**Python A blue and black logo

AI-generated content may be incorrect.**

**Python** is a **high-level, easy-to-read, and powerful programming language** used to build software, automate tasks, analyze data, and much more. It was created by **Guido van Rossum** and first released in **1991**.

print(‘Hello Word’)

>Hello Word

**Variables:**

message = "Hello Python world!"

print(message)

We’ve added a variable named message. In this case the value is the "Hello Python world!" text.

Let’s expand on this program by modifying hello\_world.py to print a second message. Add a blank line to hello\_world.py, and then add two new lines of code:

message = "Hello Python world!"

print(message)

message = "Hello Python Crash Course world!"

print(message)

Now when you run hello\_world.py, you should see two lines of output:

Hello Python world!

Hello Python Crash Course world!

**#Naming and Using Variables**

When you’re using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that’s easier to read and understand. Be sure to keep the following variable rules in mind:

* Variable names can contain only letters, numbers, and underscores.

They can start with a letter or an underscore, but not with a number.

For instance, you can call a variable message\_1 but not 1\_message.

* Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, greeting\_message works, but greeting message will cause errors.
* Avoid using Python keywords and function names as variable names; that is, do not use words that Python has reserved for a particular programmatic purpose, such as the word print. (See “Python Keywords and Built-in Functions”).
* Variable names should be short but descriptive. For example, name is better than n, student\_name is better than s\_n, and name\_length is better than length\_of\_persons\_name.
* Be careful when using the lowercase letter l and the uppercase letter O, because they could be confused with the numbers 1 and 0.

**Note::**  The Python variables you’re using at this point should be lowercase. You won’t get errors if you use uppercase letters, but uppercase letters in variable names have special meanings that we’ll discuss in later chapters.

**#Avoiding Name Errors When Using Variables**

We’ll write some code that generates an error on purpose. Enter the following code, including the misspelled word mesage shown in bold:

message = "Hello Python Crash Course reader!"

print(**mesage**)

When an error occurs in your program, the Python interpreter does its best to help you figure out where the problem is. The interpreter provides a traceback when a program cannot run successfully.

A traceback is a record of where the interpreter ran into trouble when trying to execute your code. Here’s an example of the traceback that Python provides after you’ve accidentally misspelled a variable’s name:

**Traceback (most recent call last):**

❶ File "hello\_world.py", line 2, in <module>

❷ print(mesage)

❸ NameError: name 'mesage' is not defined

The output at ❶ reports that an error occurs in line 2 of the file hello\_world.py. The interpreter shows this line ❷ to help us spot the error quickly and tells us what kind of error it found ❸.

**Data Types:**

In Python programming, **data types** define the kind of data a variable can hold. Python is dynamically typed, so you don’t need to declare the data type explicitly—Python figures it out at runtime. Here’s is break down of main data types

**🔢 1. Numbers**

**a. int → Integer (whole number)**

x = 5

print(type(x)) # <class 'int'>

**b. float → Decimal number**

pi = 3.14

print(type(pi)) # <class 'float'>

**c. complex → Complex number**

z = 2 + 3j

print(type(z)) # <class 'complex'>

**🔤 2. String (str)**

Used for text data.

name = "Monta"

print(type(name)) # <class 'str'>

Strings can use single ' or double " quotes.

**✅ 3. Boolean (bool)**

Only True or False

is\_active = True

print(type(is\_active)) # <class 'bool'>

**📦 4. List (list)**

A list of values that can change (mutable).

fruits = ["apple", "banana", "cherry"]

print(type(fruits)) # <class 'list'>

You can access by index:

print(fruits[1]) # banana

**📦 5. Tuple (tuple)**

Like a list, but **cannot be changed** (immutable).

colors = ("red", "green", "blue")

print(type(colors)) # <class 'tuple'>

**🗂️ 6. Dictionary (dict)**

Stores data in **key-value** pairs.

person = {"name": "Monta", "age": 25}

print(type(person)) # <class 'dict'>

Access by key:

print(person["name"]) # Monta

**📚 7. Set (set)**

An unordered collection of **unique** items.

numbers = {1, 2, 3, 3, 2}

print(numbers) # {1, 2, 3}

print(type(numbers)) # <class 'set'>

**📚 8. Frozen Set (frozenset)**

Like set but immutable.

fset = frozenset([1, 2, 3])

print(type(fset)) # <class 'frozenset'>

**🪙 9. None Type (NoneType)**

Represents a **null** or **no value**.

data = None

print(type(data)) # <class 'NoneType'>

**📖 Summary Table**

| **Type** | **Example** | **Mutable?** |
| --- | --- | --- |
| int | 10 | ❌ |
| float | 3.14 | ❌ |
| complex | 2 + 3j | ❌ |
| str | "Hello" | ❌ |
| bool | True, False | ❌ |
| list | [1, 2, 3] | ✅ |
| tuple | (1, 2, 3) | ❌ |
| dict | {"a": 1, "b": 2} | ✅ |
| set | {1, 2, 3} | ✅ |
| frozenset | frozenset([1, 2]) | ❌ |
| NoneType | None | ❌ |

**Changing Case in a String with Methods:**

One of the simplest tasks you can do with strings is to change the case of the words in a string. Look at the following code, and try to determine what’s happening:

name = "ada lovelace" print(name.title())

print(name)

⟩⟩ Ada Lovelace

name = "Ada Lovelace"

print(name.upper())

print(name.lower())

This will display the following:

ADA LOVELACE

ada lovelace

**Using Variables in Strings :**

In some situations, you’ll want to use a variable’s value inside a string. For example, you might want two variables to represent a first name and a last name respectively, and then want to combine those values to display someone’s full name:

first\_name = "ada"

last\_name = "lovelace"

full\_name = f"{first\_name} {last\_name}"

print(full\_name)

To insert a variable’s value into a string, place the letter f immediately before the opening quotation mark. Put braces around the name or names of any variable you want to use inside the string. Python will replace each variable with its value when the string is displayed.

These strings are called f-strings. The f is for format, because Python formats the string by replacing the name of any variable in braces with its value. The output from the previous code is:

ada lovelace

**Adding Whitespace to Strings with Tabs or Newsline :**

In programming, whitespace refers to any nonprinting character, such as spaces, tabs, and end-of-line symbols. You can use whitespace to organize your output so it’s easier for users to read. To add a tab to your text, use the character combination.

In Python you can add whitespace to strings in two ways:

1. Using special escape sequences:

* \t - tab (horizontal space, usually 4 or 8 spaces depending on environment)
* \n - newline(line break)
* \r - carriage return(moves cursor to start of line)
* \v or \x0b - vertical tab
* \f - form feed (page break in printers, but works like newline in terminals)

**Underscore in Number :**

Underscores in Numbers When you’re writing long numbers, you can group digits using underscores to make large numbers more readable:

>>> universe\_age = 14\_000\_000\_000

**Constant :**

A constant is like a variable whose value stays the same throughout the life of a program. Python doesn’t have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:

MAX\_CONNECTIONS = 5000