

Python Summer Course

Course 1: Python Basics & Objects

Théophile Gentilhomme

September 5, 2025



Introduction



My first code

Python Code ↺ Start Over ▶ Run Code

```
1 print("Hello World!")
2 print("Welcome to the Python Summer Course!")
```

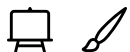
⏮ main

Downloading package: ipython

What is programming ?

“Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages.”

— Wikipedia



Why Learn with Python?

- ✓ Easy to read and write; its simple syntax is close to English
- ✓ Beginner-friendly: widely used in teaching and well-documented
- ✓ Versatile: used in data science, AI, web development, automation...
- ✓ Cross-platform: runs on Windows, macOS, and Linux
- ⚠ Slower performance compared to compiled languages like C/C++
- ⚠ Not ideal for mobile app development or real-time systems



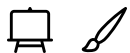
Setting Up Python: How It Works

Python is an interpreted language:

- You write source code in `.py` files
- The Python interpreter reads and executes your code line by line

To run Python code, you need:

- The **Python interpreter**
- A code editor (e.g. VS Code, Thonny, Jupyter Notebook)



What do I use?

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 import sys # We will see what it means later
2 # By the way, this is how we write comments in Python
3
4 print("I print who is running my Python")
5 print(sys.executable)
```

[↺ main](#)

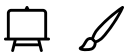


Installation Options

- ✓ Option 1: Install Python Locally (e.g. Download from python.org)
- ✓ Option 2: Use Environment management (e.g. Anaconda)
- ✓ Option 3: Use Python in the Browser (e.g.. Google Colab)

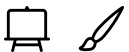
G Google Colab

1. Go to colab.research.google.com
2. You may have to login to your Google account
3. Create a new Notebook
4. Start programming!



notebooks in 2 mins

- Write text (Markdown) => + Text
- Write code (Python) = > + Code
- Run cells



Python as a Calculator

You can just write operation directly into the cells

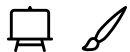
Python Code

↺ Start Over

▶ Run Code

```
1 3 + 1
```

Your turn: try it out using $+$ $-$ $*$ $/$ $\%$ $**$ $//$



`print ()` and `input ()` in Python

These are basic but powerful tools for **interacting with the user**.

`print ()`: Display output

`input ()`: Get user input

Used to **ask the user for information**. Always returns a **string**.



Example

```
1 user_name = input("What is your name? ")
2 print("Nice to meet you,", user_name)
```



If you need a number from `input()`, use `int()` or `float()`:

```
1 age = int(input("Enter your age: "))
2 print("In 5 years, you'll be", age + 5)
```

Variable

- A variable is a name that stores a value in your program, like a labeled box that holds data.
- It lets you remember values (like numbers, text, results of calculations)
- You can use or change the value later
- Python creates the variable when you assign it a value

Variable: example

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 # Assign values to variables
2 name = "Alice"
3 age = 30
4 height = 1.65
5
6 # Use variables in expressions
7 print("Name:", name)
8 print("Age in 5 years:", age + 5)
9 print("Height in cm:", height * 100)
10
11 # Change the values
12 name = "Bob"
13 age = age - 5
14 print("Name: ", name)
15 print("Age: ", age)
```

[main](#)

Types

A type in Python defines what kind of data a value is, and what you can do with it.

Python Code

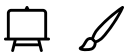
↺ Start Over

▶ Run Code

```
1 # Integer
2 age = 25
3 print("Age:", age, "->", type(age))
4
5 # Float
6 height = 1.75
7 print("Height:", height, "->", type(height))
8
9 # String
10 name = "Alice"
11 print("Name:", name, "->", type(name))
12
13 # Boolean
14 is_student = True
15 print("Is student:", is_student, "->", type(is_student))
```

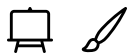
BACK main

Python Basics & Objects



Everything is an Object in Python

- Objects have data (attributes) and behaviors (methods)
- You can call methods with dot syntax: `object.method()`
- Even simple things like numbers and strings are full objects



Everything is an Object in Python

An object is an instantiation of a `class` (we will see what a `class` is later).

Python Code

↺ Start Over

▶ Run Code

```
1 # String is an object with methods
2 text = "hello"
3 print(text.upper())          # 'HELLO'
4 print(text.replace("l", "x")) # 'hexxo'
5
6 # Method (function) of an object is also an object!
7 func = text.upper
8 print(type(func))
9
10 # Even types are objects
11 print(type(42))
12 print(type("hi"))
13 print(type(int))
```



BACK main

String Manipulation in Python

Strings are sequences of characters: you can access, combine, and transform them easily.

Python Code

↺ Start Over

▶ Run Code

```
1 text = "Python"
2 print(text[0])
3 print(text[-1])
4 print(text[1:4])
```



Common string methods

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 print("hello".upper())  
2 print("HELLO".lower())
```

[↺ main](#)

Common string methods

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 print("Python".startswith("Py"))
```

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 print("data,science".split(","))
```

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 print("  clean  ".strip())
```

Common string methods

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 text = "My name is {} and I am {}".format("Bob", 40)
2 print(text)
```

[↺ main](#)

Common string methods

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 # Chain of methods
2 text = " my veRy UGly?? ## text  "
3 clean_text = text.strip().replace("?", "").replace("#", "").replace("  ", " ").lower()
4 print(clean_text)
```

[↺ main](#)

Concatenation and repetition

```
Python Code ↺ Start Over ▶ Run Code  
1 "Py" + "thon"      # 'Python'  
2 "ha" * 3           # 'hahaha'
```

Strings are **immutable**: you can't change them directly, but you can create new ones.

Boolean Conditions in Python

Boolean conditions are used to **ask questions** in your code: they return either `True` or `False`.

Common comparison operators:

- `==` → equal
- `!=` → not equal
- `<`, `>`, `<=`, `>=` → less/greater than (or equal)

Python Code

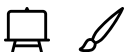
↺ Start Over

▶ Run Code

```
1 x = 10
2 print(x > 5)      # True
3 print(x == 10)    # True
4 print(x != 7)     # True
```

BACK

main



Combine conditions using:

- `and`: both must be True
- `or`: at least one must be True
- `not`: negates the condition

Python Code ↺ Start Over ▶ Run Code

```
1 age = 20
2 has_id = True
3
4 print(age >= 18 and has_id)
5 print(age < 18 or not has_id)
```

The `if` Statement

The `if` statement allows your program to **make decisions** based on conditions.

- Runs code only when the condition is `True`
- Can include `elif` (else-if) and `else` branches
- Python uses indentation to define the blocks

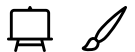
```
1 if condition:  
2     # do something  
3 elif other_condition:  
4     # do something else  
5 else:  
6     # fallback
```

The `if` Example

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 age = 18
2
3 if age >= 18:
4     print("You can vote!")
5 else:
6     print("Sorry, too young.")
```

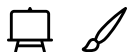
[↺ main](#)

The `for` Loop

The `for` loop lets you **repeat a block of code** for each item in a sequence.

- Commonly used to loop over lists, strings, or ranges
- Automatically stops when the sequence ends

```
1 for item in sequence:  
2     # do something with item
```



Example: Looping over a list

A list in Python is an ordered collection of items (like numbers, strings or any objects) that can be changed, added to, or removed; written with square brackets `[]`. (see Course 2)

Python Code

↺ Start Over

▶ Run Code

```
1 fruits = ["apple", "banana", "cherry"]
2
3 for fruit in fruits:
4     print("I like", fruit)
5
6 # If you need the index too
7 for i, fruit in enumerate(fruits):
8     print("Fruit number", i)
9     print("I like", fruit)
```



← main

Example: Looping with `range()`

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1  for i in range(1, 4):  
2      print("Number:", i)
```

[↺ main](#)

Example: Looping with `break` and `continue`

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 stop = 5
2 v for i in range(10):
3 v     if i < 3:
4         continue
5         print("Number:", i)
6
7 v     if i == stop:
8         break
```

[main](#)

The `while` Loop

A `while` loop repeats a block of code **as long as a condition is `True`**.

- Good for loops where you **don't know in advance how many times to repeat**
- The condition is checked **before** each loop

```
1 while condition:  
2     # code to repeat  
3     # Can use continue and break
```


Example: Counting with `while`

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 count = 1
2
3 while count <= 3:
4     print("Count is:", count)
5     count += 1
6
7 while True:
8     count -= 1
9     if count < -3:
10        break
```

[main](#)

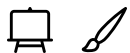
Your turn!

Write a Python program that:

1. Greets the user using their name (`input()` + `print()`)
2. Asks 3 multiple-choice questions (list of questions + list of answers)
3. Uses if statements to check answers (use object method to make it UPPER or lower case and then compare to the right answer)



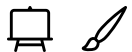
4. Uses a for or while loop to ask questions one by one (and `input()` + `print()`)
5. Keeps track of the score using a variable and `if-else` statements to increment the score.
6. Prints the final score and a custom message



More references

[Python course for data analysis](#)

[The Python tutorial](#)



Solution

[Show Solution](#)