PART 1 OF 8

Getting Started with Python

Welcome to your first Python session! In this booklet, you will discover what Python is, how to write and run your first program, and how to work with variables, strings, and numbers. By the end, you will have a solid foundation to build on in every session that follows.

Learning Goals

By the end of this session, you should be comfortable with all of the following:

1

What Python Is

Explain what Python is and why it is widely used.

2

Scripts & Interpreter

Describe what a Python script is and how the interpreter runs code.

3

First Program

Create and run a simple Python program that prints output.

4

Variables

Store values in variables and reassign them.

5

Strings & f-Strings

Use basic string methods and f-strings to build messages.

6

Math & Numbers

Perform calculations with integers and floats, including division and exponents.

CONCEPT 1

What Python Is

Why Python?

Python is one of the most popular programming languages in the world. It is praised for being easy to read and learn – almost like writing plain English instructions.

A complete Python program can be just one line:

```
print("Python is ready!")
```

Key Characteristics

- **Readable** – often described as "executable pseudocode"
- **High-level & interpreted** – no manual compilation step needed
- **Cross-platform** – works on Windows, macOS, and Linux
- **Huge ecosystem** – thousands of libraries for any task

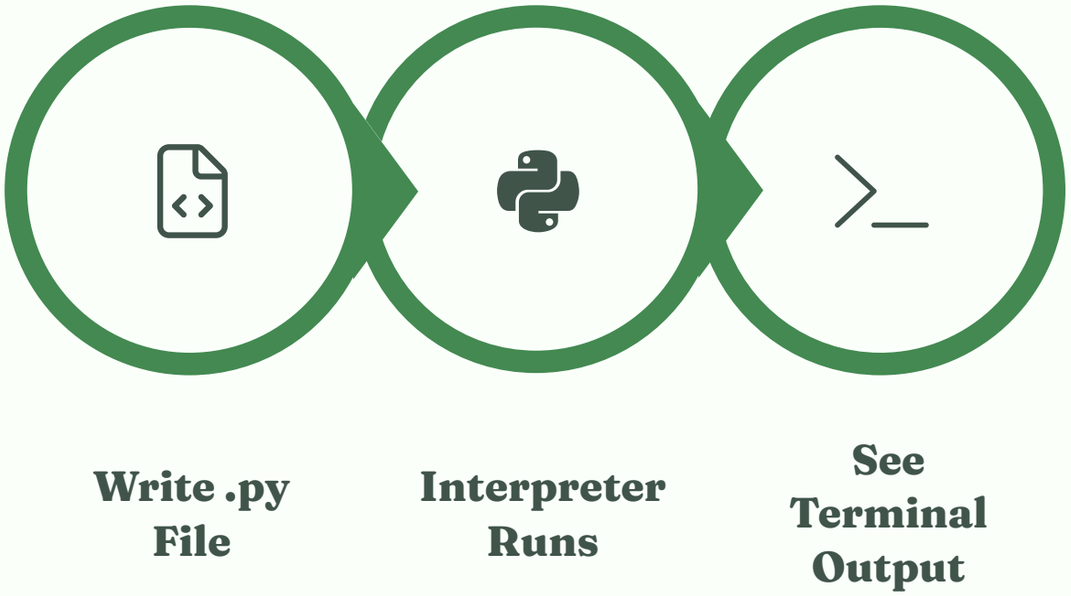
Common Uses

- Web development
- Data analysis & visualization
- Machine learning / AI
- Automation & scripting

CONCEPT 2

The Python Programming Environment

You can write Python in many editors – IDLE, VS Code, PyCharm, or online notebooks. No matter which you choose, the workflow is always the same three-step process:



This top-to-bottom execution is fundamental. Python always runs your code in order, starting from line 1.

```
print("Line 1")
print("Line 2")
```

Python prints `Line 1` first, then `Line 2` – always in the order you write them.

CONCEPT 3

Your First Program and `print()`

Hello, World!

The classic first program in any language. In Python, use the built-in `print()` function to display output:

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

`print(...)` displays output. `"Hello, world!"` is a **string** – text inside quotes.

Understanding Exit Codes

Exit code 0 – program finished normally, no errors.

Exit code 1 – an error occurred; Python will show a **traceback**.

Tracebacks include:

- The filename and line number
- The error type (e.g., `SyntaxError`, `NameError`)

Case sensitivity matters! `Print("Hello")` causes a `NameError`. Python only recognizes the lowercase `print`.

CONCEPT 4

Comments and Style Notes

Comments are notes for humans – Python ignores them completely when running your code. Use them to explain *why* you wrote something, not just what it does.

Single-line Comment

Start with a `#` symbol. Everything after it on that line is ignored.

```
# This is a comment
print("This will run.")
```

Multi-line String (Triple Quotes)

Sometimes used like a comment in beginner examples. It is actually a string literal – it won't print unless you explicitly print it.

```
"""
This is a multi-line string.
Often used as a comment.
"""
print("Done.")
```

CONCEPT 5

Variables — Storing and Reusing Values

A **variable** is like a label attached to a piece of data. Create one by assigning a value with `=`.

```
message = "Hello, Python!"
print(message)
```

Assigning

Use `=` to store a value. Python creates the variable automatically – no special keyword needed.

```
message = "Hello!"
```

Reassigning

You can give a variable a new value at any time. Python always uses the most recent value.

```
message = "Hello again!"
print(message)
```

⚠️ Changing Type

Python allows changing the type stored in a variable. This is valid but can cause confusing bugs for beginners.

```
message = "Hello!"
message = 29.5
```

CONCEPT 6

Variable Naming Rules

✅ Valid Names

```
user_name = "alex"
_tempVar = 10
greeting_1 = "hi"
```

- Use letters, numbers, underscores
- Start with a letter or underscore
- No spaces allowed
- Prefer lowercase and descriptive names

❌ Invalid Names

- `1stname` – cannot start with a number
- `user name` – spaces are not allowed
- `for`, `print`, `class` – reserved Python keywords

Best Practice

Choose names that describe the data they hold. `user_name` is much clearer than `u`. Good names make code self-documenting.

CONCEPT 7

Strings and String Methods

A **string** is text enclosed in quotes: `"hello"` or `'hello'`. Python provides many built-in methods to transform strings. Three of the most common:

.title()

Converts to Title Case – first letter of each word is capitalized.

```
"john doe".title()
# → "John Doe"
```

.upper()

Converts all characters to UPPERCASE.

```
"john doe".upper()
# → "JOHN DOE"
```

.lower()

Converts all characters to lowercase.

```
"JOHN DOE".lower()
# → "john doe"
```

You can store the result in a new variable (`name_title = name.title()`) or print it directly (`print(name.title())`). The original variable is unchanged unless you reassign it.

CONCEPT 8

f-Strings — Formatted Strings

f-strings let you embed variables and expressions directly inside a string. Add an `f` before the opening quote, then wrap variables in `{}`.

```
first_name = "john"
last_name = "doe"

full_name = f"{first_name} {last_name}"
print(f"Hello, {full_name.title()}!")
```

What Happens Step by Step

1. Combines first and last name with a space in between
2. Stores the combined name in `full_name`
3. Calls `.title()` inside the f-string to format neatly
4. Prints: `Hello, John Doe!`

Why Use f-Strings?

f-strings are the modern, preferred way to build dynamic messages in Python. You can even call methods or do math inside the `{}`:

```
price = 9.99
print(f"Total: {price * 2}")
```

CONCEPT 9

Tabs and Newlines — `\t` and `\n`

Special **escape characters** let you control how text appears on screen without writing multiple `print()` calls.

`\n` — New Line

Moves the cursor to the next line, just like pressing Enter.

`\t` — Tab

Adds a horizontal indent, useful for creating clean, aligned output.

Example combining both:

```
print("Languages:\n\tPython\n\tC\n\tJavaScript")
```

This prints a small indented list with each language on its own line – all from one `print()` call.

CONCEPT 10

Stripping Whitespace

Whitespace – spaces, tabs, newlines – often creeps into strings from user input or file data. Python gives you three methods to clean it up:

1

`.lstrip()`

Removes whitespace on the **left** side only.

2

`.rstrip()`

Removes whitespace on the **right** side only.

3

`.strip()`

Removes whitespace on **both sides**.

```
name = "  python  "

print(name.rstrip())   # "  python"
print(name.lstrip())   # "python  "
print(name.strip())    # "python"
```

CONCEPT 11

Removing Prefixes and Suffixes

The Problem

Sometimes strings have a common prefix or suffix you want to strip away – like `https://` from a URL.

The Methods

- `.removeprefix("...")` – removes from the start
- `.removesuffix("...")` – removes from the end

Example

```
url = "https://example.com"
print(url.removeprefix("https://"))
# → example.com
```

⚠️ Important

These methods do **not** change the original variable. To save the result, you must reassign:

```
url = url.removeprefix("https://")
```

CONCEPT 12

String Quote Errors

Strings must open and close with **matching quotes**. One of the most common beginner errors is accidentally breaking a string with an apostrophe.

❌ This Causes a SyntaxError

```
message = 'One of Python's strengths is...'
```

The apostrophe in Python's closes the string early, breaking it.

✅ Fix 1 — Use Double Quotes

```
message = "One of Python's strengths is..."
```

The outer double quotes mean the apostrophe inside is harmless.

✅ Fix 2 — Escape the Apostrophe

```
message = 'One of Python\'s strengths is...'
```

The backslash \' tells Python to treat the apostrophe as text, not the end of the string.

CONCEPT 13

Numbers — Integers and Floats

Python uses two main number types. You don't need to declare which one – Python figures it out automatically based on the value you write.

Integers (`int`)

Whole numbers – no decimal point.

```
-2   0   42
```

```
print(type(5))
# <class 'int'>
```

Floats (`float`)

Numbers with a decimal point.

```
3.14   0.0   -2.5
```

```
print(type(2.0))
# <class 'float'>
```

Tip

Use `type()` any time you want to check what kind of value a variable holds.

CONCEPT 14

Basic Math and Order of Operations

Addition

```
2 + 3  # 5
```

Subtraction

```
3 - 2  # 1
```

Multiplication

```
2 * 3  # 6
```

Division

```
3 / 2  # 1.5
```

Order of operations works just like in math class – multiplication and division happen before addition and subtraction. Use parentheses to change the order:

```
print(2 + 3 * 4)    # 14  (multiplication first)
print((2 + 3) * 4)  # 20  (parentheses first)
```

CONCEPT 15

Division Details — `/` vs `//`

The Rules

- `/` always returns a **float**, even if the result is a whole number
- `//` does **integer division** – truncates the decimal part
- Mixing `int` and `float` in one expression always produces a `float`

Examples

```
print(4 / 2)    # 2.0  ← float!
print(5 // 2)   # 2    ← integer division
print(5 / 2)    # 2.5
print(1 + 2.0)  # 3.0  ← mixed = float
```

Common Mistake

Beginners often expect `4 / 2` to return `2`. In Python 3, it always returns `2.0`. Use `//` if you need a whole number result.

CONCEPT 16

Exponents with `**`

Use the double-asterisk `**` operator to raise a number to a power. This is Python's exponentiation operator.

Syntax

`base ** exponent`

$2^3 = 8$

```
print(2 ** 3)  # 8
```

$3^2 = 9$

```
print(3 ** 2)  # 9
```

$10^6 = 1{,}000{,}000$

```
print(10 ** 6)  # 1000000
```

The formula for 2^3: $2^3 = 8$. Python's `**` maps directly to this mathematical notation.

CONCEPT 17

Extra Number Tips

Readable Large Numbers with Underscores

Python ignores underscores in integers – use them like commas to make big numbers easier to read.

```
universe_age =
14_000_000_000
print(universe_age)  #
14000000000
```

Multi-Assignment on One Line

Assign several variables at once, separated by commas.

```
x, y, z = 0.1, 0, 0
print(x)  # 0.1
```

Constants by Convention (ALL CAPS)

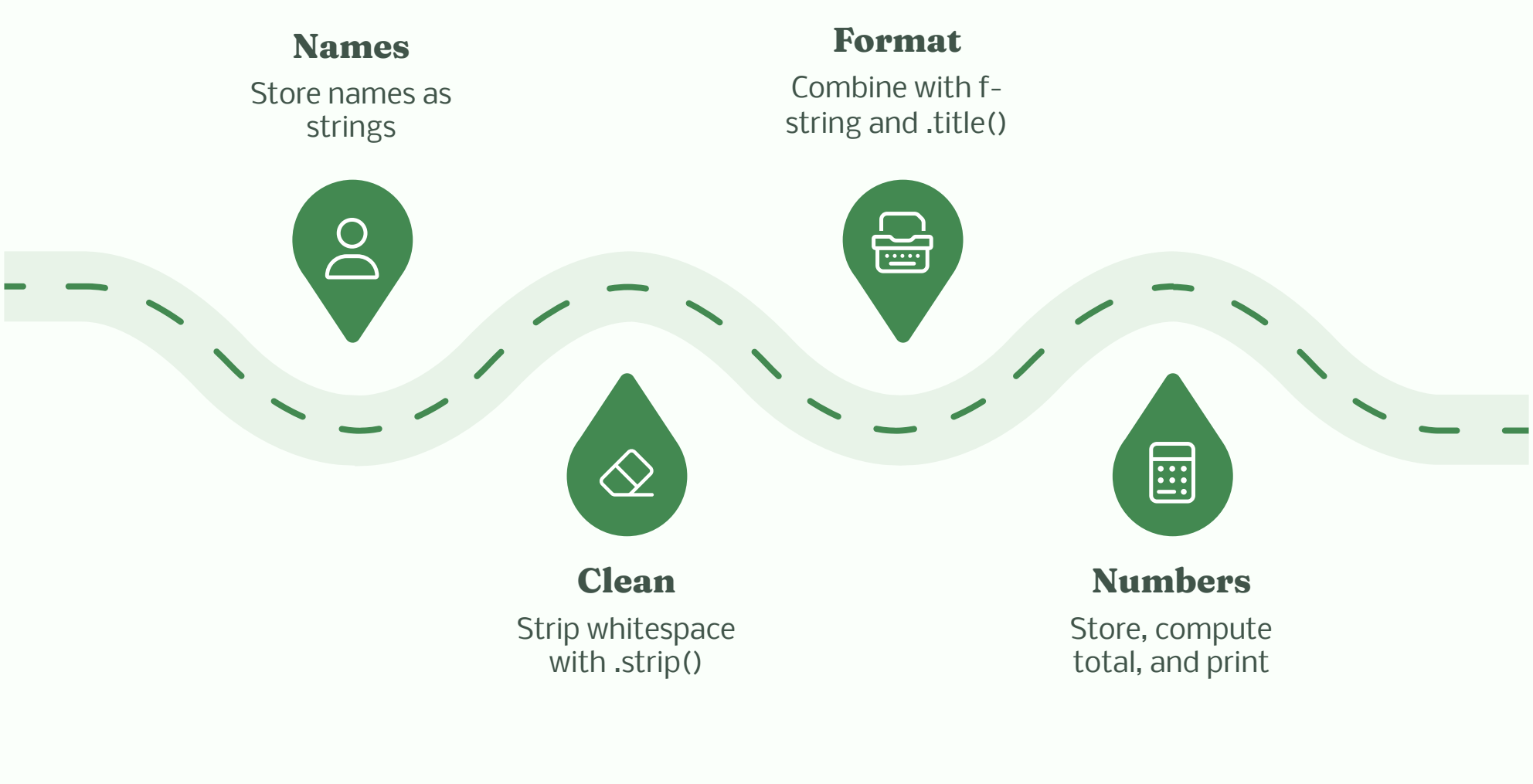
Python has no enforced constants. By convention, ALL CAPS names signal "do not change this value."

```
MAX_CONNECTIONS = 5000
```

Step-by-Step Example

Here is a small, runnable program that combines everything from this session – variables, string cleaning, f-strings, and number math:

```
# A beginner-friendly program combining key basics

first_name = "  aLex "
last_name = "  chen  "

# Clean up whitespace
first_name = first_name.strip()
last_name = last_name.strip()

# Build a full name and format it nicely
full_name = f"{first_name} {last_name}"
full_name = full_name.title()

# Work with numbers
items = 5
price_each = 2.5
total = items * price_each

# Display formatted output using \n and \t
print("Receipt")
print("\tCustomer:", full_name)
print(f"\tItems: {items}")
print(f"\tPrice each: {price_each}")
print(f"\tTotal: {total}")
```



Common Mistakes to Avoid

❌ Wrong Capitalization of Built-ins

Writing `Print()` instead of `print()` causes a `NameError`. Python is case-sensitive – built-in functions are always lowercase.

❌ Unmatched Quotes in Strings

Forgetting the closing quote or using an apostrophe inside single-quoted strings causes a `SyntaxError`.

❌ Invalid Variable Names

Starting with a number (`1stname`) or including spaces (`user name`) will cause a `SyntaxError`.

❌ Expecting `/` to Return an Integer

`4 / 2` returns `2.0`, not `2`. Use `//` for integer division.

❌ Forgetting Parentheses on Methods

`name.title` refers to the method itself – it doesn't call it. You need `name.title()` with parentheses to actually run it.

Practice Tasks

Work through these tasks on your own to reinforce what you have learned. Try each one without looking back at the concepts first.

01

Reassigning Variables

Create a variable `message` with a short sentence and print it. Then reassign it to a new sentence and print again.

02

String Methods

Store a name in a variable. Print it in lowercase, uppercase, and title case using the appropriate methods.

03

f-String Greeting

Create `first_name` and `last_name` variables. Use an f-string to print: `Hello, <Full Name>!` with `.title()` applied.

04

Whitespace Stripping

Make a string like `" city "` and demonstrate `rstrip()`, `lstrip()`, and `strip()`, printing each result.

05

Arithmetic to 8

Write four lines of code using `+`, `-`, `*`, and `/` that each produce the result `8`, and print each one.

Exploration Questions

These questions encourage you to experiment beyond the examples. There are no single right answers – try things out in your editor and observe what happens.

1

Whitespace Experiment

What changes if you replace `strip()` with `rstrip()` in the step-by-step example? Try names with extra spaces on both sides and observe the difference.

2

Division Patterns

Test `//` and `/` with different numbers (e.g., `7 / 2`, `7 // 2`, `9 / 3`, `9 // 3`). What pattern do you observe?

3

Changing Variable Types

Store a number in a variable, then later store a string in the same variable. In a longer program, what kinds of confusion might this cause?

Session Summary

You have covered all the building blocks of Python in this first session. Here is what you learned:

Scripts & Interpreter

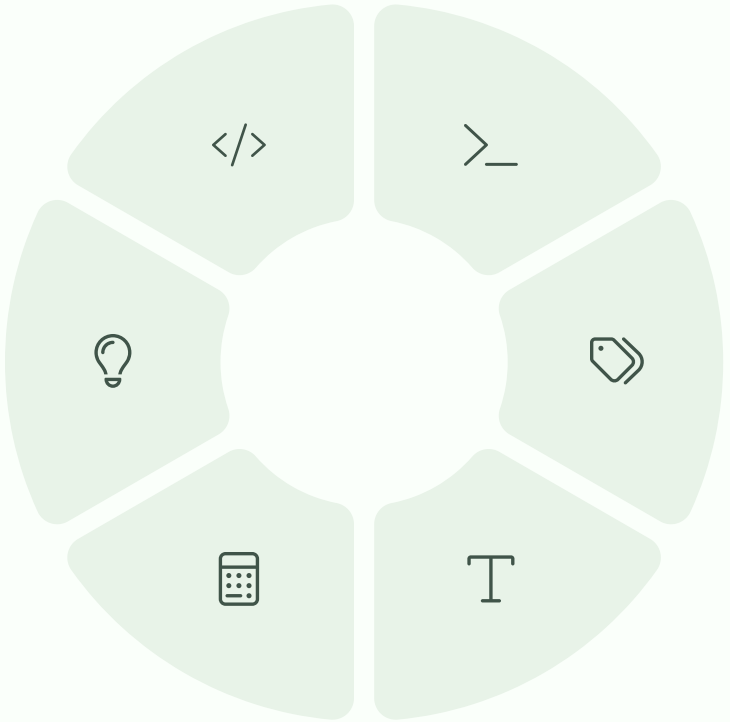
Python runs `.py` files line by line, top to bottom.

Good Habits

Comments, descriptive names, and consistent style make code readable.

Numbers & Math

Integers, floats, operators, order of operations, and `**` for powers.



print() & Errors

Use `print()` for output. Read tracebacks to find and fix errors.

Variables

Store and reassign values with `=`. Follow naming rules.

Strings

Methods like `.title()`, `.strip()`, and f-strings for dynamic text.

🎉 Great work completing Part 1 of 8! In the next session, you will build on these basics and explore more of what Python can do. Make sure to complete the Practice Tasks before moving on.