

Get started with Mojo

On this page, we'll show you how to create the classic "Hello world" starter program with Mojo. If you'd rather read how to write Mojo code, see the [introduction to Mojo](#).

By installing Mojo, you understand and agree to our [software license](#).

1. Create a new project

To create a new Mojo project, we'll use [Magic](#)—a virtual environment manager and package manager based on conda.

1. Install Magic on macOS or Ubuntu Linux with this command:

```
$ curl -ssl https://magic.modular.com | bash
```

Then run the `source` command printed in your terminal.

2. Create a Mojo project called "hello-world":

```
$ magic init hello-world --format mojoproject
```

This creates a directory named `hello-world` and installs the Mojo project dependencies—the only dependency for a Mojo project is the `max` package (Mojo is [bundled with MAX](#)).

3. Start a shell in the project virtual environment:

```
$ cd hello-world && magic shell
```

That's it! The `magic shell` command activates the virtual environment so you can now start using Mojo. For example, you can check your Mojo version like this:

```
$ mojo --version
```

Click here to install the Mojo nightly build.



2. Run code in the REPL

First, let's use the Mojo [REPL](#), which allows you to write and run Mojo code in a command prompt:

1. To start a REPL session, type `mojo` and press `Enter`.
2. Type `print("Hello, world!")` and press `Enter` twice (a blank line is required to indicate the end of an expression).

The result looks like this:

```
$ mojo
Welcome to Mojo! 🔥
```

```
Expressions are delimited by a blank line.
Type `:quit` to exit the REPL and `:mojo help repl` for further assistance.
```

```
1> print("Hello world")
Hello world
```

3. To exit REPL, type `:quit` and press `Enter`, or press `Ctrl + D`.

You can write as much code as you want in the REPL. You can press `Enter` to start a new line and continue writing code, and when you want Mojo to evaluate the code, press `Enter` twice. If there's something to print, Mojo prints it and then returns the prompt to you.

The REPL is primarily useful for short experiments because the code isn't saved. So when you want to write a real program, you need to write the code in a `.mojo` source file.

3. Run a Mojo file

Now let's write the code in a Mojo source file and run it with the [mojo](#) command:

1. Create a file named `hello.mojo` (or `hello.🔥`) and add the following code:

```
fn main():
    print("Hello, world!")
```

That's all you need. Save the file and return to your terminal.

2. Now run it with the `mojo` command:

```
$ mojo hello.mojo
```

Hello, world!

4. Build an executable binary

Finally, let's build and run that same code as an executable:

1. Create an executable file with the [build](#) command:

```
$ mojo build hello.mojo
```

The executable file uses the same name as the `.mojo` file, but you can change that with the `-o` option.

2. Then run the executable:

```
$ ./hello
```

```
Hello, world!
```

The [build](#) command creates a statically compiled binary file, so it contains all the code and libraries it needs to run.

You can now deactivate the virtual environment by just typing `exit`:

```
$ exit
```

Now let's try running an existing code example.

5. Run an example from GitHub

Our Mojo code examples in GitHub include a Magic configuration file so you can simply clone the repo and run the code with `magic`. For example:

1. Clone the Mojo repo:

```
$ git clone https://github.com/modularml/mojo.git
```

Only if you installed the nightly build, also checkout the nightly branch:

```
$ git checkout nightly
```

2. Navigate to the examples:

```
$ cd mojo/examples
```

3. Run some code:

```
$ magic run mojo hello_interop.mojo
```

```
Hello Mojo 🔥!
```

```
9
```

```
6
```

```
3
```

```
Hello from Python!
```

```
I can even print a numpy array: [1 2 3]
```

6. Install our VS Code extension (optional)

To provide a first-class developer experience with features like code completion, quick fixes, and hover help, we've created a [Mojo extension for Visual Studio Code](#).

Update Mojo

To update the Mojo version in your project, use `magic add` and specify the `max` package version.

For example, if you want to always use the latest version of Mojo, you can use the `*` wildcard as the version and then simply run `magic update` (you must run `magic add` within the project path):

```
$ cd hello-world
```

```
$ magic add max
```

```
$ magic update
```

Although the wildcard option allows `magic update` to always install the latest version, it also updates the `magic.lock` file in your project with the explicit version you've installed. This ensures that anybody else who initializes the project also gets the same package version (until you run `magic update` again).

To be more specific with your package version, you can use any of the [Python package version specifiers](#). For example:

```
$ magic add "max~=24.4"
```

```
$ magic add "max>=24.4,<24.5"
```

Next steps


- If you're new to Mojo, we suggest you learn the language basics in the [introduction to Mojo](#).
- To learn more about the `magic` tool, read [Get started with Magic](#).
- Explore more code examples in the [the Mojo repo](#). In addition to several `.mojo` examples, the repo includes [Jupyter notebooks](#) that teach advanced Mojo features.
- To see all the available Mojo APIs, check out the [Mojo standard library reference](#).

If you have issues during install, check our [known issues](#).

Note

To help us improve Mojo, we collect some basic system information and crash reports. [Learn more](#).

Was this page helpful?  

 Edit this page

