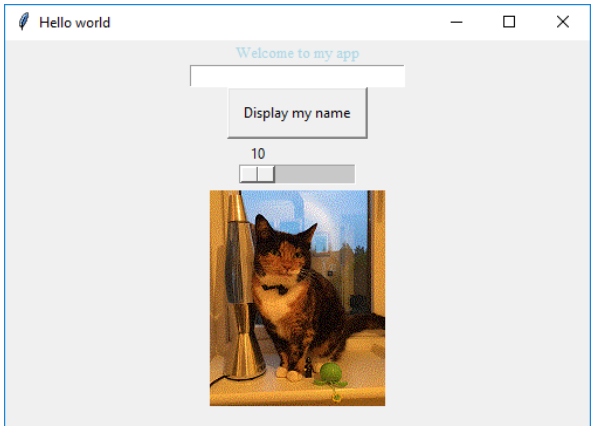
Projects

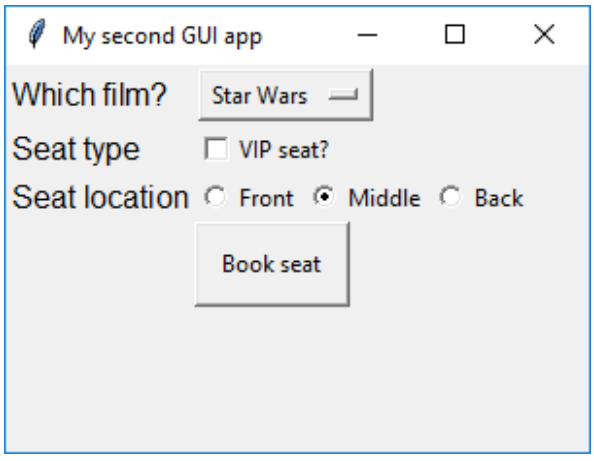
Getting started with GUIs

Make simple graphic user interfaces in Python

Step 1 What you will make

In this resource you will create two simple GUIs (graphical user interfaces) in Python.





What you will learn

By creating a GUI in Python, you will learn about:

- Using functions
- Event-driven programming, and how it differs from procedural programming

This project covers elements from the following strands of the Raspberry Pi Digital Making Curriculum (http://rpf.io/curriculum):

- Can effectively combine functionality from multiple libraries or APIs and refer to documentation (https://curriculum.raspberrypi.org/programming/developer/)

Step 2 What you will need

Hardware

- Computer (PC/Mac/Linux/Raspberry Pi)

Software

- Python 3
- `guizero` (https://lawsie.github.io/guizero/) Python module

How to install Python 3

If Python 3 or IDLE isn't installed on your computer, follow the installation instructions below for your operating system:

- Microsoft Windows
- macOS
- Raspberry Pi OS and Linux

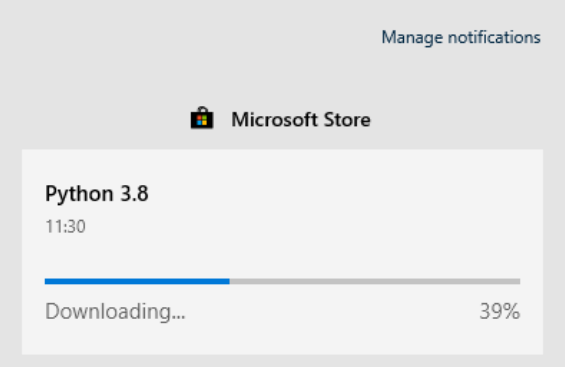
Microsoft Windows

It is recommended that you install Python via the Microsoft Store . If this is not possible, you can also use a Python installer from www.python.org (https://www.python.org).

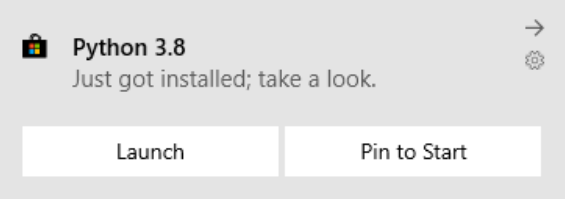
Microsoft Store (recommended)

- Open the Python 3.8 application in the Microsoft Store (ms-windows-store://pdp/?ProductId=9MSSZTT1N39L).
- Click the Get button to download and install Python 3.8.

- Python 3.8 will be downloaded and installed. Progress will be shown in the notification bar.



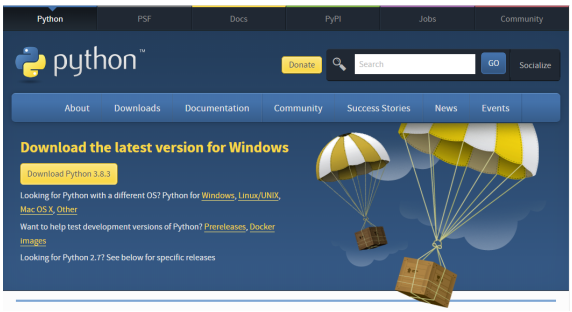
- When the installation process is complete, a notification will appear.



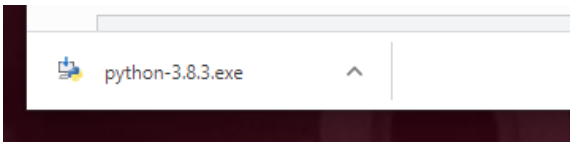
Python installer

- Open your web browser and navigate to www.python.org/downloads (https://www.python.org/downloads).

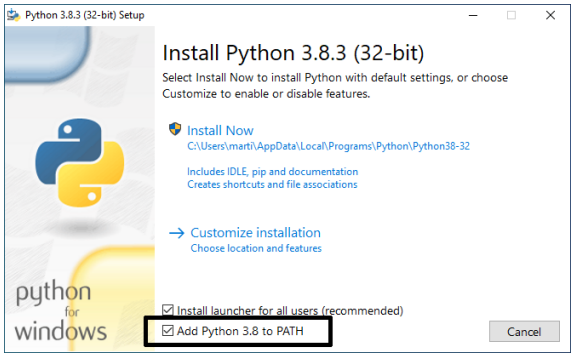
- On this web page, you will see a button to install the latest version of Python 3. Click the button, and a download will start automatically.



- Click on the `.exe` file to run it. (It will have been saved in your `Downloads` folder, or wherever your computer saves downloaded files by default.)



- In the dialogue box that opens, it is important that you first tick the box next to Add Python 3 to PATH.



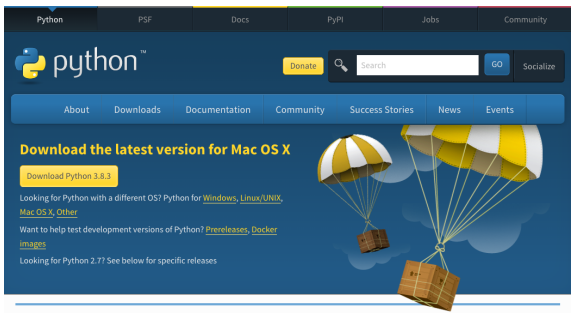
- Click on Install Now and follow the installation guide. The setup process will take a little time.

- Once the setup is complete, click on Done, and then close your web browser. Now, you can go to the Start menu to open IDLE.

macOS

- Open your web browser and navigate to www.python.org/downloads [(https://www.python.org/downloads)](https://www.python.org/downloads).

- On this web page, you will see a button to install the latest version of Python 3. Click the button, and a download will start automatically.



- Click on the download in the dock to start the installation process.

- Click on Continue and follow the installation guide. The installation may take a little time.

- When the installation process is complete, click on Close.
- Open IDLE from your Applications.

Raspberry Pi OS and other Linux (Debian-based) distributions

Most distributions of Linux come with Python 3 already installed, but they might not have IDLE, the default IDE (interactive development environment), installed.

Use `apt` to check whether they are installed and install them if they aren't.

- Open a terminal window and type:

```
sudo apt update
sudo apt install python3 idle3
```

This will install Python 3 (and IDLE), and you should then be able to find it in your Application menu.
You can use `pip` to install the `guizero` module:

Installing Python modules with pip

Installing Python modules with pip

`pip` or `pip3` is a command line tool for installing Python 3 modules.

Modules can be downloaded as packages from the Python Package Index (https://pypi.python.org/pypi) and installed on your computer automatically.

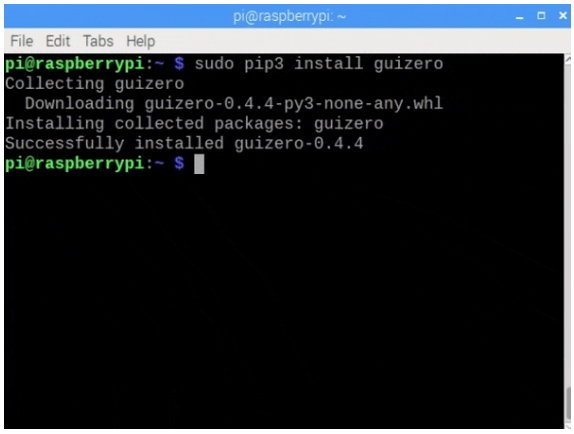
To install a module, use the `pip3 install name_of_module` command, replacing `name_of_module` with the module you wish to install.

Follow the instructions below for your operating system.

Raspberry Pi

- Open a terminal window by clicking Menu > Accessories > Terminal.
- Enter this command to install a module:

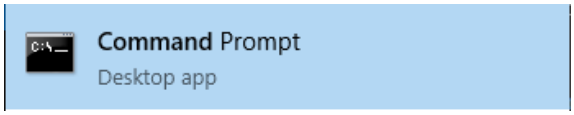
```
sudo pip3 install name_of_module
```



If you experience problems, have a look at our guide *Using pip on Raspberry Pi* (https://projects.raspberrypi.org/en/projects/using-pip-on-raspberry-pi).

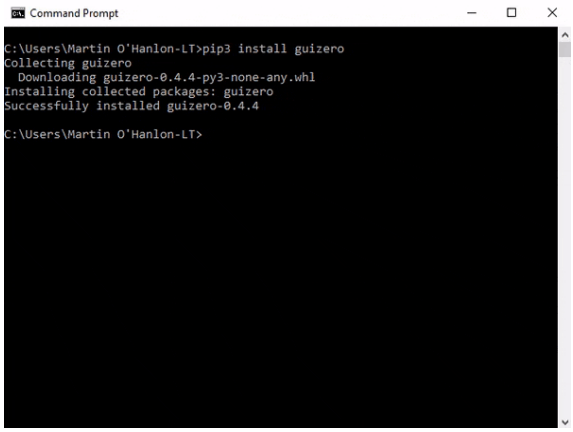
Windows

- Open a command prompt by clicking Start > Windows System > Command Prompt, or by typing 'command' into the start menu's search bar.



- Enter this command to install a module:

```
pip3 install name_of_module
```



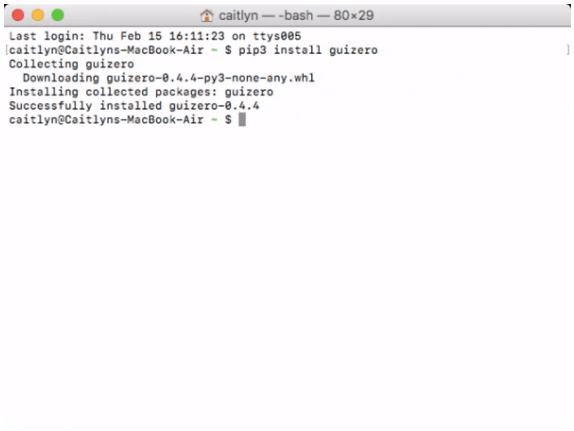
If you experience problems, have a look at our guide *Using pip on Windows* (https://projects.raspberrypi.org/en/projects/using-pip-on-windows).

macOS

- Open a terminal window by clicking Applications > Utilities > Terminal, or by typing 'terminal' into the desktop's search bar.

- Enter this command to install a module:

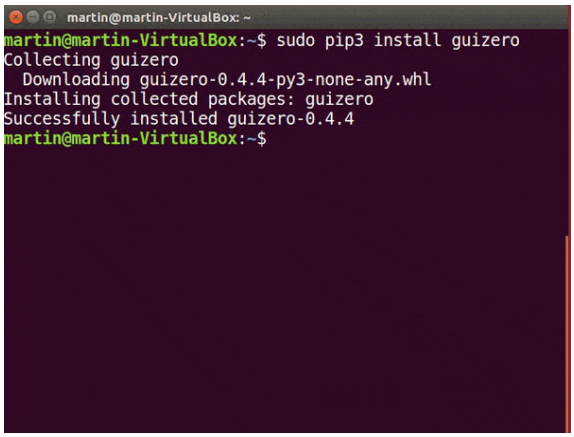
```
pip3 install name_of_module
```



Linux

- Open a terminal window.
- Enter this command to install a module:

```
sudo pip3 install name_of_module
```



Troubleshooting installation issues

There is comprehensive documentation for pip at pip.pypa.io (https://pip.pypa.io) which will help you with troubleshooting. Here are a few of the common issues, to help you identify problems.

Installation issues

If the installation of a package fails you may see an error message similar to these:

```
Could not find a version that satisfies the requirement <package-name (from versions: )>
```

```
No matching distribution found for <package-name>
```

The most common source of these errors is a misspelled package name.

You should also check that you are using the package name and not the module name. e.g. the package name for PIL (Python Imaging Library) is `pillow` and not `PIL`.

Module import issues

If the package installs but an error occurs when you try to import the module, check the following:

1. Which version of Python pip is installing packages into?

   If you have multiple versions of Python on your computer, pip might be installing modules for a different version than the one your program is using.

   It may be a case of using the right version of the pip command, make sure you are using `pip3`.

2. Is the package included in your package list?

   You can use the following command to list all the Python packages you have installed.

   ```
   pip3 list
   ```

Upgrading a package

When you install a Python package that is already on your computer, it will not update it to the latest version.

Use this command to update a Python package to the latest version:

```
pip3 install --upgrade name_of_module
```

Uninstalling a package

Use this command to uninstall a Python package:

```
pip3 uninstall name_of_module
```

Step 3 What is a GUI?

GUI stands for 'graphical user interface', and it is pronounced like 'gooey'.

If you have written Python programs before, these will probably have dealt with input as text typed in by the user, and with output appearing on the screen.

Adding a GUI to your program lets the user interact with it using buttons, menus, text boxes, and other familiar user interface features.

Step 4 Getting started

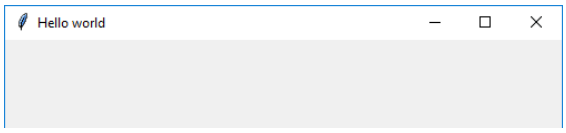
- Open up Python 3 (IDLE).
- Click on File > New File, and save your file as `gui_test.py`.
- Add a line of code at the top of your file to import the `App` class from the `guizero` module:

```
from guizero import App
```

- Now add two more lines of code to create an `App` and then display it on the screen:

```
app = App(title="Hello world")
app.display()
```

- Save your file and press **F5** to run it. You should see a GUI window that looks like this:



Congratulations, you have just built your first GUI app!

Step 5 Adding widgets

Let's start adding content to the GUI. We refer to items you can add to a GUI (such as text, text boxes, buttons, etc.) as widgets. There are a couple of rules to follow when adding a widget.

- If you want to use a new type of widget, you must import it. The first line of code in your program looks like this:

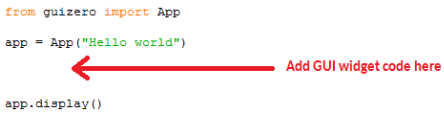
```
from guizero import App
```

As an example, if you wanted to use the `Text` widget, you would add it to the `import` statement, like this:

```
from guizero import App, Text
```

We will ask you to import various types of widgets while you work through this project. Each type of widget only needs to be added to the `import` statement once, and then you can use it as many times as you want on your GUI.

- All code that creates a widget must be added above the event loop, meaning above the `app.display()` line of code, and below the line of code where you create the app:



Throughout this guide, whenever we ask you to add widgets to the GUI, you should add them anywhere between these two lines of code.

This is because the line of code `app.display()` starts the event loop. Once it is executed, the GUI will be waiting for the user to do things such as click a button – these user actions are called events. The GUI app will constantly check whether the user has done anything new, and it will automatically update the display if necessary. The event loop blocks, so code written after the event loop will never execute. So this loop acts rather like a `while True:` loop, which is an infinite loop you may have used before when writing a Python script.

Step 6 Text widget

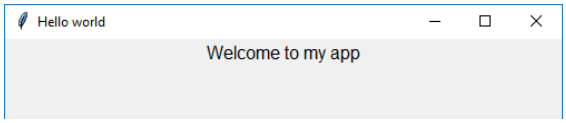
Probably the simplest widget you can add is the `Text` widget, which displays some text on the screen.

- Add `Text` to the `import` statement (check back in the Adding widgets section if you are not sure how to do this).
- Add a `Text` widget to the GUI (check back in the Adding widgets section if you are not sure where to put this code):

```
welcome_message = Text(app, text="Welcome to my app")
```

  Here we have created a `Text` widget with the name `welcome_message`. The first argument (in the brackets) tells the widget who its boss is! To be more specific, we are telling this `Text` widget that it will be controlled by the `app` object that we created earlier. The first argument given to any widget always tells it the name of its boss.

- Run your code by pressing **F5**. You should see this text displayed on your GUI:



Something has gone wrong!

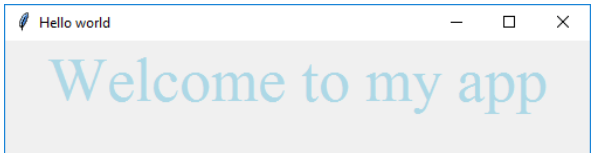
Your code should look like this, with `Text` imported at the start and the `Text` widget called `welcome_message` created above `app.display()`:

```
from guizero import App, Text

app = App(title="Hello world")

welcome_message = Text(app, text="Welcome to my app")

app.display()
```

Did you notice that we could tell the `Text` widget what content we wanted it to display by specifying `text="Welcome to my app"`? This is called a keyword argument, because we have specified the keyword `text` and the value we want. We can specify other keyword arguments too by just adding them after the `text` one and separating them by commas:

```
welcome_message = Text(app, text="Welcome to my app", size=40, font="Times New Roman", color="lightblue")
```

Here, we have used keyword arguments for the `size`, `font`, and `color` (note the American spelling!).



You can specify any font your computer has installed. Colours can be specified as colour names, but not every possible colour has a name, so you can also use Hex codes (e.g. `#ff0000`) to define colours.

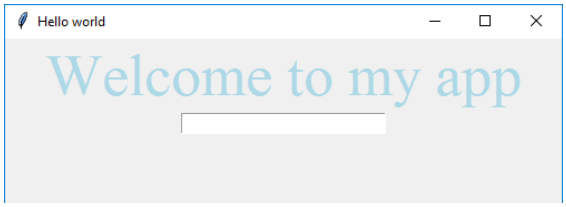
Step 7 TextBox widget

A `TextBox` widgets allows the user to type in data, a bit like the GUI version of Python's `input()` function, which you may have used before. Here's how to add a `TextBox`:

- Add `TextBox` to your `import` statement.
- Add a `TextBox` widget to the GUI:

```
my_name = TextBox(app)
```

- Run your code by pressing **F5**. You should see a small text box appear. The widget has an optional keyword argument `width`, which you can add if you wish to make the text box wider.



Step 8 PushButton widget

You can use a `PushButton` widget to create a button, and write code so that when the button is pushed, a function is called.

To keep your script nice and tidy, it is a good idea to put all the code which defines functions at the top of your program, immediately below the `import` line.

- Above the code which creates the GUI app, write a function called `say_my_name`:

```
def say_my_name():
    welcome_message.value = my_name.value
```

So this `say_my_name` function refers to the `Text` widget that you've written earlier `welcome_message`. It will set the `value` of `welcome_message` to what was typed into your `TextBox` widget (`my_name`).

You can use the `value` property with many widgets to retrieve their current value or to set it to something new.

- Add `PushButton` to your `import` statement.
- Add a `PushButton` widget to the GUI:

```
update_text = PushButton(app, command=say_my_name, text="Display my name")
```

The first argument tells the `PushButton` that the `app` is its boss. Then we use two keyword arguments: `command` tells the button which function to call when it is pressed, and `text` is the text which will be displayed on the button.

- Press **F5** to run your code. Type your name into the text box and then press the button. You should see your name displayed in large text at the top.

I need a hint

This is what your code should look like:

```
from guizero import App, Text, TextBox, PushButton

def say_my_name():
    welcome_message.value = my_name.value

app = App(title="Hello world")

welcome_message = Text(app, text="Welcome to my app", size=40, font="Times new roman", color="lightblue")
my_name = TextBox(app, width=30)
update_text = PushButton(app, command=say_my_name, text="Display my name")

app.display()
```

The `PushButton` widget is a good example of how the event loop works: the GUI waits for an event (in this case, you clicking the button), and it calls a function in response to the event. The function may contain code to change something on the GUI, and if it does, the display is updated accordingly.

Step 9 Slider widget

A slider lets users pick a value from a range of values easily, since it works rather like moving a volume control slider.

- Above the code which creates the GUI app, write a function `change_text_size`.

```
def change_text_size(slider_value):
    welcome_message.size = slider_value
```

This function will be called whenever the slider is moved, which is why it takes a parameter called `slider_value`. We must add a parameter inside the brackets so that when the slider is moved, its current value will automatically be sent to the function.

The code inside the function uses the current slider value to set the `size` of the `welcome_message`.

- Add `Slider` to your `import` statement.

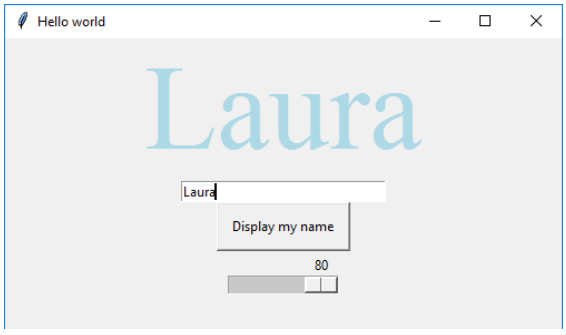
- Add a `Slider` widget to the GUI:

```
text_size = Slider(app, command=change_text_size, start=10, end=80)
```

The `command` argument specifies the function that will be called when the slider is moved — the `change_text_size` function we just created.

The values of the `start` and `end` arguments determine the largest and smallest values of the slider. Since we are using it to change text size, we have specified these so that the font does not get too large or small — the smallest it can be is 10pt and the largest is 80pt.

- Save your code and press **F5** to run it. Move the slider from side to side and watch the size of the `welcome_message` change.



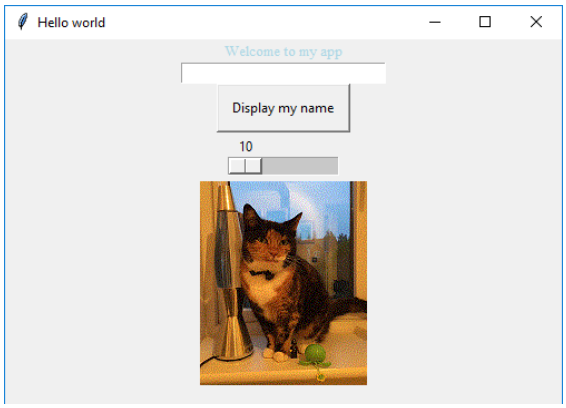
Step 10 Picture widget

You can add pictures to your GUI, as long as they are in GIF format. Sadly, animated GIFs will only display as stills.

- Find a picture in GIF format that you would like to use, or save an existing picture as a GIF file. Make sure that the picture is in the same folder as your `gui_test.py` Python file.
- Add `Picture` to your `import` statement.
- Add a `Picture` widget to the GUI:

```
my_cat = Picture(app, image="cat.gif")
```

- Press **F5** to run your code. You should see your picture appear on the GUI.



Step 11 Review

You have now learnt how to use simple GUI widgets. This is the full code for this first part:

```
from guizero import App, Text, TextBox, PushButton, Slider, Picture

def say_my_name():
    welcome_message.value = my_name.value

def change_text_size(slider_value):
    welcome_message.size = slider_value

app = App(title="Hello world")

welcome_message = Text(app, text="Welcome to my app", size=40, font="Times new roman", color="lightblue")
my_name = TextBox(app, width=30)
update_text = PushButton(app, command=say_my_name, text="Display my name")
text_size = Slider(app, command=change_text_size, start=10, end=80)
my_cat = Picture(app, image="cat.gif")

app.display()
```

You can also download a complete working example here (https://projects-static.raspberrypi.org/projects/getting-started-with-guis/698057af66a3d762f29954580d64108f0ccc4025/en/resources/gui_test.py).

Next we'll make a simple GUI for booking cinema tickets to look at some a little more complicated GUI widgets: combo menus, check boxes, and radio buttons.

Step 12 A new app

- Create a new file and save it as `gui_test2.py`.
- Add a line of code at the start of your file to import the `App` class from the `guizero` module:

```
from guizero import App
```

- Now add two more lines of code to create an `App` object and display it on the screen:

```
app = App(title="My second GUI app", width=300, height=200, layout="grid")
app.display()
```

Note that we have used some new keyword arguments: `width` and `height` set the size of the app window, and `layout` allows us to set out our widgets on an invisible grid.

Step 13 Combo widget

The `Combo` widget allows you to select an option from a drop-down menu.

- Add `Combo` to your `import` statement.
- Add a `Combo` widget to the GUI:

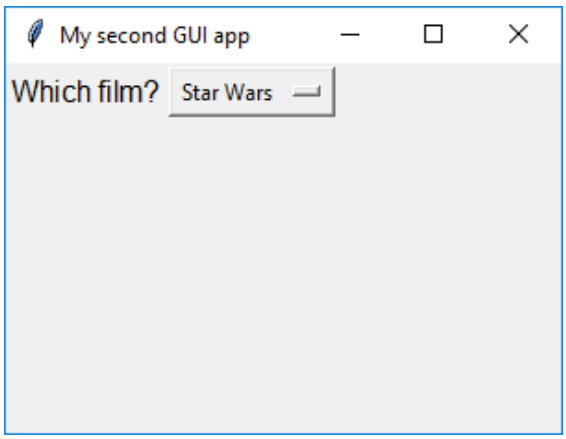
```
film_choice = Combo(app, options=["Star Wars", "Frozen", "Lion King"], grid=[1,0], align="left")
```

  - As usual, we have specified the `app` parameter to tell the `Combo` widget that the app is its master.
  - The `options` argument is a list of options we wish to display in the widget.
  - Because we specified `layout=grid` in the `app` object, we have to now include a `grid` argument with each widget to tell it where to appear. The `grid` argument should be a list containing `[x,y]` values for where you would like the widget to appear on the grid. `[0,0]` on the grid is the top left-hand corner. We can also `align` the widget within the grid square, in this case on the `"left"`.
- Save your code and press `F5` to run it. Note that the `Combo` widget appears in the very top left of the screen, even though we specified its grid position as `[1,0]`. This is because grid square `[0,0]` is empty, and empty grid squares have no height or width.

- Add the `Text` widget to your `import` statement, then add a `Text` object in grid square `[0,0]` to provide some description of what the person will be selecting using the `Combo` widget:

```
film_description = Text(app, text="Which film?", grid=[0,0], align="left")
```

  Run the program to check that the `Text` and the `Combo` are both displayed properly.



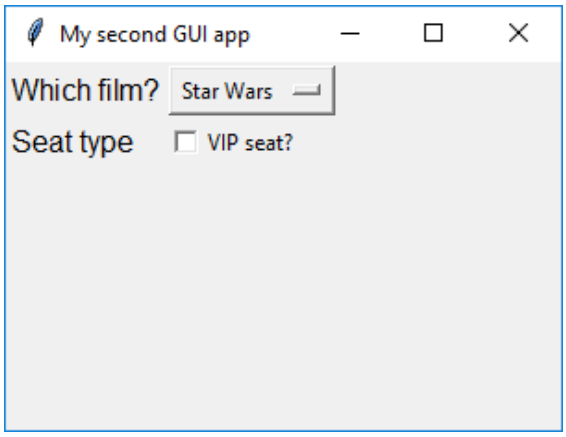
Step 14 CheckBox widget

The `CheckBox` widget allows you to select or deselect an option.

- Add `CheckBox` to your `import` statement.
- Add a `CheckBox` widget to the GUI:

```
vip_seat = CheckBox(app, text="VIP seat?", grid=[1,1], align="left")
```

- You can also add a `Text` widget in grid `[0,1]` to explain what the checkbox is for.
- Press `F5` to run your code. You should see the `CheckBox`, you should be able to tick and untick it, and if you've added a `Text` widget as a label, this should be visible next to the check box.



Step 15 ButtonGroup widget

The `ButtonGroup` widget allows you to create a group of radio buttons on your GUI so that users can choose one of a set of options.

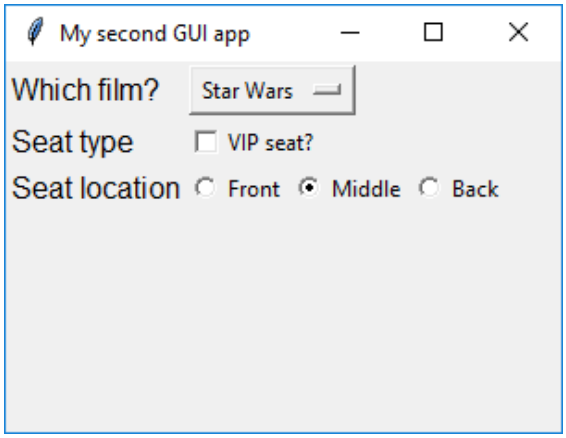
- Add `ButtonGroup` to your `import` statement.
- Add a `ButtonGroup` widget to the GUI:

```
row_choice = ButtonGroup(app, options=[ ["Front", "F"], ["Middle", "M"],["Back", "B"] ],
selected="M", horizontal=True, grid=[1,2], align="left")
```

Let's look at this code more closely:

- As before, `app` tells the `ButtonGroup` that the app is the boss.
- `options` is a list of options which will appear as buttons. Note that each option is itself a list containing the button label and a hidden value associated with that option. (You will find out later what the hidden value is useful for.)
- `selected` tells the `ButtonGroup` which button is selected to begin with.
- `horizontal=True` tells the `ButtonGroup` to display in a horizontal line.
- As with other widgets, `grid` specifies where the `ButtonGroup` will be placed on the grid, and `align="left"` positions the `ButtonGroup` on the left of the grid square.

- You can also add another `Text` widget in `[0,2]` to explain what the `ButtonGroup` is for.
- Press **F5** to run your code. You should see three buttons: the 'Middle' option should be selected when the program begins, and you should be able to switch between the options.

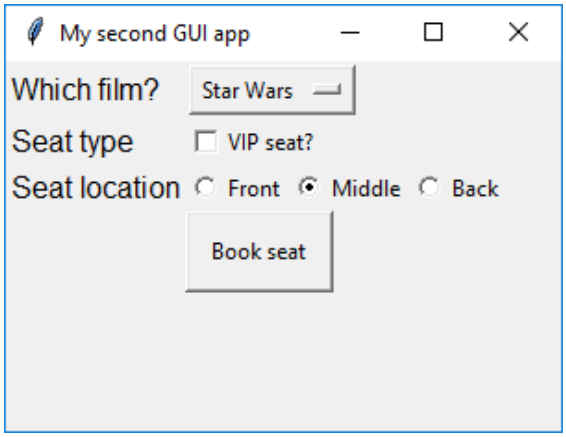


Step 16 Finishing the program

Now that users of your ticket-booking GUI can choose a film and a seat, it's time to complete the GUI script so that they can place their booking.

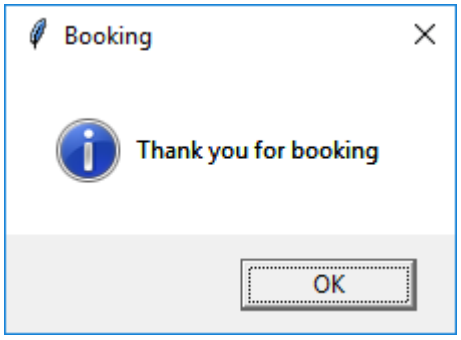
- Add `PushButton` to the `import` statement, then add a `PushButton` widget in `[1,3]` which calls a function called `do_booking` when it is pressed.

```
book_seats = PushButton(app, command=do_booking, text="Book seat", grid=[1,3], align="left")
```



- Add `info` (all lower case) to the `import` statement so that you can use the info box function of `guizero`.
- Above the GUI, write the function `do_booking()`. This will make an information box pop up.

```
def do_booking():
    info("Booking", "Thank you for booking")
```



You'll want to be able to retrieve the options the user chose, so that you can keep track of which seats in your cinema are booked.

- Add this code to your `do_booking()` function to access the values of your widgets:

```
print( film_choice.value )
print( vip_seat.value )
print( row_choice.value )
```

Note: `CheckBox` returns `0` if it is not checked and `1` if it is checked, and `ButtonGroup` returns the hidden value (`F`, `M`, or `B`) instead of the full label of the option that was chosen.

```
Star Wars
O
M
```

Step 17 What next?

Try making your own GUI program.

- Choose an old program you've written that makes use of input and output, and create a GUI for it.

There are a few other GUI widget included in the `guizero` module that we didn't talk about here.

- Have a look at the `guizero` documentation (https://lawsie.github.io/guizero/) and experiment with some of the other widgets.