Logographer

Hankertrix

November 11, 2024

Contents

1 Overview

Logographer is a Python program for drawing logos. It requires at least **Python version 3.11** to run.

2 Installation

Logographer requires Python to be installed, so please install Python before trying to run the program. This only applies to Windows, and macOS and Linux both have Python come preinstalled, so there's no need to install Python on these operating systems. However, you may need to update your system's version of Python or install a newer Python version for the program to run successfully. This program requires at least **Python version 3.11**.

Minimum Python version: 3.11

2.1 Installing Python on Windows

2.1.1 Manual installer

Download the Python installer from this link (this links to the official 64-bit Windows installer for Python 3.12.3) and run the installer. Follow the steps to install Python and you should be done.

2.1.2 Winget (package manager)

Winget is the built-in package manager that comes with Windows. Open up the terminal by using Window + R to open the run dialogue, and type in `cmd` and press enter to open the Command Prompt. Copy and paste the following command into the command prompt, press enter, and you should be done.

```
winget install -e --id Python.Python.3.11
```

2.1.3 Scoop (package manager)

To use the Scoop package manager, you will need to download and install Scoop first. Feel free to skip this step if you already have Scoop installed. Open up the terminal by using Window + R to open the run dialogue, and type in `powershell` and to open Powershell. Copy and paste the following command into the command prompt, press enter, and you should be done installing Scoop.

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

Test that Scoop is working properly by using the command below:

```
scoop
```

Now to install Python, copy and paste the command below, press enter, and you should be done installing Python.

```
scoop install main/python
```

2.1.4 Chocolatey (package manager)

To use the Chocolatey package manager, you will need to download and install Chocolatey first. Feel free to skip this step if you already have Chocolatey installed. Open up the terminal by using Window + R to open the run dialogue, and type in `powershell` and press Ctrl + Shift + Enter to open Powershell in Administrator mode. Copy and paste the following command into the command prompt, press enter, and you should be done installing Chocolatey.

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol = `
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
iex ((New-Object System.Net.WebClient).DownloadString(
'https://community.chocolatey.org/install.ps1'
))
```

Test that Chocolatey is working properly by using the command below:

```
choco
```

Now to install Python, copy and paste the command below, press enter, and you should be done installing Python.

```
choco install python
```

2.2 Updating your Python version on macOS

2.2.1 Manual installer

Download the Python installer from this link (this links to the official 64-bit macOS installer for Python 3.12.3) and follow the instructions to install the newer version of Python and you should be done.

2.2.2 Homebrew (package manager)

To use the Homebrew package manager, you will need to download and install Homebrew first. Feel free to skip this step if you already have Homebrew installed. Open up the terminal and copy and paste the following command into the command prompt, press enter, and you should be done installing Homebrew.

```
/bin/bash -c \
"$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Test that Homebrew is working properly by using the command below:

```
brew help
```

Now to update your Python version, copy and paste the command below, press enter, and you should be done updating Python.

```
brew install python3
```

2.3 Updating your Python version on Linux

Updating your Python version on Linux is just updating your system. The default Python version that comes on all modern Linux distributions should be Python 3.11, so there should be no need to do this.

2.3.1 Arch-based distributions (Arch, Manjaro, EndeavourOS, etc.)

```
sudo pacman -Syu
```

2.3.2 Debian-based distributions (Debian, Ubuntu, Linux Mint, etc.)

```
sudo apt update && sudo apt upgrade -y
```

2.3.3 Fedora and Fedora derivatives (Nobara, Asahi Linux, etc.)

```
sudo dnf upgrade
```

2.3.4 Gentoo

```
sudo emaint -a sync
sudo emerge -avuDN @world
```

2.3.5 Void Linux

```
sudo xbps-install -Su
```

2.4 Installing `tk` on macOS and Linux

On macOS and Linux, the `tk` library isn't installed by default, so it needs to be installed. This shouldn't be a problem on Windows.

2.4.1 macOS

If you installed Python **manually** using the **installer**, then there is **no need** to install the `tk` library.

Otherwise, if you used Homebrew to install Python, then use the command below to install the `tk` library.

```
brew install python-tk
```

2.4.2 Arch-based distributions (Arch, Manjaro, EndeavourOS, etc.)

```
sudo pacman -S tk
```

2.4.3 Debian-based distributions (Debian, Ubuntu, Linux Mint, etc.)

```
sudo apt install tk-dev
```

2.4.4 Fedora and Fedora derivatives (Nobara, Asahi Linux, etc.)

```
sudo dnf install tk-devel
```

2.4.5 Gentoo

Update your USE flags for Python.

```
sudo cat "USE=\"tk\"" >> /etc/portage/make.conf
```

Re-emerge your packages.

```
sudo emerge -avuDN @world
```

2.4.6 Void Linux

```
sudo xbps-install -S tk
```

3 Running the program

Go to the folder that contains the program, which is most likely where you have extracted the zip file to, or where you have cloned the repository. Once you are in the directory for the program, simply run the command below to start the program:

```
python main.py
```

Alternatively, you can run the program by opening the `main.py` file using Python. On most systems, this should be just double-clicking on the file after you have Python installed. Otherwise, you might have to right-click on the file in your file explorer and use the "Open with" option to select Python to run the `main.py` file.

4 How to use the program

For the most part, everything should be pretty clear from the dialogues and prompts given by the program when running it. The only part of the program that may be a bit more unintuitive is the file explorer, which can be interacted with by using the mouse to click on the items, or with the keys documented below.

4.1 File explorer key binds

Key	Action
Up Arrow Key	Move the cursor up 1 item
Down Arrow Key	Move the cursor down 1 item
Left Arrow Key	Go to the parent folder
Right Arrow Key	Select a file or enter a folder
w	Move the cursor up 1 item
a	Go to the parent folder
s	Move the cursor down 1 item
d	Select a file or enter a folder
h	Go to the parent folder
j	Move the cursor down 1 item
k	Move the cursor up 1 item
l	Select a file or enter a folder
c	Select the current folder (for saving a file)

These key binds should be quite intuitive, maybe except `hjkl`, which are Vim key binds, so they should be quite intuitive if you are a Vim user.

5 Logo file format

The program has the ability to load and save logo files, which is in a standard file format called TOML. The logo file is a TOML table with two keys, one called `data` and another called `background_colour`.

5.1 Background colour (`background_colour`)

The key called `background_colour` can either be a string, containing a colour name like `black`, or a hex string like `#000000`, or a list of numbers representing the RGB value of the colour, like `[0, 0, 0]`. This determines the background colour of the screen. This key can be left out, which means it is **optional**, and will **default** to turtle's background colour, which is **white** if it is not given. Below are a few examples:

```
background_colour = "#000000"

background_colour = [255, 0, 0]  # red

background_colour = "blue"
```

5.2 Data (`data`)

The key called `data` is where all the data for each of the icon should be. It is a list containing either dictionaries (TOML tables) or lists of dictionaries. You can nest as many lists as you want, as long as every single item of every list is a dictionary in the format documented below.

5.2.1 Vertices (`vertices`)

Each dictionary (TOML table) **must contain** a list of vertices, or points. It is required for the program to be able to draw the icon from the file. Each vertex, or point, is a matrix, or a list of list of integers or floats. For example, the matrix below corresponds to the point (2, 1).

```
[[2], [1]]
```

The **square brackets** wrapping each of the numbers **is important**, and must not be left out. The program will tell you that your file is invalid if the square brackets are left out.

The list of vertices must contain either **2 points, or 4 points**. The program will say that your file is invalid if the list of vertices contains any other number of vertices. For a **straight edge**, the list of vertices should have **2 points**, i.e.

```
{ vertices = [[[3], [5]], [[7.7], [9.9]]] }
```

The program will draw a straight line between the two points.

For a **curved edge**, the list of vertices should have **4 points**, as it is drawn as a Bézier curve. The first and the last points are the start and end points of the curve respectively, and the second and third points are the points on the control polygon of the Bézier curve. For example:

```
{ vertices = [[[3.3], [5.5]], [[7], [9]], [[11], [13]], [[15], [17]]] }
```

5.2.2 Fill colour (`fill_colour`)

This should be self-explanatory, this is the colour to fill the shape with. This option is **optional** and can be left out. If not given, the fill colour will **default to the last fill colour** that was used. It can either be a string, containing a colour name like `black`, or a hex string like `#000000`, or a list of numbers representing the RGB value of the colour, like `[0, 0, 0]`. A few examples are shown below:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = "#000000" }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = "blue" }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = [0, 255, 0] }  # green
```

This option will usually accompany the option below.

5.2.3 Start fill (`start_fill`)

This should also be self-explanatory, it tells the program to start filling the shape being drawn with the fill colour (`fill_colour`) given. This option is also **optional** and will default to `false` if it's not given. It is a boolean that can be either `true` or `false`, but you'll usually want to set it to `true`. This should be given together with the fill colour, otherwise the option doesn't do anything, and the fill colour option will also not do anything, since the program doesn't fill anything with colour. Below are some examples:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = "#000000", start_fill = true }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = "blue", start_fill = true }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], fill_colour = [0, 255, 0], start_fill = true }  # green
```

5.2.4 End fill (`end_fill`)

This is another self-explanatory option. This tells the program to stop filling the shape with the fill colour. For this to take effect, the program must already be filling a shape with colour, otherwise there is nothing to stop. This option is also **optional** and defaults to `false` if it's not given. It is a boolean that can either be `true` or `false`, but you'll usually want to set it to `true`. This can be given on its own without the fill colour, since it just tells the program to stop filling the shape with colour. Below is an example:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], end_fill = true }
```

5.2.5 Pen colour (`pen_colour`)

Yet another self-explanatory option. This tells the program what colour to use to draw the outline of the shape. This option is **optional** and can be left out. If it is left out, the pen colour will **default** to the **previous pen colour** being used. This option can either be a string, containing a colour name like `black`, or a hex string like `#000000`, or a list of numbers representing the RGB value of the colour, like `[0, 0, 0]`. A few examples are shown below:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], pen_colour = "#000000" }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], pen_colour = "blue" }

{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], pen_colour = [0, 255, 0] }  # green
```

5.2.6 Pen size (`pen_size`)

Once again, yet another self-explanatory option. This tells the program what pen size to use when drawing the edge, be it a curved edge or a straight edge. It affects how thick (or how wide) the line will be when drawn. It is another option that is **optional** and will **default** to a **pen size of 1** when it is not given. This option takes an integer that determines the thickness of the pen. An example is shown below:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]]
], pen_colour = "#000000", pen_size = 10 }
```

However, do take note that this option will not scale well if you decide to transform the logo after drawing it, such as by scaling or shearing the logo, as those transformations only affect the vertices and not the pen size. If you would like to have the pen thickness be affected by the transformations, consider creating an outer shape that is filled with the desired colour, and an inner shape that is smaller than the outer shape by the pen thickness and is filled with the background colour.

5.2.7 Number of segments (`number_of_segments`)

Finally, an option that isn't self-explanatory. This option tells the program to draw the Bézier curve in the given number of segments. Simply put, this represents the resolution of the Bézier curve drawn. This option should be given with a list of vertices with **4 points**, as that is a Bézier curve. Otherwise, this option will have no effect. This option is **optional**, and defaults to **100 segments** if it is not given, which should be sufficient for most icons. The higher the number, the smoother the Bézier curve will look. However, a higher number also means that the program will take a longer time to draw your icon. The icons you see in the next major section are all drawn with this option set to the default value of 100, so it is definitely more than enough to draw beautiful icons. An example of setting this option is given below:

```
{ vertices = [
    [[3], [5]],
    [[7.7], [9.9]],
    [[11], [13]],
    [[15.5], [17.7]]
], number_of_segments = 500 }
```

5.2.8 Trivial example of the full data

Below is an example of a blue square:

```
data = [
    { vertices = [
        [[-300], [300]],
        [[300], [300]]
    ], fill_colour = "blue", pen_colour = "blue", start_fill = true },
    { vertices = [
        [[300], [300]],
        [[300], [-300]]
    ]},
    { vertices = [
        [[300], [-300]],
        [[-300], [-300]]
    ]},
    { vertices = [
        [[-300], [-300]],
        [[-300], [300]]
    ], end_fill = true },
]
```

5.3 Trivial example of a logo file

Below is an example of a logo file that has a green rectangle on an orange background.

```
background_colour = "orange"
data = [
    { vertices = [
        [[-300], [200]],
        [[300], [200]]
    ], fill_colour = "green", pen_colour = "green", start_fill = true },
    { vertices = [
        [[300], [200]],
        [[300], [-200]]
    ]},
    { vertices = [
        [[300], [-200]],
        [[-300], [-200]]
    ]},
    { vertices = [
        [[-300], [-200]],
        [[-300], [200]]
    ], end_fill = true },
]
```

5.4 Non-trivial examples

For non-trivial examples of what a logo file looks like, take a look at the `logo_files` folder. Inside the folder are the logo files for the logos drawn in the next section. Specifically, it includes the World Bank logo (`world_bank_logo.toml`), the CERN logo (`cern_logo.toml`) and the logo of the Rust programming language (`rust_logo.toml`).

6 Drawn logos

Just a few examples of what the program can do!

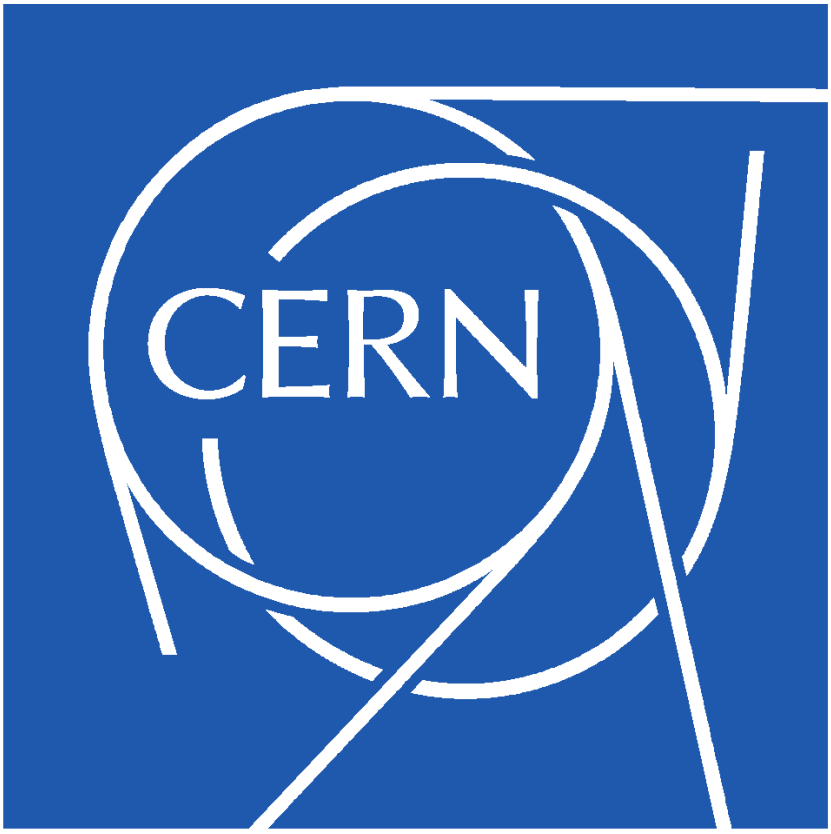
6.1 World Bank Logo

6.1.1 Original

6.1.2 Reflected about the line y = x

6.2 CERN Logo

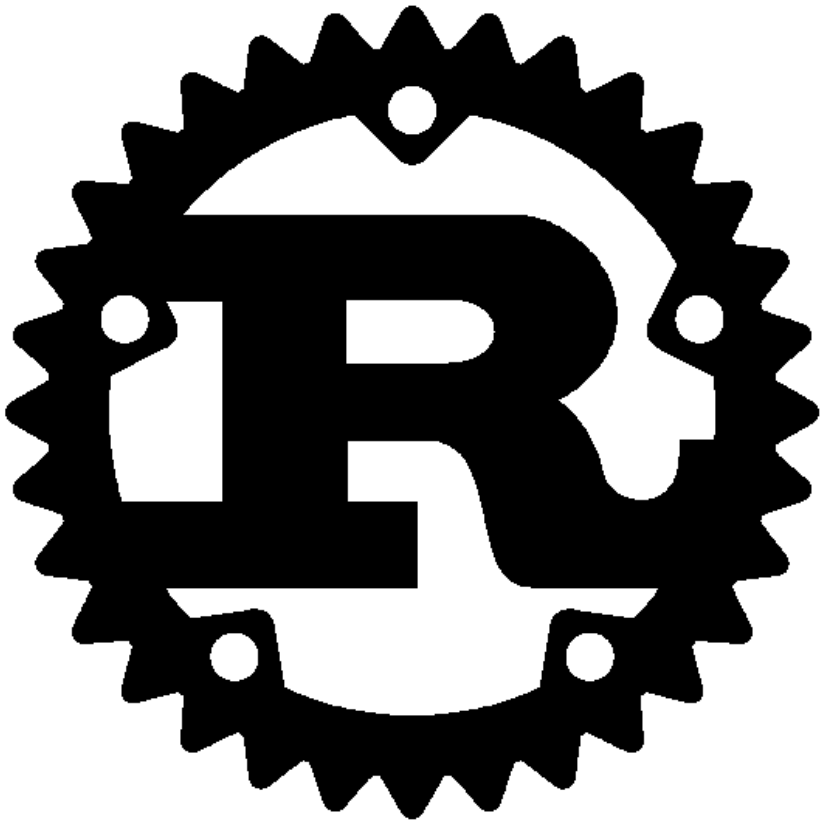
6.2.1 Original



6.2.2 Scaled down two-thirds and rotated 45 degrees clockwise

6.3 Rust Programming Language Logo

6.3.1 Original



6.3.2 Sheared with x shear factor of 1 and y shear factor of 0

7 Strengths and weaknesses

7.1 Strengths

- No additional Python packages are used, as this program is built entirely using Python's standard library.
- Can draw pretty much any logo you want if you can turn it into list of points and put it into the logo file format documented above.
- Homogeneous coordinates make it easy to transform a logo. It also allows for multiple transformations to be applied at once.
- Circles and circular arcs are turned into Bézier curves that only have an error of 0.0273% of the circle's radius in their approximation.
- It can transform your logos in 5 different ways, translation, scaling, rotation, shearing, and reflection.
- You can use it to create an icon manually, through the graphical user interface, either by typing in the coordinates manually or by clicking on the screen. However, this is an extremely slow way to create an icon, especially if it is more complex than just a few rectangular shapes. A better and faster way to draw an icon using the program is to create a logo file in the format documented above.

7.2 Weaknesses

- Can only draw one logo at a time. The screen needs to be cleared to draw another logo.
- Creating a logo is extremely tedious. Using the program's interface to create a logo is even more tedious than creating a logo file and loading the logo file.
- Having a graphical user interface makes the program quite complicated and hence it has a lot of files and directories.

8 Features

- Has a beautiful title screen (subjective).
- Has a purely graphical user interface (GUI), so there's no need to touch the command line or the terminal!
- Has a built-in file explorer, so you don't have to type the files you want to load or save manually, or wonder about why the file you have given isn't found.
- Automatically resizes the screen to make sure the logo drawn is fully visible. No more cut-off logos and scrolling in the turtle window to view the full logo, unless your logo is bigger than your entire screen, then that's on you.
- You can create an icon programmatically by using this program as a library, using the functions in the `math_utils.py` file to much more easily create your own logos. The `math_utils.py` file contains a lot of convenience functions to make logo creation easier. After creating the logo, the logo can be exported as a logo file using the functions in the `save_lib.py` file. You can find a few examples in the `default-logos` folder.
- The code is fully documented with comments and documentation strings.
- Uses TOML as the logo file format, which is easy to read and learn.

9 Licence

This program is licensed under the GNU AGPL V3 licence. You can view the `LICENCE.txt` file that is distributed with the program for the licence text.