono

```
1  def greet(name):      ⟵ Parameter
2      return f'Hi {name}'
3
4  print(greet('Mosh')) ⟵ Argument
```

Return vs. Print

Print
Displays text to the console (A task).

Return
Sends a value back to the caller (A calculation).

Note: Functions return 'None' by default if no return statement is used.

Advanced Arguments

Keyword Arguments (Readability)

```
increment(2, by=1)
```

JetBrains Mono

Explicitly names the argument for clarity.

Default Arguments (Flexibility)

```
def increment(number, by=1):
```

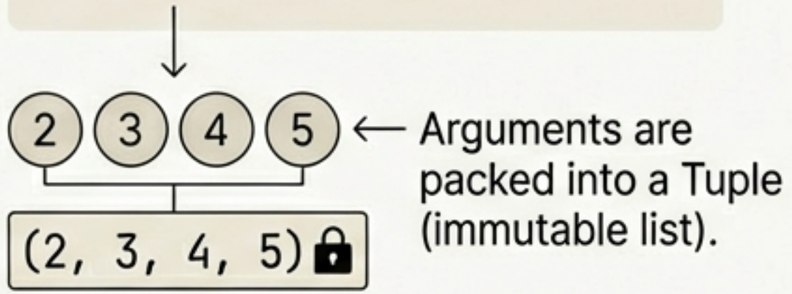
The 'by' parameter is optional. If missing, it defaults to 1.

Variable Arguments (Tuples)

```
def multiply(*numbers):
```

The * allows passing any number of arguments.

```
multiply(2, 3, 4, 5)
```



Arguments are packed into a Tuple (immutable list).