Hello, World
Now that you’ve installed Cairo through Scarb, it’s time to write your first Cairo program.
It’s traditional when learning a new language to write a little program that
prints the text `Hello, world!` to the screen, so we’ll do the same here!
> Note: This book assumes basic familiarity with the command line. Cairo makes
> no specific demands about your editing or tooling or where your code lives, so
> if you prefer to use an integrated development environment (IDE) instead of
> the command line, feel free to use your favorite IDE. The Cairo team has developed
> a VSCode extension for the Cairo language that you can use to get the features from
> the language server and code highlighting. See [Appendix F][devtools]
> for more details.

[devtools]: ./appendix-06-useful-development-tools.md
Creating a Project Directory
You’ll start by making a directory to store your Cairo code. It doesn’t matter
to Cairo where your code lives, but for the exercises and projects in this book,
we suggest making a _cairo_projects_ directory in your home directory and keeping all
your projects there.
Open a terminal and enter the following commands to make a _cairo_projects_
directory.
For Linux, macOS, and PowerShell on Windows, enter this:
```shell
mkdir ~/cairo_projects
cd ~/cairo_projects
```
For Windows CMD, enter this:
```cmd
> mkdir "%USERPROFILE%\cairo_projects"
> cd /d "%USERPROFILE%\cairo_projects"
```
> Note: From now on, for each example shown in the book, we assume that
> you will be working from a Scarb project directory. If you are not using Scarb, and try
to run the examples from a different directory, you might need to adjust the commands
accordingly or create a Scarb project.

Creating a Project with Scarb
Let’s create a new project using Scarb.
Navigate to your _cairo_projects_ directory (or wherever you decided to store your
code). Then run the following:
```bash
scarb new hello_world
```
It creates a new directory and project called _hello_world_. We’ve named our project
_hello_world_, and Scarb creates its files in a directory of the same name.
Go into the _hello_world_ directory with the command `cd hello_world`. You’ll see that
Scarb has generated two files and one directory for us: a _Scarb.toml_ file and a _src_
directory with a _lib.cairo_ file inside.
It has also initialized a new Git repository along with a `.gitignore` file

> Note: Git is a common version control system. You can stop using version control system by using the `--no-vcs` flag.
> Run `scarb new --help` to see the available options.

Open _Scarb.toml_ in your text editor of choice. It should look similar to the code in Listing {{#ref scarb-content}}.

<span class="filename">Filename: Scarb.toml</span>

```toml
[package]
name = "hello_world"
version = "0.1.0"
edition = "2024_07"
# See more keys and their definitions at https://docs.swmansion.com/scarb/docs/reference/manifest
[dependencies]
# foo = { path = "vendor/foo" }
```

{{#label scarb-content}}

<span class="caption">Listing {{#ref scarb-content}}: Contents of _Scarb.toml_ generated by `scarb new`</span>

This file is in the [TOML][toml doc] (Tom’s Obvious, Minimal Language) format, which is Scarb’s configuration format.

The first line, `[package]`, is a section heading that indicates that the following statements are configuring a package. As we add more information to this file, we’ll add other sections.

The next three lines set the configuration information Scarb needs to compile your program: the name of the package and the version of Scarb to use, and the edition of the prelude to use. The prelude is the collection of the most commonly used items that are automatically imported into every Cairo program. You can learn more about the prelude in [Appendix D][prelude].

The last line, `[dependencies]`, is the start of a section for you to list any of your project’s dependencies. In Cairo, packages of code are referred to as crates. We won’t need any other crates for this project.

> Note: If you're building contracts for Starknet, you will need to add the `starknet` dependency as mentioned in the [Scarb documentation][starknet package].

The other file created by Scarb is _src/lib.cairo_, let's delete all the content and put in the following content, we will explain the reason later.

```cairo,noplayground
mod hello_world;
```

Then create a new file called _src/hello_world.cairo_ and put the following code in it:

<span class="filename">Filename: src/hello_world.cairo</span>

```cairo,file=hello_world.cairo
fn main() {
    println!("Hello, World!");
}
```

We have just created a file called _lib.cairo_, which contains a module declaration referencing another module named `hello_world`, as well as the file _hello_world.cairo_, containing the implementation details of the `hello_world` module.
Scarb requires your source files to be located within the _src_ directory.
The top-level project directory is reserved for README files, license information, configuration files, and any other non-code-related content.
Scarb ensures a designated location for all project components, maintaining a structured organization.
If you started a project that doesn't use Scarb, you can convert it to a project that does use Scarb. Move the project code into the _src_ directory and create an appropriate _Scarb.toml_ file. You can also use `scarb init` command to generate the _src_ folder and the _Scarb.toml_ it contains.
```txt
% % %  Scarb.toml
% % %  src
%    % % %  lib.cairo
%    % % %  hello_world.cairo
```
<span class="caption"> A sample Scarb project structure</span>
[toml doc]: https://toml.io/
[prelude]: ./appendix-04-cairo-prelude.md
[starknet package]: https://docs.swmansion.com/scarb/docs/extensions/starknet/starknet-package.html
Building a Scarb Project
From your _hello_world_ directory, build your project by entering the following command:
```bash
$ scarb build
   Compiling hello_world v0.1.0 (file:///projects/Scarb.toml)
    Finished release target(s) in 0 seconds
```
This command creates a `sierra` file in _target/dev_, let's ignore the `sierra` file for now.
If you have installed Cairo correctly, you should be able to run the `main` function of your program with the `scarb cairo-run` command and see the following output:
```shell
$ scarb cairo-run
Running hello_world
Hello, World!
Run completed successfully, returning []
```
Regardless of your operating system, the string `Hello, world!` should be printed to the terminal.
If `Hello, world!` did print, congratulations! You've officially written a Cairo program. That makes you a Cairo programmer — welcome!
Anatomy of a Cairo Program
Let's review this "Hello, world!" program in detail. Here's the first piece of

the puzzle:
```cairo,noplayground
fn main() {
}
```
These lines define a function named `main`. The `main` function is special: it
is always the first code that runs in every executable Cairo program. Here, the
first line declares a function named `main` that has no parameters and returns
nothing. If there were parameters, they would go inside the parentheses `()`.
The function body is wrapped in `{}`. Cairo requires curly brackets around all
function bodies. It's good style to place the opening curly bracket on the same
line as the function declaration, adding one space in between.
> Note: If you want to stick to a standard style across Cairo projects, you can
> use the automatic formatter tool available with `scarb fmt` to format your code in a
> particular style (more on `scarb fmt` in
> [Appendix F][devtools]). The Cairo team has included this tool
> with the standard Cairo distribution, as `cairo-run` is, so it should already be
> installed on your computer!

The body of the `main` function holds the following code:
```cairo,noplayground
    println!("Hello, World!");
```
This line does all the work in this little program: it prints text to the
screen. There are four important details to notice here.
First, Cairo style is to indent with four spaces, not a tab.
Second, `println!` calls a Cairo macro. If it had called a function instead, it would be
entered as `println` (without the `!`).
We'll discuss Cairo macros in more detail in the ["Macros"][macros] chapter. For now,
you just need to know that using a `!` means that you're calling a macro instead of a
normal function and that macros don't always follow the same rules as functions.
Third, you see the `"Hello, world!"` string. We pass this string as an argument to `println!
`, and the string is printed to the screen.
Fourth, we end the line with a semicolon (`;`), which indicates that this
expression is over and the next one is ready to begin. Most lines of Cairo code
end with a semicolon.
[devtools]: ./appendix-06-useful-development-tools.md
[macros]: ./ch11-05-macros.md
{{#quiz ../quizzes/ch01-02-hello-world.toml}}
# Summary
Let's recap what we've learned so far about Scarb:
- We can install one or multiple Scarb versions, either the latest stable or a specific one, using asdf.
- We can create a project using `scarb new`.
- We can build a project using `scarb build` to generate the compiled Sierra code.
- We can execute a Cairo program using the `scarb cairo-run` command.

An additional advantage of using Scarb is that the commands are the same no matter which operating system you’re working on. So, at this point, we’ll no longer provide specific instructions for Linux and macOS versus Windows.

You’re already off to a great start on your Cairo journey! This is a great time to build a more substantial program to get used to reading and writing Cairo code.