

# Lab 1: raylib install & tinkering

---

Installations are difficult because they are machine specific, and change over time. Each depends on your hardware, OS, prior installs, coding environment, and individual preferences.

As developers we progress over time. We learn each time we add/configure a new environment, adapt our setup, change web engine, profile, shell default and so on. Each time we struggle, it is frustrating, we are worried, but it is also an opportunity to learn. In this class we are fortunate because we have the chance to learn from each other, helping each other in turn.

Gaining CS experience includes knowing our system configuration, and because we struggle and hence learn maintaining it.

For this course, I am using a new system, so my configuration is fresh. I am giving you indications so you can compare/model but you will have to adapt it to your machine. There will be classmates, whose systems may closer than mine, because you have been taking the same class(es), or you run the same OS. We are in it together and we should learn from each other, and share lessons.

Setting up your laptop is part of this first lab.

## Installations

Figure out how to globally install **Python raylib** on your laptop.

I recommend reading through search links, and official webpage and source code readme. The search terms are embedded in the first search below.

- [Qwant search](#)
- Official [pip package](#)
- Package [source code](#)

Personally I have installed

- **Python 3.9.6**, and **pip 25.3**

**pip** is a package manager (package management system) written in Python and is used to install and manage software packages. The Python Software Foundation recommends using pip to install Python applications and its dependencies during deployment. ([Wikipedia](#)).

As *"most distributions of Python come with pip preinstalled. Python 2.7.9 and later (on the python2 series), and Python 3.4 and later include pip by default"*, you should check what you have.

You likely can check your installed versions at the shell with the following commands

```
%python3 --version
%pip3 --version
```

I installed

- **pip3 install raylib**

and have the following installed

```
> pip3 list
Package      Version
-----
altgraph     0.17.2
cffi         2.0.0
future       0.18.2
macholib     1.15.2
pip          25.3
pyparser     2.23
raylib       5.5.0.4
setuptools   58.0.4
six          1.15.0
wheel        0.37.0
> pip3 index versions raylib
raylib (5.5.0.4)
```

Another package installer on macOSX you might have encountered is [brew](#), which you might have also installed. The following Formulae might be all you need [pkgconf](#) and [raylib](#).

In **Visual Studio Code**, I also have the following extensions installed.

- Python language support from Microsoft
- Vim from vscodevim
- Print
- Markdown PDF
- Code Spell Check

## Testing [python raylib](#)

To test your [raylib](#) installation, you should use the following code:

```
from pyray import *

init_window(800, 450, "Hello")
while not window_should_close():
    begin_drawing()
    clear_background(WHITE)
    draw_text("Hello world", 190, 200, 20, VIOLET)
    end_drawing()
close_window()
```

Saved as [test\\_raylib.py](#), you can run it with the Run button or the following command

```
> python3 test_raylib.py
```

---

## Three Programs

As you are reading code below, I encourage you to learn some about the `raylib` calls, by referring to

- [raylib cheatsheet](#)
- [its math cheatsheet](#)

### 1. `fixme.py`: `raylib` in Python...

To introduce you to `raylib` in Python the first program requires you to correct `fixme.py`.

Try to run it. The function calls are not proper. Use what I explained to make that code run. Do the following in turn.

1. Fix the code outside the loop, and run it to make sure it works.
2. Uncomment loop block and fix it as well

### 2. `core_input_keys.py`

Run and play with `core_input_keys.py`: manipulating the ball, which isn't constraint to the screen.

Your task is to update the code so that the ball stops at all the window boundaries. The user controls should not allow any part of the ball to disappear.

Think through and make sure to have a plan before adding code. Update the code a line at a time, and check it incrementally. No much code is needed.

You should add a constant so that the step is not a magic number and it isn't as tedious to interact when you check your progress.

### 3. `core_random_sequence.py`

Run `core_random_sequence.py` which displays random sequences of bars. Experiment with its execution and controls, as you read the code so as to orient yourself connecting the various displays to the actual `raylib` lines that produce each.

Read the above `raylib` documentation sufficiently to be able to get intuition about the code inner working.

## Warm-up

Where are the following functions defined, i.e. coming from?

- `load_random_sequence`: \_\_\_\_\_

- 
- `remap`: \_\_\_\_\_
- 

Fill in above and write each call description.

## Task

Add a **press RIGHT** feature that makes the displayed random chart sequence moves one from the right.

To do so, you must

- implement one function and
- add text display that indicate that new feature

**Bonus:** Do something of your choice, like sort with LEFT.

Show me before the end of the lab, everything you have achieved.