Twisted Python Projects

DAN ALDRED

MANNING

Twisted Python Projects MEAP V02

1. Copyright_2023_Manning_Publications
2. welcome
3. 1_Hello_GUI
4. 2_Are_you_funnier_than_a_hyena?
5. 3_F.A.R.T._box,_a_disgusting_soundboard
6. Appendix_A._Installing_Python
7. Appendix_B._Installing_Python_on_other_operating_systems
8. Appendix_C._Installing_guizero_on_other_operating_systems_and_features_of_guizero

MEAP Edition

Manning Early Access Program

Twisted Python Projects

12 wacky, useful, tricky, cool, fun things to do

Version 2

Copyright 2023 Manning Publications

https://livebook.manning.com/#!/book/twisted-python-projects/discussion

For more information on this and other Manning titles go to

manning.com

welcome

Thank you for your purchase of the MEAP for *Twisted Python Projects*! For people who are new to Python, it's important to have fun while learning. I understand that, and that's why I've combined Python with guizero, a simple program for creating apps, to bring you some twisted and entertaining projects. From a joke machine to a fart soundboard, a number guessing game, a prank quiz, a pixel painter, and so much more, each chapter covers everything you need to know to build these fun and unusual apps. And don't worry, there's even a sensible chapter where you'll learn to build an app to check the security of passwords.

Throughout the book, I guide you through each stage of the code, explaining what it does and why we use it. Through a combination of diagrams, callouts, graphics, and code listings, you'll learn essential Python and guizero programming skills. Plus, at the end of each chapter, you'll find extra challenges to make your app even more twisted, with all the code included for you to experiment with.

With the skills you'll learn, you'll be able to create your own fun apps beyond the ones presented in the book. Thank you again for your interest and support. I'm excited to hear your feedback on the platform and look forward to helping you become a twisted Python programmer!

Please be sure to post any questions, comments, or suggestions you have in the liveBook discussion forum. I look forward responding to you on the platform.

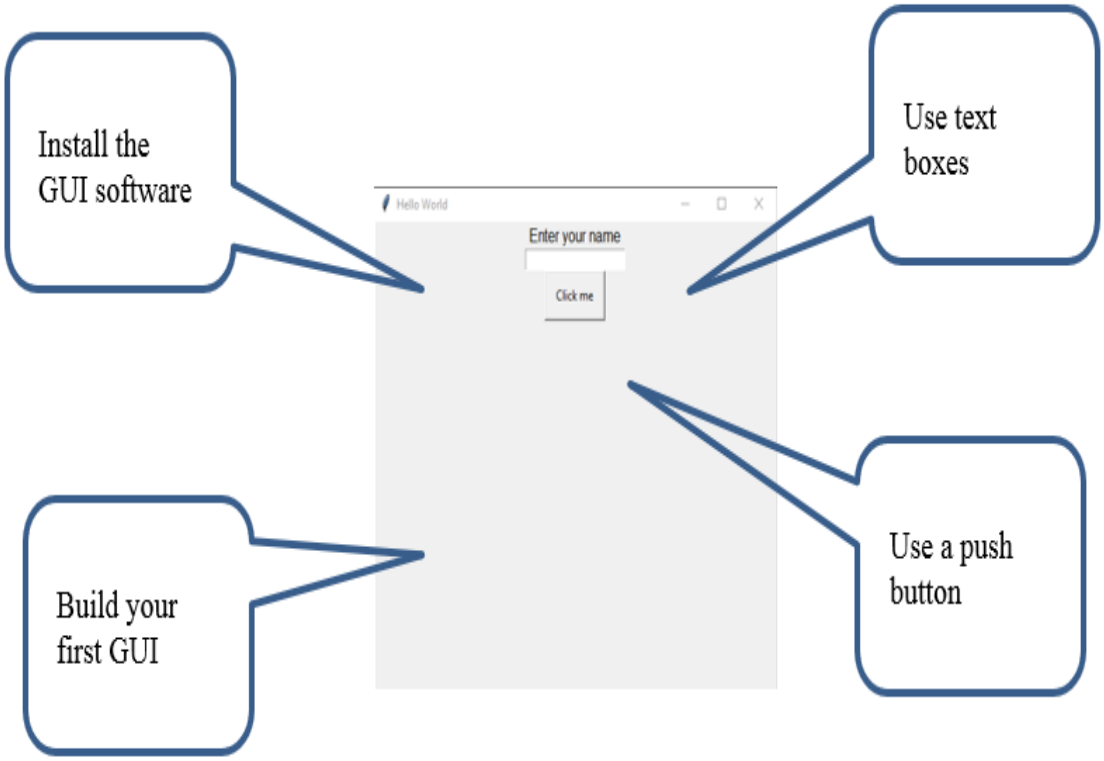
—Dan Aldred (TeCoEd)

In this book

Copyright 2023 Manning Publications welcome brief contents 1 Hello GUI 2 Are you funnier than a hyena? 3 F.A.R.T. box, a disgusting soundboard Appendix A. Installing Python Appendix B. Installing Python on other

operating systems Appendix C. Installing guizero on other operating systems and features of guizero

1 Hello GUI



This chapter covers

- Understanding what Python is
- Knowing what a GUI is
- Installing the required software
- Building your first GUI

Welcome to *Twisted Python Projects*, some outrageous projects and 1 sensible one. This book walks you through building these fun projects, such as the F.A.R.T soundbox, a personal timer, a BAE or Bust calculator, even a pixel painter. Each chapter presents a new and exciting project that is fun in its own right and teaches you programming skills in Python and guizero. Python is a popular and versatile coding language, and guizero is an easy way to create interactive graphical interfaces, GUIs. You can find GUIs

everywhere, for example, the menus and buttons that enable you to interact with your smartphone or tablet. The apps and games that you can download and play, the Operating System on your computer. GUIs are everywhere. The software we will use to create our GUIs is called guizero, more on this later on.

Each chapter covers building a new GUI App with different features. When you're done with each project you can combine your new programming skills and techniques from each chapter to create your own or new versions or the GUIs.

This first chapter is a comfortable introduction to creating your first GUI using the Python programming language. You will be walked through every step of building a simple project, from installing the software, creating a file, writing the program code, testing your code, and then making improvements. This first chapter contains all the guidance that you need and will walk you through all the steps and concepts in detail.

Most of this book's projects use the Microsoft Windows platform; however, Python and guizero are compatible with Linux and Apple computers . Appendix A covers installing Python on a Windows computer. Appendix B covers how to install the required software on a Linux or Apple computer.

Just what is Python?

Python is one of the fastest-growing programming languages. You can use Python to create a wide variety of programs, such as websites and games, or for automating tasks, data analysis and so much more.

Tell me more about Python

Python was created by Guido van Rossum as a side project and first appeared in February 1991, which is a long time ago! When asked why his project was called Python, Guido said that at the time he was reading scripts from a 1970s BBC TV comedy program called Monty Python's Flying Circus. He needed a short and memorable name for his new programming

language and chose to call it Python. Although Python has a funny name, it is used professionally and used as a main language to teach coding in many schools and colleges across the world.

So why is Python so popular? One reason is because Python can be used to write programs for so many different purposes. However, its real strength lies in the simplicity of using the code. For example, consider the beginners “hello world” program, which in the coding world is a well-known first program to create when learning a new language. The program prints out the words “hello world.”

In some programming languages a “hello world” program would look like this:

```
void main() {
  print('Hello, World!');
}
```

In Python it is simply,

```
print ("Hello World")
```

The advantage of Python code being so simple to write is that you are less likely to make errors. Some programming languages require special punctuation and spacing. Python still does but, it is less complex. However, do not assume Python is a basic language. You can still create really interesting and engaging GUIs as you will see throughout the chapters; it is just that the Python code is easier to write than other programming languages.

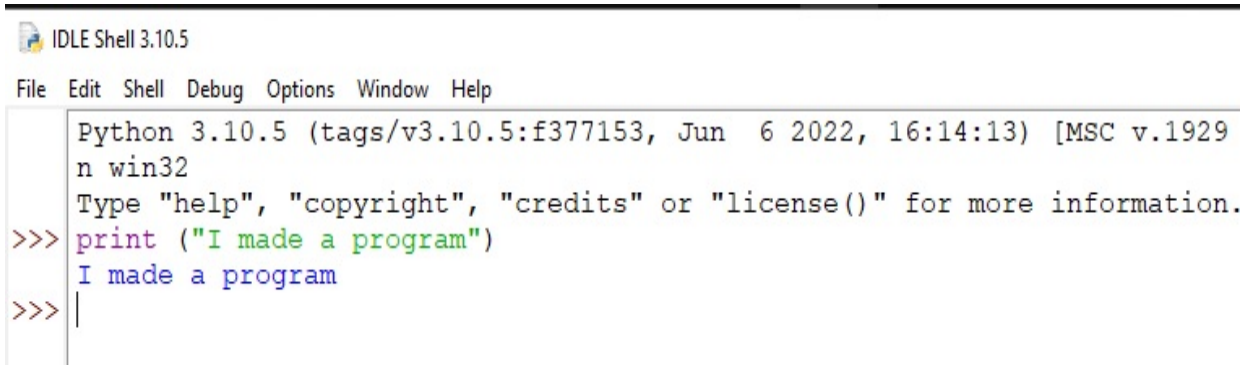
If you already have Python installed, then you can move on to section 1.2. Remember to check out appendix E, which covers commonly used Python features and techniques. If you need to first install Python, then go to Appendix A before reading 1. 2.

Once Python is installed, you can test that Python is working correctly by,

1. Opening the Python IDLE program.

2. In the window that opens (known as the Shell Window) type out the line of code, `print ("I made a program")` (see figure 1.1).
3. Press the **Enter** key on the keyboard and the program will print out the message, *I made a program*.

Figure 1.1 Testing that Python works



What is a GUI and what is guizero?

GUI stands for Graphical User Interface. It's pronounced "gooey," like the inside of a Cadburys' Cream Egg. You are probably already familiar with several computing devices. Such as laptops, game consoles and your smartphone. All these devices work using a text-based language that we call a programming language or code. Using text to create code is challenging. It is easy to make mistakes or omit a required part of the code even if you are concentrating really hard. This results in glitches—that is, bugs or the program failing.

An easier way to interact with your computer or device is to use graphics and images to create a visual way to interact with your computer; in short, a GUI. (figure 1.2). The Home Screen on your smartphone is GUI, if you want to make a new folder using a GUI, you press the folder icon. If you want to take a photo, you press the camera icon. This is a simpler method of interaction and easier to remember rather than having to write the code to create the folder. Icons are easier to remember and use, and this is why GUIs are found everywhere, from Web Browsers, ATMs, self-service checkout tills, GPS satnav, and so on.

Figure 1.2 A modern phone GUI

So, you now know what a GUI is, but what is guizero? Guizero was created by Laura Sach and Martin O'Hanlon and they describe it as, "a Python 3 library for creating simple GUIs. It is designed to allow new learners to quickly and easily create GUIs for their programs." (https://lawsie.github.io/guizero/about/) A library is a group of smaller programs called functions and modules that have been written and packaged together. (Functions are covered in more detail in section 1.3.3)

There are other versions of GUI programs that work with Python, and you can read more about these in the appendix D.

What is a function?

A function is a reusable block of code that can be called and used anywhere within a program. Functions save time and make the program more efficient.

This means that you do not have to write the code from scratch; libraries save you a lot of time and reduce your effort in writing programs, For example, a library function can play a sound or display an image. At the time of writing this book there are over 137,000 Python libraries. Let's get started and install guizero.

Installing guizero

There are several methods to install guizero and it has even been designed so that you can download the files and make GUIs without installation. This is useful if you are not permitted to download and install the software on the computer. It means that you can just get started with the projects with no need to install anything. Check out Appendix C which covers the guizero easy install method. This section of the chapter covers how to install guizero on a Windows computer. It assumes that you have already installed Python, if you have not, then check out appendix A and follow the steps, then come back here. If you are using an Apple, Linux or other computer, then head over to Appendix B and follow the steps to install guizero on your computer.

To install guizero, we are going to use the command prompt and write a short instruction to pull down the required program files. Before you start the installation, ensure that you are connected to the Internet as you are going to download the installation files.

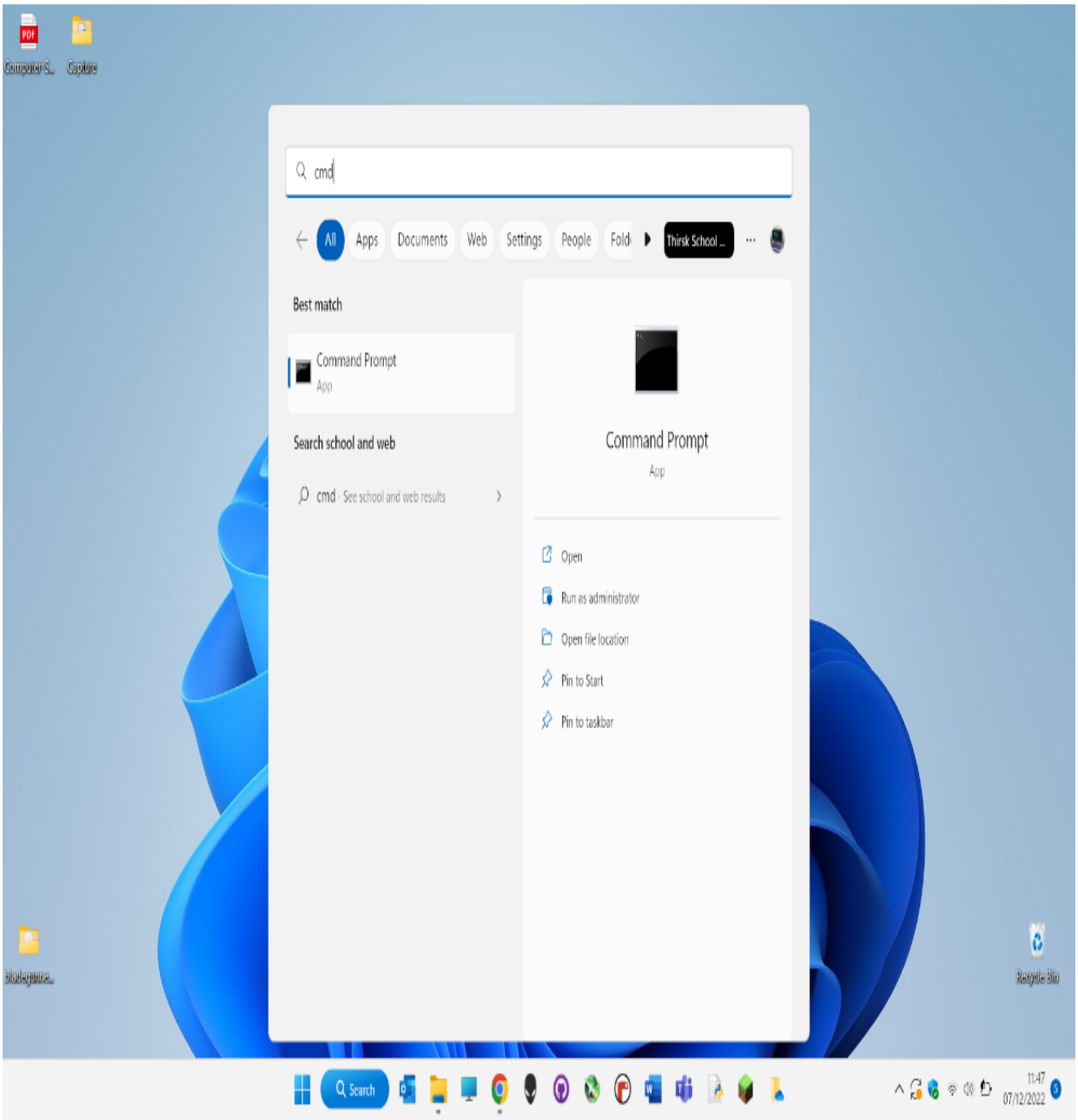
Command Prompt

Command Prompt is a command line interpreter application available in most Windows operating systems. It's used to execute entered commands. Most of those commands automate tasks via scripts and batch files, perform advanced administrative functions, and troubleshoot or solve certain kinds of Windows issues.

Command Prompt is officially called Windows Command Processor, but it's also sometimes referred to as the command shell or cmd prompt, or even by its filename, cmd.exe.

To open the command prompt,

1. Select and open the Windows start menu.
2. Type the letters cmd.
3. Let Windows search for the command prompt app
4. Select the app and open it.



The command prompt app will load, and you will be presented with the following window.

5. Click the window to select the prompt and type in the following command, (a command can be thought of as an instruction, you are writing an instruction for your computer to perform)

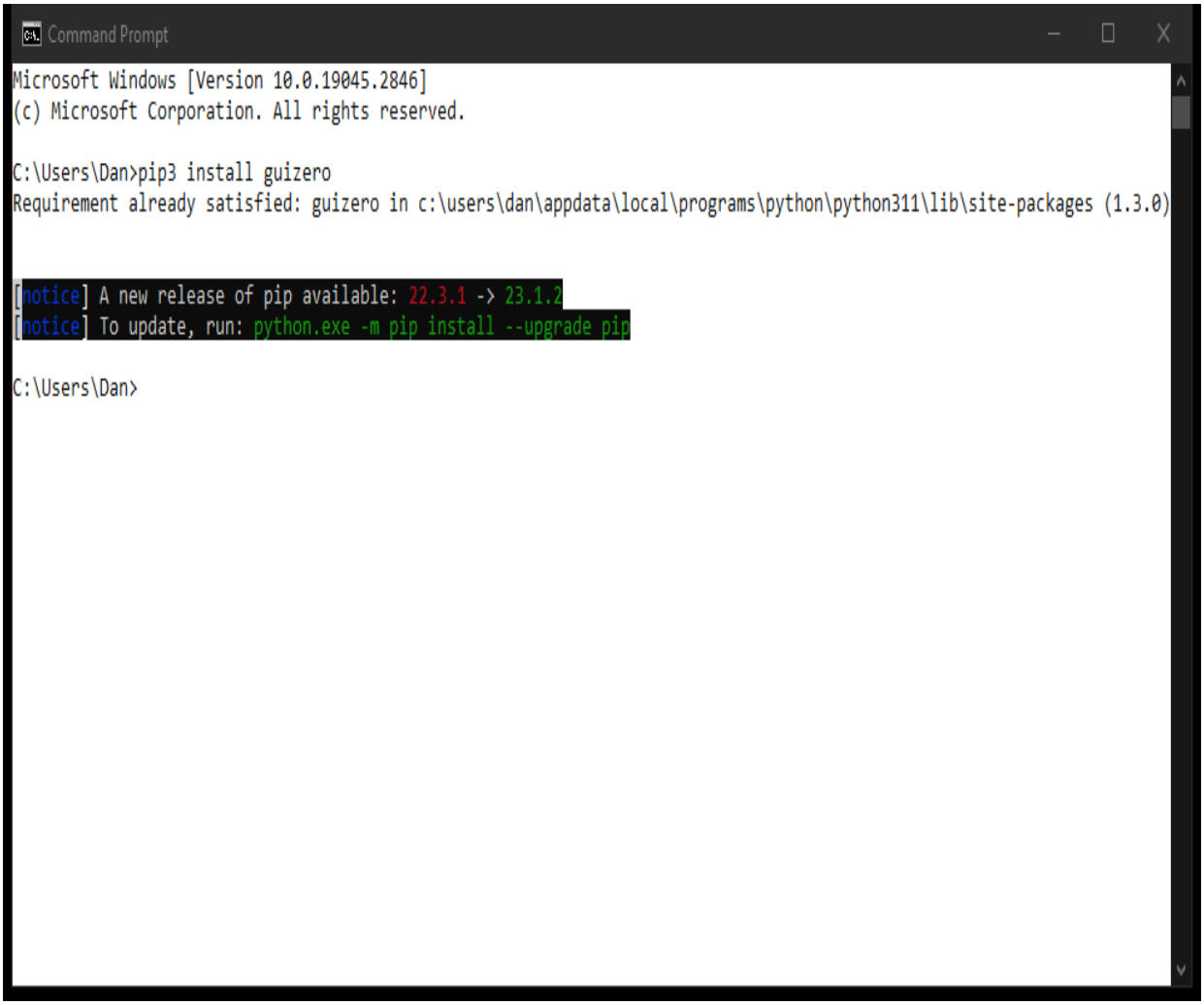
```
pip3 install guizero
```

6. This instruction tells your computer to use a program called pip3 to install the guizero program. Press the **Enter** key on your keyboard to begin the installation.

Pip3

Pip3 is a program sometimes called a package manager that manages the installation of Python libraries, modules and packages.

7. Guizero will now install, taking a little time. Once complete, close the command prompt.



Remember that this installation method only works if you have already installed Python using the method in appendix A. You can also check out the instructions on the main guizero webpage is https://lawsie.github.io/guizero/ and this sets out the installation instructions.

Guizero elements and widgets

For ease of understanding guizero, you can split guizero into two categories. Category one is the main elements of the software, these elements enable you to build a GUI. For example, Layout is an element. You can use the Layout to adjust the size of the GUI and its buttons and pictures. You can customize your GUI's layout as you wish. The Event elements are used to cause or trigger a reaction (called triggering an action) when something

happens; for example, when a person double-clicks the left mouse button on a picture, then some text is displayed.

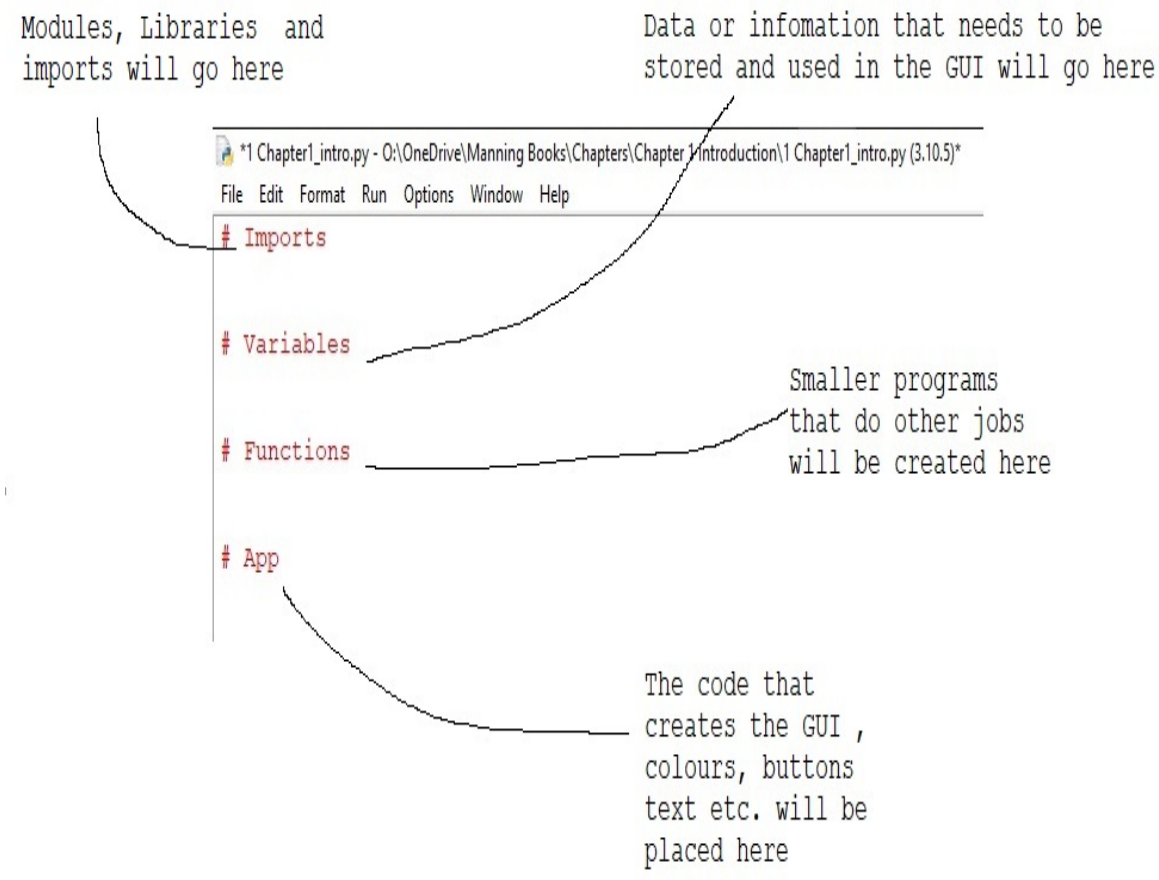
The second category is widgets, these are the main components of the GUI and there are fifteen widgets that you can use in your GUI. For example, the PushButton widget creates a button with a text label that can be clicked. The ButtonGroup widget creates an option button where you can select from several choices. You will use the widgets throughout the various projects, and you can refer to appendix D for more details.

Writing a program with Python and guizero

To code a program, you need to use an Integrated Development Environment (IDE). This is an interface that enables you to write programs, test them and run them on your computer. The Python installation includes its own IDE, called IDLE, which is pronounced, ‘idol.’ There are other IDEs available; an overview of the alternatives is covered in appendix A.

When writing each program, the code will always have four main sections (figure 1.3). These sections break up the lines of code into organized and manageable chunks. This organization will make it easier to check your program for errors and will also make it easier when you wish to make edits or changes. When you organize your code, good organization makes it easier to find the code that you need.

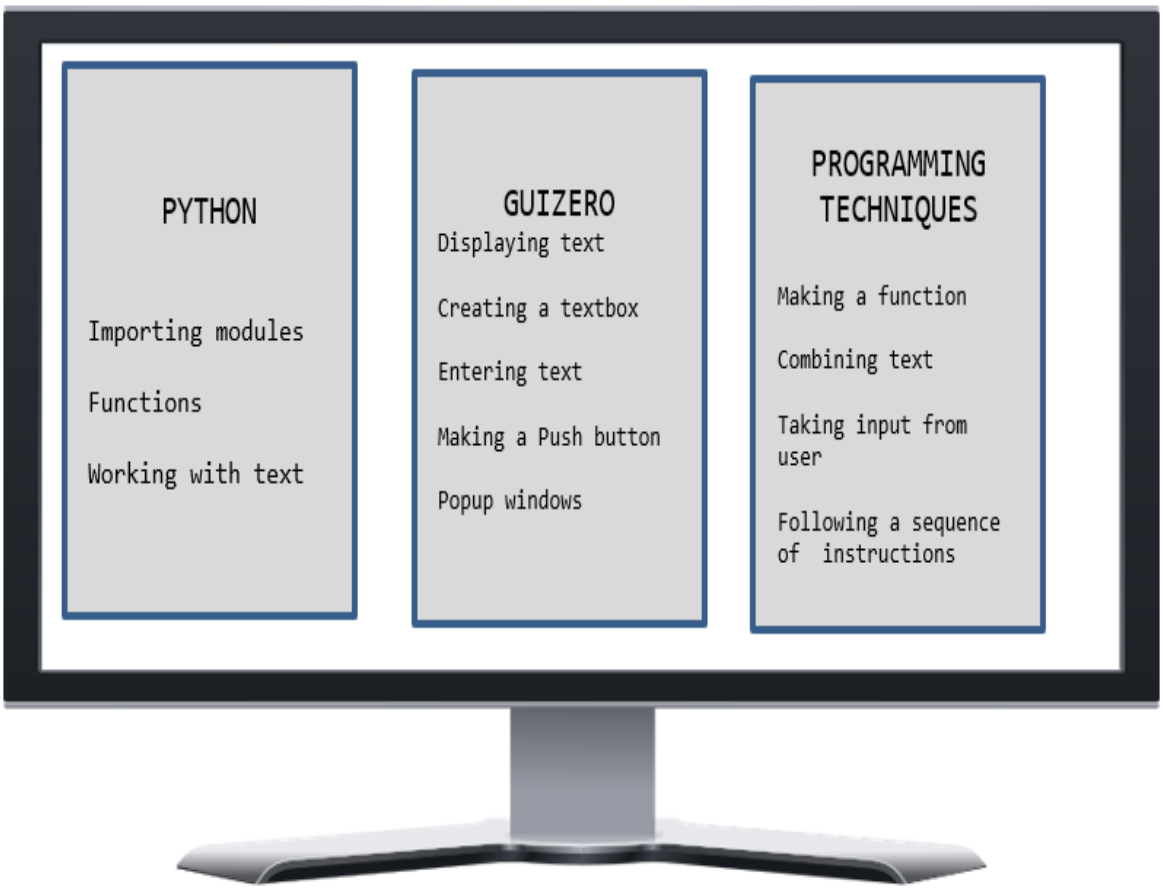
Figure 1.3 How we will structure the GUI program.



Hello GUI: Coding your first GUI

Now for the fun to begin! This section of the chapter walks you through creating your first GUI program. Or you can skip ahead to another project that has caught your eye; I know that you really want to build the F.A.R.T soundboard. You can always return to this chapter later if you need to.

Figure 1.4 Project uses the following programming skills, are we still including this in the chapters?



Earlier in this chapter I referred to the Hello World program, which is considered the beginner's program in any computer programming language. Traditionally, the program simply prints out the words hello world. It takes no inputs and has no outputs.

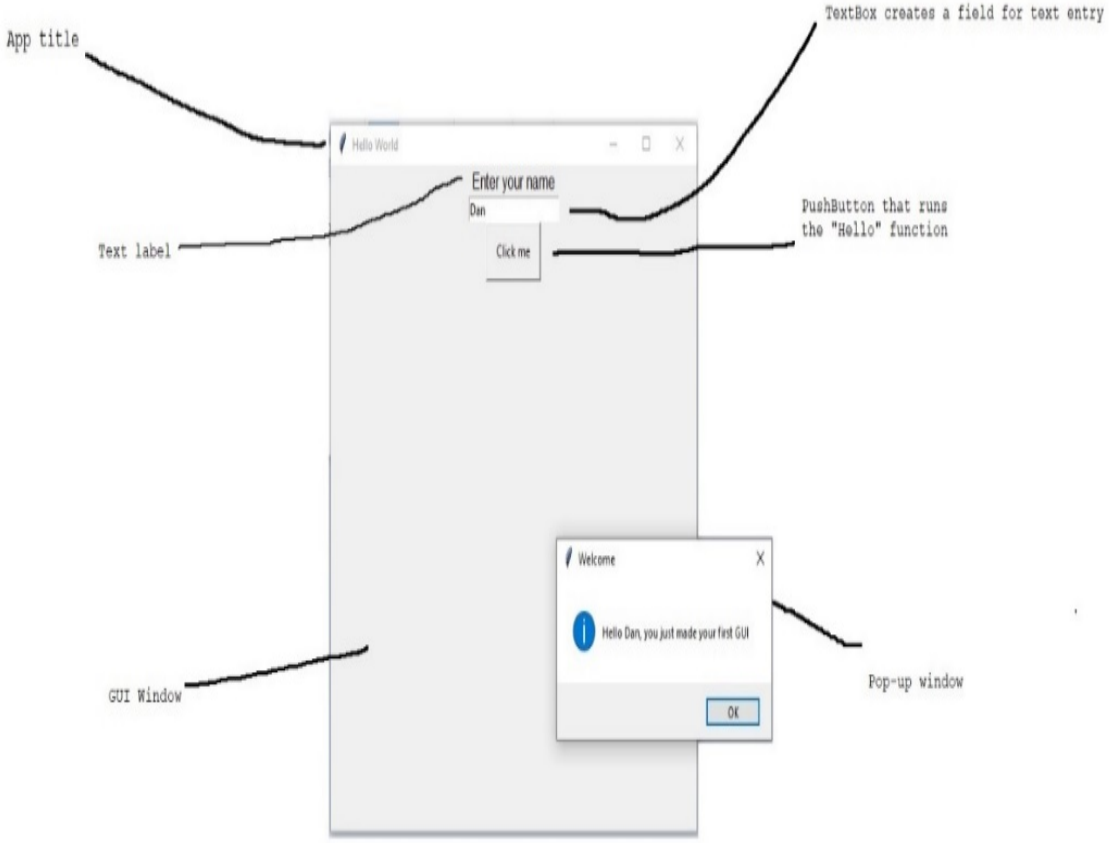
Inputs and Outputs

An input is defined as one of two things. Firstly, an input can be a device that is used to control the computer; for example, a mouse, a keyboard, and even a gamepad is an input you can use to control your computer. The second definition of an input is data or information that is put into program. One example is when you log onto your computer, and it requires you to enter your username and your password. You input your username and your password into the software. Output refers to a device that is used to display

data or information. The most common output is a screen or monitor which displays what a program is creating that is supposed to be visible; that's called a program's output.

In this version of the hello world program, we are going to add and input in the form of a text box that will enable you to enter your name or any name that you want to. The GUI also features a Button that, when pressed, tells the program to display a pop-up window that will say "hello" followed by your name, as shown in figure 1.5.

Figure 1.5 Your first GUI

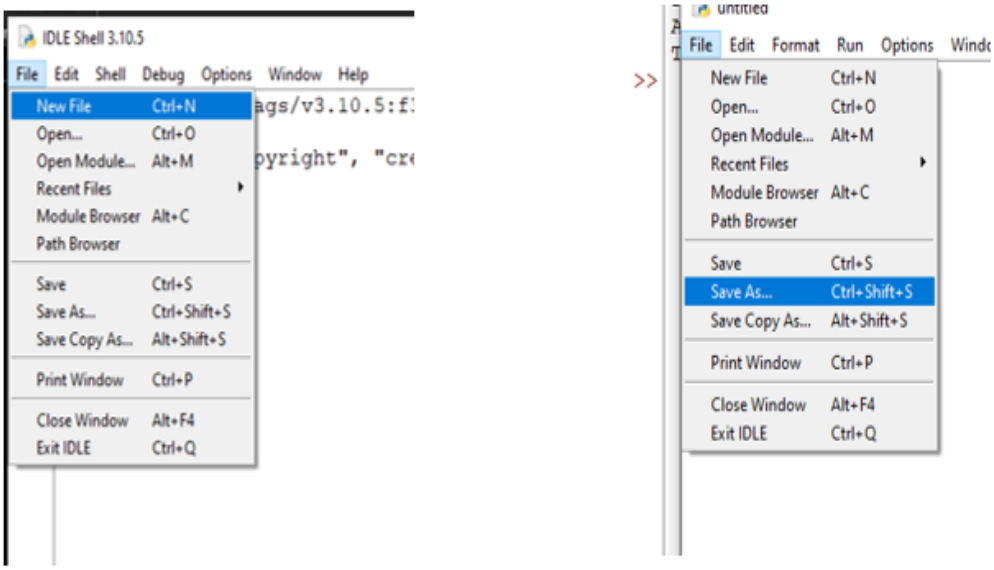


To begin writing your program, first load Python.

1. Assuming that you followed the appendix A installation of Python, then go to the Taskbar and select the Python icon, or open the windows menu and search for Python, typing in the word, ‘Python’. Click to load Python.
2. Create a new program by choosing the **File** > **New File** option. You will now be automatically prompted to save your program; do so. Saving your files right away is good practice, just in case the computer crashes and you lose your file and have to start all over again.

3. If you have installed guizero using the installation instructions in this chapter, then you can simply save the Python file. (However, if you have downloaded the guizero files for the quick start appendix B, then you must save the file **HelloGUI.py** into the same folder as you download, see figure 1.6.)
4. Save the file by choosing the option **File** > **Save As** from the menu.
5. The Python editor will prompt you to save the file.
6. Name the program **HelloGUI.py** and choose **Save**.

Figure 1.6 Save your Python file



Importing the modules

Each GUI program that you write first requires you to import the required modules. You can think of modules as smaller programs that allow you to add and use different features in the program. Both guizero and Python contain modules, so you will be using modules from both. For example, in this project we will be adding a pressable button to your GUI. (Figure 1.5). To do this, you import the `PushButton` module, which contains all the lines of program code required to instruct Python how to create a button. You do

this at the start of your program by simply using the code, `from guizero import PushButton`

Guizero modules

Guizero stores the modules in a folder called **guizero**. If you are interested, you can open this web link https://github.com/lawsie/guizero/tree/master/guizero and see the modules and code for yourself. You will notice that the name of the file is the same name of the module that you use when importing them. For example, the PushButton module is named PushButton.py.

The module enables you to create the widgets (appendix D) that we can then use in the program. Modules save us time and complexity. For example, the PushButton module contains prewritten code to program features of the buttons such as, the size of the button, the color of the button, what the button does when pressed, the amount of padding around the button and more.

Lucky for us we can simply use the `PushButton()`function which is a single word. Functions are so useful and efficient. You can find a list of all the Elements and Widgets and what they do in appendix D.

In your Python program file, type the code shown in listing 1.1.

Listing 1.1 Import the modules

```
# Imports

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

The program begins with the hashtag symbol `#` used on `# Imports` is a comment. Comments are useful for telling the reader what your program code does. It is useful for organizing and managing the sections of your program.

Comments

A comment is a reminder or notice that tells you something about the code. A little bit like a keep of the grass sign, it tells you what to do and which grass to keep off! Comments begin with the # symbol and turn anything in front of the # to red, making it a comment. They do not run; they just help you to organize and understand what the code is doing.

The next line of the program begins with the code,

```
from guizero import
```

which tells your Python program to go to the guizero library and then import the modules listed in the same line. The modules used in the HelloWorld GUI program and each of their purposes are,

- `App`, used to create the main window that holds all the widgets
- `Info`, this displays a pop-up message window
- `PushButton`, this creates a button that can be clicked
- `TextBox`, is used to enable the user to enter text which can be used by the program
- `Text`, displays the title of the GUI

Creating the variables

The next stage in writing your program is to create the variables. A variable is a location in the computer's memory that stores data or information. The data is only stored whilst the program is using it. When you close the program, the data is forgotten. The content or value stored in the variable can change; that is, it can vary.

Programs need to be able to find data quickly, so each variable you create must be given a unique label. Think about how food in your house may be in a jar or a box which has a label (figure 1.7). The label helps you to identify what food is stored in the jar.

Figure 1.7 Labels on Jars are like variables; they tell you what is stored in each Jar

In a similar way, labeling a variable means the program can find the memory location quickly and go straight there and collect the data.

You can label a variable anything that you want; however, it's good practice to use a name that represents what data is being stored in the variable. For example, if the variable was storing the name of a person, then you might label it `persons_name`. When creating the label for the variable, it has no spaces between the words. Should you use spaces, then the variable won't work, and you'll get an error message,

Constants

The opposite of a variable is a constant. Guess what, constants stay the same and do *not* change. For example, Christmas Day is always on 25 December. That makes it a constant. The present or gift that you get each year, is a variable as it changes each year, unless you are over 50 years old, and then you always get given socks!

In this program we are using some variables to create the widgets, the text, the textbox, and the button. The code for these widgets is written in the #App section of the program, so it makes sense to store these variables in the #App section of the program. Although it will remain empty for this program, add the `#Variables` comment to the program, as we will be using variables in the chapter 3 and it is good practice.

Creating a function to display the pop-up message

The next section of the program requires you to create a function named Hello, that you will assign to a button in the GUI. When the button is pressed, it will call the function named Hello, then run the code contained in the function, to combine your name with a short message and display the name and message in a pop-up window. This action happens every time that you press the button.

A function is a reusable block of code that can be called and used at any point in your program. Imagine you are playing your favorite computer game and you are down to your last life; you can die or lose a life at any point in the game. So potentially, there are millions of points during the game where it's game over and the game would end. If the game programmers didn't use a function, they would have to program the "game over" code millions of times, once for every scenario where you lose your last life. This would be very tedious and make the program's code massive. Instead, developers use a flag, which is triggered when the game is over. The flag informs the program to run the game over function.

Using a function means the developers can write the "game over" code once and then when the game ends, the program calls the function, and it runs. You only need to write the block of code once, then you can use it as many times as you like, which saves processing power and makes the program more efficient.

Listing 1.2 Function code

```
# Functions

def Hello():
    info("Welcome", "Hello " + textbox.value + ", you just made
your first GUI")
```

The line of code in listing 1.3 looks long and has a lot of elements, so let's break it down. To create a function in Python, you use the keyword `def`, which indicates to the Python program that it should expect a function to follow. Then you give the function a name and type it in. In our program, the function is called `Hello()`. The name `Hello` is used to call the function,

calling means that the program runs the function when a particular action is performed by the user. In this program the function is called when a button is pressed.

Remember that when labeling a variable, you should use a suitable name that helps you identify what the variable holds? Well, the same applies here; the function's name should be related to the function's purpose; that is, the name should indicate what the function does.

Using suitable function names will help the other programmers that look at your program code understand how the program works. It also makes error checking easier. For example, if the error is in the function, Python will inform you of the error and you can easily find the relevant area of your code. You can find the function in your code, without having to check every line for the error. Python will inform you where it found the error; it will display a message something along the lines of

```
SyntaxError: invalid syntax. Perhaps you forgot a comma in the
function Hello():
```

A syntax error indicates that the way the code has been written does not follow the required Python structure rules. In some ways, syntax is like grammar. For example, Thor from the Avengers owns a hammer, which only he and several other characters have ever been worthy enough to lift. (Do you know who they are?) The correct grammar or syntax for referring to that hammer is "Thor's Hammer." That is, you use an apostrophe to indicate that the hammer belongs to Thor. The incorrect syntax is Thors Hammer, which implies that there is more than one Thor. You can read more about types of errors in Python code in appendix E.

A function ends with

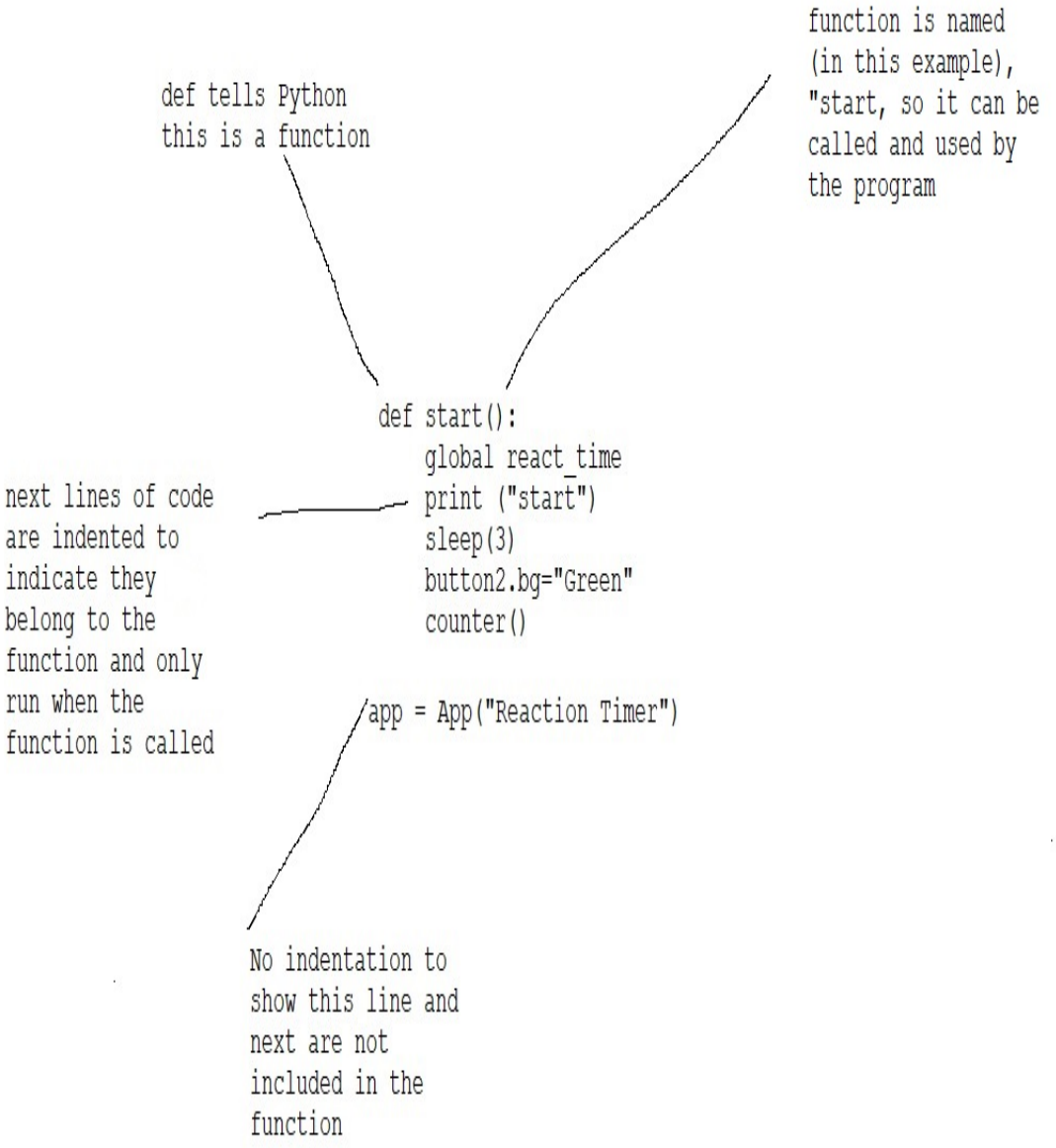
```
():
```

which is the syntax that Python uses to indicate that anything in the next section of the code belongs to the function. You may have noticed that the line of code under the function name is slightly indented to the right. To get the amount of indentation correct you can press the TAB key once or the

spacebar four times. Ensure that you always use the same amount of indentation throughout your program.

By indenting the lines of code (see figure 1.8), you tell the program which lines to include within the function, and to only include these lines. So how does the function know what the last line of code in the function is?

Figure 1.8 Parts of the Function

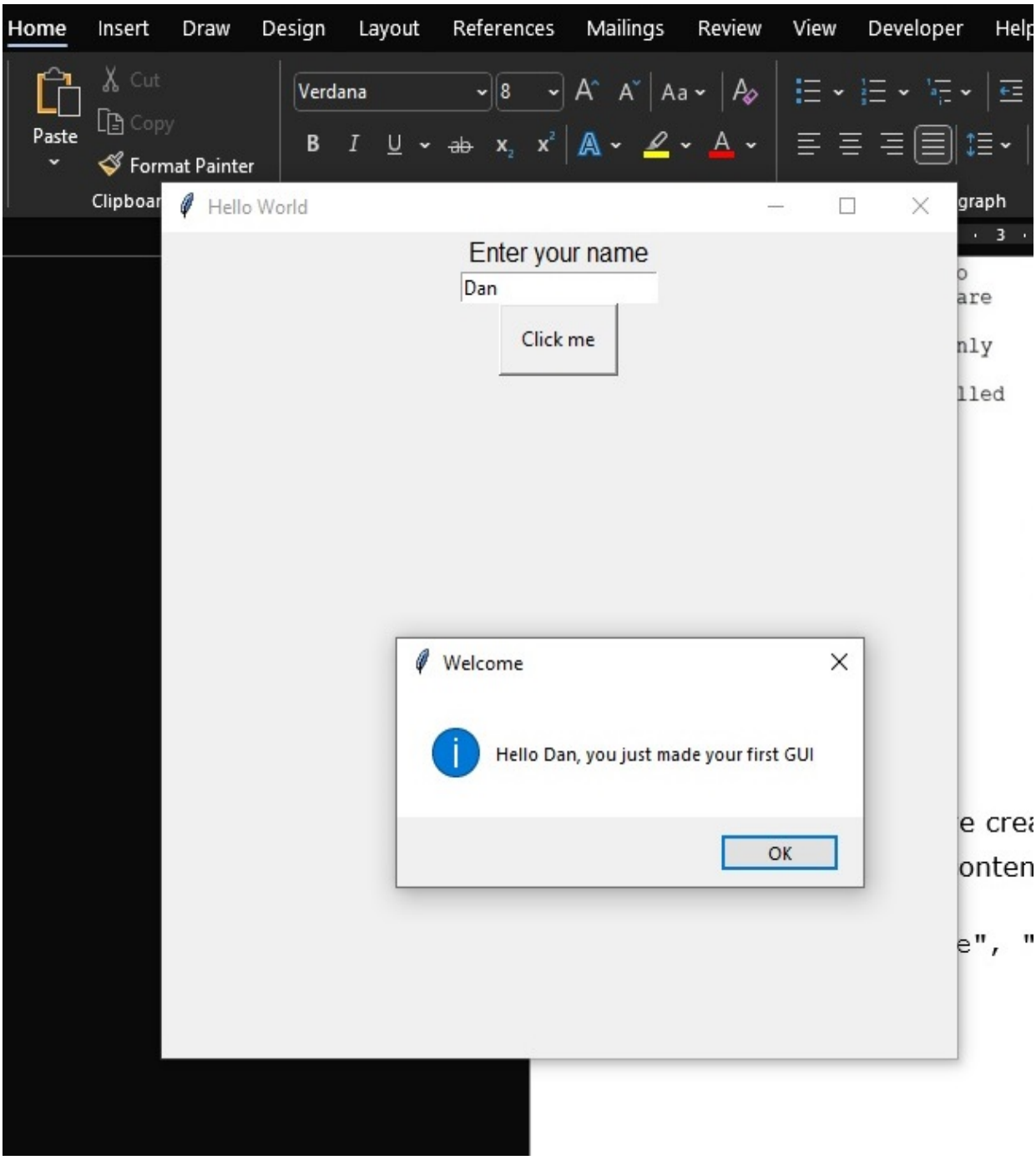


The function knows when the function ends when the next fully **left-aligned** line of code appears. Now that you have created the function and you know what it does and why we are using it, you can add the function's contents. The first part

```
info()
```

tells Python to create and display an information message box (figure 1.9).

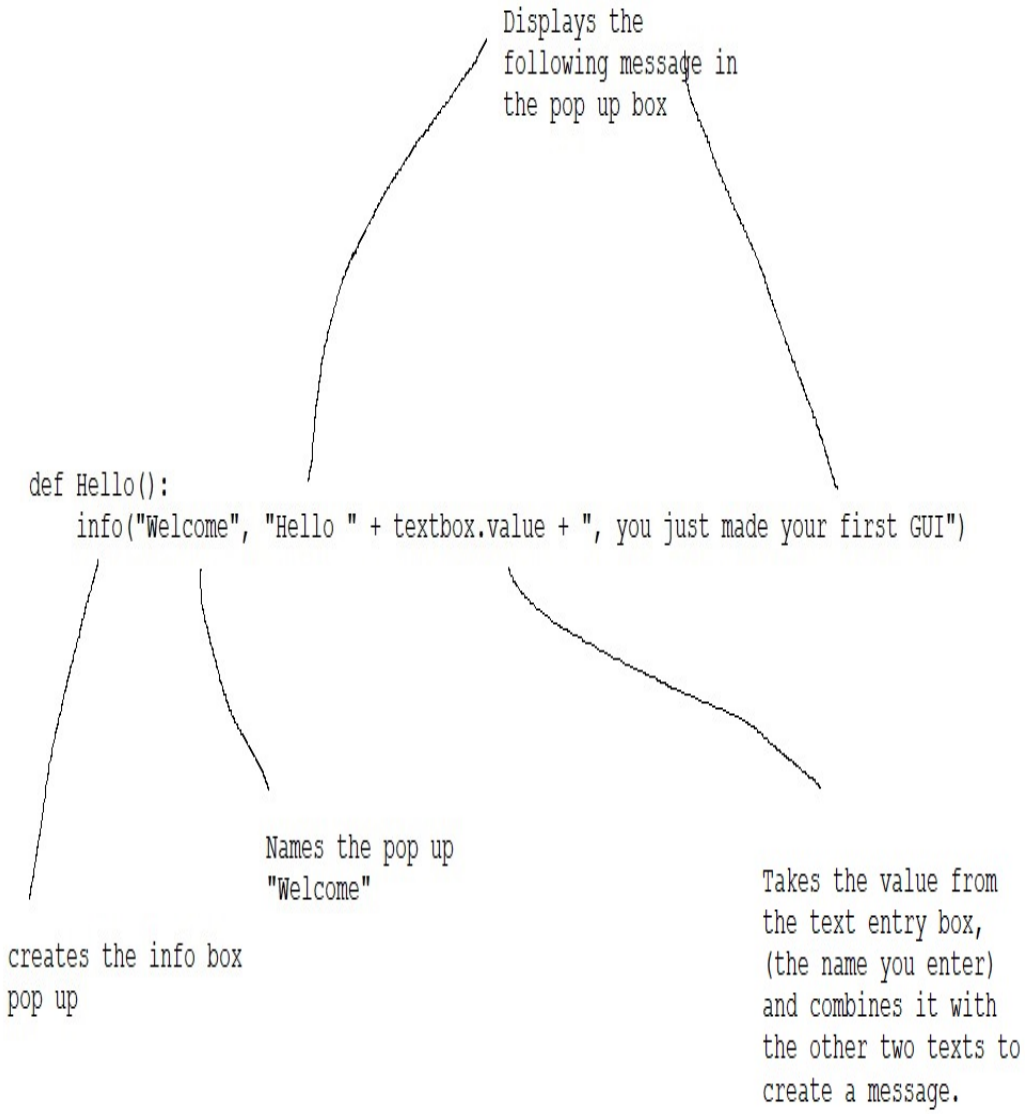
Figure 1.9 An information pop-up



The information box pops up when you press the “click me” button (which we will code in the next section) and combines the name that you enter into

the text box and the string, “Hello” to create a final message (figure 1.10).

Figure 1.10 The parts of a Function



A string is a series of characters. A character can be a letter, number, or symbol. For example, “this sentence is a string”, “Hello World” is also a string. In Python, strings start and end with the " symbol, like this: `"This is a string."` Strings are used to hold content that is going to be displayed or printed out in the program. For example, consider the web banner on Amazon’s homepage. All the options headings will be saved as strings (figure 1.11). Most commonly, strings are printed or saved in a variable, I’ll talk more about this in appendix E.

Figure 1.11 Strings used on a web banner

Digits and numbers can also be a string, for example, “C3P0” is a string. Now you might say that this is the name of the droid that you are looking for, therefore it uses letters and numbers, similar to a car registration plate or your zip / post code. A car license plate or a postal code are both created with letters and numbers. However, a list of digits or a number can be a string; for example, this number “221” is a string because the numbers are enclosed within quotes. However, the number has no value. It is not two hundred and twenty-one, it is simply the symbols 2, 2, and 1.

If you look at again at code listing 1.3, you will see the strings being added together,

```
"Hello " + textbox.value + ", you just made your first GUI")
```

We know that it is impossible to add text in the mathematical sense, so this line of code uses the + symbol to combine the two strings and the name that

you input, into a longer string, a full sentence that will be displayed.

Creating the GUI window and outputting the message

The last part of the program is to add the lines of code that will build (create) the GUI and the interaction between the parts: the textbox, pushbutton, and the pop-up window. You may remember we introduced variables and noted that the program uses several variables as discussed in section 1.2.2. As a reminder, a variable is a location in the computer's memory that stores data. In this Hello World program, there are four variables.

- The first variable labelled, `app`, creates the main window of your GUI.
- The second, `text`, holds the instruction that tells you what to do (the instruction tells you to enter your name).
- The third variable, `textbox`, stores the data to create the text box and also stores the name that you enter into the test box.
- The final variable, `button`, holds the information about the button.

Add the code in listing 1.3 to your program.

Listing 1.3 Creating the GUI main window

```
# App

app = App("Hello World")
text = Text(app, text="Enter your name")
textbox = TextBox(app, width=20)
button = PushButton(app, command=Hello, text="Click me", )
app.display()
```

The first line of the code creates the GUI window and positions the name "Hello World" at the top left of the window. You can name the GUI window differently if you wish; just ensure that you enclose the name in quotes because, yes, you have guessed it, this is a string. The `App` is the main window which contains all the other widgets. You will notice in line two that the word `app` also appears:

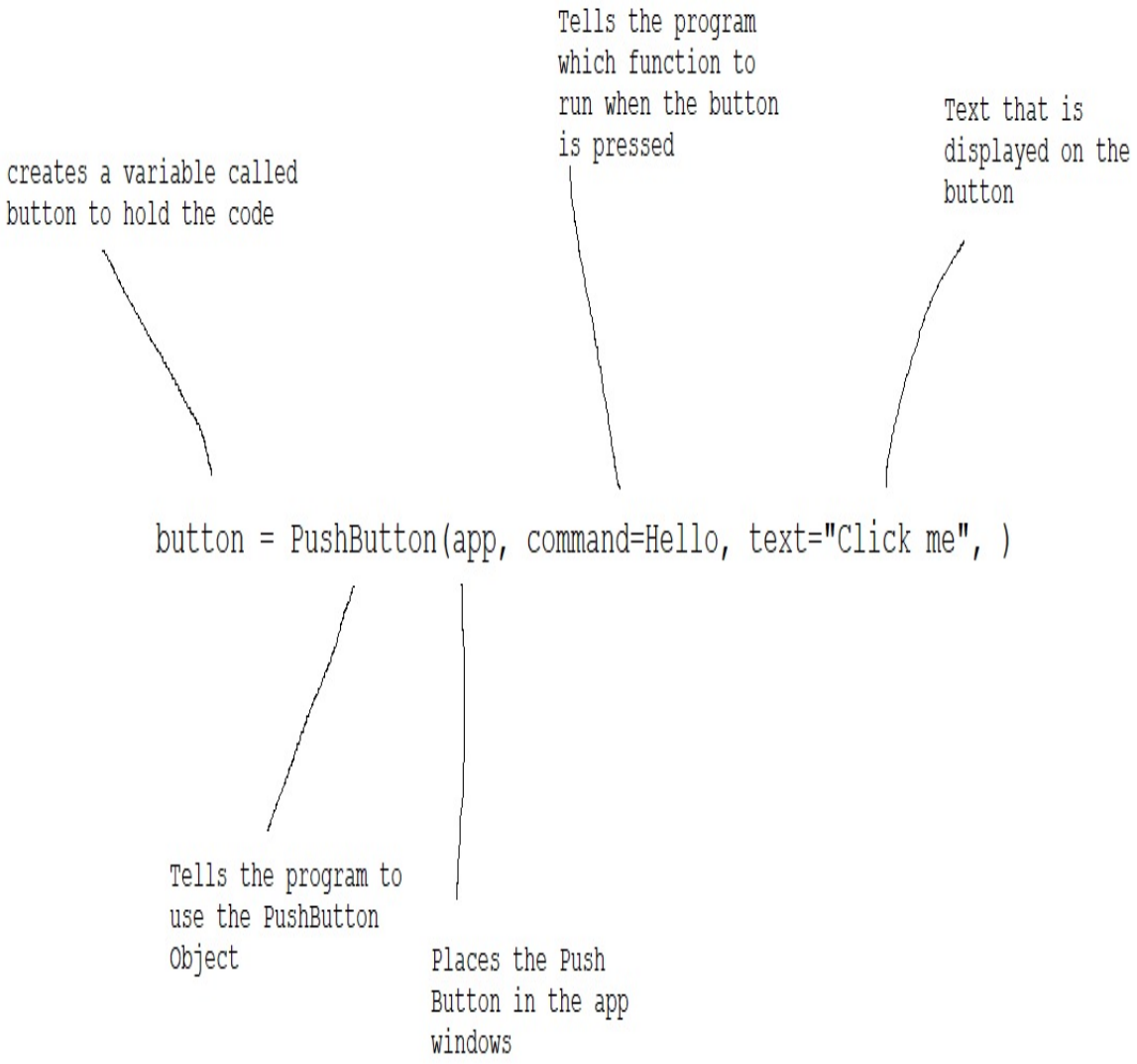
```
Text(app, text="Enter your name")
```

This is because we use the `app` object to edit and add content to the main GUI window. (There is also a `Box` object that works in a similar way to the `App`; you can add and edit content. A box is usually held within an App. You will come across boxes in chapter 3 when you make a FART soundboard!) An object can be thought of as a way to define the properties and features, so the app object enables you to set the properties, say the size and color of the app window.

The second line creates and displays a line of text with the instruction to "Enter your name". Notice that it uses `app` after the `Text()`. This tells the program where to display the text; in this case, to display the text in the App window. The third line is responsible for creating the text box where you enter your name.

Next, on line 4, add the code to create the button. You need to tell guizero which function you want the button call, (which function to run) when it is clicked (figure 1.12). Notice here that, confusingly, the function is called a command. When you create a function like you did in section 1.2.3, it is called a function. When you write code in guizero to call or run this function then you use the code, `command =`. Finally, you need to tell the program to display all the elements that you have just coded. To do so, simply add the line `app.display()`to the end of the program.

Figure 1.12 The PushButton code



Running and testing your first GUI program

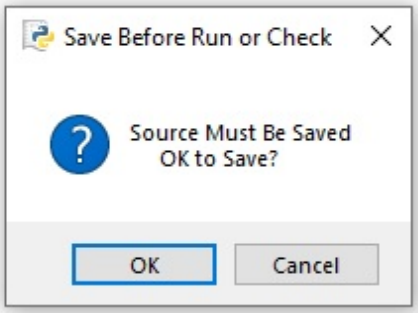
Congratulations, you have built your first GUI. Well done! Now to test that it works correctly. Do not worry if the program fails the first time. If you are new to programming or even experienced, then it is likely that you made an

error. Such errors are usually syntax errors, as explained under listing 1.3. A syntax error is like a grammar error; Python expects you to use certain syntax for the program to work correctly.

So, let's try to run the program. Press **F5** on the keyboard. (If your keyboard does not have function keys then, at the top of the IDLE window click, **Run** and select **Run Module** from the

drop down menu. You may be prompted to save the program file (figure 1.13). Choose **OK** and the program will save, and then the program will run. Fingers crossed!

Figure 1.13 The Python save window



If your program fails or does not run as expected check for these common errors:

1. Have you installed guizero correctly?
2. Did you save the program into the GUI folder?
3. Do all the comment headings start with a # and appear in red font, `# Imports, # Variables, # Functions, # Apps`?
4. Are all the required modules in the import section?
5. Have you used the correct case? Some syntax calls for capital letters in the middle of a word, for example, `TextBox`, `PushButton`.

6. Have you used "" for each string?
7. Did you spell all the words correctly?
8. Check out the final code listing and compare it with your program code.

Listing 1.4 Final code listing

```
# Imports

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

# Variables


# Functions

def Hello():
    info("Welcome", "Hello " + textbox.value + ", you just made
your first GUI")

# App

app = App("Hello World")
text = Text(app, text="Enter your name")
textbox = TextBox(app, width=20)
button = PushButton(app, command=Hello, text="Click me", )
app.display()
```

Three other things to try

This section of the chapter introduces three extra things or edits for you to add to your program. These ‘extra things’ can be used to personalize your GUI, and can make your GUI look and feel quite different. You will come across different changes throughout the chapters and you can try them out with any of your projects.

Changing the background color

The first change to make to the program is to change the color of the background of the GUI window. Color can add a different feel to your

projects. The line of code that adds color is,

```
bg = "red"
```

where `bg` is short for background. The line of code is saying “make the background color equal to red.” In the App section of the program, locate the first line where you named the GUI, “Hello World.” On the next line, add the `bg = “red”` line of code.

```
app = App("Hello World", bg ="red")
text = Text(app, text="Enter your name")
```

Press F5 on the keyboard to save and run the program. The background will now be red. There are many colors that you can choose from. Check out this website, https://wiki.tcl-lang.org/page/Color+Names%2C+running%2C+all+screens, which lists the names of each of the colors. Try some out.

Resizing the text box

Secondly you can change the width of the textbox. Changing the width is useful if you have a longer name and need more space, or if you have a shorter name and need less space. To change the width of the text box, add the code `width=40` to the TextBox line of code. This is the line before the PushButton line. Try experimenting with the width until you find a suitable size.

```
textbox = TextBox(app, width=40)
button = PushButton(app, command=Hello, text="Click me", )
```

Press F5 on the keyboard to save and run the program.

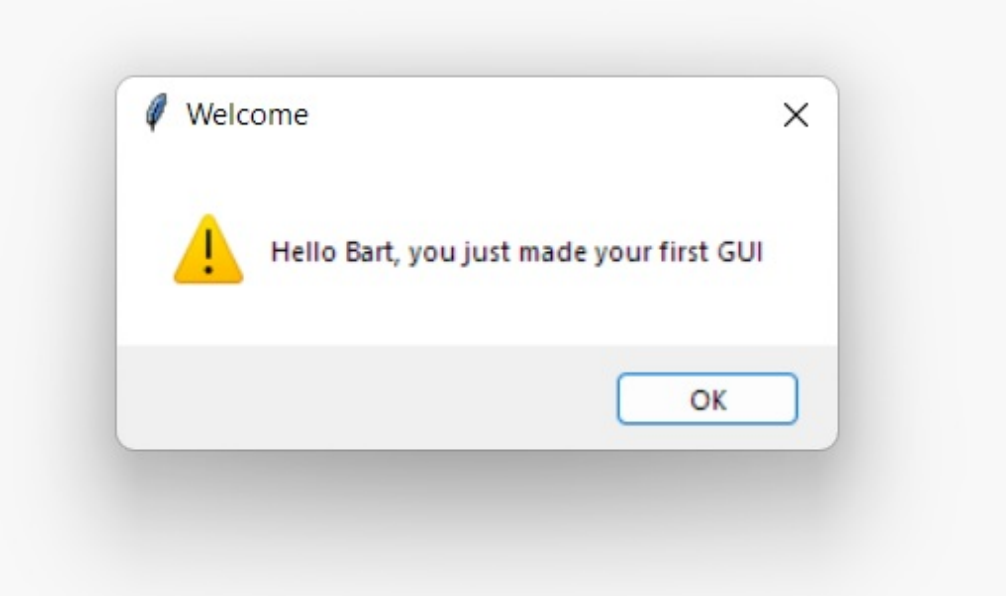
Using a warn window instead of the info window

In the last change to the program, edit the function so that a warn window is displayed instead of the info window (figure 1.14). The warn pop-up window has an image of a yellow warning sign. It is often used in GUIs to grab the user’s attention and warn them that an event is about to happen. To

make this edit, change the word `info` to `warn` on the line of code held in the Function,

```
def Hello():
    app.warn("Welcome", "Hello " + textbox.value + ", you just
made your first GUI")
```

Figure 1.14 The warn window pop-up



Press F5 on the keyboard to save and run the program.

Conclusions

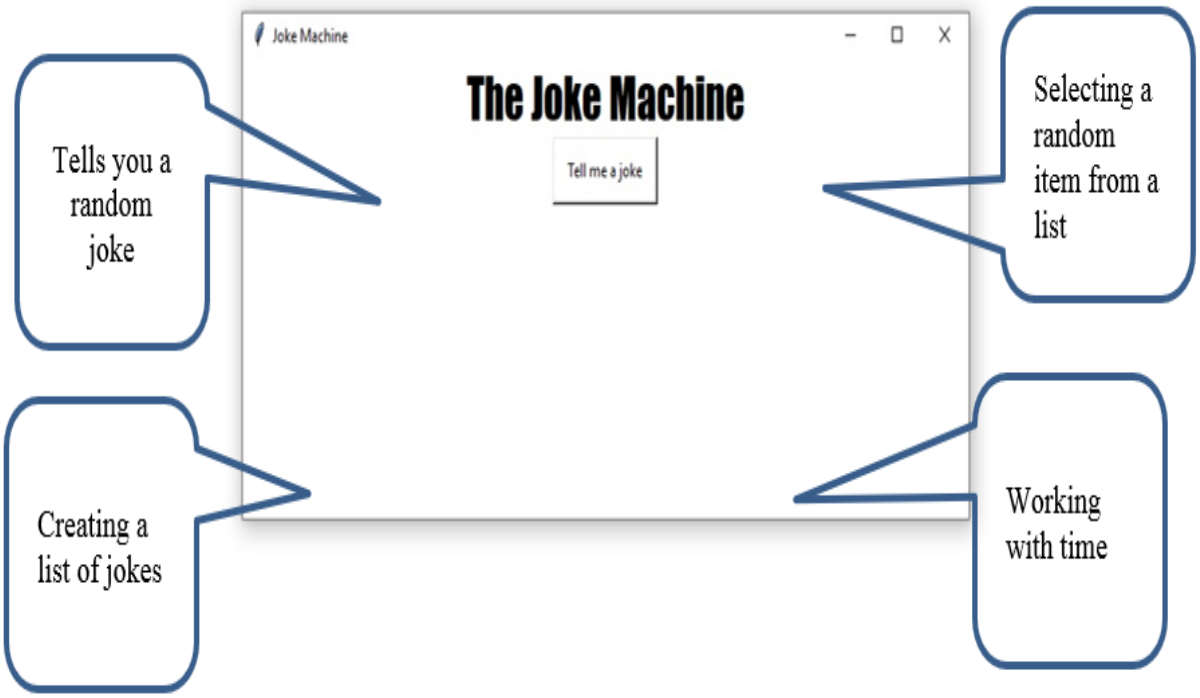
In this chapter you have created your first GUI. This project is important as it has introduced you to some of guizero's basic features. In this chapter you were introduced to the programming language Python. This language is a popular language amongst developers and can be used to create websites, games, perform data analysis and much more. You can download the chapter code here, https://github.com/TeCoEd/Twisted_Python_Projects/tree/main/Project%20Codes/Chapter%201

If you are new to Python, you can read more about the features of the language in appendix E before you start your next project. Or just jump straight in. Have fun!

Statements of fact

- Guizero is a package that is installed and enables programmers to easily create GUIs using the Python programming language.
- Widgets are the guizero name for the elements or items that appear on the GUI. Widgets enable you to interact with the GUI.
- Functions are reusable blocks of code that can be called and used anywhere within a program. Functions save time and make the program more efficient.
- A module is a library of code that contains a set of pre-built functions. Modules are imported at the start of a program.
- A variable is a location in the computer’s memory which stores information and data.
- A string consists of either text, numbers or charters enclosed within quotation marks. Strings can be stored and combined together to create a longer string. Numbers that are strings are valueless.
- Popups are message boxes that appear on your screen. Programmers use pop-ups to display information.
- The App windows holds all the elements and widgets of the GUI.
- A PushButton is a pressable button. Programmers assign functions to a button; when the button is pressed, the program calls and runs the function.

2 Are you funnier than a hyena?



This chapter covers

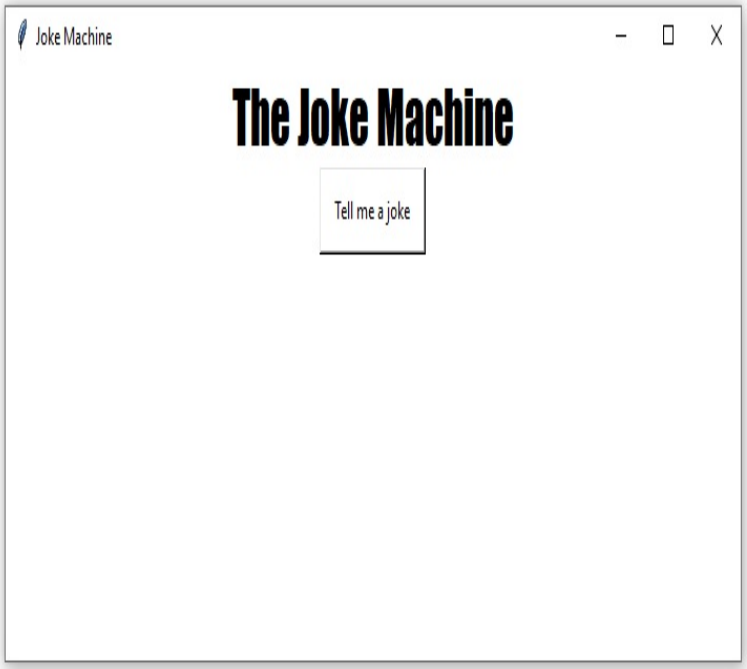
- Creating a joke machine
- Creating a list of jokes for the joke machine
- Creating a GUI for the joke machine
- Using a button to display a random joke from the list

There is a well-known saying that “Laughter is the best medicine.” The original quote is not quite as catchy: “A merry heart doeth good like a medicine.” The point of the quote is still the same; laughing makes you feel good. It can make you feel happy and make you feel better. If a good story or joke made you laugh, you are likely to share the joke with others and spread the good medicine!

What you will build in this chapter

In this chapter, since laughter is the best medicine, you will build a Joke Machine (figure 2.1) to spread laughter all around. The Joke Machine consists of a simple button which, when pressed, will display a random joke. Don’t like the joke? Not a problem. Simply press the button again and a new joke will be displayed. As people tell you or you hear more jokes, you can collect them and add them to your Joke Machine so that it always makes people laugh. You will need four jokes to start with for this project, you can use the ones in this chapter and if you hear any good ones, write them down so that you can use them in this project.

Figure 2.1 The Joke Machine GUI



While building the project you will learn the skills shown in table 2.1, including the random function, which is used to select a random item from a list. Choosing random items is a common feature of games such as Minecraft where the game world is randomly generated. A staple method of most user interaction with a device, from mobile phones, smart TVs even washing machines is via buttons. This project covers how to set up and add interaction to a push button, this is a visual button in the GUI that you interact with to make something happen. You press the button, and the joke

is displayed on the screen. Text is used everywhere, and this project covers how to display and edit text.

Table 2.1 Programming skills taught in this chapter.

Python	guizero	Programming techniques
Random module **Variables** **Lists** Strings Functions	Displaying text Push button **Resizing the window** **Looping a program**	Storing data **Displaying data** **Searching for data** **Iteration**

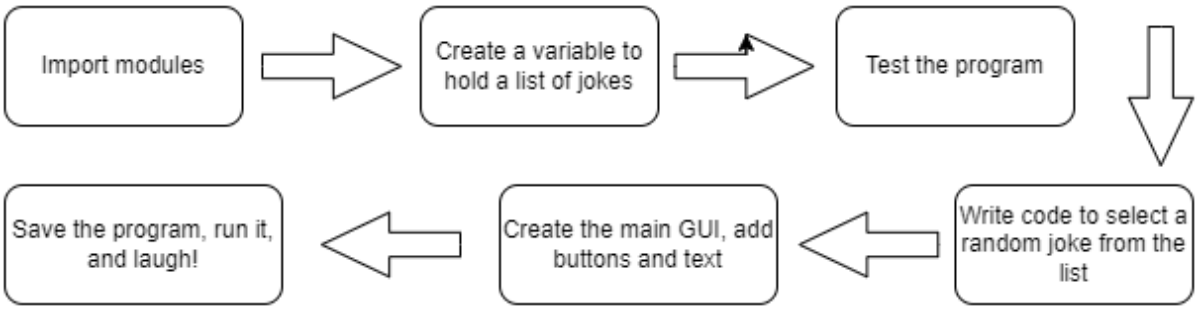
Creating the project

Before you start the project, you may want to gather a number of jokes that you can use in your Joke Machine. Or you can use the example jokes in the code and then replace them with yours later.

Here are some jokes to get you started.

Let's begin to build the Joke Machine. The project consists of several steps, which are shown in figure 2.2. Follow the steps in order.

Figure 2.2 Overview of the project



Importing the modules

The first step of the joke program is to import the modules that the program requires. You may remember from previous chapters that modules contain prewritten functions that perform different tasks, such as displaying text, displaying images, and much more. In this context, the modules provide all the code needed to create the various elements and widgets of your GUI. Prewritten functions save you time as you don't have to create them from scratch.

You are also going to create a list. Lists are a useful method of holding more than one item in a single variable. We will go into more detail about lists later in this section. For this program, you will use the list to store the jokes and assign them to a variable.

Creating your new program

Open a new Python program file in your editor. Refer to chapter 1 and appendix A for a reminder about editors. From the **File** menu,

1. Choose **File** > **New File**. Your Python editor will prompt you to save the file.
2. Save the file with the name Joke_Machine.py and choose Save.

Remember

(as with the previous projects, if you used the guizero quick install method from Appendix C then you must ensure that you save the file into the same

folder as the guizero.)

Importing the needed modules

Begin the Joke Machine program code by importing the required modules (listing 2.1) from the guizero library. You installed this library chapter 1. The modules provide all the code you need to create the various elements and widgets of your GUI.

Listing 2.1 import the modules

```
# Imports

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

In this program, you are importing the following guizero modules:

- App, to build the app window,
- Text, to enable the program to display the Jokes,
- PushButton, to create a button that displays a joke when pressed.

This program also uses the `random` module which is imported from the Python program. This module is used to select and display a random joke from the list. This is instead of always displaying the jokes in the same order each time the program runs. It would be unfunny if you always already knew the joke and its punchline.

Creating a list of jokes

Next, you need to create a list to store your jokes. You use lists all the time. Think of a to do list, a list of friends or a shopping list (figure 2.3)—these are both examples of lists that contain all the things you must do or, in the case of a shopping list, all the items that you need to buy. You can read the list, cross items off, and even add to the list.

Figure 2.3 A shopping list is a list of items.

Creating your list of jokes

Your list of jokes is stored in a variable with the label `list_of_jokes`. You may recall from chapter 1 that the label cannot have spaces in between the words. But it's hard to read words when they're crammed together without spaces. To solve this, you can join the words using underbars (_). The variable `list_of_jokes` is easier to read than `listofjokes`.

A variable is a location in the computer's memory that stores the jokes. The name references the location of the memory location. RAM stores the information your computer is actively using so the information can be accessed quickly by the program. The Joke Machine program needs to be able to find the memory location where the jokes are stored, so you name or label the location so the program can quickly go straight there and collect your jokes. Write up and add the code below in listing 2.2, underneath your imports.

Listing 2.2 Creating a list to hold the jokes

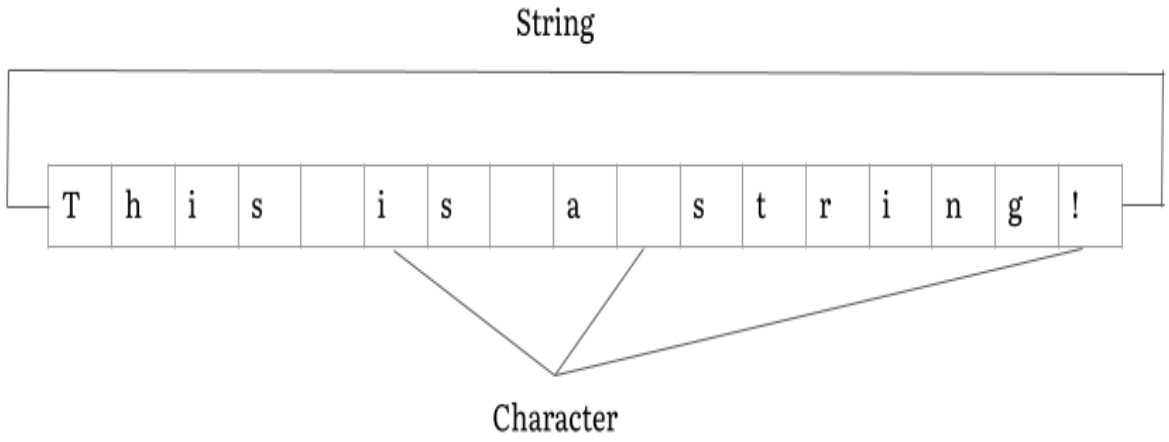
```
# Variables

list_of_jokes = ("What do you call a man with no shins? Tony
(Toe – Knee)",
                 "Nothing begins with N and ends with G",
                 "What kind of tree can fit in one hand? Palm
tree",
                “[CA]”"Why do owls not go dating in the rain?
Because it’s too wet to woo")

print (list_of_jokes)
```

Each joke in the list is entered as string. A string is a series of characters. A character can be a letter, number, or symbol, figure 2.4. For example, the letters in this sentence are characters. In Python, strings start and end with the " symbol, like this: `"This is a string."` Notice that each joke is a string, and each joke, apart from the last joke, where the comma is optional.

Figure 2.4 Anatomy of a string

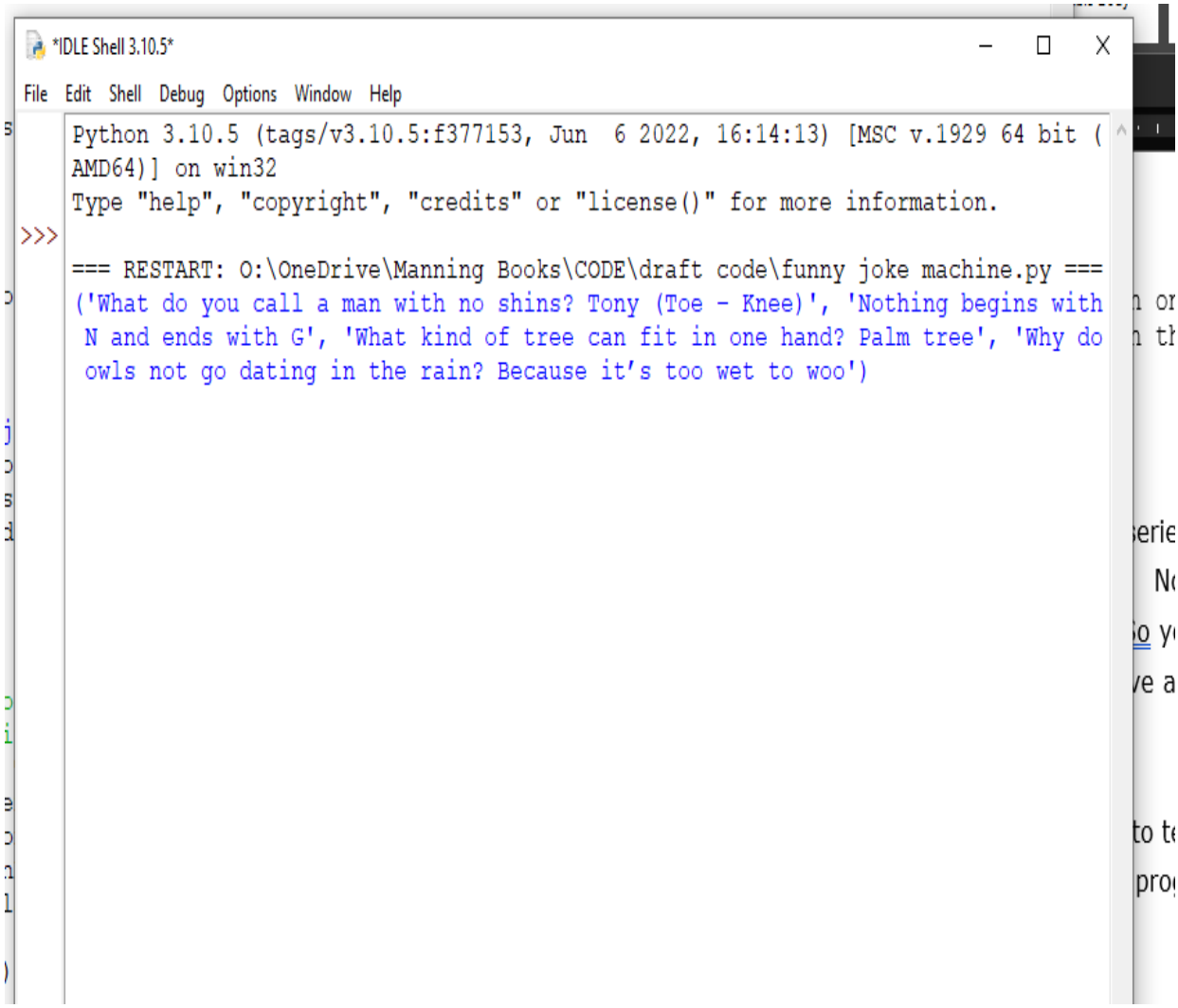


The last joke in the list ends with a bracket to close the list, `"Why do owls not go dating in the rain? Because it's too wet to woo").` You can continue to add any number of jokes to the list, but start with a few to test that the program works correctly. For now, just add four jokes; using a small number of jokes will make finding errors and testing easier.

Running your program to test if it works

On the last line, the code `print(list_of_jokes)` is used to test that your program is working correctly. Without the final GUI window, this is sometimes helpful to test parts of your program in small pieces. Press **F5** on the keyboard to save and run your program; you should see all your jokes displayed in a long line in the window of the Python editor, as shown in figure 2.5. When you have tested the program change the last line of code to `#print (list_of_jokes)`, this will stop it printing out all the jokes.

Figure 2.5 Output of the test



Troubleshooting your program

Programs don't always run perfectly the first time. Programmers are used to this. To find and fix the problems that make their code run poorly, programmers use a process called troubleshooting. The first step of troubleshooting is to check for common errors. Even the best programmers make them!

If your program fails or does not run as expected, check for these common errors:

1. Have you used quotation marks ("") around each string?
2. Is the closing parenthesis—the —present at the end of the list?

3. Did you label the variable `list_of_jokes` and not `list of jokes`?
4. Did you save the program into the GUI folder as shown in chapter 1?

Creating a function to display a random joke

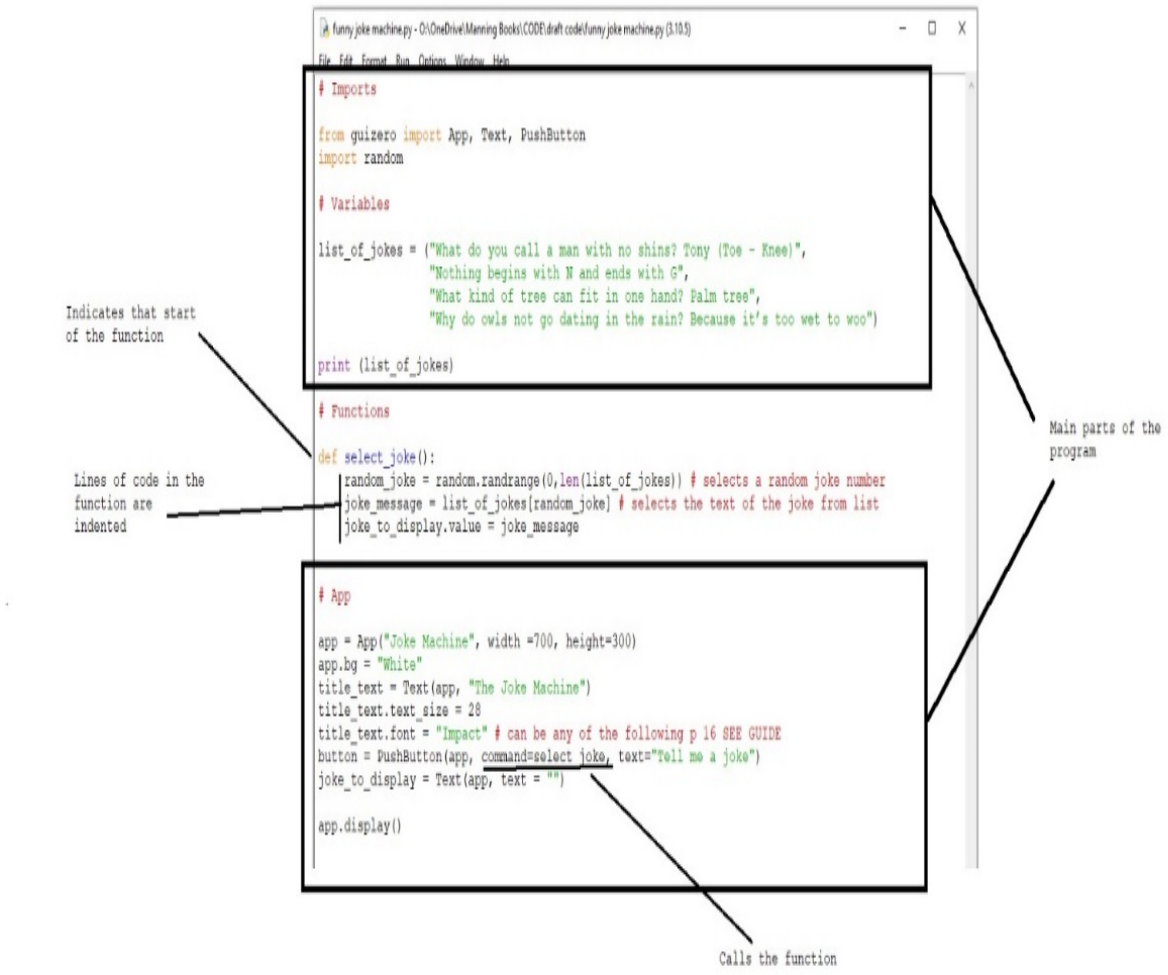
Now you are ready to take the next step: writing the function that will select a random joke from your list of jokes.

Why create a Function?

Imagine you are playing your favorite computer game and you are down to your last life; you can lose your final life at any point in the game. So potentially, there are millions of points during the game where it's, game over. Most games contain a "game over" animation where the game character dies, faints, falls over, melts, dissolves, and even flashes. If the game developers didn't use a function, they would have to program the "game over" animation code millions of times, once for every possible scenario where you lose your last life. This would be very tedious and make the program code massive.

You only need to write the block of code once (figure 2.6), then you can use it as many times as you like, which saves processing power and makes the program more efficient.

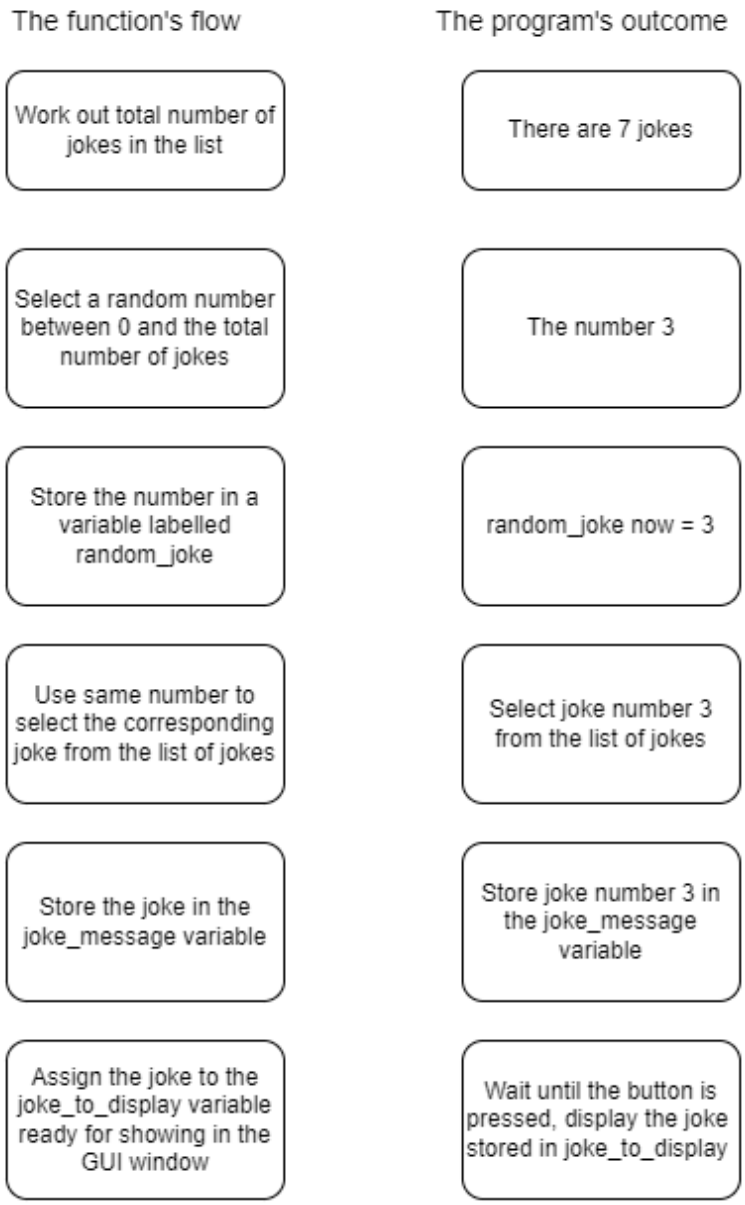
Figure 2.6 Creating a function to select a random joke from the list



The function in the Joke Machine program will be assigned to a button in the GUI, which when pressed will call the function and then run the code to select a random joke. This happens every time the button is pressed. Figure

2.7 shows the flow of the function on the left side, and on the right, what the outcome is in the program.

Figure 2.7 Flow of the function that selects the joke



Add the lines of code in listing 2.3 to create the function on the next line of your program.

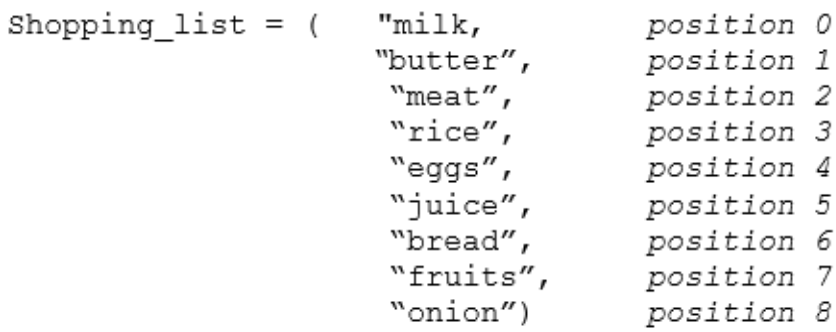
Listing 2.3 Create a function to select a random joke

```
def select_joke(): #A
    random_joke = random.randrange(0,len(list_of_jokes)) #B
    joke_message = list_of_jokes[random_joke]
    joke_to_display.value = joke_message #C
```

A function begins with the text `def`, which stands for define. This tells Python that you are creating or defining a new function. Then you must name the function so that you can call it by its name and reuse it. (If you have a dog, it has a name; usually when you call their name, they come up to you! Not all the time. Likewise with cats! But functions must come every time that you call them) The function in the Joke Machine program is called `select_joke():` as its purpose is to select a joke from the list. Note the use of the `():` at the end of the function, you must use this syntax, so Python knows that you are creating a function.

The next line of code selects a random item. In this program, the random item is a random joke from the list. Consider the image in figure 2.3 of a shopping list (in the style of a Python list) and number each of the items starting from zero to the end. The first item, milk, would be numbered 0, then butter numbered, 1, then meat 2, rice 3, and so on. The line of code uses `random.randrange` which means a random selection within the range, and you may know from maths that a range is from the first number to the last number in a list.

Figure 2.8 The shopping list as a Python list



In the example of the shopping list, our first item on the list is milk and the last item, an onion. The position of the onion is eighth, so the range of the

shopping list, is 0 to 8. Now, you need to tell the program what the range of the list of jokes is so that it knows the total number of jokes in the list and can select a position that exists in the list. (Basically positions 9 and above, do not exist and the program would fail). Finding the start of the list is simple as it starts at zero, it starts at the beginning. You then set the end of the range as three since we have 4 jokes and joke 0, joke 1, joke 2 and joke 3 will give us four positions to hold each individual joke.

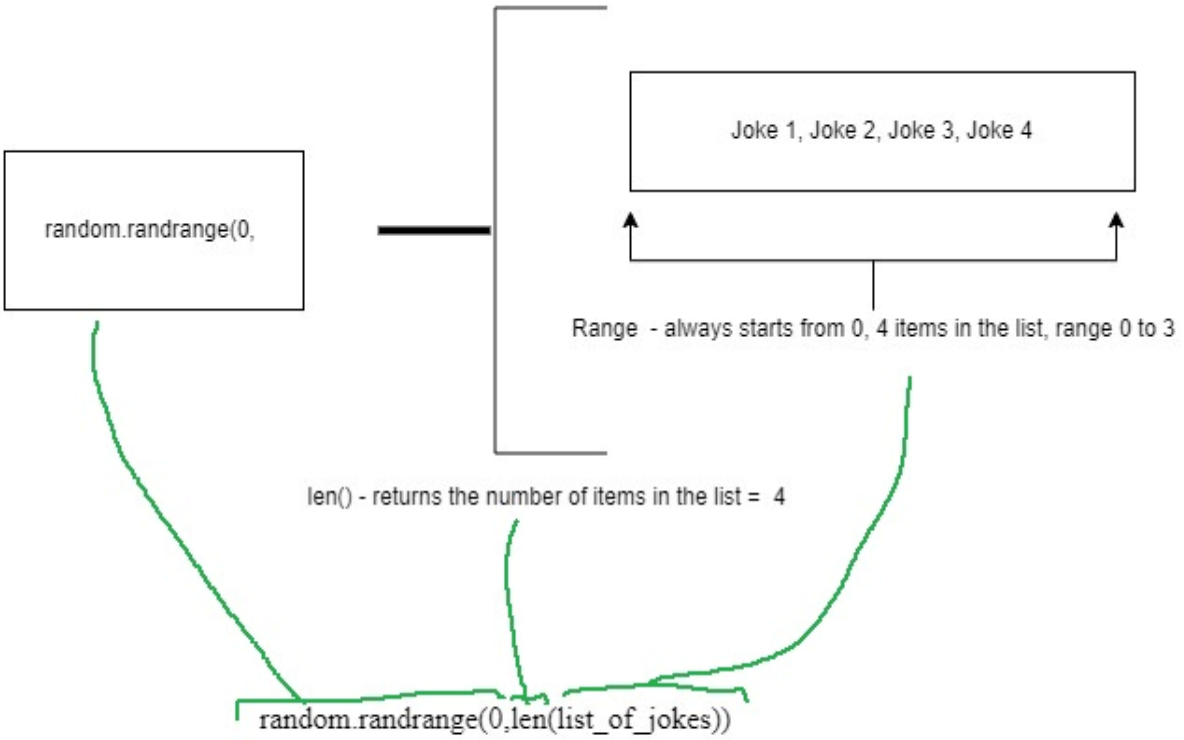
Why start from zero?

Counting positions start from 0 because Martin Richards, creator of the BCPL language, designed lists to start from at 0 as the natural position.

However, what happens if you add more jokes? That would mean you would have to change the range. For example, if you add 8 jokes you would need to change the end of the range to 7. If you forget to do this, it will create an error and the Joke Machine would stop working (figure 2.9).

The solution is to instead, use the code `len(list_of_jokes)` which finds and returns the length of a list with any number of items stored in it. The `len()` function, short for length, enables the program to find the length or the end of the variable called list of jokes that we created in listing 2.2. So, this line of code is basically looking up how many jokes you have stored in the list and then returning the end value of the range.

Figure 2.9 How the program selects a random number from a list of any length



Since the program knows the length of the list, it can then use the `random.randrange` to select a random joke from your list of jokes. This is a prewritten function from the Python random module that is included with Python. The function consists of a program that can select a random number from a range of numbers defined by the user, in the same way that you can roll a standard die and it will land on one of size numbers.

Note that at this stage the program is only returning the position of the item, not the actual joke. This position value is then returned and stored in a variable named `random_joke`.

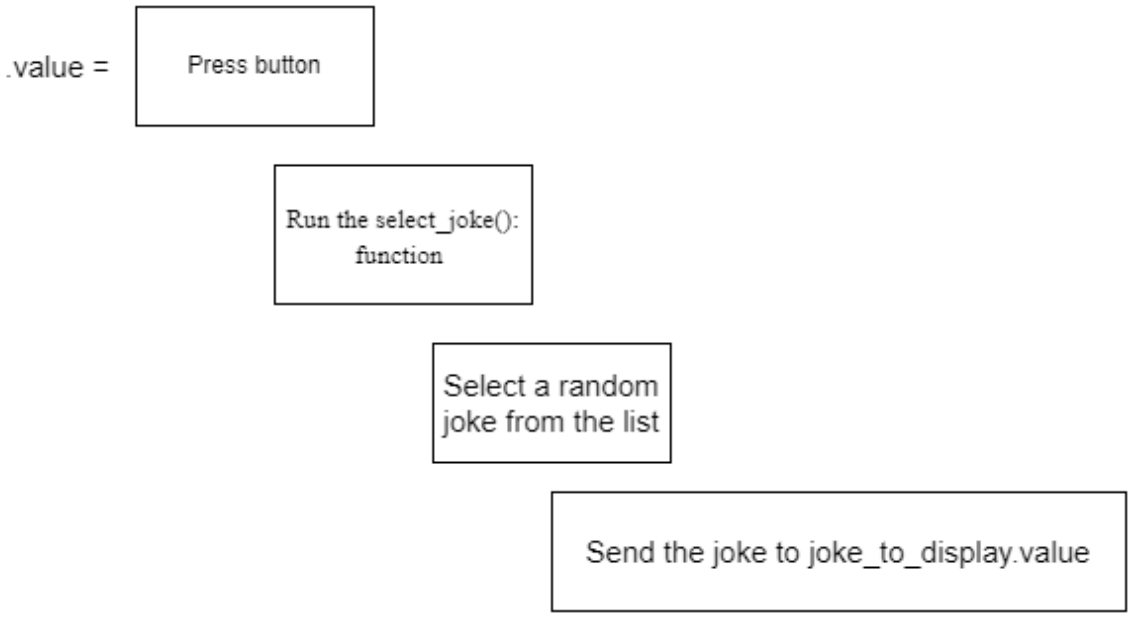
So now the variable `random_joke` holds a random position number from the list of jokes (note that this is just the number of the item and not the physical text of the joke). In order to extract or collect the joke that will be displayed, use the code `joke_message = list_of_jokes[random_joke]` which creates a new variable named `joke_message`, selects the item number from the list of jokes, then returns the joke as a string and saves it into the variable ready to be used in the program, figure 2.10.

Figure 2.10 Selecting a random joke from the list of jokes

The last line of code assigns the joke, as a string, to a new variable so that computer can pass the joke to the GUI and display it on the Joke Machine window. The term “passing” describes the process of taking the joke that is stored in the computer’s memory and making it available for the program to display the joke as text in the GUI window. The code uses the function `.value` at the end of the variable, to tell the program that the data the variable holds will be updated. You will see how this works later in listing 2.5.

The `.value` code is responsible for replacing the existing value of the joke. What this means is that the program displays a joke. Then the next time you press the button, the program calls the function again and *overwrites* the current joke with the new joke.

Figure 2.11 How the .value code works



If the value was not updated, then all the new jokes would be displayed, and the program would end up with long list of jokes cluttering up the GUI window. Note that this variable, `joke_to_display.value`, currently has no value as the program has not run and the function has not been called. Therefore this variable is empty.

Creating the GUI

Now you are ready to create the App section of your program (listing 2.4). The App section brings together all the features into a GUI.

The first line of code names your GUI. The name will appear in the top left of the GUI window.

Next, we set the background color of the GUI to white. You can change the color by replacing the word White with the color that you want the background to change to. Simply replace White with the word Green, to change the background to the color green.

Then display the title of the program in your GUI and set the size of the text. You can change this if you want to call your Joke Machine something else, like “Ha Joke Teller!” Simply replace the text but remember to use the " ".

Next, as you have done before in previous chapters, you can set the GUI's height and width. You may need to change these values depending on the length of your jokes. If you have chosen longer jokes, then you may need the GUI window to be wider.

Finally, set the font (what the text looks like) to Impact. You do this using this line of code:

```
title_text.font = "Impact”
```

You put the name of the font you want use between the quotation marks. We will cover changing font styles, colors, and background in more detail in chapter X.

Add the code from listing 2.4 to your program.

Listing 2.4 Set up the GUI

```
# App

app = App("Joke Machine", width=900, height=300) #A
app.bg = "White" #B
title_text = Text(app, "The Joke Machine") #C
title_text.text_size = 28 #D
title_text.font = "Impact" #E
```

Adding a control button

The last part of your program code adds the button, which, when pressed, will run the function from listing 2.3. When the function runs, it displays a random joke from the list.

Start by creating a variable which will hold the code instructions to create the button. In listing 2.5, the `command=` tells the program which function to run—the function that you created. The code basically says that when the button is pressed, run the function called `select_joke`.

Lastly, add the text that you want to display on the button. Something simple like “Tell me a joke” provides the user with an instruction about what the button does when it is pressed.

If you look at listing 2.3, you see the last line of code uses the variable `joke_to_display.value`. This variable holds the joke that is going to be displayed. However, the program needs to firstly be expecting a joke.

Imagine you invite six friends round to your house, and an extra friend turns up. So, there are now seven friends, not six. You were not expecting them, so you don’t have a chair for them to sit on. Now as a kind human you would share or maybe give up your chair for the extra person. However, alas computers are machines and would not.

They follow exactly what the program instructions ask them to do, computers cannot adapt to deal with unexpected or additional requirements. So, in the computer world, if a seventh friend turned up and the computer was expecting six, then this would cause an error as the program was not expecting it.

In the program we can create a list of six jokes, but creating the jokes is not the same as telling the program to expect a joke. You have to program in the expectation!

To solve this issue, we create the `joke_to_display` variable, and use the `Text` command so that the program knows that it will be holding a string, because it is going to display the joke as text. Next set the text to empty/blank using two quotation marks with nothing in between them, like this "". This means that when the program runs, the variable is storing an empty string. The line of code is basically getting a chair ready for when your friend arrives. They are not seated in the chair until they arrive; however, you still get the chair ready for them. You can think of this line of code as creating a placeholder for the joke. It must be blank at the beginning of the program because the joke is only displayed when the button is pressed.

The final line of code simply tells your program to display the GUI when you run it. Copy the lines of code from listing 2.5 into your program.

Listing 2.5 Add the control button

```
button = PushButton(app, command=select_joke, text=”Tell me a
joke”)
joke_to_display = Text(app, text = "")

app.display()
```

Well done. You have now completed the Joke Machine. Press **F5** to save and run your program and have a giggle! If the program doesn’t run, check out the troubleshooting section.

Listing 2.6 Final Listing

```
# Imports

from guizero import App, Text, PushButton
import random

# Variables

list_of_jokes = ("What do you call a man with no shins? Tony
(Toe – Knee)",
                 "Nothing begins with N and ends with G",
                 "What kind of tree can fit in one hand? Palm
tree",
                 "Why do owls not go dating in the rain? Because
it’s too wet to woo",)

print (list_of_jokes)

# Functions

def select_joke():
    random_joke = random.randrange(0,len(list_of_jokes)) #
selects a random joke number
    joke_message = list_of_jokes[random_joke] # selects the text
of the joke from list
    joke_to_display.value = joke_message
```

```
# App

app = App("Joke Machine", width =700, height=300)
app.bg = "White"
title_text = Text(app, "The Joke Machine")
title_text.text_size = 28
title_text.font = "Impact" # can be any of the following p 16
SEE GUIDE
button = PushButton(app, command=select_joke, text="Tell me a
joke")
joke_to_display = Text(app, text = "")

app.display()
```

Automating the Joke Machine

In this section, you will adapt your original program so that it automatically displays a joke after a set time. This process of making a program automatic is called automation. Automation is an essential part of computer programs, as much of today's software and hardware is required to run without human intervention. Automation is defined as “the application of technology, programs, robotics or processes to achieve outcomes with minimal human input” (https://www.cognixia.com/).

Imagine if you were listening to music and every time you wanted to change to a different song it required you to firstly stop the song you're listening to, locate the new song you want to listen to, load that song, and press play. (This is how listening vinyl records works.) It would be frustrating and time consuming.

Instead what happens is the software is automated. You simply click the songs or playlist you want to listen to, and the program automatically plays through the songs.

An automated Joke Machine is a great motivational tool. If you are working or studying (sometimes called revising in other countries) and need to take a break, you can set the program to display a joke every 15 minutes; then you

know you have completed 15 minutes of work and you get to laugh! For more customizable timers, check out chapter X.

To automate your Joke Machine, you are going to add a line of code that makes the program repeat itself. A program or a section of a code that repeats is called iteration. An iteration is used to run a part of the program repeatedly as long as a given condition is met. In the Joke Machine, the condition is that the set time between jokes has passed.

CONFUSING TERMS

Often the terms loop or looping and repeat are used to denote the use of iteration.

Open your original Joke Machine program (**Joke_Machine.py**). First you need to save the program with a new name. This is so that you don't overwrite your first program.

1. From the Python editor menu, choose **File** > **Save As**.
2. Save your Python file as Joke_Machine_Auto.py. This will ensure that your original program is not overwritten or deleted.
3. Next, in the App section of the program, find the line of code,

```
joke_to_display = Text(app, text = "")
```

Underneath this line, enter the code in listing 2.7:

Listing 2.7 Automating the Joke Machine

```
joke_to_display = Text(app, text = "")
joke_to_display.repeat(1000, select_joke)

app.display()
```

This line of code is very simple and consists of a repeat function to instruct the program to repeat the function called `select_joke`. Then you tell the program how often you want it to repeat the function. In listing 2.6, the

repeat time is set to 1000. This means that the repeat time is set to 1000 milliseconds, so the function runs every second. The Joke Machine automatically will display a new joke every second, this is probably too fast as it will not give you time to read the joke. The joke may also be very funny, and it makes you laugh a lot. By the time you have finished your laughing fit you will have missed maybe 20 other jokes. However, one second is a suitable time period for testing to observe that the program is working correctly.

The program uses milliseconds to store the time so you will need to be able to convert seconds into milliseconds. You need to calculate the milliseconds before you add them to your program code. Use the simple formula shown below,

Milliseconds = seconds * 60

However, you will also have to covert minutes into seconds which uses the following formula,

Seconds = minutes * 60

Let’s look at an example, if you want your Joke Machine to display a new random joke every 5 minutes, then you would calculate the required number of milliseconds as follows,

5 minutes is 5 * 60 seconds,

this is a total of 300 seconds

300 * 1000 is 300,000

So, 5 minutes in milliseconds is 300,000 seconds

The line the line of code in your program would therefore be

```
joke_to_display.repeat(300,000, select_joke)
```

For your convenience, table 2.2 provides the time in minutes, seconds, and milliseconds of common time intervals.

Table 2.2 Times in second and milliseconds of common time intervals.

Minutes	Seconds	Milliseconds
1	60	60,000
2	120	120,000
5	300	300,000
10	600	600,000
20	120	120,000
30	1800	1800,000
45	2700	2700, 000
60	3600	3600,000

Once you have added the extra line of code and decided on your time interval, press **F5** on the keyboard to save and run your program. (It's always useful to set the time interval to 1000 at first, so that you can test the code is working correctly). Congratulations, you now have an automated Joke

Telling Machine but, the real question is are your jokes funnier than a Hyena?

Three other things to try

In this chapter you created a funny Joke Machine that either displays jokes when you click a button or automatically displays a joke every X number of minutes. Remember that you can add your own jokes in order to personalize the Joke Machine, but remember that you may need to readjust the height and width so the jokes are displayed correctly. Here are some other features that you might want to add.

Changing the font color and type

Firstly, you can change the colour of the font and the font type. This edit can be made to any of the text, but would work better if you change the joke's font color. The color will grab the attention of you audience. On the last but one line of the program, **before** the `app.display()`, add the following line of code:

```
joke_to_display = Text(app, text = "")
joke_to_display.text_color = "Green"
```

You can also change the style of font by adding this line of code after the line of code that changes the color:

```
joke_to_display.font = "fontname"
```

Replacing the name of font with the name of the font that you want to use. The fonts that are available depends on which operating system you are using.

Changing the GUI's background color

Changing the background color of the GUI window will make your Joke Machine more unique and appealing. Simply add the code

```
app.bg = "White"
```

into the #App section of the program code.

Displaying a funny picture

There are so many funny pictures that it seems a shame not to display one in your GUI. You can choose a meme if you like (more on memes in chapter 14).

Start by saving the picture that you want to display into the same folder as the GUI code is stored. Remember that the image file needs to be either a .PNG or a GIF.

Add the following line of code below to the #App section of the program code

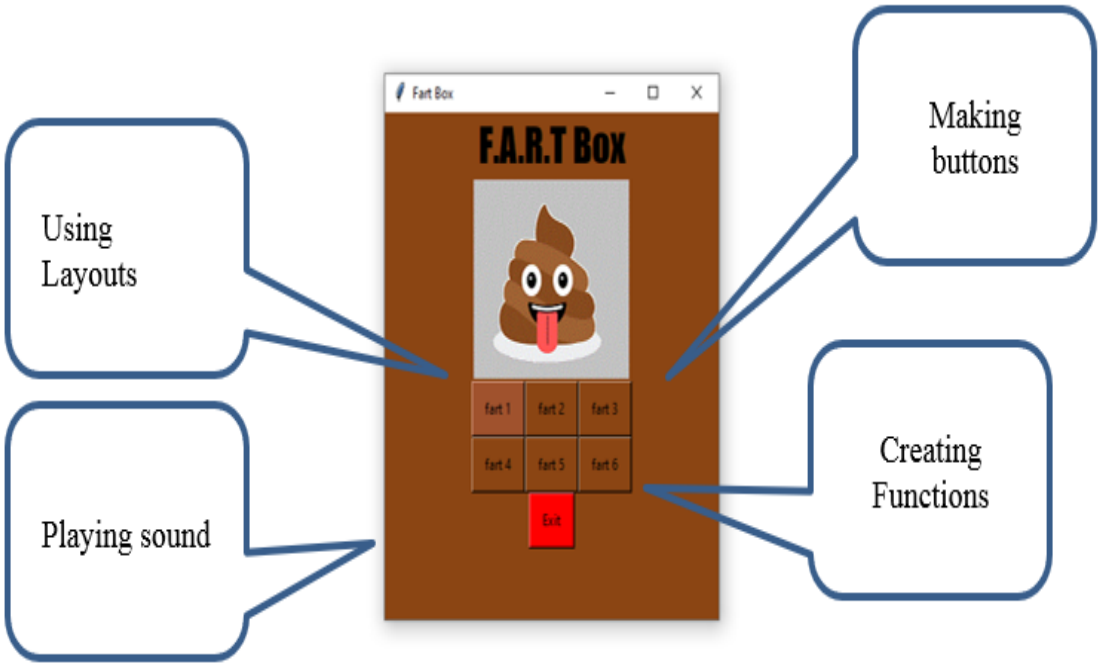
```
picture = Picture(app, image="name of file.gif")
```

Replacing the name of file with the file name of your picture.

Statements of fact

- A variable is a location in the computer’s memory which stores information and data. You create variables and use them to store data and strings.
- A list is a type of variable that can store more than one item. You create lists when you want to store more than one item in a location.
- Iteration is repeating or looping parts of the program. You use iteration to tell the program to repeat part of, or to automate parts of, the code. Iteration reduces the amount of human input required.
- When testing programs that use time, pauses and delays, it is always useful to initially set the time interval to 1000 milliseconds at first, so that you can test the code is working correctly.

3 F.A.R.T. box, a disgusting soundboard



This chapter covers

- Using layouts
- Making pushbuttons
- Assign sounds to Pushbuttons and playing audio
- Creating functions

Have you heard the term “soundboard”? ” A soundboard usually consists of a computer program, a physical device, or a website that has a selection of buttons. When you press a button, it plays an audio clip. The sound clip is commonly a quote from a film or a lyric from a song, and has even been sound effects. Soundboards are self-contained, requiring no outside media player; you simply press a button and sound is played.

You may not know this, but you produce about 500 to 1,500 milliliters of gas per day and expel it in 10 to 20 farts. Trumps, bottom burps, gas, noises are funny and make most people laugh. Farts have an amazing habit of being able to distract anyone hearing one, especially if they are playing an online game. So, in this project we will build a fun GUI that combines a selection of fart noises to create a Fluff Audio Resonance Transmission (F.A.R.T.) sound board.

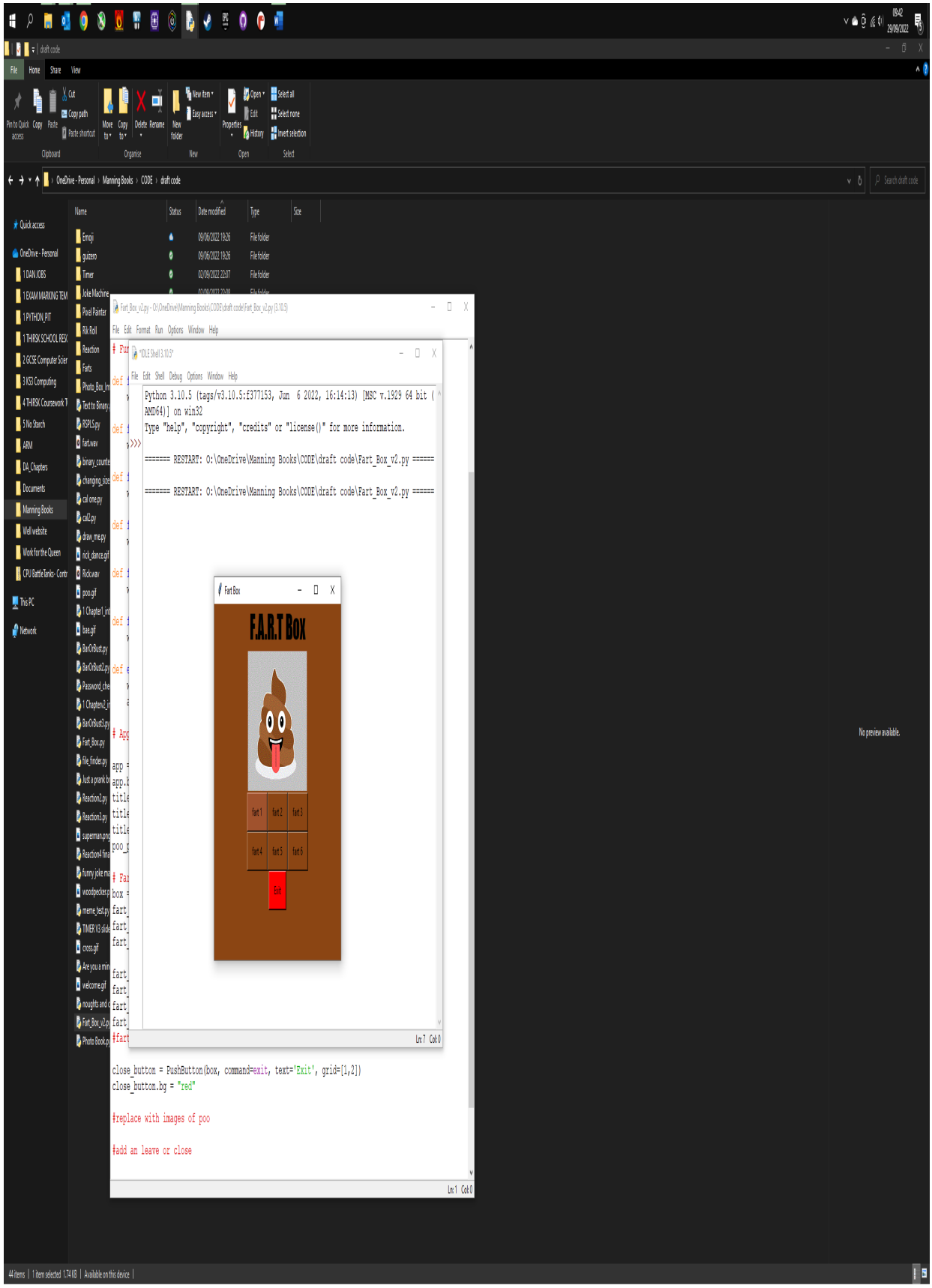
WHAT YOU WILL BUILD

In this chapter you will build a F.A.R.T. Box soundboard (figure 3.1). This soundboard is a GUI with seven buttons. When the user presses a button, the app plays a fart sound. Press a different button and it plays a different fart sound. The last button closes the soundboard.

Make your own

However, if you prefer, you can collect your own sound clips and make your own soundboard; for example, cat meows, famous quotes from a TV show, and so on. The program code and the process are still the same; you just replace the sound clips with your own audio.

Figure 3.1 The Fluff Audio Resonance Transmission (F.A.R.T.) soundboard


draft code
File Home Share View
Pin to Quick access Copy Paste Cut Copy path Paste shortcut
Clipboard
Move to Copy to Delete Rename
Organize
New folder New item Easy access
New
Properties Open Edit History
Open
Select all Select none Invert selection
Select
OneDrive - Personal › Manning Books › CODE › draft code
Search draft code
Quick access
OneDrive - Personal
1 DAN JOBS
1 EXAM MARKING TEM
1 PYTHON_PIT
1 THRSX SCHOOL RESV
2 GCSE Computer Scier
3 KS3 Computing
4 THRSX Coursework T
5 No Starch
ARM
DA_Chapters
Documents
Manning Books
Well website
Work for the Queen
CPU BattleTanks- Contr
This PC
Network
Name
Status
Date modified
Type
Size
Emoji
09/06/2022 19:26
File folder
quizzes
09/06/2022 19:26
File folder
Timer
02/09/2022 22:07
File folder
Joke Machine
Pixel Painter
Rik Roll
Reaction
Farts
Photo_Box_Im
Text to Binary
RSPLS.py
fart.wav
binary_counte
changing_size
cal one.py
cal2.py
draw_me.py
nick_dance.gif
Rickv.wav
poo.gif
1 Chapter1_Int
bae.gif
BarOrBust.py
BarOrBust2.py
Password_che
1 Chapterv2_Int
BarOrBust3.py
Fart_Box.py
file_finder.py
Just a prank bo
Reaction2.py
Reaction3.py
superman.pn
Reaction4 fina
funny joke ma
woodpecker.p
meme_test.py
TIMER V3 slide
cross.gif
Are you a min
welcome.gif
noughts and c
Fart_Box_v2.p
Photo Book.p
Fart_Box_v2.py - O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py (3.10.5)
File Edit Format Run Options Window Help
IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py ======
====== RESTART: O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py ======
Fart Box
F.A.R.T Box
fart 1
fart 2
fart 3
fart 4
fart 5
fart 6
Exit
Ln: 7 Col: 0
close_button = PushButton(box, command=exit, text='Exit', grid=[1,2])
close_button.bg = "red"
#replace with images of poo
#add an leave or close
Ln: 1 Col: 0
No preview available.
44 items | 1 item selected 1.74 KB | Available on this device

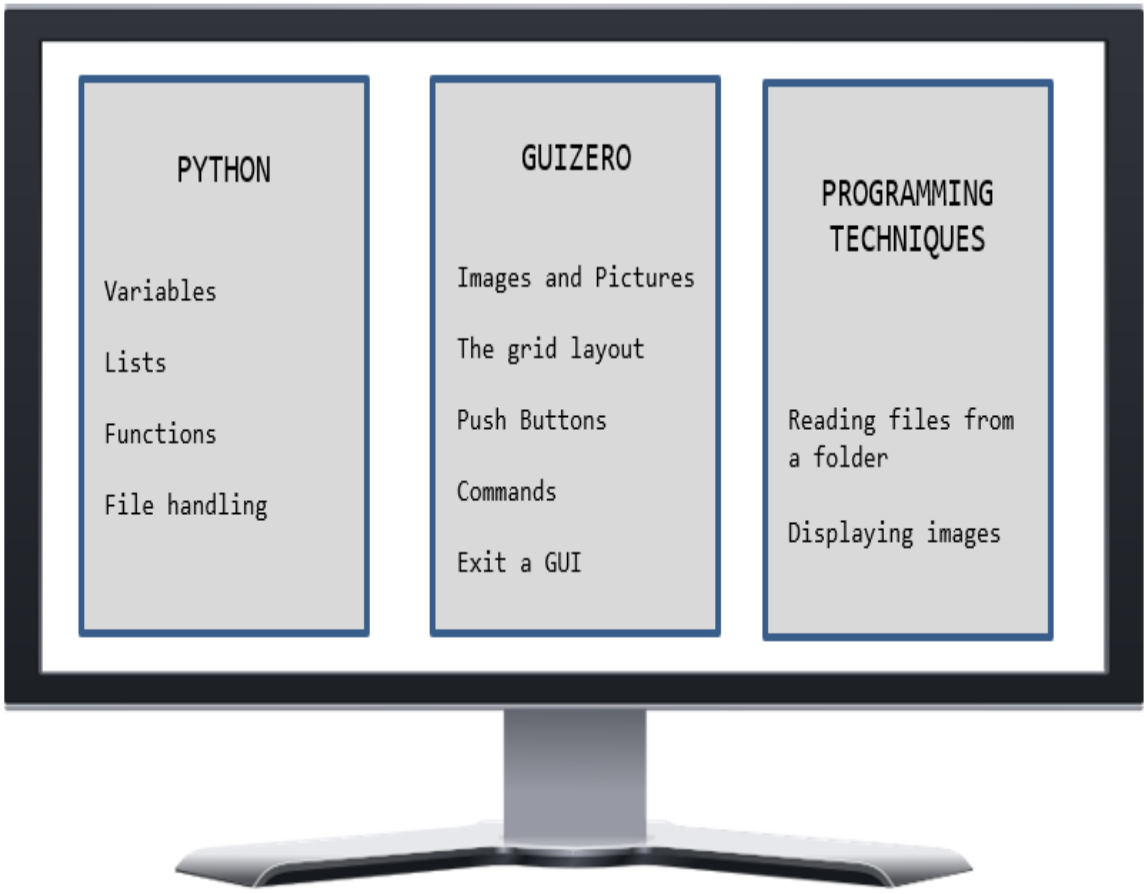


Table 3.1 Skills that you will learn

Python	guizero	Programming techniques
Variables Lists Functions File handling	Images and Pictures The grid layout Push Buttons Commands Exit a GUI	Reading files from a folder Displaying images

Creating the project

Before you start the project, you will need to download the six audio clips from 888, (or you collect your own audio files. These sound files need to be .wav (waveform) audio file format, as this file format is compatible with the Python module that we are using to play sound. More detail on this later, in the paragraph after we import the modules in the listing 3.1.

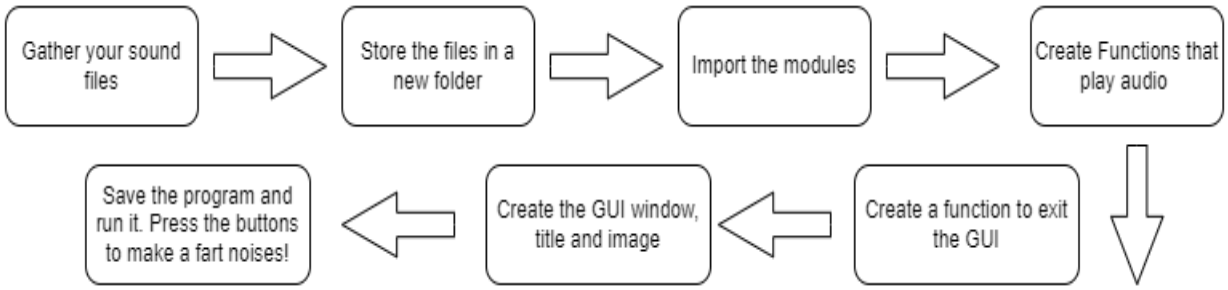
The project has six buttons. Therefore, the project requires six fart sounds. The soundboard also has an exit button which, when pressed, closes the GUI window, but not before leaving you with the sound of a loud burp! The GUI also displays a suitably related image file, of a poo, that is 160 pixels wide and 180 pixels high. The image will be displayed in the GUI window.

Guizero uses .png and .gif file formats, so ensure that your image file is a correct. If working on an Apple computer, macOS only supports the GIF file format.

Let’s begin to build the F.A.R.T sound board. The project consists of several steps which are shown in figure 3.2. The main steps of the project are to

1. Download the collection of audio files of the fart noises and the image .
2. Create a new folder, then copy and store the audio files in this folder.
3. Start a new Python file and import the required modules.
4. Code six versions of a function that will each play one of the audio files when a corresponding button is pressed.
5. Create a function to exit the soundboard once you have had enough of all the fart noises.
6. Create the main GUI window to hold the title, image, and buttons.
7. Create the required buttons.
8. Save and test the soundboard
9. Turn up the volume, press a button and make a fart sound!

Figure 3.2 Flow diagram of the building the Fluff Audio Resonance Transmission (F.A.R.T.) soundboard

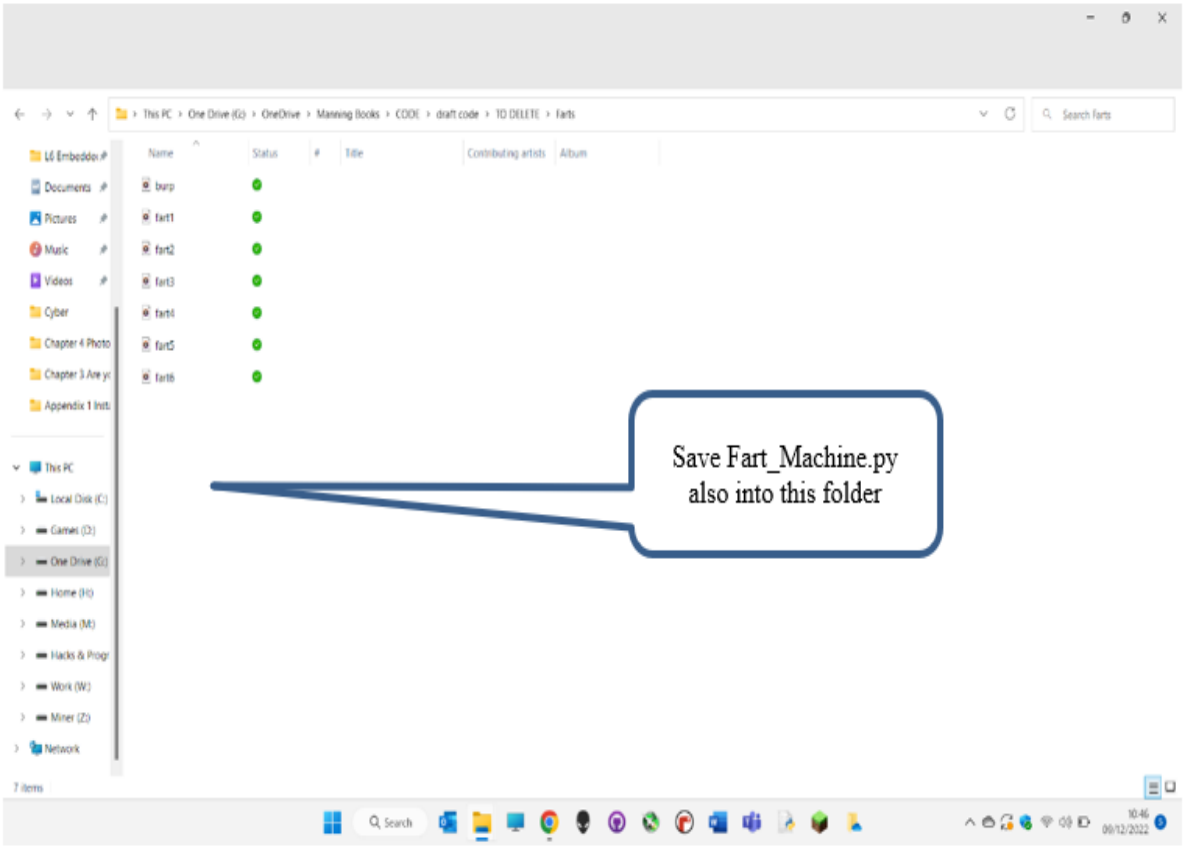


Downloading and storing the audio files

There are several options that you can choose from to collect the audio clips for your soundboard. The simplest is to head over to the projects page and download the audio files. I recommend building this project with the supplied audio files to ensure that the program works correctly. Once working, you can then source your own sounds and customize your soundboard. The audio sound files need to be stored in a folder that both Python and guizero can access. Begin by opening the folder where you are saving your projects,

1. Create a new folder and name it, Farts.
2. Open your web browser and enter the address, https://github.com/TeCoEd/Twisted_Python_Projects/tree/main/Project%20Codes/Chapter%203
3. Click on the folder called **Fart_sounds** and download it the audio clips and image.
4. Open the folder and extract the audio files.
5. Copy and paste, drag, or save all the audio files into the Farts folder, figure 3.3.

Figure 3.3 Fart files in the folder.



You are now ready to write the program code and create your disgusting or not so disgusting soundboard.

1. Load your Python editor.
2. Open a new Python program file, from the **File** menu, choose **File** > **New File**.
3. A new window opens. Choose **File** > **Save**. The Python editor prompts you to save the file.
4. You must save the Python file in the same folder that also holds the audio clips— Farts folder. If you do not do this, the GUI will not be able to find the sound files. Name your file **Fart_Machine.py** and choose **Save**. (Remember, as with the previous projects, if you used the guizero quick install method from chapter 1, then you must save the file into the same folder as the guizero.)

As with the Hello GUI project in chapter 1, we begin the program by importing the required modules (listing 3.1) from the guizero library. Modules contain prewritten functions that perform different tasks, such as displaying text, displaying images, and much more. In this context, the modules provide all the code needed to create the various elements and widgets of your GUI.

Listing 3.1 Importing the modules

```
# Imports

from guizero import App, Box, Picture, PushButton, Text
import winsound
```

You may recognize the `PushButton` and `Text` modules from chapter 1. These modules import the prewritten code to display text and create a button that you can push. In this GUI, we also use the `Picture` module to add a picture to the GUI. The `Box` object operates as an invisible container that holds the buttons and creates an ordered layout.

Next, we import the winsound module. This module is included in the Python installation files and provides basic access to the software in Windows OS that plays sound. You can read more about the features of winsound here, https://docs.python.org/3/library/winsound.html. (If you are using a macOS or Linux, then refer to section 3.5 on the last two pages for information about a compatible audio module.) Save your program before you move on.

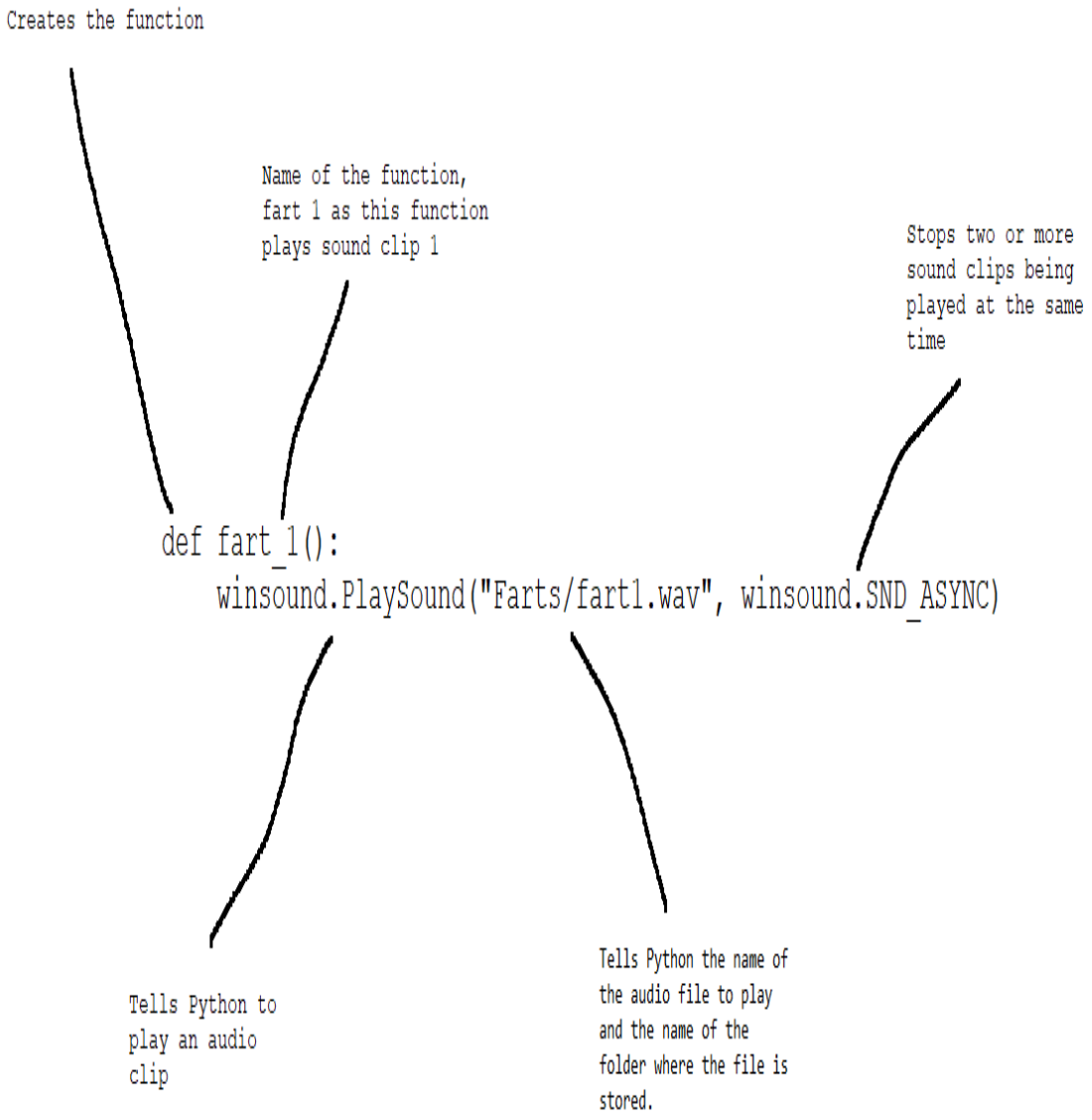
This program has no discrete variables, however, there are variables that create objects that hold the related code for the title, the image and the pushbuttons, so we will write them in the `#App` section of the program. Therefore, the variable section in this project is left blank. You may recall from chapter 1 that a variable is a location in the computer's Memory that stores data or information.

Creating the Farting Functions

Now let’s create the farts, I mean the, functions that will play the sound files that you collected earlier and stored in the **Farts** folder. If you completed the starter GUI in chapter 1, you may recall that a function is a reusable block of code that can be called and used anywhere within a program. Functions save time and make the program more efficient.

In this program, the function is assigned to a button which, when pressed, will call the function. On the button press, a Function runs, playing the audio clip (figure 3.4).

Figure 3.4 How the function works



Add the functions from listing 3.2 to your program code

Listing 3.2 Function code

```
# Functions

def fart_1():
    winsound.PlaySound("Farts/fart1.wav", winsound.SND_ASYNC)

def fart_2():
    winsound.PlaySound("Farts/fart2.wav", winsound.SND_ASYNC)

def fart_3():
    winsound.PlaySound("Farts/fart3.wav", winsound.SND_ASYNC)

def fart_4():
    winsound.PlaySound("Farts/fart4.wav", winsound.SND_ASYNC)

def fart_5():
    winsound.PlaySound("Farts/fart5.wav", winsound.SND_ASYNC)

def fart_6():
    winsound.PlaySound("Farts/fart6.wav", winsound.SND_ASYNC)
```

Each of the six functions nearly use the same code, which means you can copy the whole function and then paste the code underneath. Just remember to edit it and change the numbering and the audio file names. You just need to remember to update the number used in the name of the function and update the name of the sound file.

REMEMBER

- If you have a different folder name other than Farts/ to store your sound files, then you must replace the Farts/ folder name with the name of your folder.
- If you used a different file naming structure for your audio clips, then you need to use these names instead, replacing, for example, fart1.wav with your file's name.
- If you are using Linux or a MacOS look at section 3.5 on the last few pages of this chapter which covers how code the sound.

Closing the soundboard

Once you have added the code for the six functions that play the audio clips, you need to create a final function that will close the GUI when the Exit button is pressed. The close function uses the code `app.destroy()`, which sounds awful and destructive, but is simply a function that closes the F.A.R.T. app by closing the GUI window.

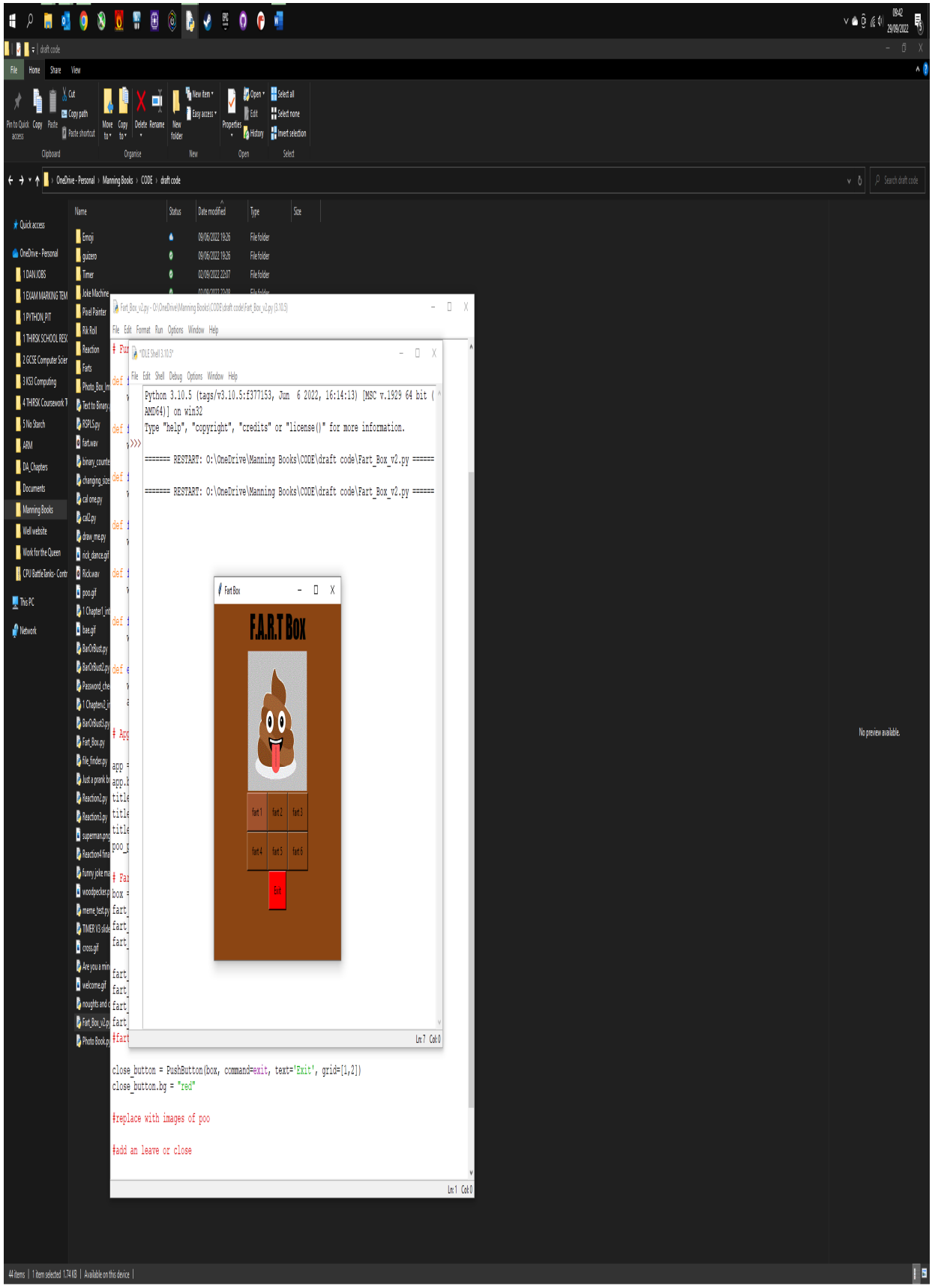
The `exit` function uses the same line of code as used in listing 3.1, except we are changing the sound to a Barney burp, and then you add the code to trigger the GUI to close. Add the code in listing 3.3 and, if you have yet to, choose the **File > Save** option from the main menu in your Python editor.

Listing 3.3 Function code to exit the GUI

```
def exit():
    winsound.PlaySound("Farts/burp.wav", winsound.SND_ASYNC)
    app.destroy()
```

Creating the GUI for the soundboard

Now that the functions are ready and added to the program, the next step is to create the F.A.R.T GUI soundboard.


draft code
File Home Share View
Pin to Quick access Copy Paste Cut Copy path Paste shortcut
Clipboard
Move to Copy to Delete Rename
Organize
New folder New item Easy access
New
Properties Open Edit History
Open
Select all Select none Invert selection
Select
OneDrive - Personal › Manning Books › CODE › draft code
Search draft code
Quick access
OneDrive - Personal
1 DAN JOBS
1 EXAM MARKING TEM
1 PYTHON_PIT
1 THRSX SCHOOL RESV
2 GCSE Computer Scier
3 KS3 Computing
4 THRSX Coursework T
5 No Starch
ARM
DA_Chapters
Documents
Manning Books
Well website
Work for the Queen
CPU BattleTanks- Contr
This PC
Network
Name
Status
Date modified
Type
Size
Emoji
09/06/2022 19:26
File folder
quizzes
09/06/2022 19:26
File folder
Timer
02/09/2022 22:07
File folder
Joke Machine
Pixel Painter
Rik Roll
Reaction
Farts
Photo_Box_Im
Text to Binary
RSPLS.py
fart.wav
binary_counte
changing_size
cal one.py
cal2.py
draw_me.py
nick_dance.gif
Rickv.wav
poo.gif
1 Chapter1_Int
bae.gif
BarOrBust.py
BarOrBust2.py
Password_che
1 Chapterv2_Int
BarOrBust3.py
Fart_Box.py
file_finder.py
Just a prank bo
Reaction2.py
Reaction3.py
superman.pn
Reaction4 fina
funny joke ma
woodpecker.p
meme_test.py
TIMER V3 slide
cross.gif
Are you a min
welcome.gif
noughts and c
Fart_Box_v2.p
Photo Book.p
Fart_Box_v2.py - O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py (3.10.5)
File Edit Format Run Options Window Help
IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py ======
====== RESTART: O:\OneDrive\Manning Books\CODE\draft code\Fart_Box_v2.py ======
Fart Box
F.A.R.T Box
fart 1
fart 2
fart 3
fart 4
fart 5
fart 6
Exit
Ln: 7 Col: 0
close_button = PushButton(box, command=exit, text='Exit', grid=[1,2])
close_button.bg = "red"
#replace with images of poo
#add an leave or close
Ln: 1 Col: 0
No preview available.
44 items | 1 item selected 1.74 KB | Available on this device

This is the interface that the user will use to interact with the soundboard. It consists of

- Setting the size of the GUI window
- Changing the background color of the GUI
- Adding the title: F.A.R.T. box
- Display a suitable image
- Six buttons that each individually trigger a different audio clip
- A button to close the soundboard

Adjusting the size and color of the soundboard

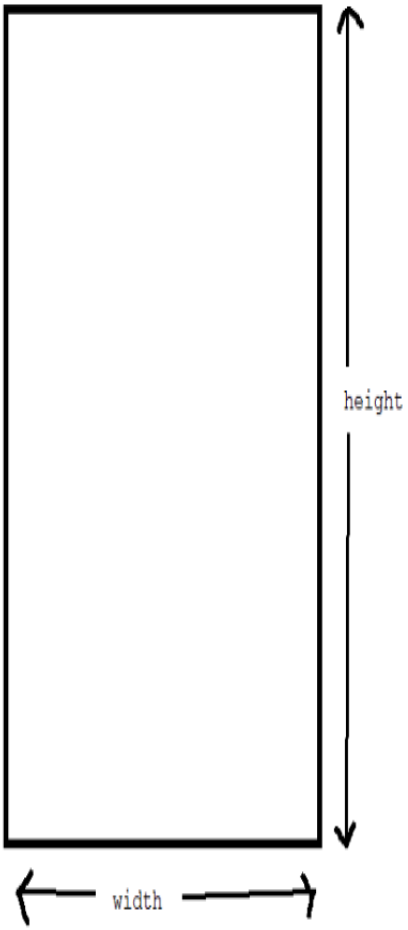
Each time we create a GUI, we use the App object. The App object is the main window of your GUI; in this program, the main App window holds all the features of our soundboard such as the text, the image, and the buttons. In chapter 1, the App is named using the code

```
app = App("Hello World",)
```

This code adds the text to the top left corner of the GUI window.

In this section of the program, we are going to display the name the GUI which is ‘F.A.R.T Box’ at eh top of the window. This title will grab the user’s attention and tempt them to press one of the buttons. We also need to set the size of the app window. The reason for setting the size of the app window is because the soundboard only has seven small buttons and a small image. Therefore, it does not require a large window, which would make it look bad and be a waste of screen space. Imagine drawing a small two centimetre by two centimetre picture on

Figure 3.5 Setting the sizes of the GUI window.



a large piece of paper, it would be a waste of the additional space on the paper! To set the size of the App window, (the GUI), we provide values for the width and height (figure 3.5).

These values are measured in pixels. We are going to set the size of the soundboard GUI to 350 x 400 pixels, that is a width of 350 pixels and a

height of 400 pixels. The code that sets the size is included after the App window's title. It does not matter whether you put the width or the height value first, line 1 in listing 3.4. Add the code from listing 3.4.

What is a pixel?

A pixel (picture element) is a single dot on a display or screen. Each pixel consists of three colors: red, green, and blue (RGB). Displays have different numbers of pixels, and this affects the quality or sharpness of the display. A standard high-definition (HD) monitor is 1920 x 1080 pixels, which means the screen contains 2,073,600 pixels; whereas a 4K display is 2160 x 3840, which is 8,294,400 pixels.

Listing 3.4 Setting up the GUI

```
# App

app = App("Fart Box", width = "350", height = "400") #A
app.bg = "saddle brown"
```

On the second line of code you will notice the connection between the color of poo and the background of the F.A.R.T. soundboard. Yep it's the background color of the GUI. The last line of code in listing 3.4 uses the code `app.bg = "saddle brown"`, like in chapter 1, to change the color to brown.

You have many color choices to use. The names and shades are all available on this website: https://wiki.tcl-lang.org/page/Color+Names%2C+running%2C+all+screens. Why not try out some other colors? But remember to keep them a shade of brown so that the color is in keeping with the project theme.

Adding text and an image to the soundboard

Now that you have created the main window of the GUI app, the next step is to add to the window the title of the GUI and the related image. To do this

we will create two variables, one to hold the title code and the second to hold the code for the image.

If you have read chapter 1, you may recall that a variable is a location in the computer's RAM that stores data or information.

Programs need to be able to find data quickly, so each variable you create must have a unique label. It is always a good practice to label the variable with a name that identifies the data or code being stored in the variable. For example, in this project the variable `poo_picture` stores the location of the poo image file.

To create a variable, simply type out its name, known as the label, followed by the equals sign; for example `poo_picture =`.

When creating the label for a variable, you can use more than one word, but if you do so, ensure that it has no spaces between the words. If you add spaces, then the variable will not work, and Python will present you with an error message when the program runs. To add a title and add the image, type the code from listing 3.5 into your program.

Listing 3.5 Code for the title and image

```
title_text = Text(app, "F.A.R.T. Box")   #A
title_text.text_size = 28   #B
title_text.font = “Impact”   #C
poo_picture = Picture(app, image="poo.gif")   #D
```

The first line of code in listing 3.6 declares the variable. Declaring is the proper name for creating and labeling a variable in a program. This variable has the label, `title_text`, which is assigned the Text widget. With this widget, we can add text to the GUI window. After the variable is created, the contents of the variable tell Python where to display the text. The word ‘app’ references the app window that you made in listing 3.4. So, the code is saying “display the title in the main GUI window; that is, the app window.”

On the second line, the code uses `.text_size` to set the size of the text. In this program, we set the size to 28, which makes it stand out and grab the viewers' attention. If you are creating your own soundboard, you can alter the text size as required.

Next, we set the font style of the text, using the font function `title_text.font = "Impact″ where` the font style is **Impact**. The font styles that you can choose depends on which operating system you are using and which fonts you like using.

The first line of code in listing 3.4 tells Python where to display the text and in a similar way, the last line of code tells Python where to display the picture assigning the image to the app window. We declare a variable labelled `poo_picture = Picture`, which assigns the `Picture` widget to the variable. The final part of this line of code (`image = "poo.gif"`) is the name of the image file.

The image dimensions are 160 pixels wide and 180 pixels high. If you decide to use your own image and replace the project image then ensure that it is a similar size to ensure that the picture fits within he app window. (You can alter the size of the GUI window to hold a larger image).

REMEMBER

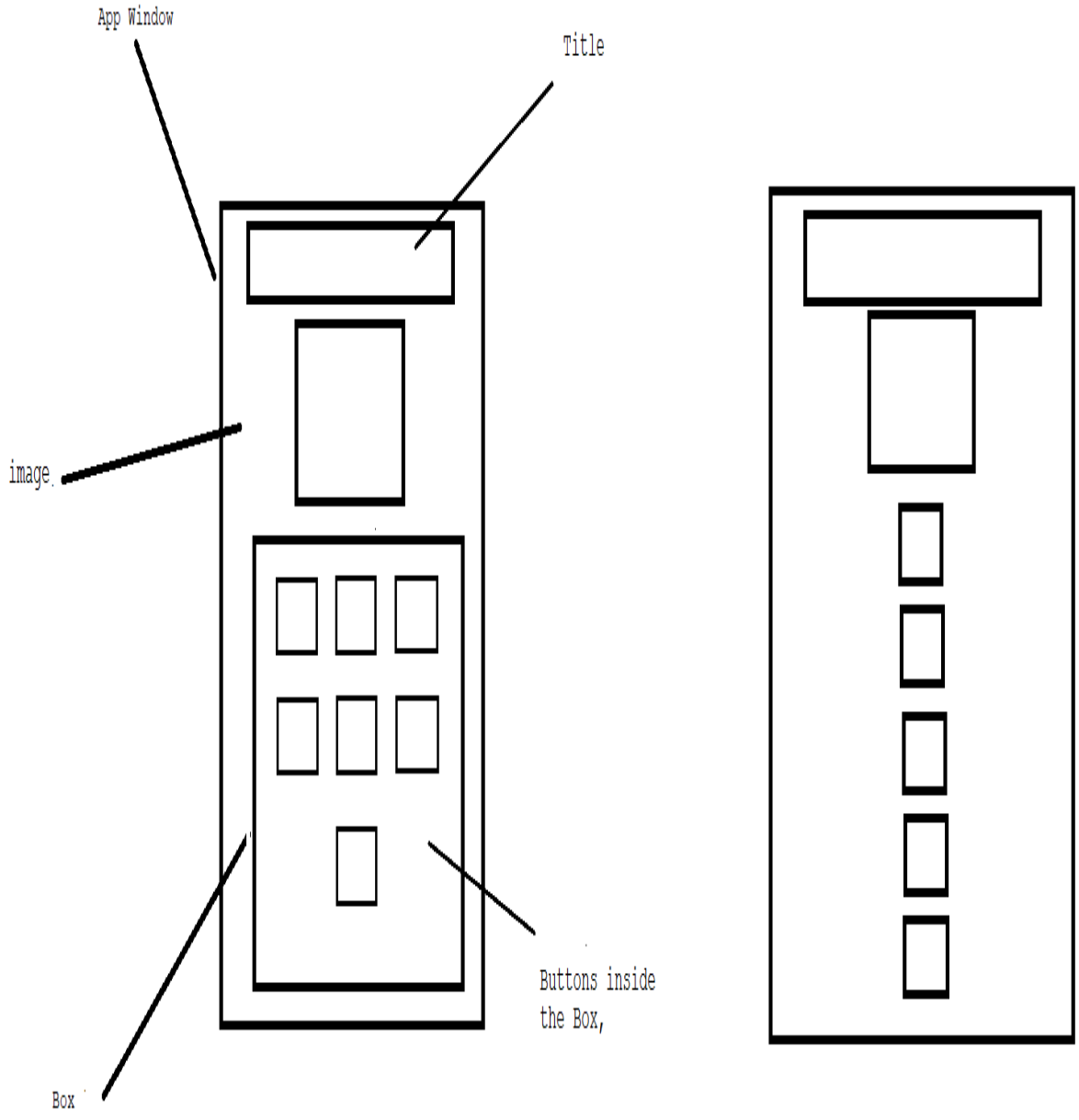
- The image files must be in .GIF or .PNG format.
- macOS only supports .GIF image files.
- If you use your own image, you need to replace the filename after `image =` with your image's filename.
- You must save the image file into the same folder as the **Fart_Machine.py** program.

Using the Box object to lay out the buttons

In guizero, the `Box` object is a container which can contain other widgets (figure 3.6). Unlike the App window, a box is invisible. Using the `Box` has the following benefit, it neatly lays out widgets and elements indie it. In this

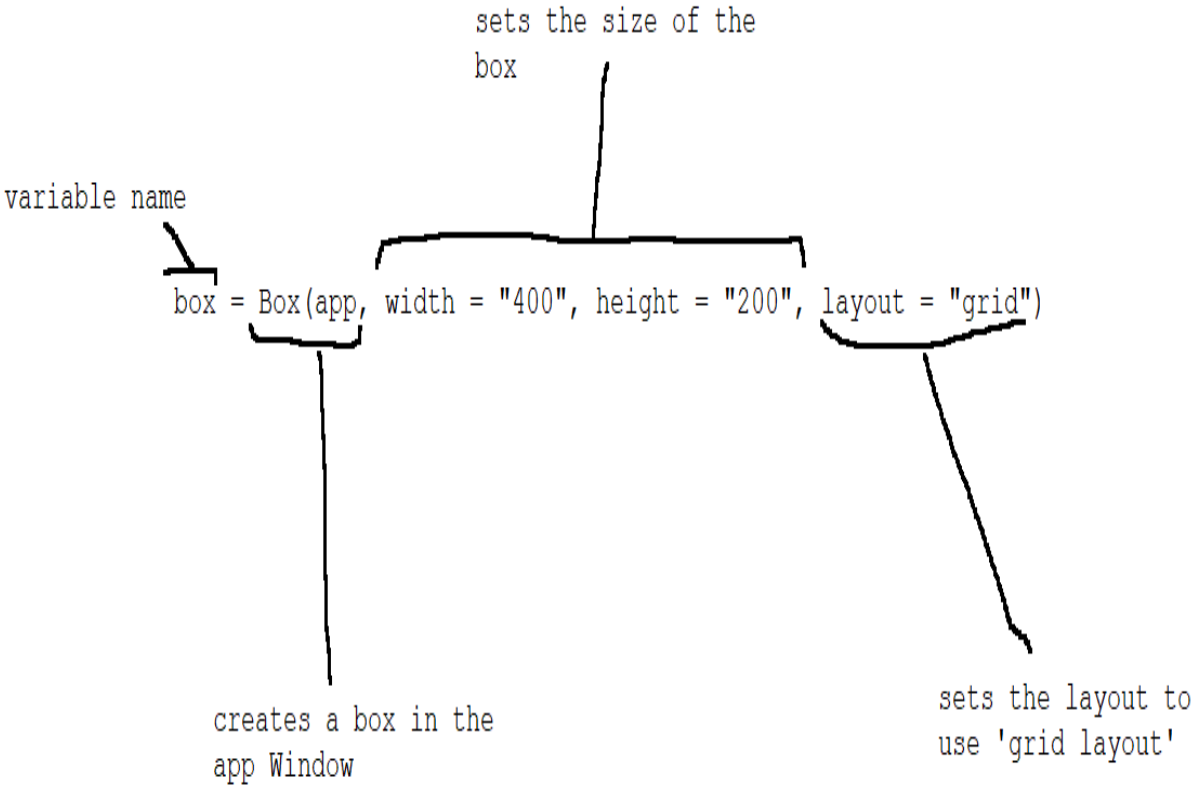
project it neatly lays out the buttons otherwise, the App window, just stacks the buttons vertically on top of each other.

Figure 3.6 Layouts when using and not using the Box



In this project we will create and use a `Box` to hold the six buttons that each trigger a Function and play a fart audio clip, figure 3.7. This means that the buttons are grouped together, and they are easy to arrange into a neat and tidy order. Add the code in listing 3.7; it starts with a variable labelled `box`. By now you will be noticing that variables are a staple part of programming.

Figure 3.7 Code to create and configure the Box within the app window



Listing 3.6 Code for the title and image

```
box = Box(app, width = "400", height = "200", layout = "grid")
```

Then we add the code to configure the box. The first word, `app`, shows that the box is part of the app window. Then we set the size of the `Box` like you

previously did in listing 3.4, where we set the width and height of the GUI window. Here we are setting the width and height of the box, which you will notice is smaller than the app window. This is so the box fits neatly inside the App window.

In chapter 1, when we added the widgets, we simply added them to the GUI and the program automatically took care of laying them out. In this project, we want to control the layout, so we use a feature called grid layout and add it to the `box` variable using the code

```
layout = "grid”
```

This layout offers more control over where the buttons are displayed. With the grid layout, you can position widgets into a virtual grid inside the box.

Imagine dividing the box into equally sized blocks, and then using an x and y value to locate one of the grids, in a similar way that you do when you play battleships or if you have ever used a cell reference in a spreadsheet to locate a specific cell. At school, math teachers used to have a phrase to remember the order of the x and y coordinates as “along the corridor and then up the stairs,” where the x value is first and horizontal (along the corridor), and the y value is second and represents to vertical values (up the stairs).

The grid layout uses an x and y value (the x and y are called a coordinate) to place the button in a specific location in the box. For example, button 1 has the following coordinate, [0, 0], where the first 0 is the x value and the second 0 is the y value. This coordinate places the first button in the first line in the box and on the left. Figure 3.8. Where it can be confusing is that [0,0] is located at the top left-hand side of the grid and not the bottom left-hand side as with graphs.

Figure 3.8 Coordinates when using grid layout.

Y/X	0	1	2
0	0,0	1,0	2,0
1	0,1	1,1	2,1
2	0,2	1,2	2,2

The second button uses the coordinates [1,0], which means the x value is 1 and the y value is 0. The x indicates the second block and the y indicates that you're still on the first line. Let's set up the first button.

Coding the buttons that trigger the fart sounds

Now to add to the program the code to create the buttons that when clicked, play the fart sound (or another sound if you are building your own soundboard). The line of code in listing 3.7 assigns the Push Button widget to the GUI and identifies where to display the button in the box.

The essential feature of the code is the *command*, which is passed to the `PushButton` widget and is used to call one of the functions from listing 3.2. The called function then plays the fart audio clip. Commands make it possible for the user to interact with the GUI; for example, when you press the button, the command calls the function which makes the GUI respond creating interaction. In the chapter 1 GUI, the command made the GUI say Hello to you. In other chapters, a command will make it possible for you to draw. In this chapter, the commands are linked to

- the functions that play the sound clips and
- the function that closes the GUI.

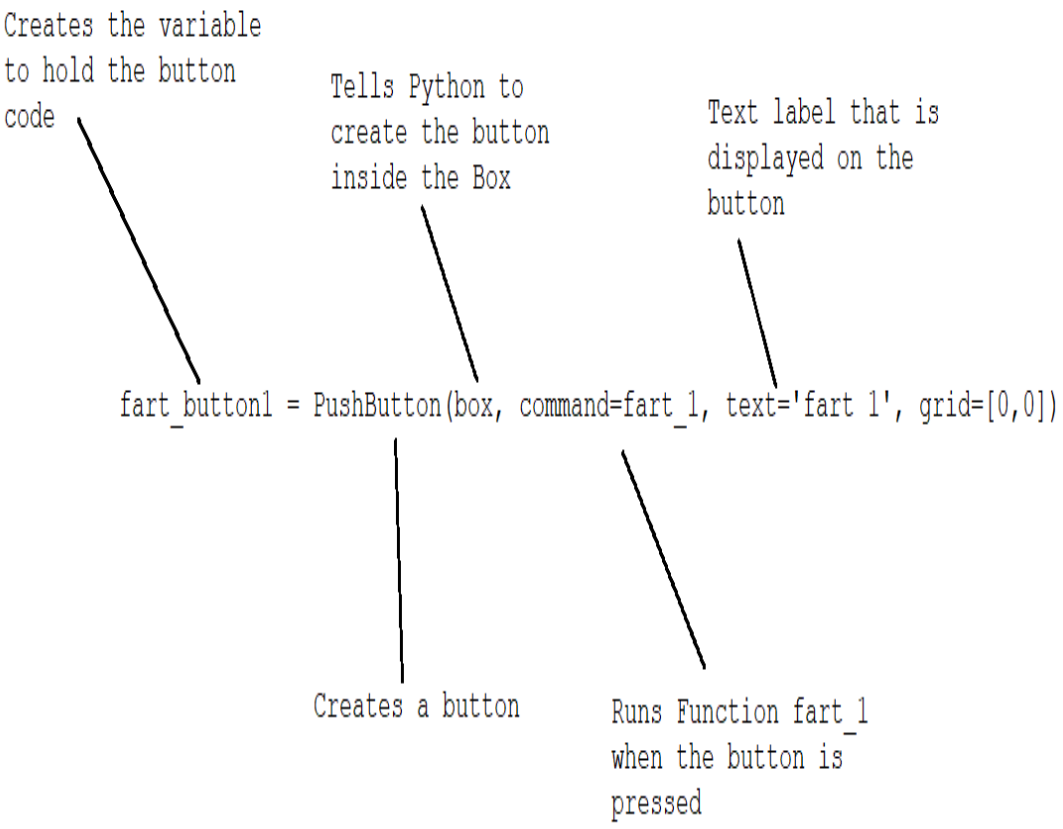
Add the code in listing 3.7 to your program.

Listing 3.7 Code to add the first button

```
fart_button1 = PushButton(box, command=fart_1, text='fart 1',
grid=[0,0])
app.display()
```

In chapter 1 we gave the button a label so the user knows what it does. Do the same thing here and label the button with fart 1. You can label the button with any text, so you could use the name of the fart instead? Such as ‘squeaker’, ‘loud and proud’, ‘silent and deadly’, ‘Trump’. Then use the grid code to align the button’s layout (figure 3.9).

Figure 3.9 How to create a fart button



At this point you can test your GUI by including the line `app.display()`. This means that you can see if the program is working correctly before you add the other buttons. It will be easier and quicker to find and correct errors in a program with fewer lines of code to check.

Do not worry if the program fails first time. With a longer program, it is likely that you may have made some errors. Commonly, errors are syntax errors. A syntax error is like a grammar error, only more serious. With human language, people can generally understand you even if you made grammatical mistakes. Programming languages, such as Python, expect you to use proper syntax for the program to work correctly.

Press **F5** on the keyboard. Python prompts you to save the program file. Choose **OK**. The program will save, and then the program will run, and the F.A.R.T. GUI soundboard will be ready. The GUI should display the title, the poo related image, and only one button, which when pressed plays fart sound clip 1.

If your program fails or does not run as expected, check for these common errors:

1. Check that your computer audio is not on mute.
2. Is the volume turned up?
3. Are the audio clips that you are using .wav files?
4. Did you create a folder called **Farts**, and are all the audio files placed inside this folder?
5. Is your **Fart_Machine.py** program saved in the same folder that holds the **Farts** folder? (Not saved inside the Farts folder)
6. If you named the folder and audio clips differently, then have you used the correct names in your program code? For example, if your folder is named Burps and you first audio file is called burping1.wav, then the code in the functions should be
   `winsound.PlaySound("`**Burps/burping1.wav**`", winsound.SND_ASYNC)`
7. When creating the buttons, have you used the same function name in the command?
8. Have you used the correct case? Some syntax has capital letters in the middle of a word. For example, TextBox, PushButton.
9. Have you used straight quotation marks ("") for each string?
10. Did you spell all the words correctly?

Once the program is working, you can then delete the `app.display()` from the last line and then add the additional buttons from listing 3.8.

Listing 3.8 Code for the additional buttons

```
fart_button2 = PushButton(box, command=fart_2, text="fart 2",
grid=[1,0])
fart_button3 = PushButton(box, command=fart_3, text='fart 3',
grid=[2,0])
fart_button4 = PushButton(box, command=fart_4, text='fart 4',
grid=[0,1])
fart_button5 = PushButton(box, command=fart_5, text='fart 5',
grid=[1,1])
fart_button6 = PushButton(box, command=fart_6, text='fart 6',
grid=[2,1])
```

Creating a button to close the F.A.R.T. soundboard

The last stage of the program is to add the code that creates a button that when pressed closes the disgusting soundboard. This is important, as the user may have had enough of the fart sounds and want to leave! Also, it is good GUI practice to include a method to cleanly close the GUI. In chapter 1, to leave the ‘Hello GUI’ program, you chose the x in the top right corner of the GUI window. This used the standard close a window feature built into the Windows OS. Creating a dedicated close or exit button means that you can customize it. Add the final code from listing 3.9 to your program code.

Listing 3.9 Adding a button to exit the GUI

```
close_button = PushButton(box, command=exit, text='Exit', grid=
[1,2])   #A
close_button.bg = “red”   #B
app.display()   #C
```

You will notice that the code for the `close_button` is the same code that we used in listing 3.7 where we created the buttons to trigger the audio clips. In listing 3.9, we use the variable `close_button` to store the information related to the button including which function to run when the button is pressed. The

button appears in the box, so the element `Box`, is included after the first bracket.

Next, assign the exit function that you created to the command using the line `command = exit`. This tells the button to run the function that closes the GUI when the user presses it. Then we name the button. Here we have called it Exit. Lastly, use the grid alignment to display the button in the middle of the very last line, underneath the other buttons.

The next line of code changes the color of the button to red, so that the user can see that it is not an audio clip. Also, the color red acts as a warning that if they press the button something else will happen, the program will stop running and the GUI will close. Finally on the last line you add the code `app.display()` to run the GUI window.

Running and testing your GUI program

That is it, you have now built a disgusting FART soundboard, well done. Now to test that it works correctly. Press **F5** on the keyboard. Again, you may be prompted to save the program file. Choose **OK** and the program will save, and then the program will run, and the F.A.R.T. GUI soundboard will be ready. Turn up the volume and press a button. Go on, I dare you. If you encounter any errors this time, then refer to the error questions listed in the paraph just after code listing 3.7.

Listing 3.10 Final code

```
# Imports

from guizero import App, Box, Picture, PushButton, Text
import winsound

# Variables

# Functions

def fart_1():
    winsound.PlaySound("Farts/fart1.wav", winsound.SND_ASYNC)
```

```
def fart_2():
    winsound.PlaySound("Farts/fart2.wav", winsound.SND_ASYNC)

def fart_3():
    winsound.PlaySound("Farts/fart3.wav", winsound.SND_ASYNC)

def fart_4():
    winsound.PlaySound("Farts/fart4.wav", winsound.SND_ASYNC)

def fart_5():
    winsound.PlaySound("Farts/fart5.wav", winsound.SND_ASYNC)

def fart_6():
    winsound.PlaySound("Farts/fart6.wav", winsound.SND_ASYNC)

def exit():
    winsound.PlaySound("Farts/burp.wav", winsound.SND_ASYNC)
    app.destroy()

# App

app = App("Fart Box", width = "350", height = "400")
app.bg = "saddle brown"
title_text = Text(app, "F.A.R.T Box")
title_text.text_size = 28
title_text.font = "Impact" #
poo_picture = Picture(app, image="poo.gif")

# Fart buttons
box = Box(app, height = "200", width = "350", layout = "grid")

fart_button1 = PushButton(box, command=fart_1, text='fart 1',
grid=[0,0])
fart_button2 = PushButton(box, command=fart_2, text="fart 2",
grid=[1,0])
fart_button3 = PushButton(box, command=fart_3, text='fart 3',
grid=[2,0])
fart_button4 = PushButton(box, command=fart_4, text='fart 4',
grid=[0,1])
fart_button5 = PushButton(box, command=fart_5, text='fart 5',
grid=[1,1])
fart_button6 = PushButton(box, command=fart_6, text='fart 6',
grid=[2,1])
```

```
close_button = PushButton(box, command=exit, text='Exit', grid=
[1,2])
close_button.bg = "red"
app.display()
```

Two other things to try

As with chapter 1, this section of the chapter introduces some additional edits for you to add to your program. They are basic; however, they can make the GUI look and feel very different. You will come across theses edits in later chapters and if you wish, you can make use the edits in your own future projects.

Changing the background color of the buttons

One easy edit to make is to change the color of the buttons. For example, you could color code the audio buttons in terms of, the browner the color the more bombastic the fart noise is! This would make it easier for the user to know which buttons would make a louder fart. For example, a lighter shade of brown could be a squeaky fart, but the dark brown is loud. Coloring the buttons is also a useful feature for future projects.

To edit the color, you simple create a new variable in the App section with the same name as the button variable, then use `.bg= color` followed by the color you want to use. For example, the first button is labeled `fart_button1`, so the code to change the color of this button is

```
fart_button1.bg = "sienna"
```

And the code for button 2 is

```
fart_button2.bg = "saddle brown"
```

I recommend that you add the line of code for the color underneath the code for the buttons. This will make it easier to keep track of.

```
fart_button1 = PushButton(box, command=fart_1, text='fart 1',
grid=[0,0])
```

```
fart_button1.bg = "sienna"

fart_button2 = PushButton(box, command=fart_2, text="fart 2",
grid=[1,0])
fart_button2.bg = "saddle brown"
```

Adding more buttons and sounds

If you can't get enough of the fart sounds, or your soundboard requires many more buttons, then you can add more buttons. This involves three stages:

1. Collect or make the additional audio sound files that you want to use in your GUI. Ensure that these files are .wav and save them into the correct folder. (In this project the folder is named Farts.)
2. Create a new function for each of the additional sound files. This uses the same code structure used in listing 3.2. In the `#Function` section of the program, add the new Function, remember to use a different name for the Functions and change the filename of the new sound clip, shown in italics below.

```
def fart_1():
    winsound.PlaySound("Farts/fart1.wav", winsound.SND_ASYNC)
```

3. Add the code to trigger the additional button. In the `#App` section under the last button, which was `fart_button6`, add a new line and type in the code for the new button:

```
fart_button7 = PushButton(box, command=fart_7, text='fart 7',
grid=[2,1])
```

Remember to change the name of the button and link the command to the new function. You will also need to add the change the grid numbers. Otherwise, when the program runs, Python will try to create two buttons in the same place. To make the new buttons fit, you may need to extend the height of the box that holds the button. Do this by adjusting the width value (shown in bold) in this line of code:

```
box = Box(app, width = "400", height = "200", layout = "grid").
```

Statements of fact

- A package is a collection of different Python modules,
- The `Text()` object displays text that cannot be edited while the GUI is running. It is useful for titles, labels, and instructions.
- Objects are a bundling of variables and functions so they work as a single unit.
- The `Picture()` object displays images in the GUI. The default file formats are .GIF and .PNG. macOS only supports GIF format.
- Commands add interactivity between the user and the GUI interface.
- A window is the main app window of the GUI. It holds the parts of the GUI, widgets, text, and buttons.
- A `Box` is an invisible container that, like a window, can hold widgets, text, and buttons. Editing a `Box` does not edit the main GUI window.
- A grid layout uses an X and Y coordinate to position a widget in a precise location within the GUI.

Using Simple Audio

If you are using a Linux or Mac OS operating system, then you will need to use an alternative sound module to winsound. This is because winsound is only packaged and available with Windows OS. We can use simple audio instead.

This module is described as providing cross-platform, dependency-free audio playback capability for Python 3 on MacOS, Windows, and Linux. So, it provides an alternative if you are not using Windows OS. You can read more about simpleaudio here, https://pypi.org/project/simpleaudio/

INSTALLATION

Download and install of simpleaudio can be completed in one command from the Terminal window. Open you Terminal windows and enter the code.

```
pip install simpleaudio
```

Wait for the program to download and install, and you are ready to make disgusting FART sounds, well, your GUI is! The program code, as the name suggests, is simple to use, first you import the audio module using the line, `import simpleaudio as sa`. Then you create an instance of the audio file and assign it to a variable.

You can now control when the sound plays using the code, `play_obj = wave_obj.play()`. The program needs to wait for the sound to play through before moving not the next line of code, to do this use the code `play_obj.wait_done()`

Listing 3.11 Alternative OS listing, Final code

```
# Imports

from guizero import App, Box, Picture, PushButton, Text
import simpleaudio as sa

# Variables

# Functions

def fart_1():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart1.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

def fart_2():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart2.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

def fart_3():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart3.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

def fart_4():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart4.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()
```

```
def fart_5():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart5.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

def fart_6():
    wave_obj = sa.WaveObject.from_wave_file("Farts/fart6.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

def exit():
    wave_obj = sa.WaveObject.from_wave_file("Farts/burp.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()
    app.destroy()

# App

app = App("Fart Box", width = "350", height = "400")
app.bg = "saddle brown"
title_text = Text(app, "F.A.R.T Box")
title_text.text_size = 28
title_text.font = "Impact" # can be any of the following p 16
SEE GUIDE
poo_picture = Picture(app, image="poo.gif")

# Fart buttons
box = Box(app, height = "200", width = "350", layout = "grid")
fart_button1 = PushButton(box, command=fart_1, text='fart 1',
grid=[0,0]) # use text
fart_button2 = PushButton(box, command=fart_2, text="fart 2",
grid=[1,0])
fart_button3 = PushButton(box, command=fart_3, text='fart 3',
grid=[2,0])

fart_button4 = PushButton(box, command=fart_4, text='fart 4',
grid=[0,1])
fart_button5 = PushButton(box, command=fart_5, text='fart 5',
grid=[1,1])
fart_button6 = PushButton(box, command=fart_6, text='fart 6',
grid=[2,1])

close_button = PushButton(box, command=exit, text='Exit', grid=
[1,2])
close_button.bg = "red"
```

```
app.display()
```

Appendix A. Installing Python

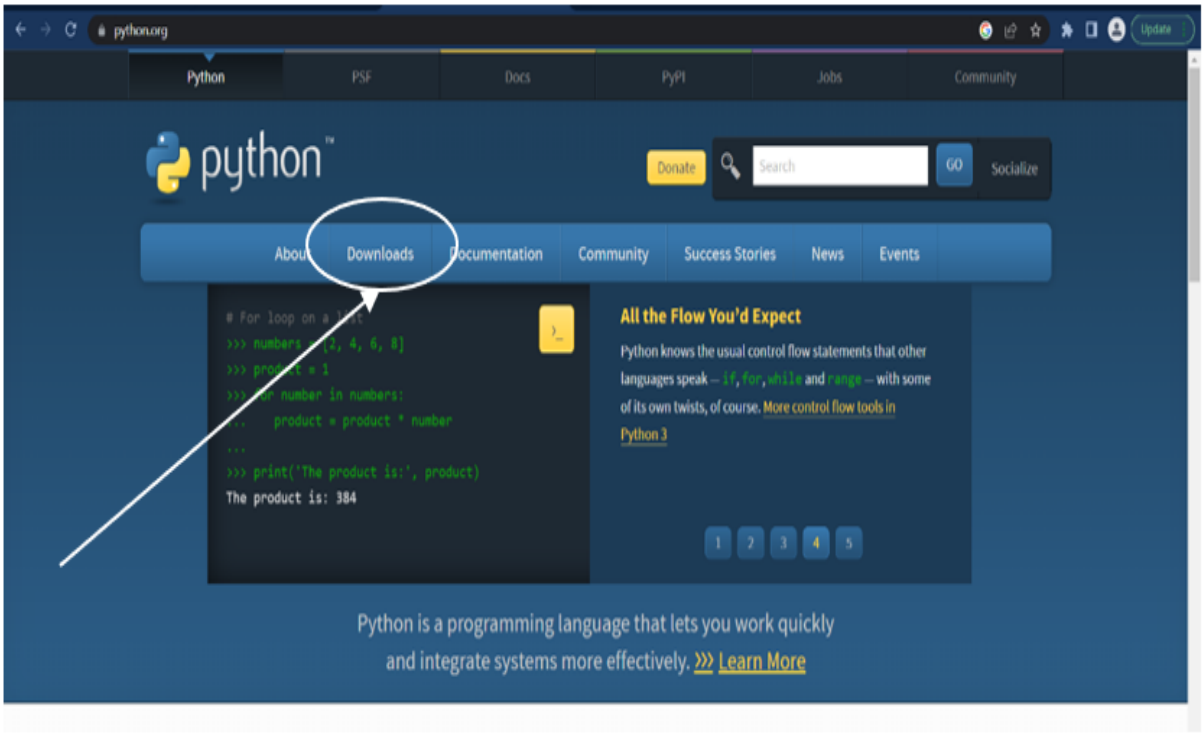
This appendix walks you through downloading, installing, and testing Python on a Microsoft Windows computer. If you are using macOS or Linux, or another operating system then check out appendix B which provides the instruction to do this.

Python is a free programming language. This means that you can use Python for free to create, edit, and share your programs. Python is popular because you can use it to write programs for many different applications and purposes. However, its real strength lies in the simplicity of using the code compared to other languages. You can learn more about Python in appendix D.

To code with Python, you need an integrated development environment (IDE). An IDE is an interface that you can use to write programs, test them, and run them on your computer. The Python installation includes its own IDE, called IDLE, which is pronounced idol. There are other IDEs available; an overview of the alternatives is covered in the last section of this appendix.

Getting Python

Let's begin. The first step is to head over to the Python website and download the Python install program so that you can install it on your computer. Open your web browser, (like Chrome, Safari, Edge etc.) and go to www.python.org. This will take you to the home page for Python. Over time newer versions of Python are released, so remember to check this site regularly if you wish to upgrade your version of Python at a later date.



Downloading Python

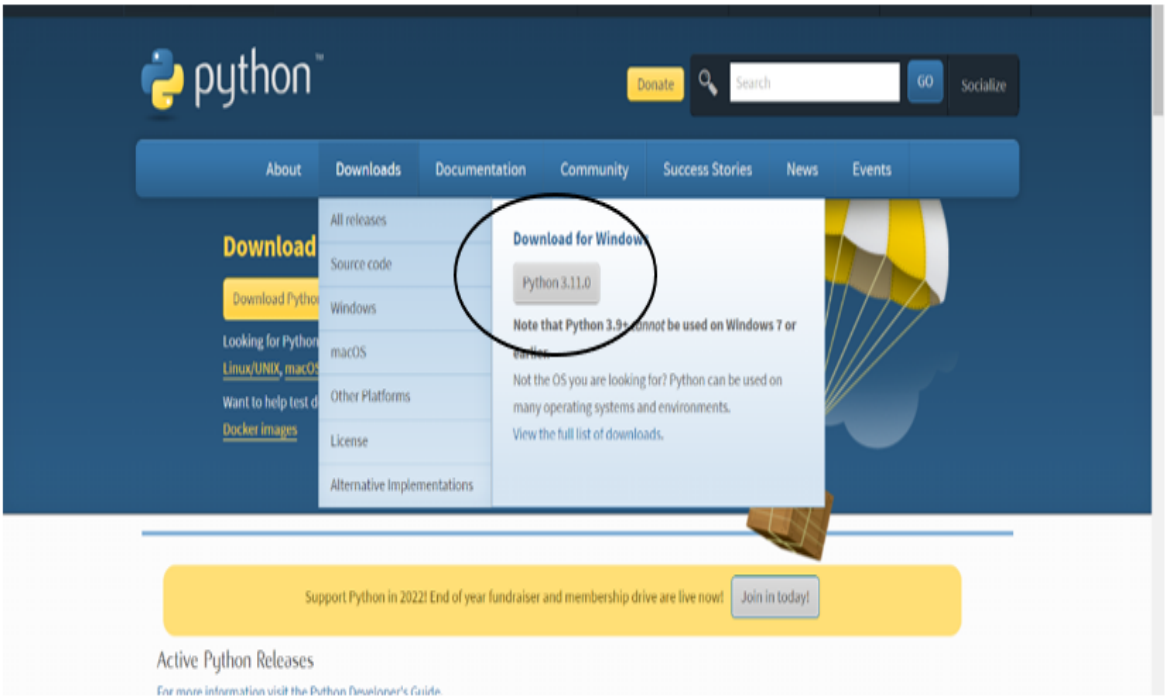
On the Python home page, find the navigation menu at the top of the website and select the **Downloads** button. This takes you to the downloads page where you can select which version of Python you want to install.

When the Downloads page opens, the site automatically displays the most up to date version of Python for the Microsoft Windows operating system. Click the button to download the latest version.

Note

If you are using Mac or Linux, click the link underneath the download button to navigate to the appropriate page. (Remember to check out appendix B for a detailed guide on installing on macOS and Linux.)

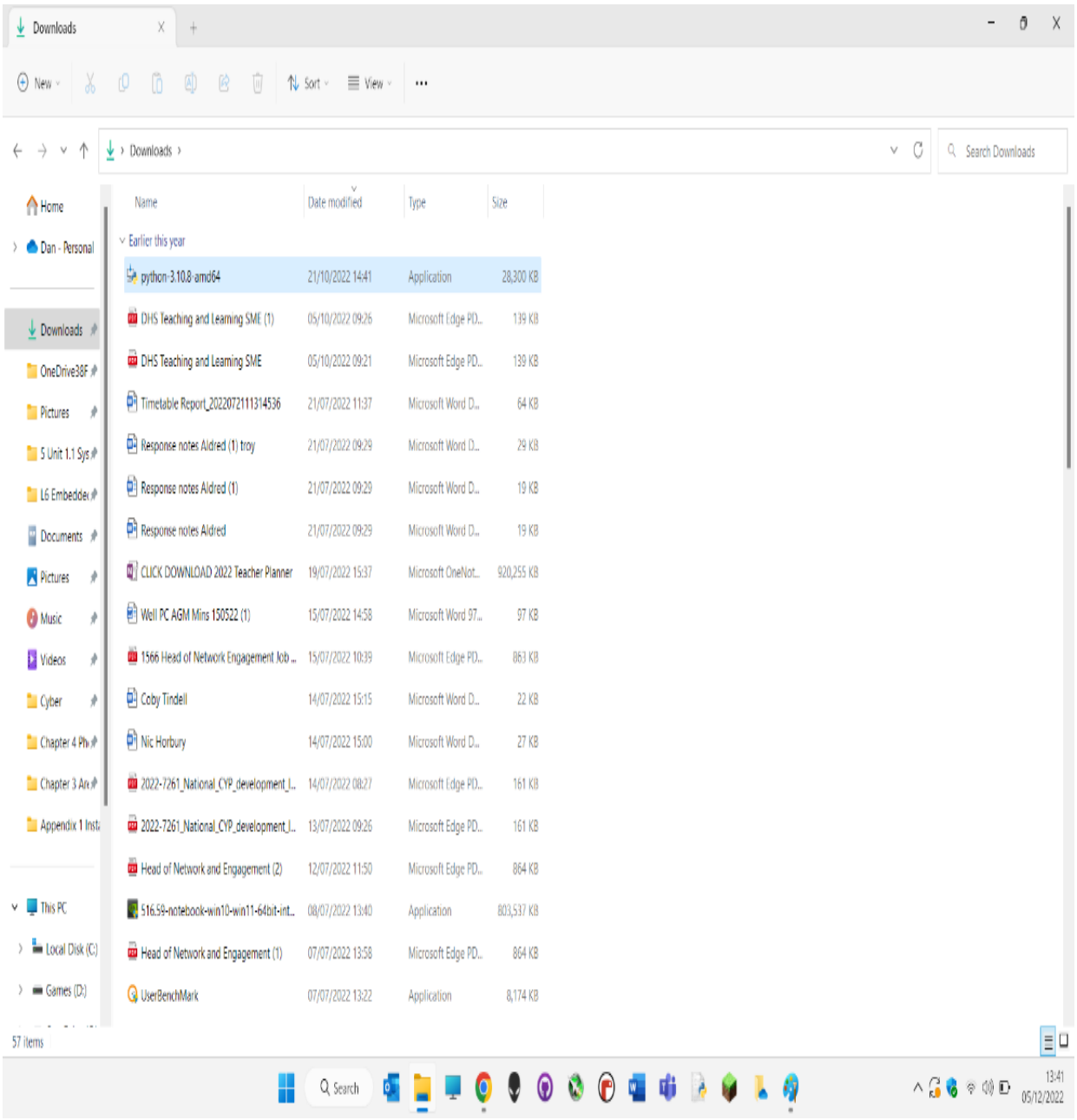
To begin downloading the Python installation file, click the download button. It’s okay if the version number is different from what you see here.



nstalling Python

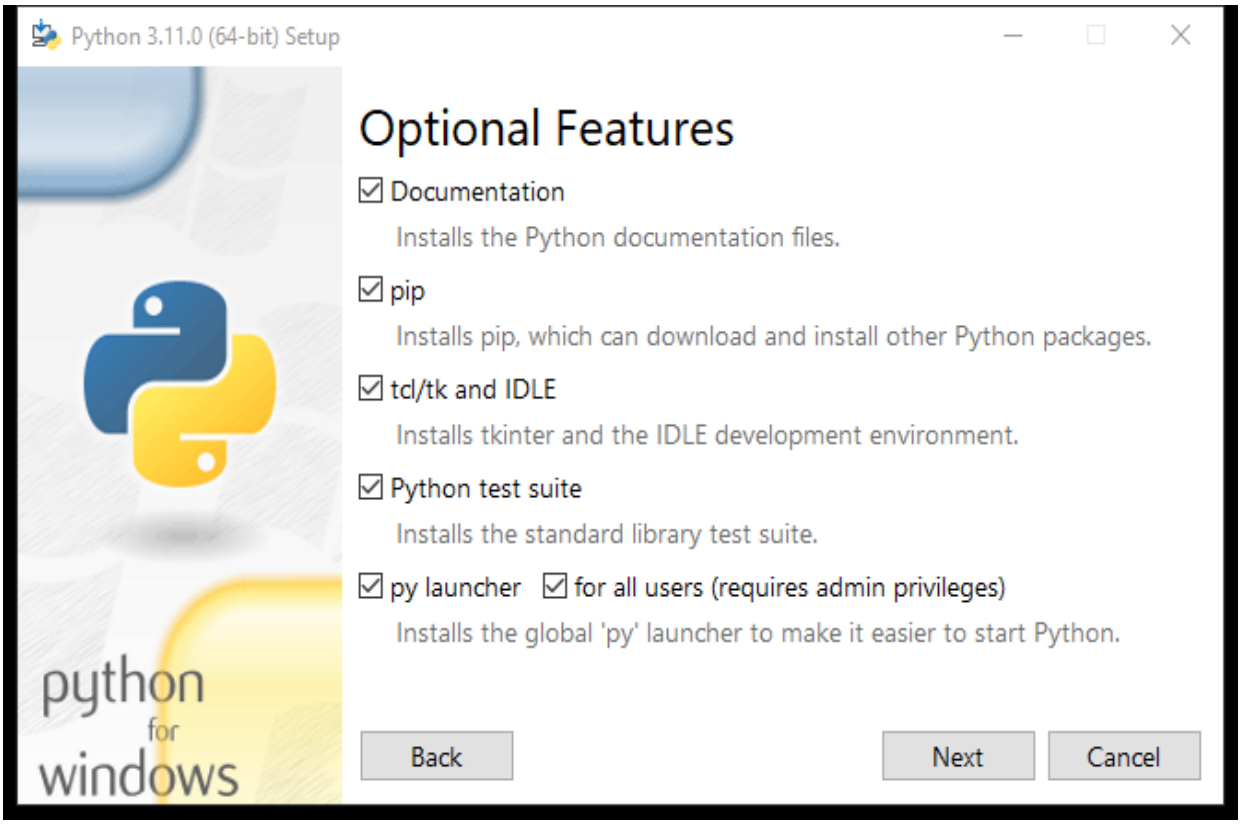
Once the Python installation file downloads, on your computer, use the folder browser and head to the folder where the Python installation file has been downloaded to. In Windows, unless you changed the download location, this is probably your Downloads folder.

1. Locate the downloaded Python file,

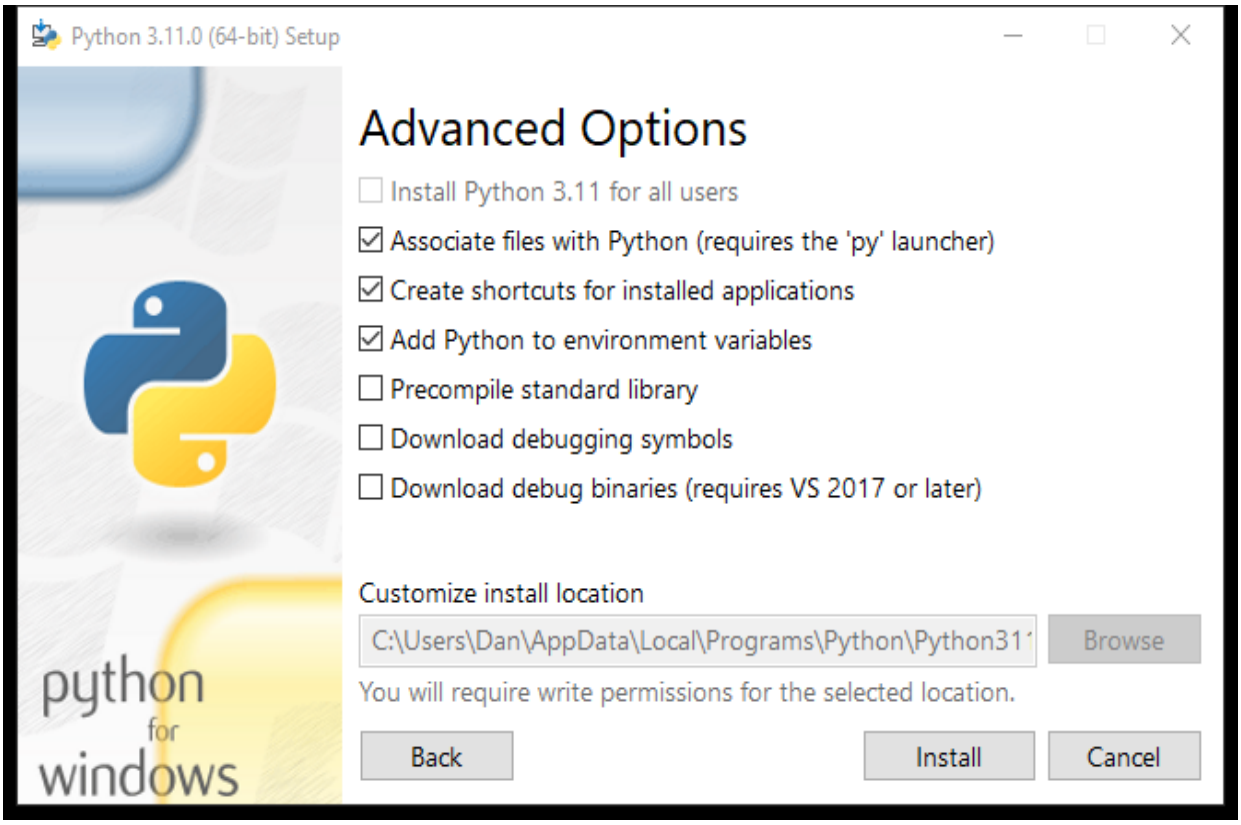


2. Once you've found the file, select it (usually double-click) to start installing Python . You will be presented with a menu window that has several options. Select the second option: **Customize installation**.

3. The Python installer loads the **Optional Features** window. All these options are ticked. This is correct.



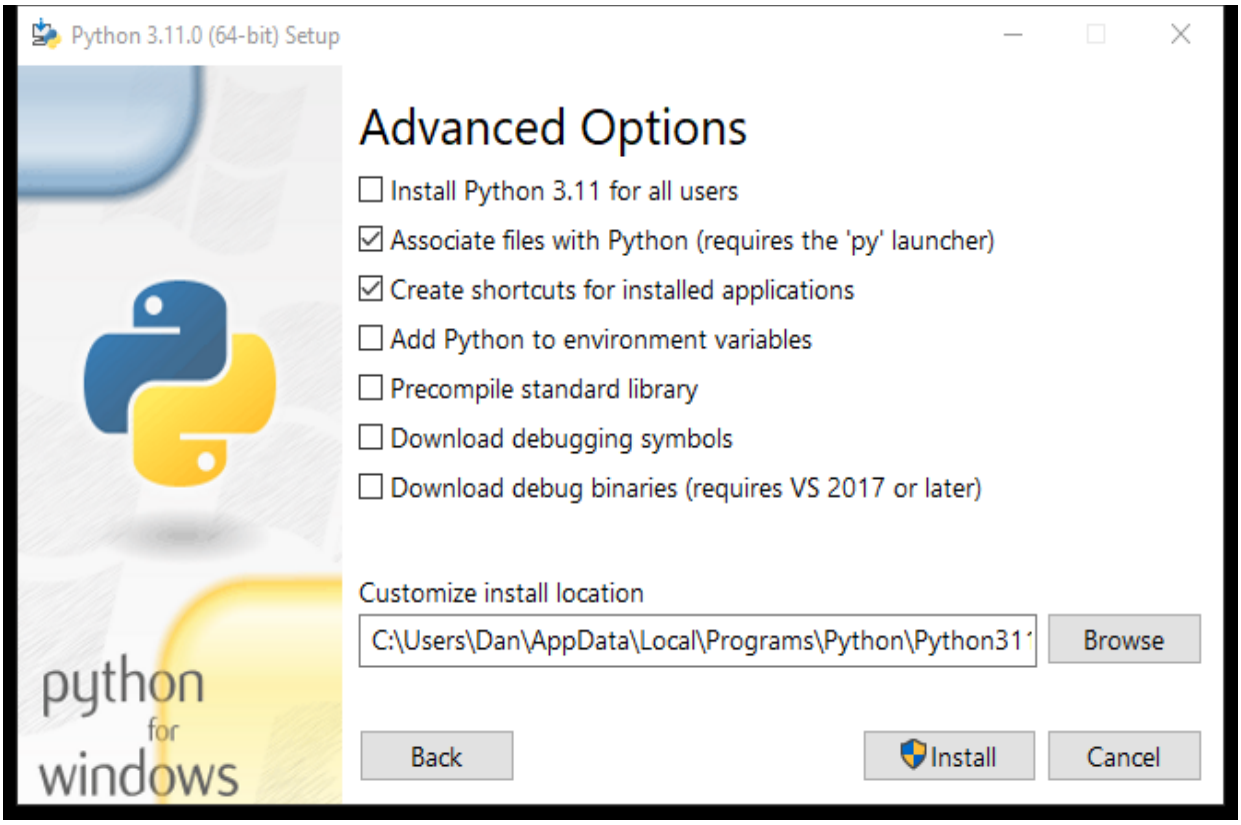
4. Press **Next** to move to the **Advanced Options** window. This window presents you with several options, some of which may already be ticked. Leave any ticked options, do not untick them. There is an additional option that you may want to select, and one option that must be selected.



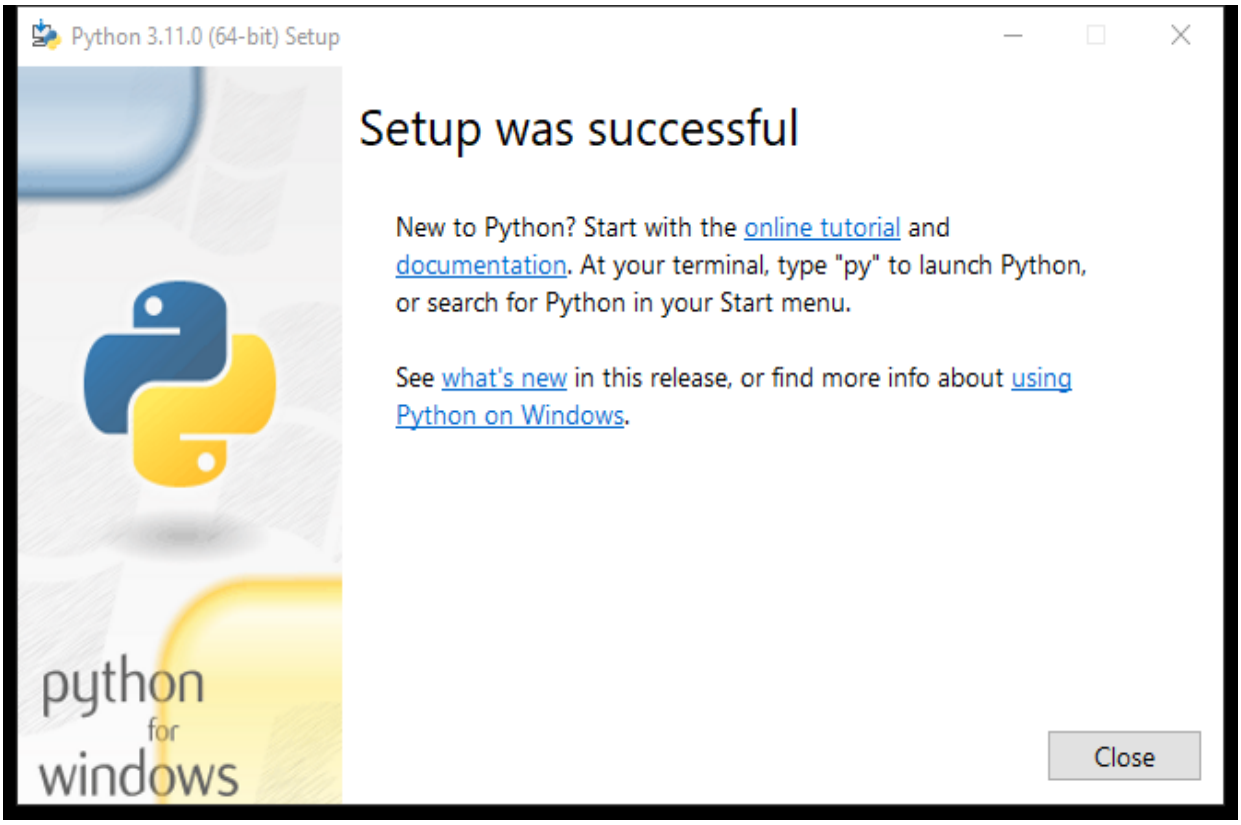
a. The first option, the first box, is to decide whether you want to **install Python for all users** of the computer. Say for example that you are using someone else's computer. They might not want Python installed. In that case, leave this option unticked. If the computer is a shared computer and used by others who may also want to build the projects in the book, then they will need Python installed, so you can tick this button.

b. The second option, the fourth box is **Add Python to environment variable**. This option ***is essential and must be ticked***. The option ensures that your Windows operating system can locate Python and all Python's associated files. Select the **Add Python to environment variable** option.

5. After you have selected these two options you are ready to install Python. Press the **Install** button. Unless you change the location, this will install Python onto the C drive of your computer



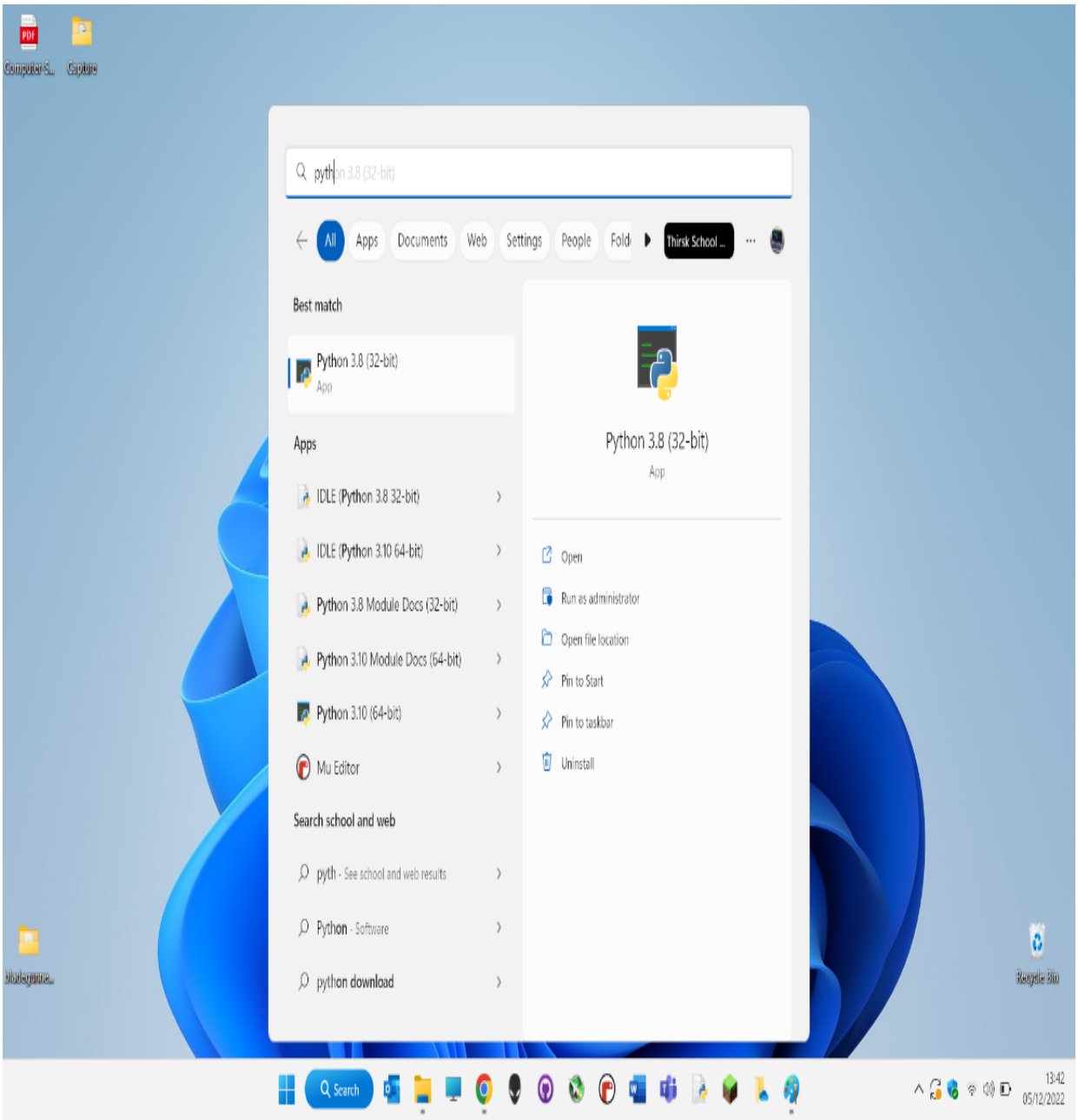
6. You will be presented with the final window that asks you to confirm that you want to install Python. Select **Yes** to begin the installation.
7. On completion of the installation, the next window confirms that the installation was successful. Well done! You have installed Python. Now to test it.



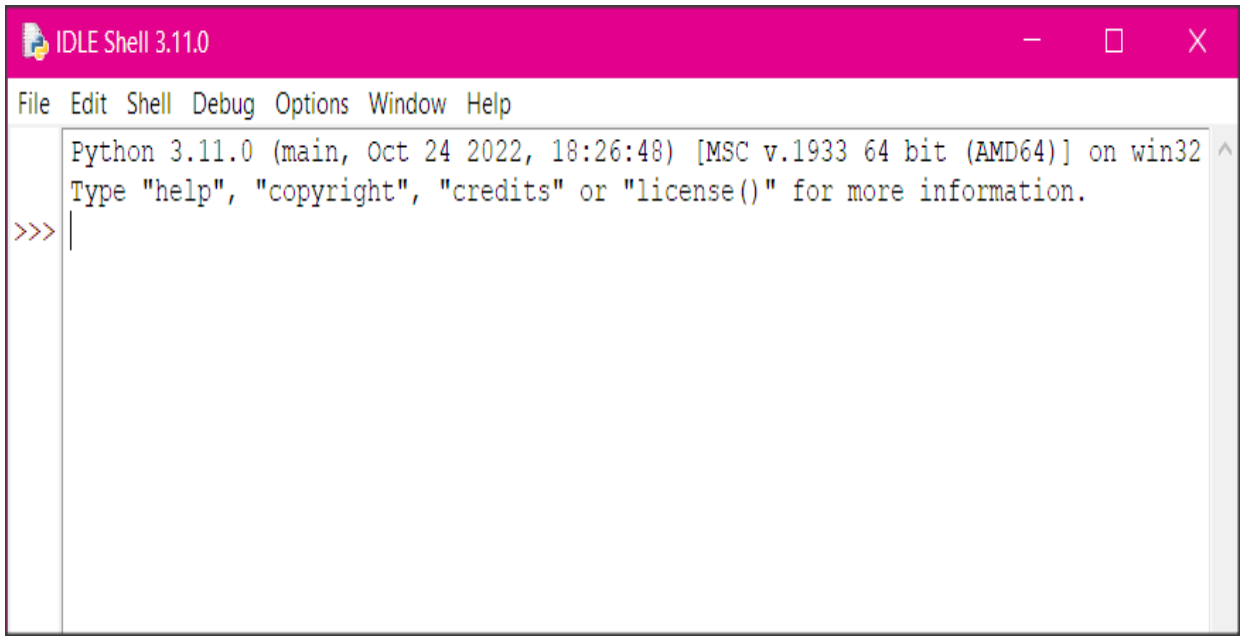
Testing your installation of Python

Now to test that Python has installed correctly and is working.

1. Open the Windows start menu. If you are using Windows 10 then you will see that Python has been added to the top of the menu under the **Recently added** section. If you are using Windows 11 then simply type Python into the search bar.



2. Double-click the IDLE program. Python loads in an **IDLE Shell** window. The shell window is an interface in which you can enter Python code one line at a time. You write the line of code, press enter, and then the code runs, displaying the output in the shell window.



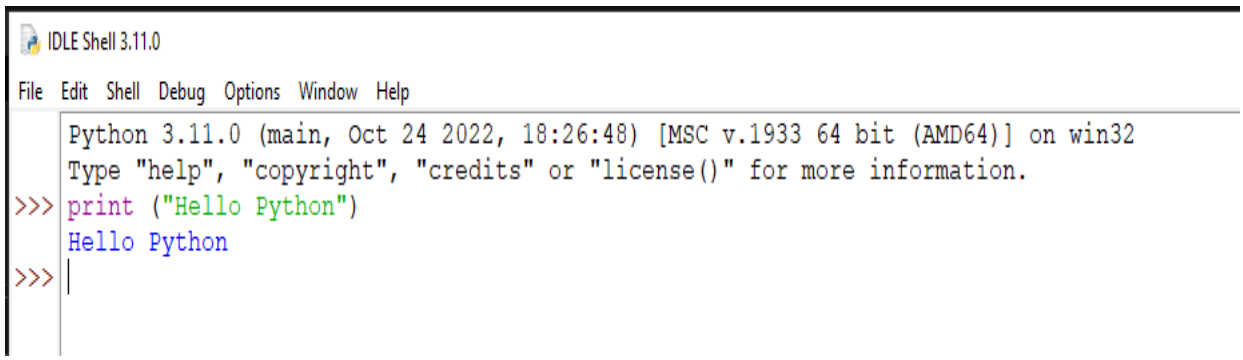
3. Locate the Python icon on your Windows task bar at the bottom of your monitor's screen. Right click and select **Pin to Taskbar**. This makes Python always on your taskbar, which saves you having to look for the program in the future.

Let's write a simple program to test Python is working

1. In the IDLE Shell window, click where the >>> is displayed. The >>> is called a prompt.
2. Type in this line of text exactly as it is shown here. This is programming code!

```
print("Hello Python")
```

3. Press **Enter** on your keyboard.
4. The Python Shell shows the words `Hello Python` underneath the line of code you just wrote.



You've successfully installed Python and written your first program. If you want to learn more about the Python programming language, then be sure to check out appendix D for an overview of Python's uses and functions.

Other IDEs you may want to try.

IDLE is a free IDE provided with Python. IDLE is standard and contains many features that are included in other IDEs. One of the common comments about IDLE is that the interface is unappealing. Therefore, you may want to try some of these other IDEs if you want a more colorful coding experience. Remember you can try other IDEs and find one that you prefer.

- Mu. This IDE describes itself as a *"simple Python editor for beginner programmers."* It has a nice eye-catching interface. Download here: https://codewith.mu/.
- Replit. This IDE is cloud based, which means you don't have to download and install any software. The advantage to cloud-based software is that you can access you projects on any device that has a live data connection or is connected to the Internet. Before using Replit, you need to create a free account. You can access Replit at https://replit.com/.
- Thonny. This IDE describes itself as a "*Python IDE for beginners.*" It provides a colorful user interface and has stripped out all the features that may distract a user. Download here: https://thonny.org/.

Appendix B. Installing Python on other operating systems

This appendix walks you through downloading, installing, and testing Python on operating systems other than Windows such as macOS, Linux, and Chromebook.

Python is a free programming language. This means that you can use Python for free to create, edit, and share your programs. Python is popular because you can use it to write programs for many different applications and purposes. However, its real strength lies in the simplicity of using the code compared to other languages. You can learn more about Python in appendix D.

To code with Python, you need an integrated development environment (IDE). An IDE is an interface that you can use to write programs, test them, and run them on your computer. The Python installation includes its own IDE, called IDLE, which is pronounced idol.

There are other IDEs available although these alternatives may not run on all operating systems. This book is written using the standard IDE that comes with Python which means that if you are searching for our own instructions or videos you should search for,

How to install python idle on x

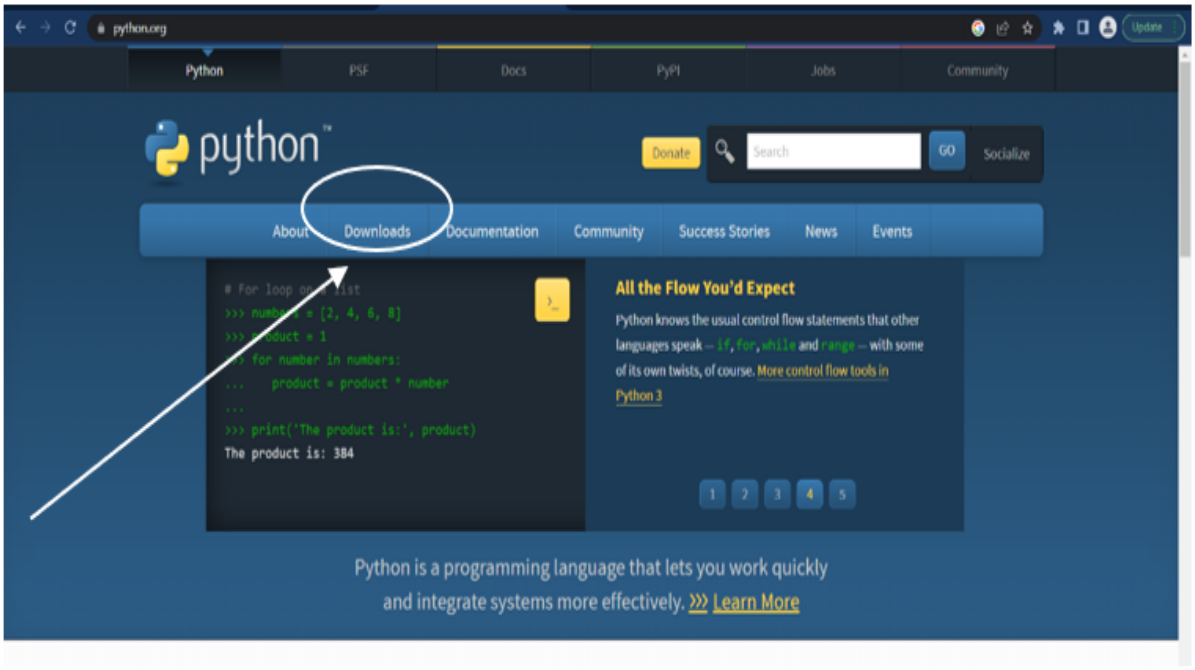
Replacing the x with your operation system for example,

How to install python idle on Linux Ubuntu

This will ensure that the guizero program will work correctly and you can build the projects in each of the chapters.

Getting Python

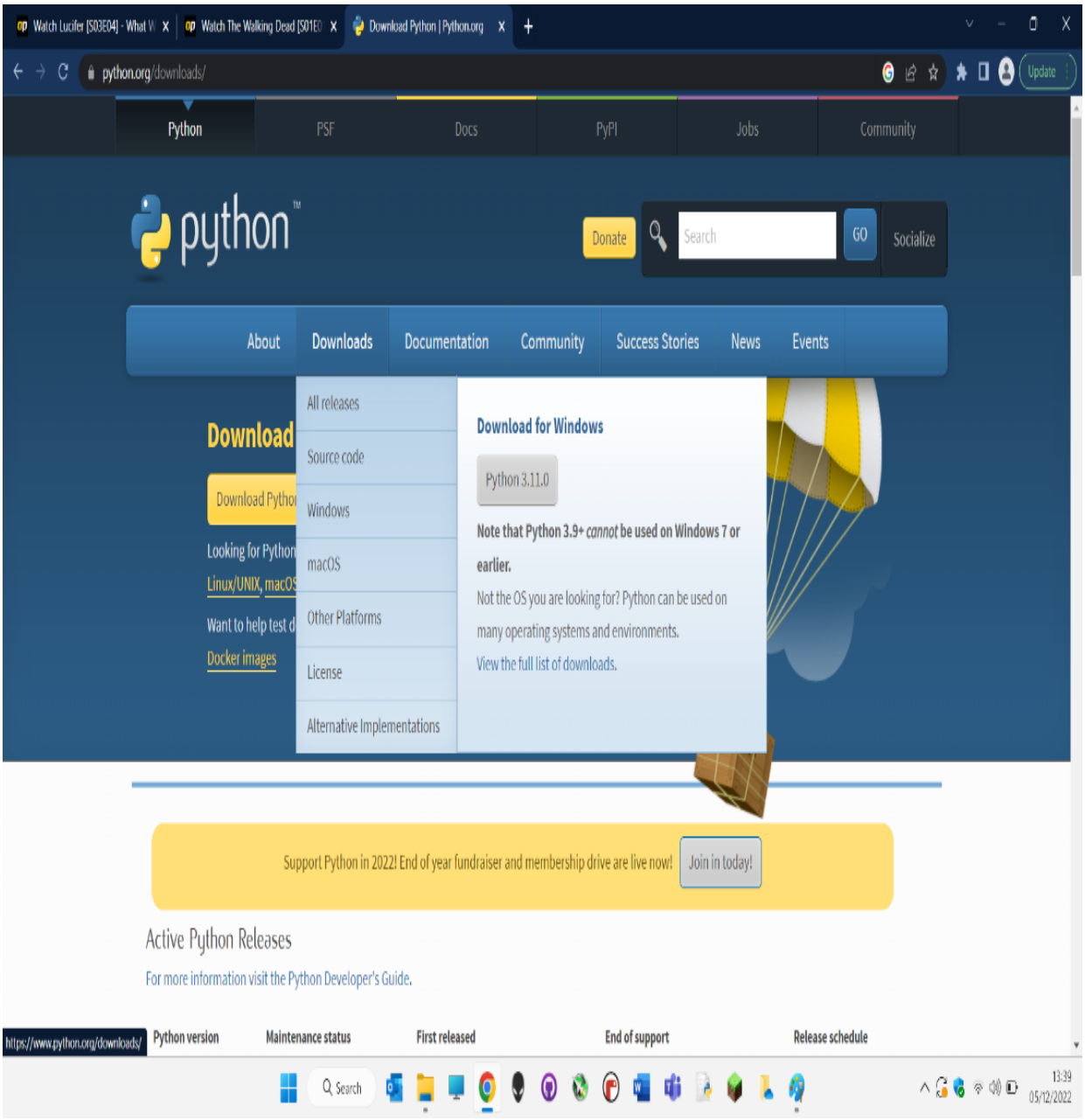
Let’s begin. The first step is to head over to the Python website and download the Python install program so that you can install it on your computer. Open your web browser, (like Chrome, Safari, Edge etc.) and go to www.python.org. This will take you to the home page for Python. Over time newer versions of Python are released, so remember to check this site regularly if you wish to upgrade your version of Python to the latest release.



Downloading Python

On the Python home page, find the navigation menu at the top of the website and select the **Downloads** button. This takes you to the downloads page where you can select which version of Python you want to install.

When the Downloads page opens, the menu displays all the download options in a list. Click the tab for the OS that you are using, for example, if you want to install Python on macOS, then select this option.

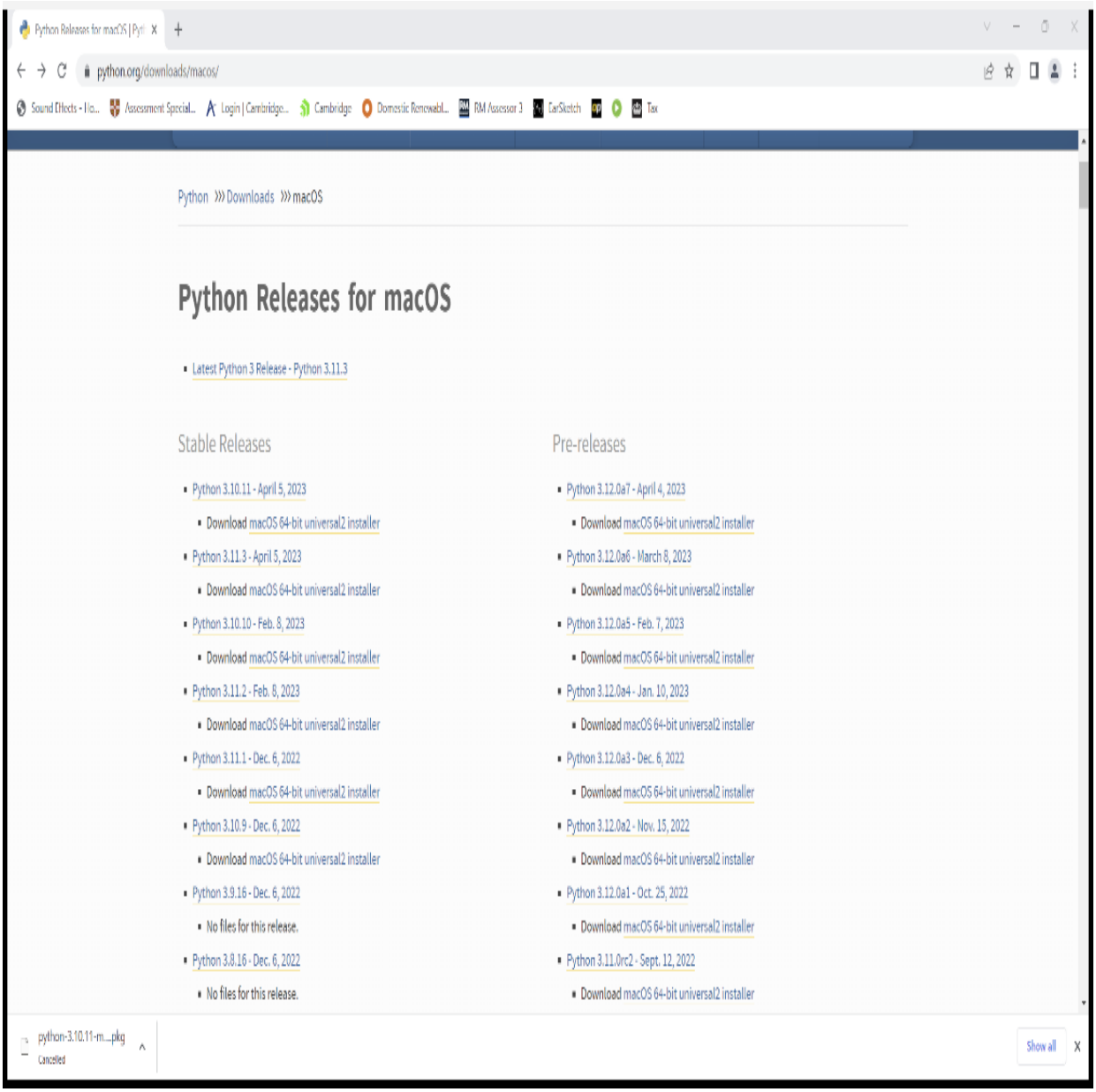


To begin downloading the Python installation file, click the download button. It’s okay if the version number is different from what you see here.

There are many versions of Python available on the main website so if you are using a different OS be sure to check out the downloads page

Installing Python on macOS

Select the macOS option which will take you to a page that displays all the available versions of Python. As shown below,

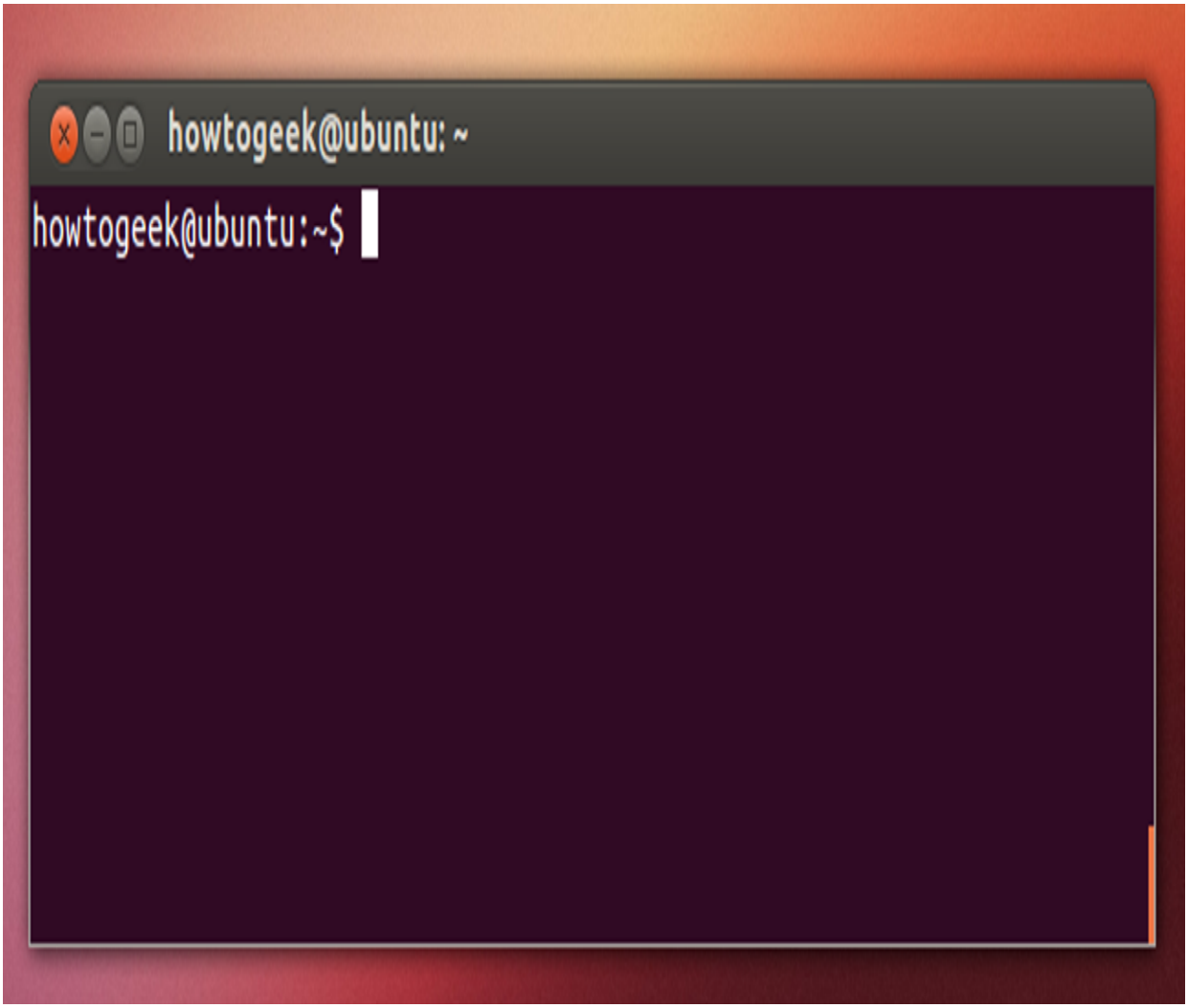


Select the version that you want, usually the latest stable release and then locate the Download and select the `macOS 64-bit universal2 installer.` This will download the Python set up program. Once the Python installation file downloads, on your computer, use the folder browser and head to the folder where the Python installation file has been downloaded to. Locate the downloaded Python file.

When you've found the file, select it (usually double-click) to start installing Python. You will be presented with a menu window that has several steps. For further details and a video see this weblink, https://www.jcchouinard.com/install-python/#Install_Python_on_MacOS.

Installing Python on Linux

There are many versions of Linux so covering the installation of all versions is not possible. Therefore, this section just covers installing Python using the command line. To use the command line, you have to have permission to access the Terminal window.



Many versions of Linux already have Python installed so the first step is to check if it is installed.

To do this, open a Terminal window and enter the following command:

```
python –version
```

If the response displayed says that Python cannot be found, then it means that Python has not been installed. Linux may then provide you with a set of instructions covering how to install Python. Follow these and Python will install. If not, then follow these simple steps:

1. In the terminal window, type `sudo apt install idle3`
2. Press **Enter** to begin the installation.
3. Once installation has completed, go to your search bar and type in **idle**
4. This will find Idle. Click the logo to load it.
5. You can watch a video about the install here: https://www.youtube.com/watch?v=l0g2_7csJs8

Installing Python on Chromebook

Installing Python on a Chromebook is similar to the Linux installation. Complete the following steps:

1. Open a terminal window.
2. Enter the command `apt install idle3` into the prompt.
3. Press **Enter**.

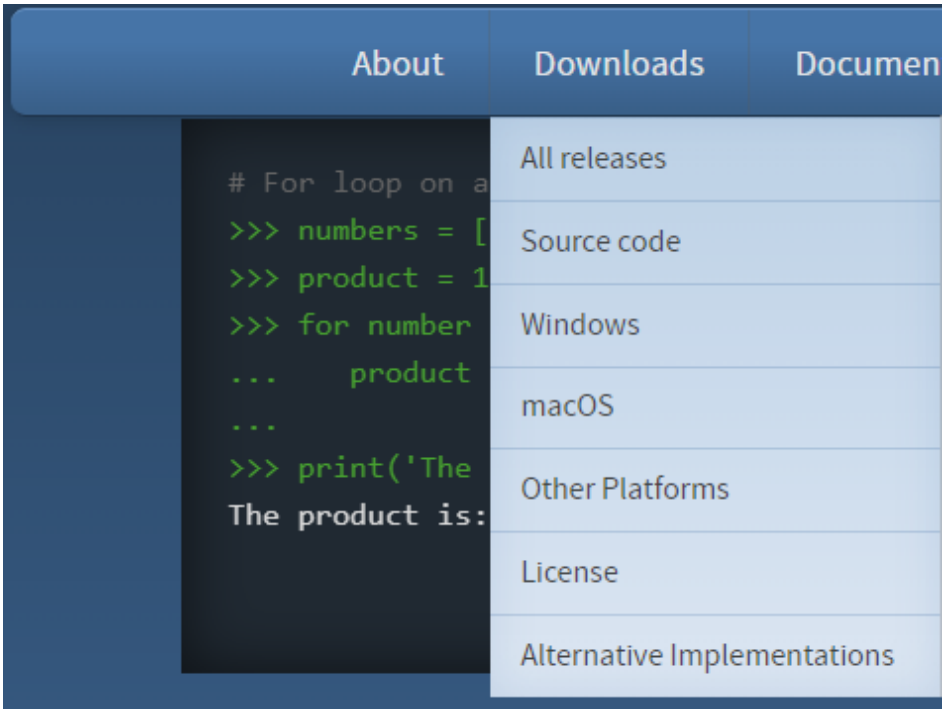
If this does not work, then you may need to install Python using pip. To do so, follow these instructions.

1. Open a terminal window.
2. Enter the command `apt install python3-pip`
3. Press **Enter**.

For further details check out this useful article: https://ninja-ide.org/install-python-on-chromebook-without-linux/

Other versions of Python

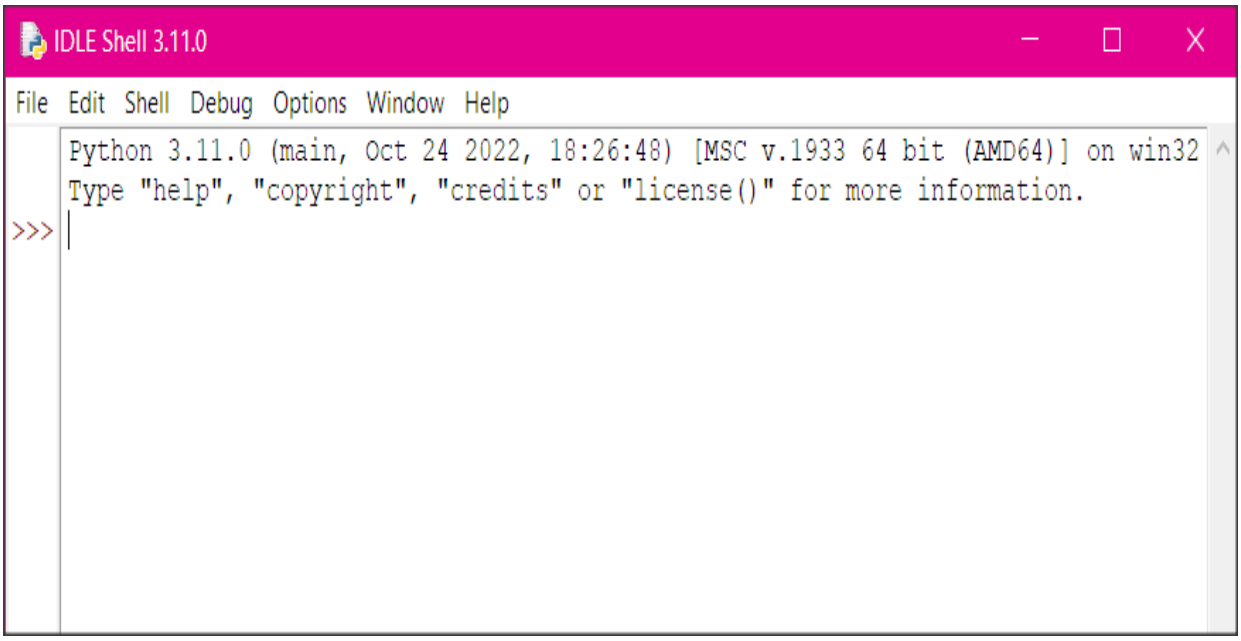
There are many versions of Python available for many operating systems, including those that run on tablets and mobile devices. Check out the Alternative Implementations on the Python downloads page to find out more.



Testing your installation of Python

Now to test that Python has installed correctly and is working. Complete the following steps.

1. Open the Python program. Depending on which operating system you are using, Python may be in a recently added section, on the desktop, or in the main menu.
2. Double-click the IDLE program. Python loads in an **IDLE Shell** window. The shell window is an interface in which you can enter Python code one line at a time. You write the line of code, press **Enter**, and then the code runs, displaying the output in the shell window.

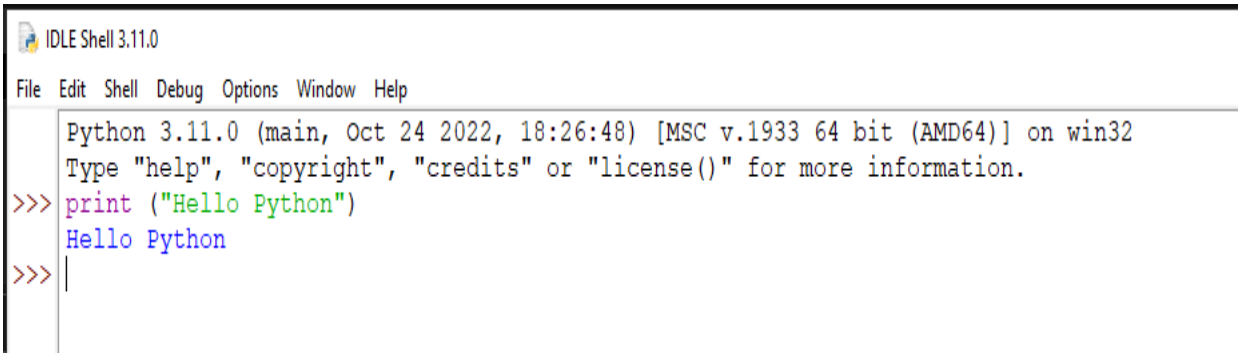


Now that you've opened IDLE, let's write a simple program to test that Python is working.

1. In the IDLE Shell window, click where the >>> is displayed. The >>> is called a prompt.
2. Type in this line of text exactly as it is shown here. This is programming code!

```
print("Hello Python")
```

3. Press **Enter** on your keyboard.
4. The Python Shell shows the words `Hello Python` underneath the line of code you just wrote.



This result means you've successfully installed Python and written your first program. If you want to learn more about the Python programming language, check out appendix D for an overview of Python's uses and functions.

Appendix C. Installing guizero on other operating systems and features of guizero

You may remember from chapter 1 that GUI stands for Graphical User Interface and is pronounced “gooey,”. A GUI offers an easy way for the user to interact with a computer or device using graphics and images. Guizero is a programming library that contains all the ode and widgets to create and build GUIs in Python.

There are several methods to install guizero onto your computer and it has even been designed so that you can simply download the program files and build GUIs.

Chapter 1 tells how to install guizero on a Windows computer. This appendix covers how to install guizero on other operating systems such as macOS, the Raspberry Pi, and Linux. It also covers the guizero Easy Install method which requires no installation; you simply download the required files and then guizero is ready to use.

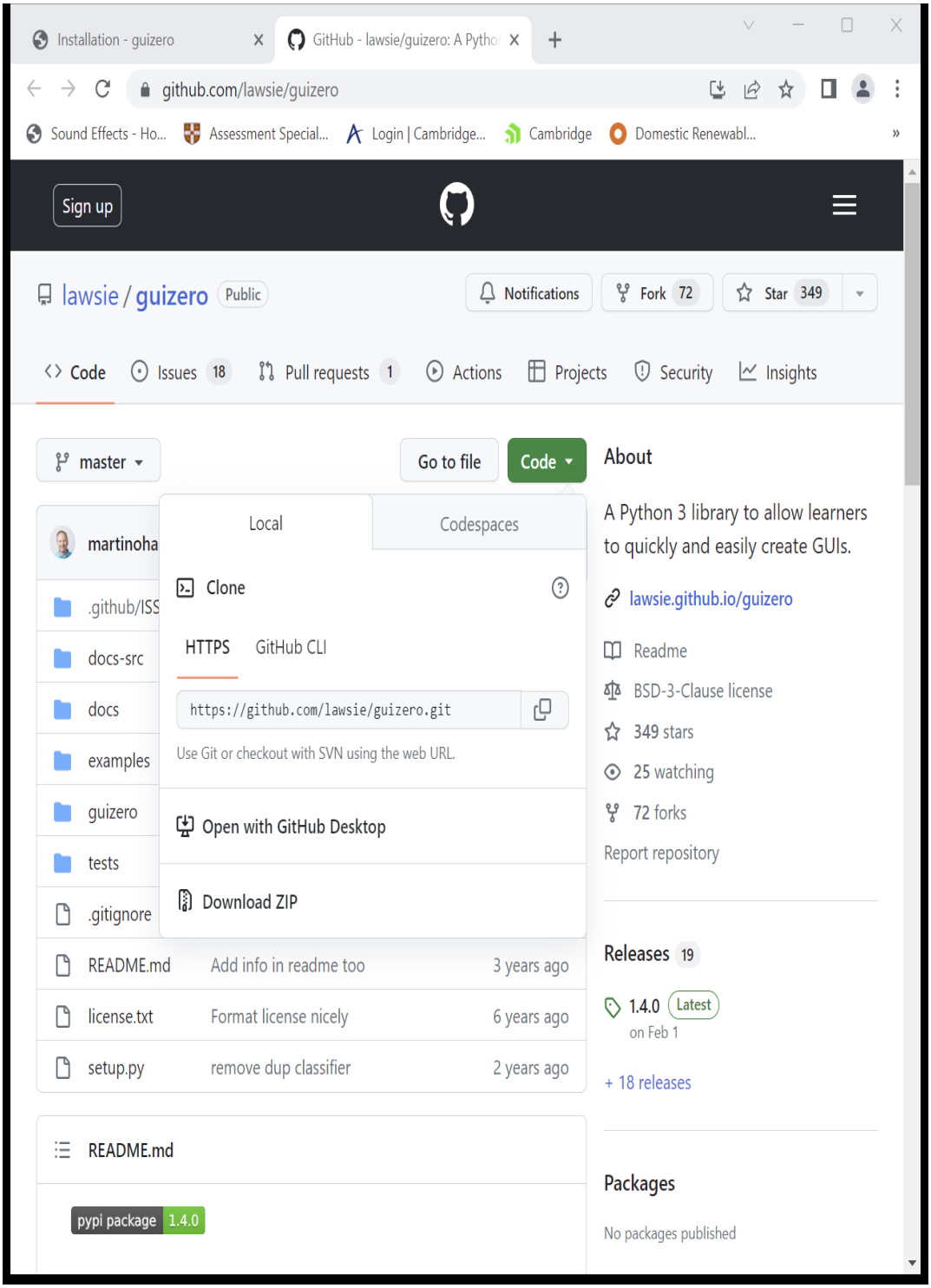
The Easy Install method is useful if you are not permitted to download and install software on your computer. For example, if you do not have the required access permissions, or you are using a computer that you share with other people. Maybe you are using someone else’s computer and they do not want you to install additional software!

The last section of this appendix covers the key features of the guizero. Let us start with the Easy Install method first.

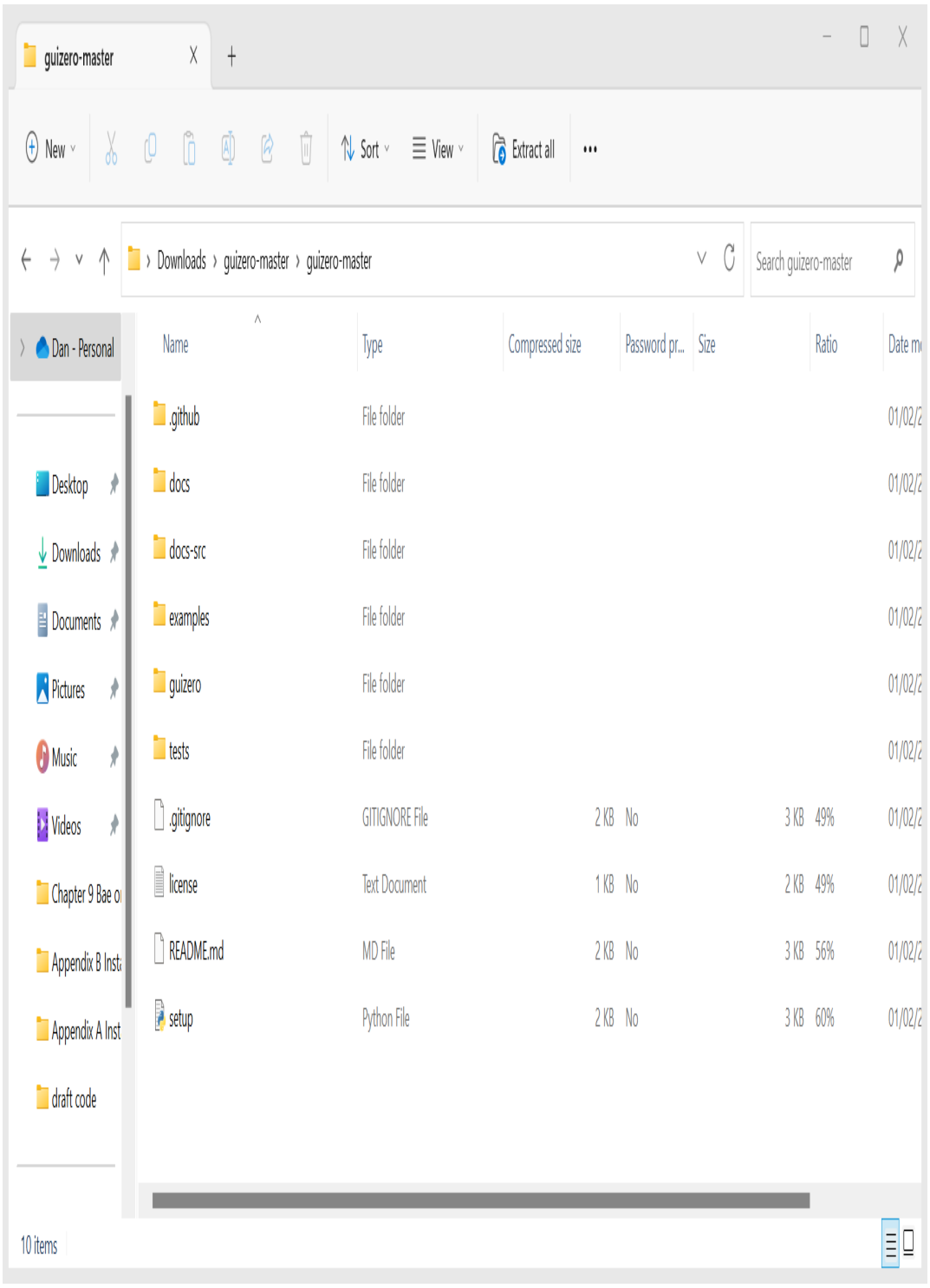
Easy installation

This installation is the simplest and easiest as it requires no installation of the software and therefore no special permissions or administrators’ rights. To use this method, complete the following steps.

1. Open your internet browser.
2. Go to the link https://github.com/lawsie/guizero.
3. This opens the guizero page, which holds all the code and documents.
4. Find the green code button and click it.



5. Select the **Download Zip** option. This downloads the guizero files.
6. Open the ZIP folder. This folder contains a folder named guizero-master.
7. Open the **guizero-master** folder.



8. Select all the files and folders and copy all the files into a folder on your computer.

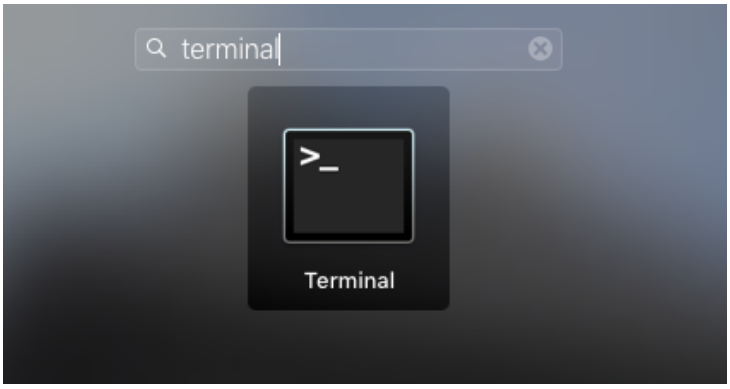
Important

Now all your GUI projects and code will run, *as long as you save your Python file into the same folder* as the guizero files. You can read more details at the official guizero website https://lawsie.github.io/guizero/.

Installing guizero on macOS

To install guizero on macOS complete the following steps.

1. Open the terminal window click selecting **Applications** > **Utilities** > **Terminal** (or type **terminal** into the search bar).
2. In the terminal window type `pip3 install guizero`.
3. Press **Enter** to install guizero.



Installing guizero on a Raspberry Pi

To install guizero on a Raspberry Pi complete the following steps.

1. Open the terminal window.
2. In the terminal window type `sudo pip3 install guizero`
3. Press **Enter**.
4. This installs guizero.

Installing guizero on Linux

To install guizero on a Linux device complete the following steps.

1. Open the terminal window.
2. Ensure that `tkinter` is installed by entering `sudo apt install python3-tk`.
3. Install guizero by typing `pip3 install guizero` or `sudo pip3 install guizero`.

There are other versions of Linux. If you are using the Debian version, then you can install guizero in the terminal using the following command:

```
sudo apt-get install python-guizero
```

Installing additional features

guizero has additional features that are used to edit images and graphics within a GUI, such as resizing images or playing an animation. Each chapter in the book covers any additional image editing that you need to do. Most projects do not use animation. If you want to use these additional features of guizero, you will need to install guizero with the `pip` command (table C.1).

Table C.1 Installing additional guizero features using the pip command

Operating system	Command
Windows and macOS	`pip3 install guizero[images]`
Linux or Raspberry Pi	`sudo pip3 install guizero[images]`

Note

These additional image features are not available if you use the Easy Install method.

Upgrading guizero

Over time, guizero will be updated and you will want to keep your version up to date. If you installed guizero using pip, then you can upgrade guizero in the terminal window using the commands in table C.2.

Table C.2 Upgrading guizero

Operating system	Command

Windows and macOS	`pip3 install guizero --upgrade`
Linux or Raspberry Pi	`sudo pip3 install guizero --upgrade`

Features of guizero

For ease of understanding guizero and its features, you can split guizero into two distinct groups. Group one contains the main categories of the software (table C.3). These categories are called elements and are the overall building features of a GUI. Elements allow you create interactivity between the user and the GUI. Interactivity is where, for example, the user presses a button, and the GUI displays a message via a popup. Or the user presses a button, and it triggers a sound to play.

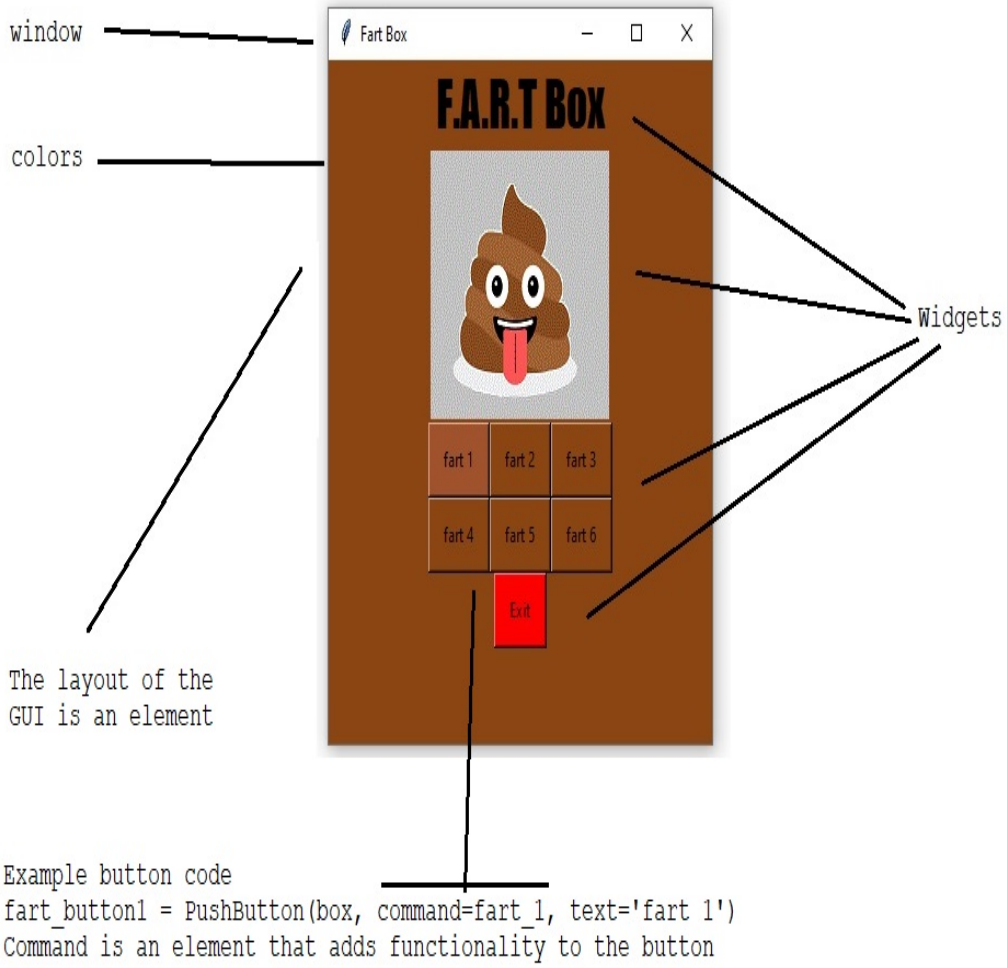
These elements also enable you to build a GUI. For example, using the Layout element, you can adjust the size of the GUI window and position buttons and text within the GUI window. With the ‘Events’ , you can program an action to respond (called triggering an action) when something happens; for example, when a user double-clicks the left mouse button on a picture, some text is displayed.

Table C.3 The elements of guizero

guizero Element	Description	Example use with button widget
Colors	Edit the colors	Color can be set on most widgets, text, and backgrounds, customizing the feel of the GUI.
Commands	Create interaction in the GUI; for example, pressing a button or selecting an option	Used to assign functions to objects, how the GUI responds to a button press.
Event	Assigned to a widget that triggers an action based on mouse and keyboard inputs.	GUI displays an image when a button is clicked once and changes the image when the button is double clicked.

Images	Manage images within the GUI window	Display an image in your GUI, an animation or make a button a picture button.
Layout	Controls how widgets are arranged in the GUI	Stacks the buttons in a particular order or uses auto layout to arrange the buttons.
Loops	Used to repeat a section of the program code and ensue the GUI always responds to user.	GUI enters a loop which waits for events (user input) to happen and then respond.
Pop-Ups	Windows that pop up with a message or information for the user	Informing or updating the user. Popups can use the warn, info and error popups, prebuilt into Windows OS.
Sizes	Used to set a widget's width and height	Makes a button a certain size within the GUI.
Widgets	The objects that appear in your GUI (see table 1.2)	Used to build the GUI and add interactivity.
Windows	Creates extra windows in your GUI	The space in which the GUI and its features are held.

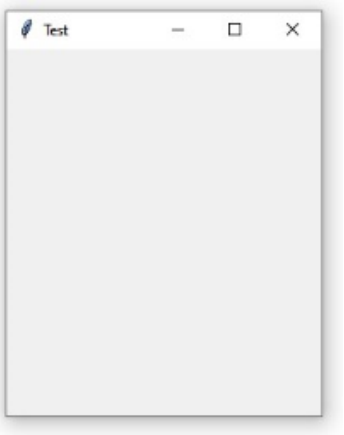
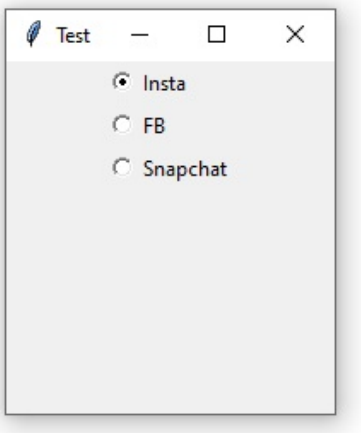
The second group is widgets. Widgets are how you physically build the GUI; they are the main building blocks of a GUI. There are fifteen widgets that you can use. Each one is an object (think of objects as an item) that appears within the GUI, everything from the app itself to text boxes, buttons, and pictures. Some of the widgets are used with the Elements of the GUI. For example, the Box widget is a container that holds other widgets and is useful for neatly grouping them together. The Box widget is used as part of the Layout Element to create and customize the layout of the GUI. Table C.4 describes what all the guizero widgets do.

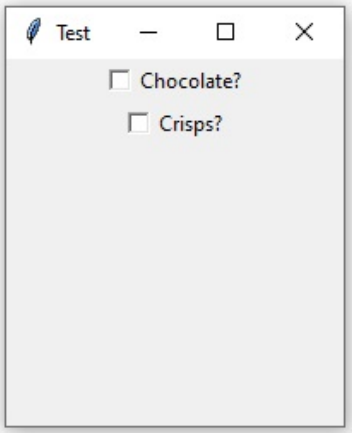
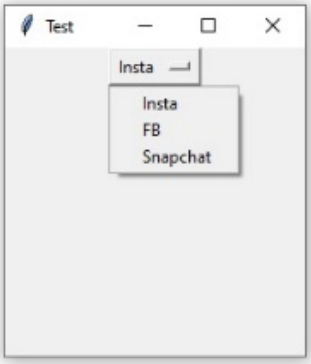
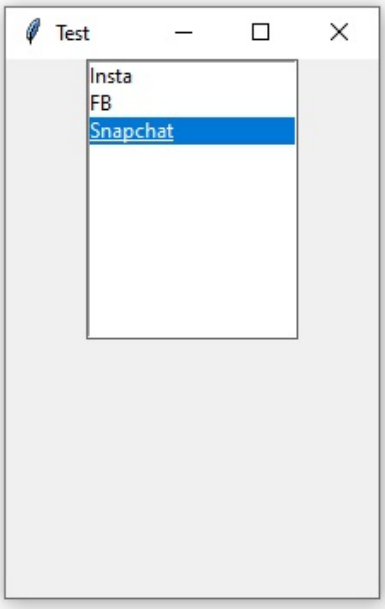


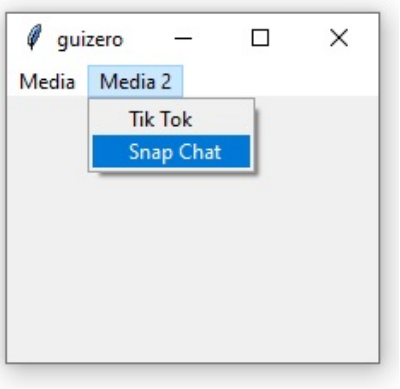
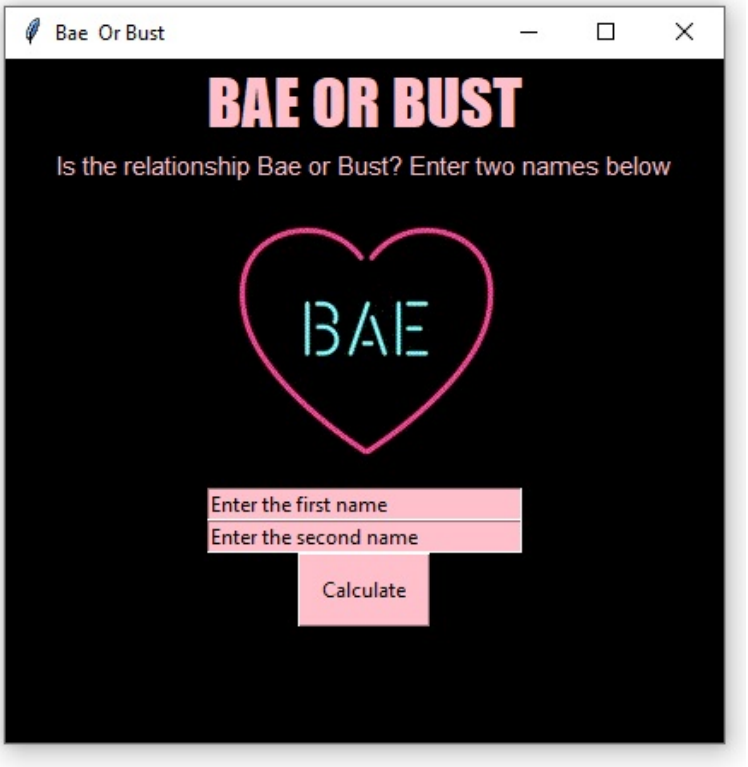
Some widgets are similar but have different properties. For example, the PushButton widget creates a clickable button, whilst the `ButtonGroup` widget creates a button with options where the user can select from several choices. You will use the widgets throughout the various projects, and you can refer to tables C.3 and C.4 as required.

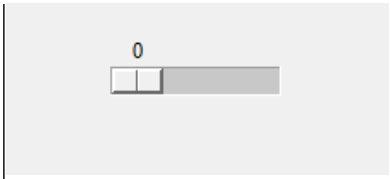
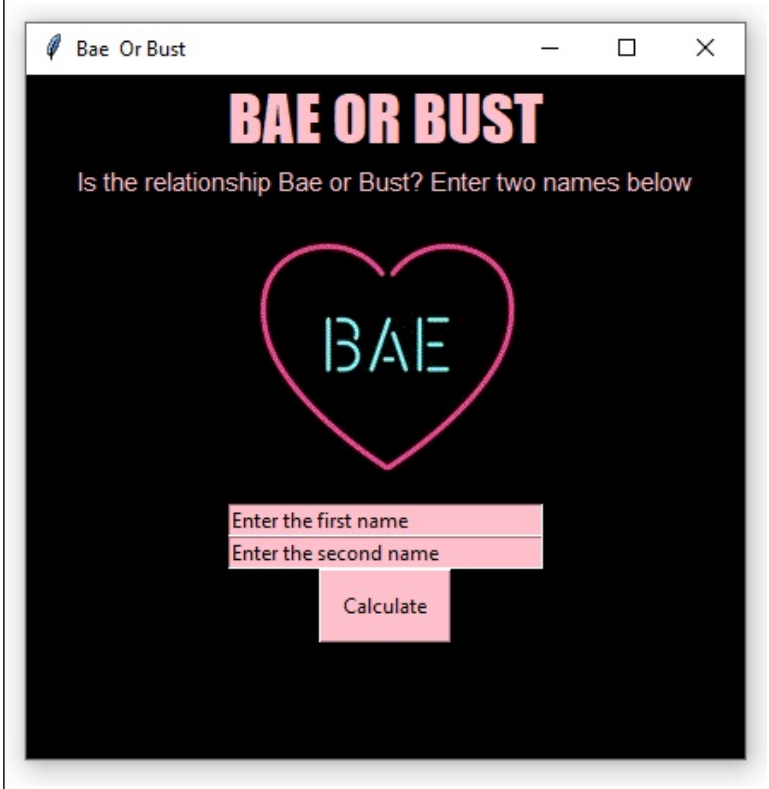
Table C.4 The guizero widgets

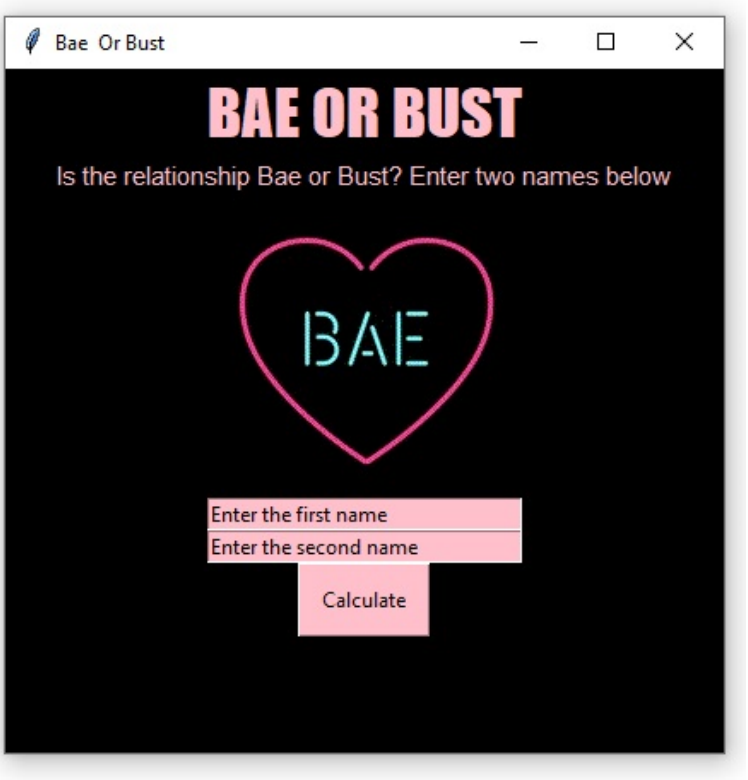
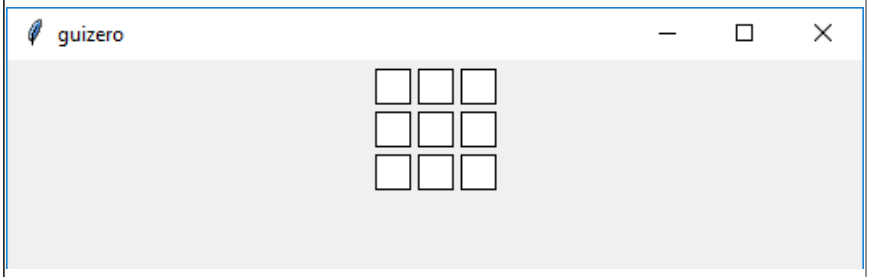
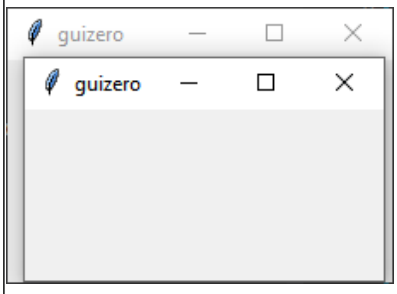
guizero Widget	Description	Example use
`App`	The main window of the GUI	

	that holds all the other widgets	
`Box`	An invisible container that holds other widgets, useful for grouping objects together	Useful for organizing and layout of the widgets in the GUI window.
`ButtonGroup`	A group of radio buttons that enable the user to make a choice	
`CheckBox`	A box that can be ticked or unticked	

		
`Combo`	Displays a drop-down box where a single item can be selected from a list	
`ListBox`	Displays a list of items where one item can be selected from	

`MenuBar`	Creates a drop-down menu at the top of the GUI	
`Picture`	Displays an image in the GUI	
`PushButton`	A button with either text or an image that runs a function when pressed	

`Slider`	Displays a bar that can be used to select a specific value within a range	
`Text`	Displays non editable text in your app	
`TextBox`	Displays a box which the user can type into	
`TitleBox`	An invisible container where other	

	widgets can be grouped together within a boarder and displaying a title	
`Waffle`	Displays a grid of squares	
`Window`	Creates a new window in the GUI	

Twisted Python Projects

DAN ALDRED

MANNING