Python Fundamentals – Day 1

A beginner's guide to mastering Python basics

CHAPTER 1

What is Python?

Python is a versatile, high-level programming language known for its simplicity and readability. Created by Guido van Rossum in 1991, it emphasizes code clarity with minimal syntax.

Python's design philosophy makes it perfect for beginners while remaining powerful enough for experts.

Why Python Matters Today

1

Most Popular

Ranked #1 in TIOBE Index

8.2M

Global Developers

Active Python community

15%

Job Growth

Annual increase in Python roles

Python's Real-World Applications

Web Development

Build websites with Django and Flask frameworks powering platforms like Instagram and Spotify.

Data Science

Analyze massive datasets using pandas and NumPy to extract meaningful insights.

AI & Machine Learning

Create intelligent systems with TensorFlow and scikit-learn for predictive models.

More Python Use Cases

Automation

Automate repetitive tasks and workflows to save hours of manual work daily.

Game Development

Build games using Pygame library for interactive entertainment experiences.

Cloud Computing

Manage cloud infrastructure on AWS, Azure, and Google Cloud platforms.

CHAPTER 2

Getting Started: Installation

Before writing Python code, you need to install it on your machine. Let's walk through the setup process step by step.

Installation Steps

01

Visit python.org

Navigate to the official Python website and locate the Downloads section.

02

Download Latest Version

Choose Python 3.12 or newer for your operating system (Windows, Mac, or Linux).

03

Run Installer

Execute the downloaded file and check "Add Python to PATH" during installation.

04

Verify Installation

Open terminal and type: python --version to confirm successful setup.

Choosing Your Code Editor

VS Code

Free, lightweight editor with excellent Python extensions and debugging tools.

PyCharm

Professional IDE designed specifically for Python development with intelligent features.

Jupyter Notebook

Interactive environment perfect for data science and learning experiments.

Your First Python Program

The traditional first program in any language. This simple line demonstrates Python's readable syntax.

Type this in your editor and save as hello.py, then run it from terminal.

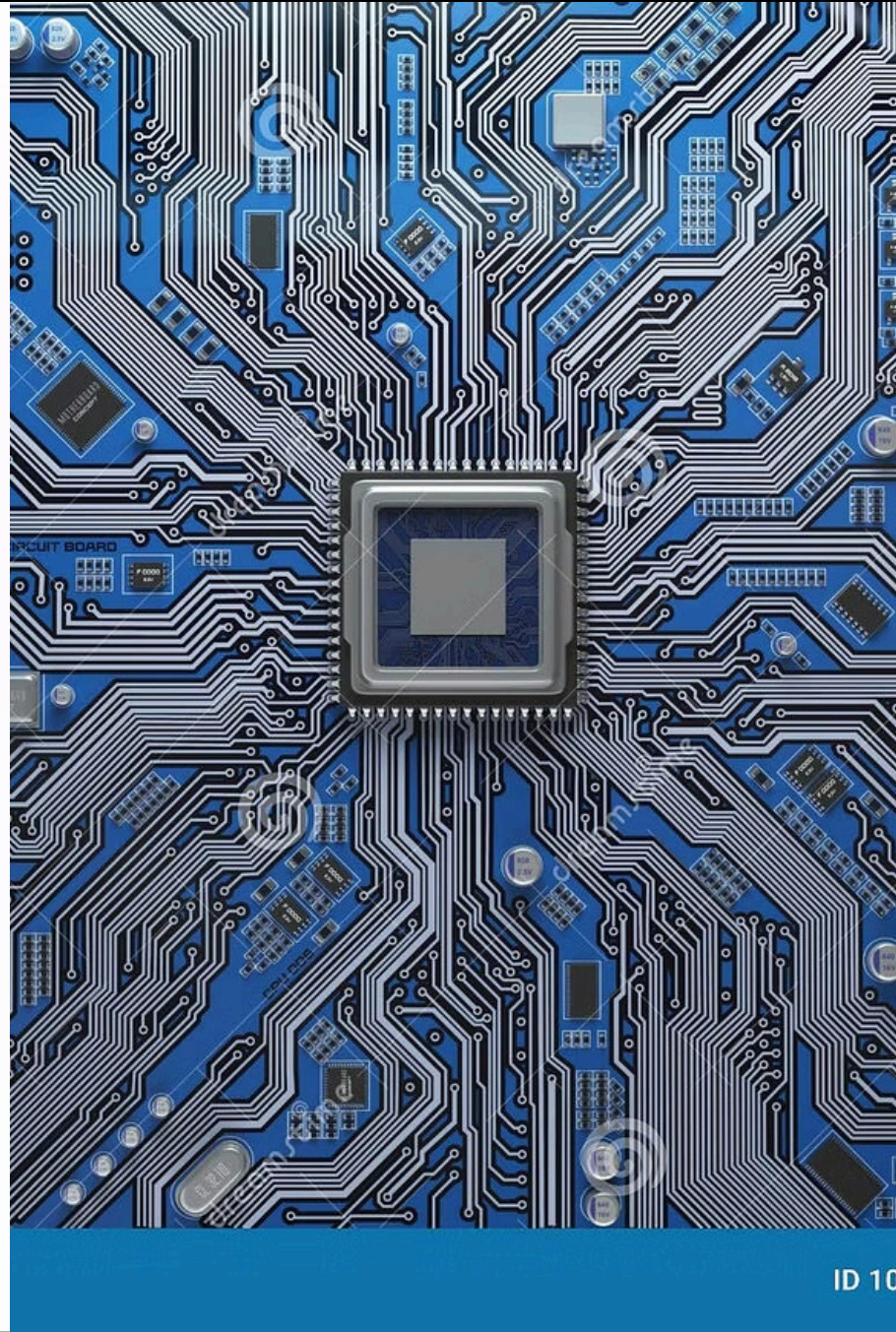
```
print("Hello, World!")
```

Output: Hello, World!

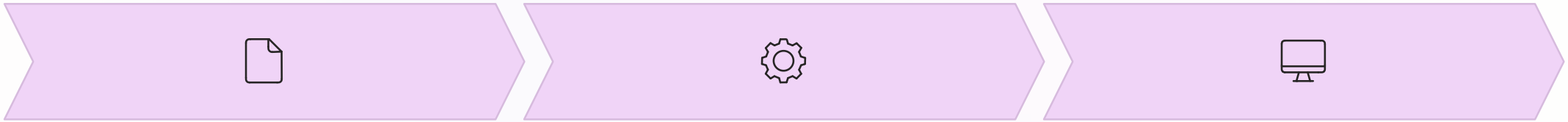
CHAPTER 3

How Python Executes Code

Understanding the execution flow helps you debug and write better code.



Python Execution Process



Source Code

You write .py files with human-readable Python code.

Compilation

Python converts code to bytecode (.pyc files) automatically.

Interpretation

PVM executes bytecode line by line to produce results.

Interactive vs Script Mode

Interactive Mode

Type `python` in terminal to enter REPL (Read-Eval-Print Loop). Perfect for testing quick commands and experimenting.

```
>>> 2 + 2
4
```

Script Mode

Write code in .py files and execute with `python filename.py`. Best for larger programs and projects.

```
python hello.py
```

CHAPTER 4

Python Syntax Rules

Python's syntax emphasizes readability through simplicity and consistency.

The Power of Indentation

Unlike other languages that use braces, Python uses indentation to define code blocks. This enforces clean, readable code.

Use 4 spaces (not tabs) as the standard indentation level.

```
# Correct
if True:
    print("Indented!")

# Wrong - will cause error
if True:
print("Not indented")
```

Indentation errors are the most common mistake for beginners!

Comments: Documenting Your Code

Single-Line Comments

```
# This is a comment
x = 5  # Comments can follow code
```

Use for brief explanations and notes within your code.

Multi-Line Comments

```
"""
This is a multi-line comment
Used for longer explanations
or documentation
"""
```

Perfect for function descriptions and detailed documentation.

Case Sensitivity Matters

Python treats uppercase and lowercase letters as different characters.

Variable names, function names, and keywords are all case-sensitive.

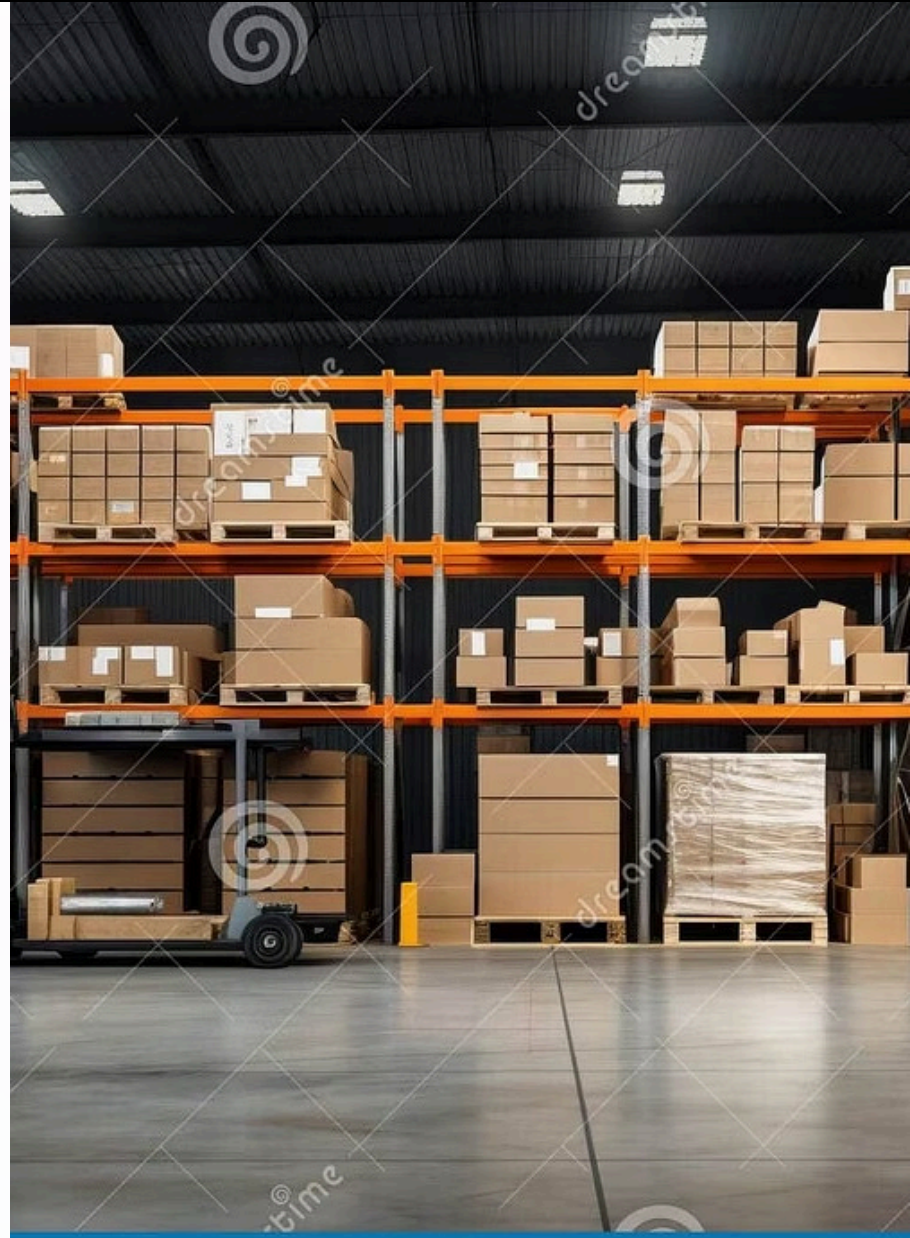
```
name = "Alice"
Name = "Bob"
NAME = "Charlie"

print(name) # Output: Alice
print(Name) # Output: Bob
print(NAME) # Output: Charlie
```

CHAPTER 5

Variables in Python

Variables are containers that store data values for later use in your program.



Creating Variables

Python uses dynamic typing—you don't need to declare variable types explicitly. The interpreter determines the type automatically.

Assignment uses the equals sign (=) to store values in variable names.

```
age = 25
name = "John"
salary = 50000.50
is_employed = True

print(age) # Output: 25
print(name) # Output: John
```

Variable Naming Rules

Must start with letter or underscore

Valid: name, _count, user1 | Invalid: 1user, $price

Can contain letters, numbers, underscores

Valid: user_name, total2, _temp | Invalid: user-name, total@2

Cannot use Python keywords

Reserved words like if, for, while, class cannot be variable names.

Use descriptive names

Prefer total_price over tp for better code readability.

Dynamic Typing in Action

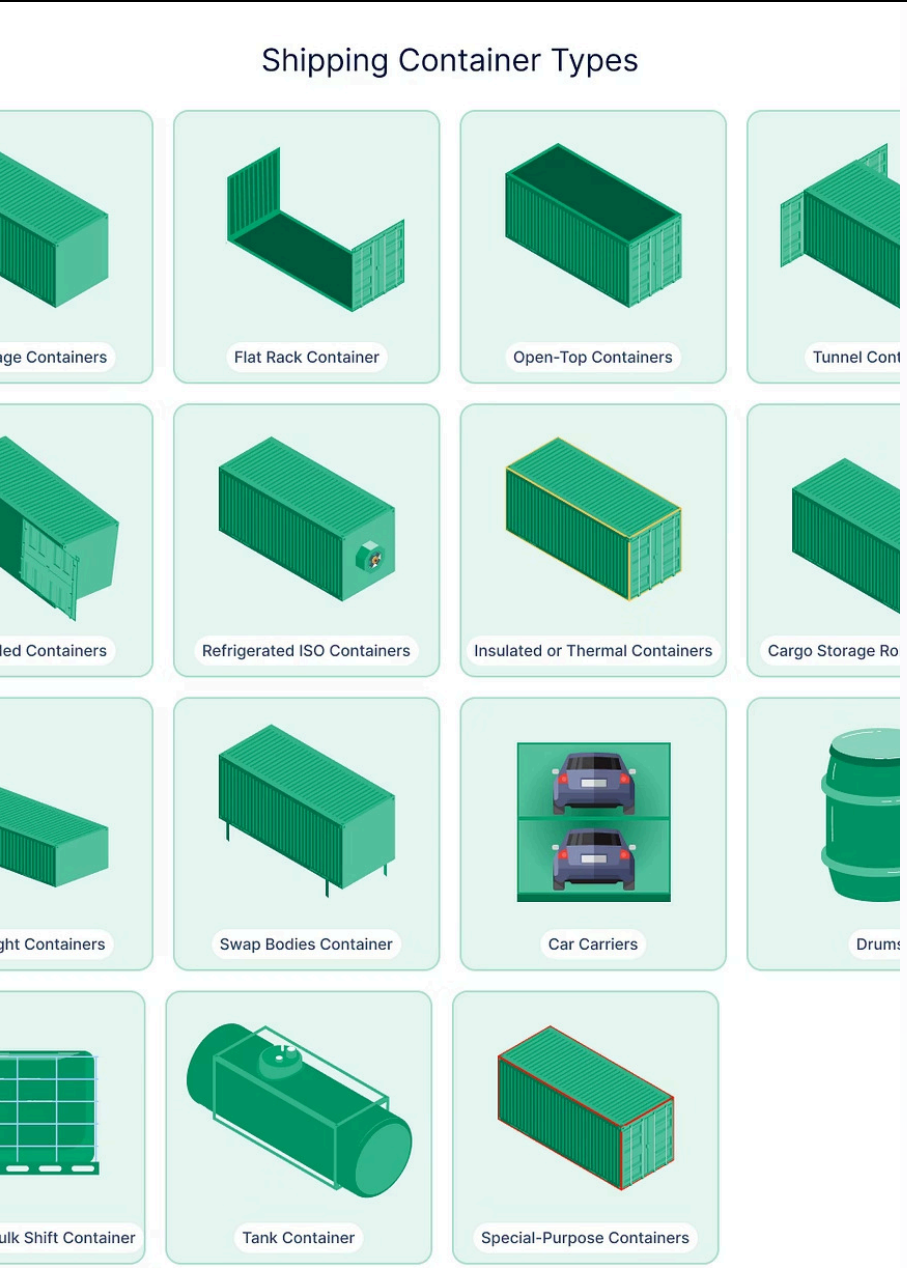
Variables can change types during program execution. Python automatically adjusts the type based on the assigned value.

This flexibility makes Python easy to learn but requires careful attention.

```
x = 10 # x is integer
print(x) # Output: 10

x = "Hello" # x is now string
print(x) # Output: Hello

x = 3.14 # x is now float
print(x) # Output: 3.14
```



CHAPTER 6

Python Data Types

Data types define what kind of values can be stored and what operations can be performed.

Core Data Types Overview

Integers (int)

Whole numbers without decimals: 42, -10, 0, 1000

Floats (float)

Numbers with decimal points: 3.14, -0.5, 2.0

Strings (str)

Text enclosed in quotes: "Hello", 'Python', "123"

Booleans (bool)

Truth values: True or False (capitalized)

Working with Integers

```
age = 25
year = 2024
temperature = -5
count = 0

# Integer operations
total = age + 10        # Output: 35
difference = year - 10  # Output: 2014
product = age * 2       # Output: 50
```

Integers have unlimited precision in Python—no overflow errors unlike other languages!

Understanding Floats

Floating-point numbers represent decimal values. Use them for measurements, calculations, and scientific notation.

Be aware of precision limits in floating-point arithmetic.

```
price = 19.99
pi = 3.14159
scientific = 2.5e3 # 2500.0

# Float operations
total = price * 2 # 39.98
half = pi / 2 # 1.570795
```

Strings: Working with Text

1

Single or Double Quotes

```
name = 'Alice'
message = "Hello, World!"
```

2

String Concatenation

```
first = "John"
last = "Doe"
full = first + " " + last
```

3

String Methods

```
text = "python"
upper = text.upper() #
"PYTHON"
length = len(text) # 6
```

Multi-Line Strings

Use triple quotes (""" or ''') to create strings that span multiple lines.

Useful for long text blocks, documentation, and formatted output.

```
message = """
This is a multi-line string.
It can span several lines.
Formatting is preserved!
"""

print(message)
```

Boolean Values

True and False

```
is_active = True
is_empty = False

print(is_active)  # True
```

Always capitalize: True and False, not true/false.

Boolean in Comparisons

```
x = 10
y = 5

result = x > y   # True
equal = x == y   # False
```

Comparison operators return boolean values.

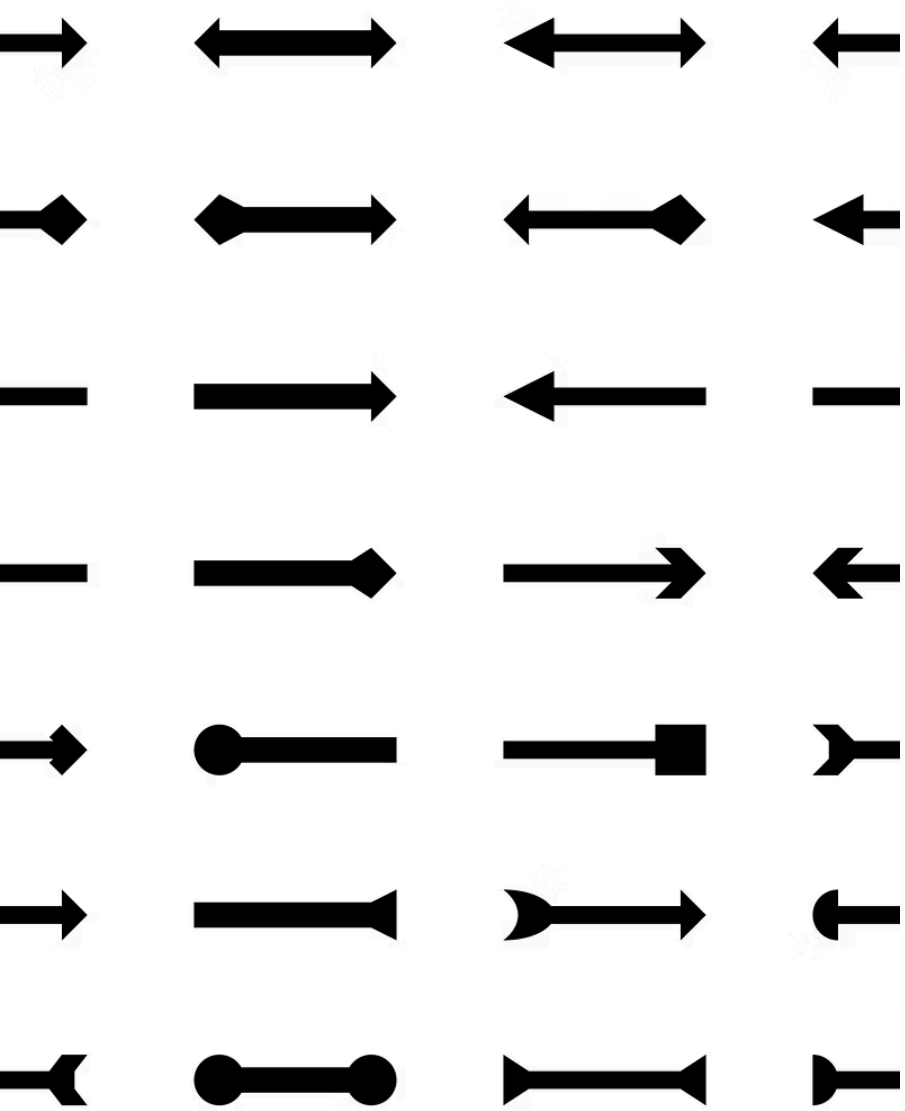
Checking Data Types

Use the `type()` function to check what data type a variable holds.

Essential for debugging and understanding your code's behavior.

```
age = 25
price = 19.99
name = "Alice"
active = True

print(type(age)) #
print(type(price)) #
print(type(name)) #
print(type(active)) #
```



CHAPTER 7

Type Casting

Type casting converts values from one data type to another explicitly.

Converting Between Types

1

To Integer

```
x = int("10")    # 10
y = int(3.9)     # 3
z = int(True)    # 1
```

2

To Float

```
a = float("3.14")  # 3.14
b = float(5)       # 5.0
c = float(True)    # 1.0
```

3

To String

```
s1 = str(100) # "100"
s2 = str(3.14) # "3.14"
s3 = str(False) # "False"
```

Practical Type Casting Example

```
# User input (always returns string)
age_input = "25"
height_input = "5.9"

# Convert to appropriate types
age = int(age_input) # 25
height = float(height_input) # 5.9

# Perform calculations
next_year = age + 1 # 26
double_height = height * 2 # 11.8

# Convert back for display
message = "Age: " + str(next_year)
print(message) # Age: 26
```

Type Casting Warnings

Invalid Conversions Cause Errors

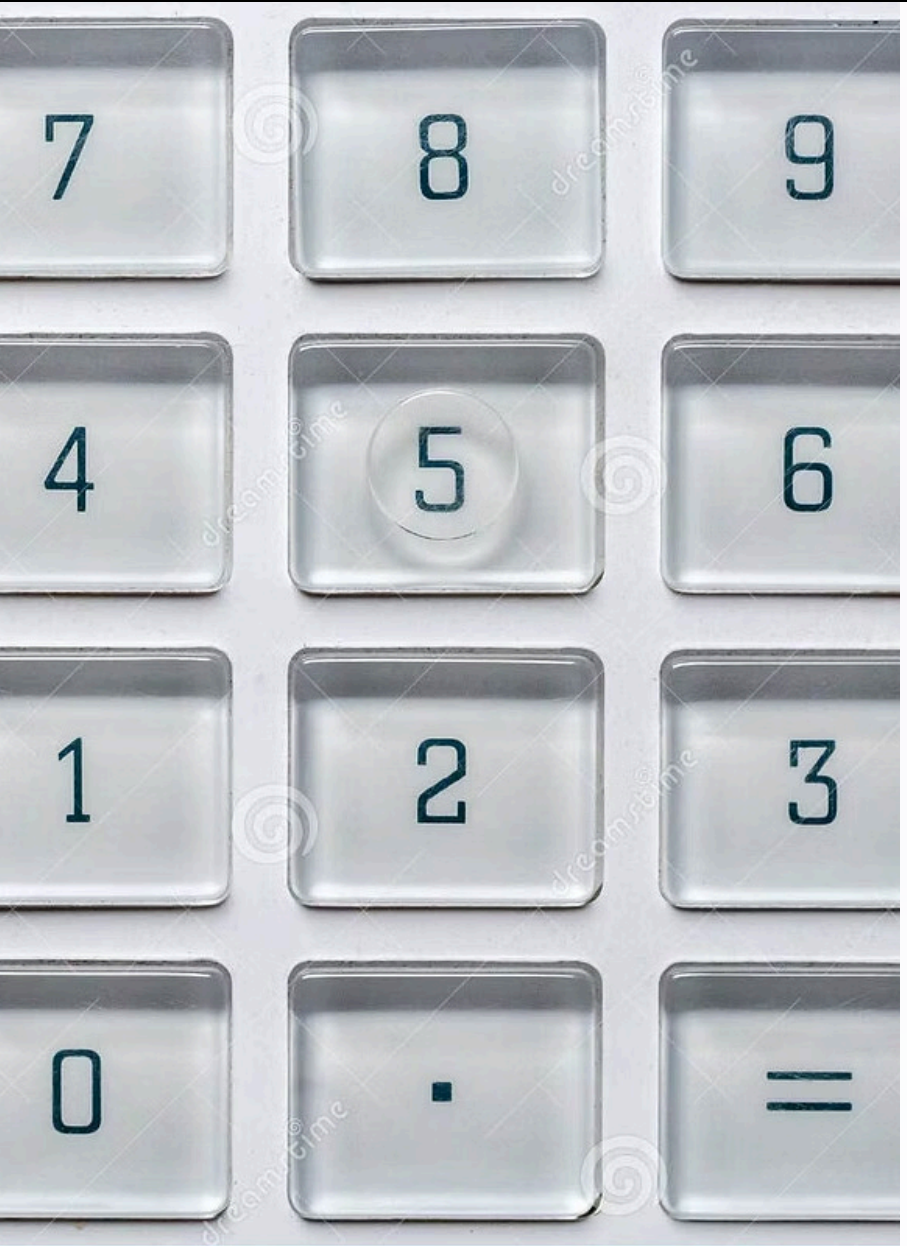
```
# This will crash!
x = int("hello")  # ValueError
```

Make sure strings contain valid numeric characters before converting.

Data Loss with int()

```
value = int(3.99)  # 3 (not 4!)
```

Converting float to int truncates decimals—it doesn't round.



CHAPTER 8

Operators and Expressions

Operators perform operations on variables and values to create expressions.

Arithmetic Operators

Addition (+)

```
x = 10 + 5   # 15
```

Subtraction (-)

```
x = 10 - 5   # 5
```

Multiplication (*)

```
x = 10 * 5   # 50
```

Division (/)

```
x = 10 / 5 # 2.0
```

More Arithmetic Operators

Floor Division (//)

```
x = 10 // 3  # 3
```

Divides and rounds down to nearest integer.

Modulus (%)

```
x = 10 % 3   # 1
```

Returns remainder after division.

Exponentiation (**)

```
x = 2 ** 3   # 8
```

Raises number to a power.

Comparison Operators

Equality & Inequality

```
x = 10
y = 5

x == y   # False (equal to)
x != y   # True (not equal to)
```

Greater & Less Than

```
x > y    # True (greater than)
x < y    # False (less than)
x >= y   # True (greater or equal)
x <= y   # False (less or equal)
```

All comparison operators return boolean values (True or False).

Assignment Operators

```
# Basic assignment
x = 10

# Compound assignment operators
x += 5   # Same as: x = x + 5  (Result: 15)
x -= 3   # Same as: x = x - 3  (Result: 12)
x *= 2   # Same as: x = x * 2  (Result: 24)
x /= 4   # Same as: x = x / 4  (Result: 6.0)
```

Compound operators provide shortcuts for common operations—cleaner and more efficient code.

CHAPTER 9

Common Beginner Mistakes

Learn from these frequent errors to accelerate your Python journey.

Top Mistakes to Avoid

1 **Forgetting indentation**

Python requires consistent indentation—always use 4 spaces for code blocks.

2 **Misspelling variable names**

userName vs username—case matters! Keep naming consistent throughout.

3 **Using = instead of ==**

Single = assigns values; double == compares them in conditions.

4 **Incorrect type operations**

Can't add string and integer directly—use type casting first.

WHAT'S NEXT

PCAP Certification Alignment

Today's content aligns with PCAP (Certified Associate in Python Programming) exam objectives covering fundamental Python concepts.

Practice daily to master these basics before moving to advanced topics like loops, functions, and data structures.

01

Master Variables

Practice creating and manipulating different data types.

02

Experiment with Operators

Write expressions using arithmetic and comparison operators.

03

Build Small Programs

Combine today's concepts into simple calculators and converters.