Your First Python Program: From Print to Program Structure

Learning Objectives

By the end of this guide, you will understand:

- What a Python program actually is
- How `print()` displays information
- What statements and expressions mean
- How Python reads a program
- How strings, numbers, and Boolean values appear in code
- How function calls work at a basic level
- How to write your first useful Python programs
- Common beginner mistakes and how to fix them
- How to structure simple Python code clearly

---

1. Your First Program Is More Than “Hello World”

When people start programming, the first example is usually:

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

It looks extremely simple.

But this one line introduces several important programming concepts:

- a function
- a function call
- an argument
- a string
- parentheses
- a statement
- output

Understanding what is happening here is more valuable than simply memorizing the line.

Programming is fundamentally about giving a computer precise instructions.

For example, imagine building a developer dashboard.

You might eventually want Python to:

1. display the user's name

2. calculate statistics
3. process data
4. make decisions
5. communicate with an API
6. store information
7. generate reports

Every large Python application starts from small instructions like these.

---

2. What Is a Python Program?

A Python program is a collection of instructions written using Python's syntax to accomplish a task.

A tiny program:

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

A slightly larger program:

```
print("Welcome to Python")
print("You are learning programming")
print("Keep building")
```

A real application can contain thousands or millions of lines of code.

The important idea is:

> A large application is built from many smaller instructions working together.

Your first goal is not to write huge applications.

Your first goal is to understand how individual instructions work.

---

3. The `print()` Function

One of the first Python tools you will use is `print()`.

It displays information as output.

```
print("Hello")
```

Output:

```
Hello
```

You can print numbers:

```
print(25)
```

Output:

```
25
```

You can print decimal numbers:

```
print(3.14)
```

You can print Boolean values:

```
print(True)
print(False)
```

---

4. What Is `print()` Actually Doing?

Consider:

```
print("Python is fun")
```

There are several parts here.

`print`

This is the name of a built-in Python function.

`()`

Parentheses are used when calling the function.

`"Python is fun"`

This is the value being passed to the function.

The complete expression:

```
print("Python is fun")
```

means:

> Call the `print` function and give it `"Python is fun"` so it can display it.

You do not need to understand the internal implementation of `print()` yet.

For now, remember:

```
function name + parentheses + value
```

---

5. Function Calls

A function is a reusable piece of functionality.

You call a function by writing its name followed by parentheses.

```
print()
```

If the function needs information, you provide it inside the parentheses.

```
print("Hello")
```

Another example:

```
len("Python")
```

The `len()` function calculates length.

You will learn functions in much greater depth later.

For now, recognize this pattern:

```
function_name(value)
```

---

6. Arguments

The value you pass to a function is called an argument.

Example:

```
print("Hello")
```

Here:

```
print → function
"Hello" → argument
```

You can pass multiple arguments:

```
print("Hafsa", "Python", "Developer")
```

Output:

```
Hafsa Python Developer
```

Arguments are separated using commas.

```
print("Python", "JavaScript", "Flutter")
```

---

7. Strings

Text in Python is usually represented using strings.

Examples:

```
"Hello"
"Python"
"haas.dev"
"I am learning programming"
```

You can use double quotes:

```
print("Hello")
```

Or single quotes:

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

Both represent strings.

You should generally choose one style and use it consistently within a project.

---

8. Strings Can Contain Spaces

A string can contain a complete sentence:

```
print("Python helps you automate repetitive tasks.")
```

Output:

```
Python helps you automate repetitive tasks.
```

Spaces inside quotes are part of the string.

Compare:

```
print("Hello World")
```

with:

```
print("HelloWorld")
```

They are different strings.

---

9. Numbers

Python can work with numbers directly.

For example:

```
print(10)
print(25)
print(1000)
```

You don't put quotes around numbers when you want Python to treat them as numeric values.

Compare:

```
print(25)
```

and:

```
print("25")
```

They may look similar when printed, but Python treats them differently.

The first is a number.

The second is text.

This distinction becomes extremely important when you start performing calculations.

---

10. Basic Calculations

Python can evaluate mathematical expressions.

```
print(10 + 5)
```

Output:

```
15
```

More examples:

```
print(20 - 8)
print(6 * 7)
print(20 / 4)
```

Output:

```
12
42
5.0
```

Python evaluates the expression first and then `print()` displays the result.

---

11. Expressions

An expression is code that Python can evaluate to produce a value.

Examples:

```
10 + 5
```

```
20 * 3
```

```
"Hello"
```

```
True
```

You can put an expression inside `print()`:

```
print(10 + 5)
```

Python evaluates:

```
10 + 5
```

and gets:

```
15
```

Then `print()` displays:

```
15
```

This distinction will become increasingly important as programs become more complex.

---

12. Statements

A statement is an instruction that forms part of a program.

For example:

```
print("Hello")
```

is a statement.

A program can contain several statements:

```
print("Welcome")
print("Learning Python")
print("Building projects")
```

Python processes these instructions in sequence.

A useful beginner mental model is:

```
Program
   ↓
Statement
   ↓
Python performs the instruction
```

```
   ↓
Next statement
```

---

13. Expressions vs Statements

The terms can feel confusing at first.

Consider:

```
10 + 5
```

This is an expression because it produces a value.

Now consider:

```
print(10 + 5)
```

This is a statement containing an expression.

Another example:

```
print("Hello")
```

The string:

```
"Hello"
```

is an expression.

The complete line is an instruction in the program.

You don't need to memorize strict language-lawyer definitions at this stage.

Focus on this mental model:

> Expressions produce or represent values. Statements perform actions or form instructions in a program.

---

14. Python Reads Your Program in Order

Consider:

```
print("First")
print("Second")
print("Third")
```

Output:

```
First
Second
Third
```

The order matters.

If you change the order:

```
print("Third")
print("First")
print("Second")
```

the output changes accordingly.

This is the beginning of understanding **program flow**.

Later, conditions and loops will allow your program to change that flow.

---

15. Multiple Statements

You can create a small program with multiple instructions.

```
print("Welcome to my application")
print("Loading profile")
print("Profile loaded")
```

Output:

```
Welcome to my application
Loading profile
Profile loaded
```

Each line represents a separate instruction.

Keeping statements on separate lines makes code easier to read.

---

16. Why Readability Matters

Python is designed to be readable.

Compare:

```
print("Welcome")
print("Learning Python")
print("Building Projects")
```

with poorly structured code:

```
print("Welcome");print("Learning Python");print("Building Projects")
```

Although Python can support multiple statements on one line using semicolons, beginners should generally keep one logical statement per line.

Readable code is easier to:

- understand
- debug
- maintain
- review
- modify

Professional development is not only about making code work.

It is also about making code understandable.

---

17. Comments

Comments allow you to write notes inside your code.

A comment begins with `#`.

```
# Display a welcome message
print("Welcome to Python")
```

Python does not execute the comment as an instruction.

Another example:

```
print("Welcome")

# Show the learning message
print("Start building projects")
```

Comments are useful for explaining:

- why something exists
- what a section does
- important assumptions
- complex decisions

Avoid comments that simply repeat obvious code.

For example:

```
# Print Hello
print("Hello")
```

The code already makes the action obvious.

---

18. Your First Useful Program

Let's create something slightly more meaningful.

```
print("Developer Profile")
print("------------------")
print("Name: Hafsa")
print("Role: Software Engineer")
print("Focus: Python Development")
```

Output:

```
Developer Profile
------------------
Name: Hafsa
Role: Software Engineer
Focus: Python Development
```

This is still a very simple program, but it introduces an important idea:

> Programming becomes useful when instructions represent a real-world task.

Later, variables will allow the information to become dynamic.

---

19. A haas.dev Example

Imagine a simple command-line version of a haas.dev resource page.

```
print("haas.dev Learning Resources")
print("---------------------------")
print("Python Development")
print("Data Structures & Algorithms")
print("Web Development")
print("Mobile Development")
print("System Design")
```

Output:

```
haas.dev Learning Resources
---------------------------
Python Development
Data Structures & Algorithms
Web Development
Mobile Development
System Design
```

This is obviously much simpler than a real website.

But the underlying idea is similar:

> Software takes information and turns it into something useful for a user.

A real application would eventually get these values from variables, databases, APIs, or other sources.

---

20. A Real Product Example

Imagine an e-commerce system displaying an order summary.

A beginner version might look like:

```
print("Order Summary")
print("-------------")
print("Product: Laptop")
print("Quantity: 1")
print("Price: 120000")
print("Status: Confirmed")
```

Output:

```
Order Summary
-------------
Product: Laptop
Quantity: 1
Price: 120000
Status: Confirmed
```

A real application would not hard-code every order like this.

It would retrieve the data dynamically.

But before learning dynamic data, you need to understand the basic building blocks.

---

21. Quotes Are Part of Python Syntax

These work:

```
print("Hello")
```

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

This does not:

```
print(Hello)
```

Why?

Because without quotes, Python interprets `Hello` as something other than literal text.

At this stage, you can think of quotes as telling Python:

> Treat this as text.

---

22. Parentheses Matter

Correct:

```
print("Hello")
```

Incorrect:

```
print"Hello"
```

The parentheses are part of the function-call syntax.

Another example:

```
print(10 + 20)
```

The expression is inside the function call.

---

23. Case Sensitivity

Python is case-sensitive.

These are different names:

```
print
Print
PRINT
```

The built-in function is:

```
print()
```

So this:

```
Print("Hello")
```

will not behave the same way.

When writing Python, capitalization matters.

---

24. Common Beginner Error: Missing Quotes

Incorrect:

```
print(Hello World)
```

Python cannot interpret this as a valid string.

Correct:

```
print("Hello World")
```

---

25. Common Beginner Error: Unclosed Quotes

Incorrect:

```
print("Hello)
```

The string starts with a quote but does not end correctly.

Correct:

```
print("Hello")
```

---

26. Common Beginner Error: Unclosed Parentheses

Incorrect:

```
print("Hello"
```

Correct:

```
print("Hello")
```

Every opening parenthesis must have a corresponding closing parenthesis.

---

27. Common Beginner Error: Incorrect Capitalization

Incorrect:

```
Print("Hello")
```

Correct:

```
print("Hello")
```

Python does not automatically assume that different capitalization means the same thing.

---

28. Common Beginner Error: Mixing Quotes

This can cause a syntax error:

```
print("Hello')
```

The opening and closing quotation marks don't match.

Use:

```
print("Hello")
```

or:

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

---

29. Printing Multiple Values

You can provide multiple arguments:

```
print("Python", "is", "powerful")
```

Output:

```
Python is powerful
```

Another example:

```
print("Name:", "Hafsa")
```

Output:

```
Name: Hafsa
```

This becomes useful when displaying multiple pieces of information.

---

30. Printing Different Value Types

You can provide different types of values:

```
print("Age:", 25)
print("Score:", 95.5)
print("Active:", True)
```

Output:

```
Age: 25
Score: 95.5
Active: True
```

You will later learn how to store these values in variables and manipulate them.

---

31. Using `print()` for Debugging

`print()` is not only for displaying final output.

Beginners often use it to understand what their program is doing.

For example:

```
print("Program started")
print("Processing data")
print("Program finished")
```

If you only see:

```
Program started
Processing data
```

you know the program did not reach the final instruction.

This is a very basic debugging technique.

Later, you will learn professional debugging tools.

---

32. Building a Tiny Menu

You can use multiple `print()` statements to create a command-line menu.

```
print("=== Developer Toolkit ===")
print("1. Learn Python")
print("2. Practice DSA")
print("3. Build a Project")
print("4. Read System Design")
```

Output:

```
=== Developer Toolkit ===
1. Learn Python
2. Practice DSA
3. Build a Project
4. Read System Design
```

The program doesn't respond to the user's selection yet.

That requires input and decision-making, which you will learn later.

---

33. Output Is the Beginning, Not the Goal

At the beginning, programming can feel like:

```
print("Something")
```

over and over again.

But `print()` is simply your first way to see what your program is doing.

The progression will eventually look like:

```
Print information
      ↓
Store information
      ↓
Receive information
      ↓
Process information
      ↓
Make decisions
      ↓
Repeat operations
      ↓
Organize code
      ↓
Work with files and APIs
      ↓
Build applications
```

That is why learning the fundamentals matters.

---

34. A Small Developer Dashboard

Let's combine what you know so far.

```
print("================================")
print("        DEVELOPER DASHBOARD")
print("================================")

print("Name: Hafsa")
print("Role: Software Engineer")
print("Language: Python")
print("Projects: 5")
print("Status: Learning")
```

Output:

```
================================
        DEVELOPER DASHBOARD
================================

Name: Hafsa
Role: Software Engineer
Language: Python
Projects: 5
Status: Learning
```

This program is still static.

The next major step will be learning how to store information instead of repeatedly writing literal values.

That is where **variables** become important.

---

35. The Important Mental Model

When you see:

```
print("Python")
```

think:

```
Python source code
      ↓
Instruction
      ↓
Call print()
      ↓
Pass "Python"
      ↓
Display output
```

When you see:

```
print(10 + 20)
```

think:

```
10 + 20
   ↓
Evaluate expression
   ↓
```

```
30
   ↓
print(30)
   ↓
Display 30
```

This mental model will help you understand more advanced Python later.

---

36. Practice Exercises

Easy

Exercise 1

Write a program that prints:

```
Hello Python
I am learning programming
I will build projects
```

Exercise 2

Print your:

- name
- profession
- favorite programming language

Exercise 3

Print the result of:

```
25 + 75
```

Medium

Exercise 4

Create a simple course card:

```
Python Development
Beginner Level
30 Lessons
Start Learning
```

Exercise 5

Create a product order summary containing:

- product
- quantity
- price
- order status

Exercise 6

Create a simple haas.dev resource menu with at least five categories.

Hard

Exercise 7

Create a developer dashboard containing:

- name
- role
- programming language
- number of projects
- years of experience
- current learning goal

Use only concepts covered in this guide.

---

37. Five Minute Challenge

Create a program called:

My Developer Introduction

The output should look similar to:

```
============================
     MY DEVELOPER CARD
============================

Name: Your Name
Role: Software Developer
Learning: Python
Goal: Build Real Projects
Website: haas.dev
Status: Learning and Building
```

Requirements:

- use at least 6 `print()` statements

- include both text and numbers
- include at least one Boolean value
- add at least one useful comment
- keep the output readable

Do not use variables yet.

The goal is to practice the fundamentals before adding more concepts.

---

38. Mini Quiz

Question 1

What does `print()` do?

A. Stores information

B. Displays information

C. Creates a variable

D. Deletes information

Answer: B

Question 2

Which is a valid Python string?

A.

```
Hello
```

B.

```
"Hello"
```

C.

```
"Hello
```

D.

```
Hello"
```

Answer: B

Question 3

What is the result?

```
print(10 + 5)
```

A. `105`

B. `"10 + 5"`

C. `15`

D. Error

Answer: C

Question 4

Which is correct?

A.

```
Print("Hello")
```

B.

```
print("Hello")
```

C.

```
print"Hello"
```

D.

```
print(Hello")
```

Answer: B

Question 5

What does this represent?

```
10 + 20
```

A. Comment

B. String

C. Expression

D. Function

Answer: C

---

39. Interview Questions

What is `print()` in Python?

`print()` is a built-in Python function used to display values or text as output.

What is a function call?

A function call is an instruction that invokes a function, usually by writing its name followed by parentheses.

Example:

```
print("Hello")
```

What is an argument?

An argument is a value supplied to a function when it is called.

```
print("Hello")
```

Here `"Hello"` is the argument.

What is a string?

A string is a sequence of text characters represented using quotes.

```
"Python"
```

What is an expression?

An expression is code that Python can evaluate to produce or represent a value.

Example:

```
10 + 20
```

Is Python case-sensitive?

Yes.

```
print
Print
PRINT
```

are treated as different names.

Why are comments used?

Comments document code and provide useful context for developers. They are not executed as normal Python instructions.

---

40. Cheat Sheet

Display text

```
print("Hello")
```

Display a number

```
print(100)
```

Display a Boolean

```
print(True)
```

Perform a calculation

```
print(10 + 20)
```

Multiple arguments

```
print("Name:", "Hafsa")
```

String with single quotes

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

Comment

```
# This is a comment
```

Multiple instructions

```
print("First")
print("Second")
print("Third")
```

Basic function call pattern

```
function_name(argument)
```

---

41. Key Takeaways

You now understand the basic structure of a Python program.

Remember:

1. A Python program consists of instructions.
2. `print()` displays information.
3. Functions are called using parentheses.
4. Values passed to functions are arguments.
5. Text is represented using strings.

6. Numbers can be used directly in calculations.
7. Expressions produce or represent values.
8. Programs execute instructions in an ordered flow.
9. Python is case-sensitive.
10. Comments begin with `#`.
11. Readable code is easier to understand and maintain.
12. `print()` is useful for both output and basic debugging.
13. Large applications are built from smaller programming concepts.

Most importantly:

> Do not memorize `print("Hello World")`. Understand what every part of it means.

Once you understand the building blocks, learning more Python becomes a process of combining them.

---

42. What Comes Next?

You have learned how to write simple instructions and display information.

But there is a problem.

What if the information changes?

For example:

```
User 1 → Hafsa
User 2 → Ali
User 3 → Sara
```

You cannot realistically write a completely new program every time the data changes.

You need a way to **store information and give it a name**.

That concept is called a **variable**.

The next stage of the Python journey is understanding how variables work, how Python stores values, how names reference objects, and how to write programs that work with changing data.

Visit haas.dev for more resources and guides.

haas.dev

https://dev-roast-app.vercel.app