Tutorial 1: Setting Up Your Programming Environment

The Art of Computer Programming

16 September 2026

Objective. In this first tutorial, you will set up on your own laptop everything needed to program in Python, C, and C++ for the rest of the semester, and you will write and run your first programs. By the end of the session, you should have:

1. Visual Studio Code (VS Code) installed, with its Python and C/C++ extensions;
2. a working Python interpreter and a working C/C++ compiler, both usable from VS Code;
3. a "Hello world" program written, compiled (for C and C++), and run in each of the three languages;
4. if time allows, practiced the material of the first lecture (data representation, variables, expressions, basic input and output) in Python and C.

Parts A to C are *mandatory*: a working environment is a prerequisite for all the following sessions. Ask for help as soon as you are stuck; do not wait until the end of the session.

Conventions. Instructions differ between operating systems; follow the box corresponding to yours (Windows, macOS, or Linux). Commands to type in a *terminal* (a text console where you type commands and press Enter) are shown with a `$` prompt when they appear in the text, as in `$ python3 --version`, and in a tinted box when they stand on their own:

```
python3 --version
```

Do not type the `$` itself: it stands for what the terminal displays in front of the commands you type. Names of files, folders, commands and options are in `typewriter` font, as in `hello.py`, and source code appears in a gray box labeled with its language. Keyboard shortcuts are given for Windows/Linux first, then macOS in parentheses, e.g., Ctrl + Shift + P (Cmd + Shift + P). Questions to answer or tasks to complete are numbered **Q1**, **Q2**, etc.

Part A: Visual Studio Code

VS Code is a free code editor working on Windows, macOS, and Linux. It is the one we will use throughout the course; if you already know and prefer another editor or IDE, you may use it, but we will only be able to help you with VS Code.

Installation

Download VS Code from `https://code.visualstudio.com/download` and install it:

Windows

Download the *User Installer* (x64; choose *Arm64* only if you know your laptop has a Snapdragon/ARM processor), run it, and accept the default options; make sure *Add to PATH* stays checked.

macOS

Download the *Universal* build (works both on Apple silicon and Intel Macs). Unzip the archive if your browser did not do it, then drag *Visual Studio Code.app* into your *Applications* folder (this step matters: do not run it from the Downloads folder). Launch it from the Applications folder or Launchpad.

Linux

Download the `.deb` (Debian, Ubuntu, Mint) or `.rpm` (Fedora, openSUSE) package and install it, e.g., with `$ sudo apt install ./code_*.deb`. On other distributions, use your package manager (the package is usually called `code`) or a Snap/Flatpak package.

First steps

Launch VS Code and take a minute to locate the following elements of its interface (see `https://code.visualstudio.com/docs/getstarted/userinterface` for a guided tour):

- the **Activity Bar** on the left, with in particular the *Explorer* (files of the opened folder) and the *Extensions* view (Ctrl+Shift+X / Cmd+Shift+X);
- the **editor** in the center, where files are opened in tabs;
- the **Command Palette** (Ctrl+Shift+P / Cmd+Shift+P), from which every command of VS Code can be run by typing its name;
- the integrated **Terminal** (View > Terminal, or Ctrl+`, the backquote key – with Ctrl, not Cmd, on macOS), which opens a terminal at the bottom of the window. This is where we will type commands.

Q1. Create a folder named `programming-1` somewhere convenient on your disk (e.g., in your *Documents* folder), and a subfolder `tutorial01` inside it. Every tutorial will get its own subfolder. In VS Code, open this folder with File > Open Folder and select `tutorial01` (the button confirming the dialog reads *Select Folder*, or *Open* on macOS). Accept to trust the authors of the folder if asked. The Explorer should now show an empty folder named TUTORIAL01.

Windows users: by default, Windows hides file extensions, so that a file named `hello.py.txt` appears as `hello.py` in the File Explorer, which causes endless confusion. Enable View > Show > File name extensions in Windows File Explorer.

Q2. Open the integrated terminal and type `$ echo hello`, then Enter. Which program interprets your commands? It is usually *PowerShell* on Windows, *zsh* on macOS, and *bash* on Linux; the name appears at the top right of the terminal panel. The terminal starts in the folder opened in VS Code: type `$ pwd` to display it.

Part B: Python

Installing Python

Windows

Go to `https://www.python.org/downloads/` and download the latest Python 3 installer (3.14 or later). Run it. On the first screen, **check the box "Add python.exe to PATH"** before clicking *Install Now*: without it, the `python` command will not be found in the terminal.
If typing `$ python` in a terminal opens the Microsoft Store instead of Python, either install Python from the Store (it also works), or disable the corresponding *App execution aliases* in the Windows settings.

macOS

Go to `https://www.python.org/downloads/` and download the macOS installer (universal, works on all Macs) for the latest Python 3 (3.14 or later); open the `.pkg` file and follow the steps. (Older versions of Python may already be installed on your Mac, but they may be outdated; the installer from python.org is the safest option.)

Linux

Python 3 is almost certainly already installed. If not:

- `$ sudo apt install python3` (Debian, Ubuntu, Mint);
- `$ sudo dnf install python3` (Fedora);
- `$ sudo pacman -S python` (Arch).

Check

Close the integrated terminal and open a new one (programs installed while a terminal is open are not always visible in that terminal). Type `$ python --version` (Windows) or `$ python3 --version` (macOS, Linux). You should see something like `Python 3.14.4`. Any version 3.10 or later is fine for this course; if you see `Python 2.x`, or an error, ask for help.

The Python extension for VS Code

In the Extensions view (Ctrl+Shift+X / Cmd+Shift+X), search for *Python* and install the extension named *Python* published by Microsoft (it comes with the *Pylance* extension, which provides code completion and error highlighting).

Hello world in Python

Q3. In the Explorer, create a new file (*New File* icon next to the folder name) named `hello.py`. The `.py` extension is what tells VS Code (and everyone else) that this is a Python program. Type in the following program and save the file (Ctrl+S / Cmd+S):

Python

```
print("Hello world")  # This is a comment.
```

Q4. Run the program in the three following ways, and check that `Hello world` is displayed each time:

(a) click the *Run Python File* button (a triangle ▷ at the top right of the editor);
(b) in the integrated terminal, type `$ python hello.py` (Windows) or `$ python3 hello.py` (macOS, Linux): this is what the button does for you;
(c) in the terminal, launch the *interactive interpreter* by typing `$ python` (Windows) or `$ python3` (macOS, Linux) alone. You get a prompt `>>>`, at which you can type any Python statement or expression and see its result immediately: try `print("Hello world")`, then `2 ** 100`, then `"Hello" + " " + "world"`. Type `exit()` to leave the interpreter. This interactive mode is very convenient to experiment with the language.

Look at the bottom right of the VS Code window: the status bar shows which Python interpreter the extension uses (e.g., `3.14.4`). If it shows an old version, or none, click on it (or use *Python: Select Interpreter* in the Command Palette) and choose the interpreter you just installed.

Q5. Programs contain errors, and the first skill to acquire is to read error messages. Modify the program as follows and run it again with the Run button. For each case, read the message displayed in the terminal, note the kind of error and the line number reported, and then fix the program.

(a) Remove the closing parenthesis: `print("Hello world"`
(b) Remove the quotes: `print(Hello world)`
(c) Misspell the function name: `prnt("Hello world")`

Notice that Pylance underlines some of these errors even before you run the program: hover the underlined text with the mouse to read the diagnostic.

Part C: C and C++

Unlike Python, C and C++ programs must be *compiled* to machine code before they can be run. We will use the GCC compiler (`gcc` for C, `g++` for C++), or Clang on macOS, which accepts the same commands.

Installing a compiler

Windows

The recommended way is to install the MSYS2 environment, which provides GCC (this is the procedure of the official VS Code documentation, `https://code.visualstudio.com/docs/cpp/config-mingw`):

1. Download the installer from `https://www.msys2.org/` and run it, keeping the default installation folder `C:\msys64`.
2. At the end of the installation, an *MSYS2 UCRT64* terminal opens (if not, launch it from the Start menu). In it, type:

```
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

Press Enter to accept the default selection (all packages), then Y to confirm. This downloads a few hundred megabytes.

3. Add the compiler to the Windows *PATH*, so that it can be found from any terminal:
   (a) in the Windows search bar, type *Settings* and open the Windows Settings;
   (b) in the Settings, search for *Edit environment variables for your account* and open it;
   (c) in the list *User variables*, select the *Path* variable and click *Edit*;
   (d) click *New* and add `C:\msys64\ucrt64\bin` (if you changed the installation folder at step 1, adapt the beginning of the path accordingly);
   (e) click *OK*, then *OK* again in the *Environment Variables* window.
4. Quit VS Code completely and start it again, so that it sees the new PATH.

Laptop with a Snapdragon (ARM) processor: in the MSYS2 *CLANGARM64* terminal, install `mingw-w64-clang-aarch64-toolchain` instead, and add `C:\msys64\clangarm64\bin` to the PATH; ask us if unsure.
Alternative: if you are comfortable with Linux, the Windows Subsystem for Linux (`$ wsl --install` in an administrator PowerShell, then reboot) gives you an Ubuntu system in which to follow the Linux instructions, together with the *WSL* extension of VS Code. This is more complex; use it only if the MSYS2 route fails.

macOS

Open the *Terminal* application (in *Applications* > *Utilities*, or search for it with Spotlight) and type:

```
xcode-select --install
```

Accept the installation of the *Command Line Developer Tools* in the dialog that appears (it downloads a few hundred megabytes). This installs Apple's Clang compiler; the commands `gcc` and `g++` exist and actually run Clang, which is fine for this course.

Linux

- `$ sudo apt install build-essential gdb` (Debian, Ubuntu, Mint);
- `$ sudo dnf install gcc gcc-c++ gdb` (Fedora);
- `$ sudo pacman -S base-devel gdb` (Arch).

Check

Open a new integrated terminal in VS Code and type `$ gcc --version` and then `$ g++ --version`. Each should print a version banner (of GCC, or of Apple clang on macOS), not an error such as *command not found* or *not recognized*.

If it does not work

On Windows, "`gcc` is not recognized" after installation almost always means the PATH step was skipped, mistyped, or VS Code was not restarted: check the folder `C:\msys64\ucrt64\bin` exists and contains `gcc.exe`. On macOS, if `$ xcode-select --install` says the tools are already installed, you are done.
If the compiler rejects the `-std=c23` option, it is too old for the 2023 version of C: ask us, and in the meantime use `-std=c2x` instead, which older compilers accept for the same language.
If you cannot get a compiler to work today, do not stay stuck: finish the session with an online compiler such as `https://www.onlinegdb.com/` (supports Python, C, and C++, including keyboard input) and come and see us so that we fix your installation.

The C/C++ extension for VS Code

In the Extensions view, search for *C/C++* and install the extension named *C/C++* published by Microsoft.

Hello world in C

Q6. Create a file `hello.c` in the `tutorial01` folder with the following program (the `.c` extension means: C program):

C

```c
#include <stdio.h>

int main() {
  printf("Hello world\n"); /* This is a comment. */
  return 0;
}
```

Q7. Compile it from the integrated terminal:

```
gcc -Wall -Wextra -std=c23 hello.c -o hello
```

This command asks the compiler to read `hello.c`, to produce an executable program named `hello` (option `-o`), to use the C23 version of the language (`-std=c23`), and to warn about any suspicious construct (`-Wall -Wextra`; always use these options, warnings are your friends). If all goes well, nothing is printed. Look at the Explorer: a new file `hello` (or `hello.exe` on Windows) appeared. This file contains machine code; it is not a text file (try opening it).

Q8. Run the program from the terminal by typing `$ ./hello` (macOS, Linux) or `$ .\hello.exe` (Windows). The `./` (or `.\`) prefix means "the program located in the current folder".

Q9. Now run it from VS Code: with `hello.c` open, click the Run button ▷ at the top right and choose *Run C/C++ File*. VS Code asks which compiler to use: pick the `gcc` entry (`clang` on macOS) that was found in your PATH. It creates a configuration file in a `.vscode` subfolder, compiles the program, and runs it in a terminal. Later on, the Run button reuses this configuration.

Q10. Introduce errors in the program and compile again from the terminal, reading the messages, which indicate the file, the line, and the column of each problem:

(a) remove the semicolon at the end of the `printf` line;
(b) remove the `#include` line;
(c) add a line `int x = 3;` just before `return 0;`.

Which of these are errors (no executable produced) and which are only warnings (an executable is produced, but the compiler complains and you should listen)? Contrast with Python: in C, most mistakes are caught *at compilation time*, before the program runs at all.

Hello world in C++

Q11. Create `hello.cpp` (extension `.cpp`: C++ program):

C++

```cpp
#include <iostream>

int main() {
  std::cout << "Hello world" << std::endl; // This is a comment.
  return 0;
}
```

Compile it with `g++` (the C++ compiler) and run it:

```
g++ -Wall -Wextra -std=c++17 hello.cpp -o hellocpp
./hellocpp
```

(`$ .\hellocpp.exe` on Windows.) Then run it with the Run button as well.

Q12. C is *almost* a subset of C++: compile `hello.c` with `g++` instead of `gcc`. Does it work? And what happens if you compile `hello.cpp` with `gcc`?

Check

Before moving on, make sure that you can, from VS Code: write a file, run a Python program, compile and run a C program, compile and run a C++ program. If any of these fails, call us.

Part D: Practice (if time allows)

The following exercises revisit the content of the first lecture. Each of them says which language it concerns: most ask for the same program in Python and in C, in which case write the Python version first and the C one next; some concern only one of the two languages, or ask you to check in one language what you observed in the other. Write each program in its own file (e.g., `greetings.py` and `greetings.c`) in the `tutorial01` folder. Exercises are roughly ordered by importance; it is fine not to finish them all.

Q13. Greetings. Write a program asking the user for their name and their age, and printing a message such as `Hello Ada! You were born in 2008 or 2007.` In Python, use `input()`, `int()`, and an f-string. In C, use `scanf` and `printf`; to read the name, use the following recipe (we will see what it means in a later lecture):

C

```c
char name[100];      /* room for 99 bytes plus the final null byte */
scanf("%99s", name); /* reads one word into name; note: no & here */
```

Q14. Bases. Write a program that reads a nonnegative integer and prints its representation in binary, octal, and hexadecimal. In Python, f-strings accept format specifiers: `f"{n:b}"`, `f"{n:o}"`, `f"{n:x}"` (also try `bin(n)`, and `int("BF", 16)` for the converse). In C, `printf` knows `"%o"` and `"%x"`; the binary format `"%b"` is new in C23 and may not be supported by your system's library yet. Check by hand the results for 42, 191, and 255.

Q15. Sizes of C types. In C, the operator `sizeof(T)` gives the number of bytes used by a value of type `T`; it is printed with `printf("%zu", sizeof(int))`. Write a program that prints the sizes of `char`, `short`, `int`, `long`, `long long`, `float`, `double`, `long double`, and `int*`. Compare your results with classmates using another operating system: which types differ?

Q16. Two's complement. In C, with `#include <stdint.h>`, write a program that stores 200 in a `uint8_t` variable, copies it into an `int8_t` variable with `int8_t s = (int8_t) u;`, and prints both with `"%d"`. Explain the result by writing 200 in binary and applying the two's complement rule of the lecture. Do the same the other way around, starting from -42. What happens when you add 1 to a `uint8_t` containing 255? In Python, check your binary computations with `f"{(-42) & 0xFF:08b}"` (the `& 0xFF` keeps only the lowest 8 bits), and observe that `2 ** 64 + 1` is computed exactly: Python integers have arbitrary size.

Q17. Floating-point numbers. In Python, evaluate `0.1 + 0.2`, `0.1 + 0.2 == 0.3`, and `1e16 + 1 == 1e16`. Explain. In C, store $6.02214076 \times 10^{23}$ in a `float` and in a `double` and print both with `"%.10e"`; then store 0.1 in a `float` and in a `double` and print both with `"%.20f"`. Compare with the IEEE-754 example of the lecture.

Q18. Characters and bytes. In Python, compare `len("héllo")` with `len("héllo".encode("utf-8"))`; try `ord("A")`, `ord("é")`, `chr(949)`. In C, print `strlen("héllo")` (with `#include <string.h>`; the result is printed with `"%zu"`), then `printf("%d", 'A')` and `printf("%c", 66)`. Explain each result with the lecture on character encodings.

Q19. Mathematical functions. Write a program reading the coordinates of two points of the plane and printing the distance between them, with 3 decimal digits. Python: `float(input())`, `math.sqrt`, and the format specifier `f"{d:.3f}"`. C: `double x;` then `scanf("%lf", &x);` (note `%lf` for reading a `double`, but `%f` or `%.3f` for printing one), `sqrt` and `pow` from `<math.h>`. On Linux, you may need to add `-lm` at the end of the compilation command to link the math library.

Q20. Addresses. Type in the pointer example of the lecture and, before running it, predict the values of `i`, `j`, and `k`; then print them. Print the address of `i` with `printf("%p", (void*) &i)`: does it change from one run to the next? In Python, run

Python

```python
a = 42
b = a
print(id(a) == id(b), a is b)
b = 43
print(id(a) == id(b), a is b)
```

and explain the output using the notions of address and of typing seen in the lecture.