

Computing

Lesson 1: GUIs

Programming Part 5: Strings and Lists

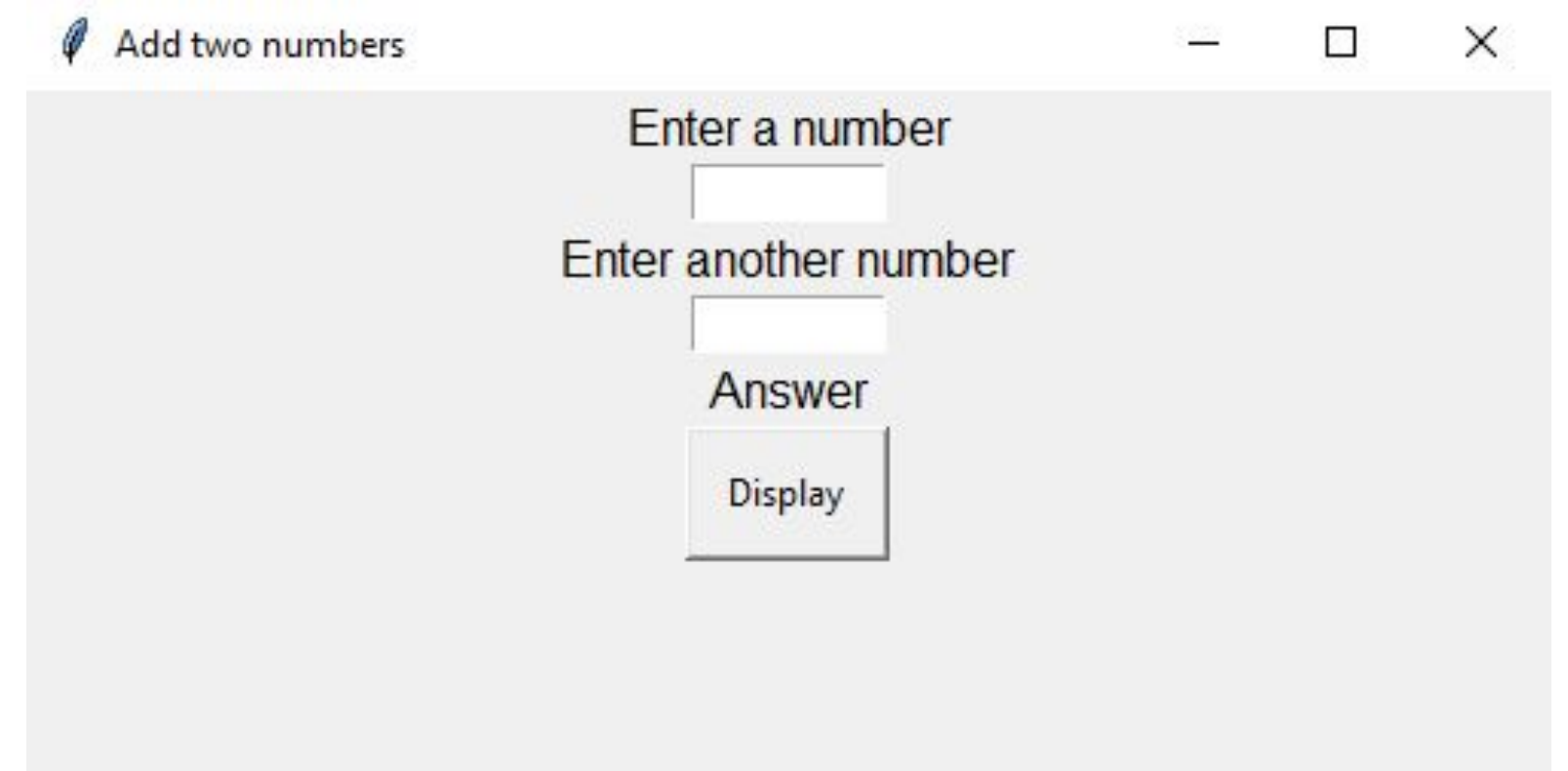
Ben Garside

¹ *Materials from the Teach Computing Curriculum created by the National Centre for Computing Education*



Task 1 - adding two numbers

For this mini project you will be creating an app that allows you to enter two numbers and add them together. Your app will look like the screenshot (note that Mac users will see a slight variation to the screenshots presented here)



Task 1 - adding two numbers

Step 1

The first step is to set up and test your app using the code below:

```
1 # import the modules required from guizero
2 from guizero import App
3
4 # create the app with the window title "Add two numbers"
5 app = App(title="Add two numbers")
6
7 # display the app, this should always be last
8 app.display()
```

Step 2

Test that the program works successfully.



Task 1 - adding two numbers

Step 3

Import all of the widgets that you need .

This program uses the following widgets:

App, Text, TextBox, PushButton

Make sure that all of the widgets have been imported on the top line of code.



Task 1 - adding two numbers

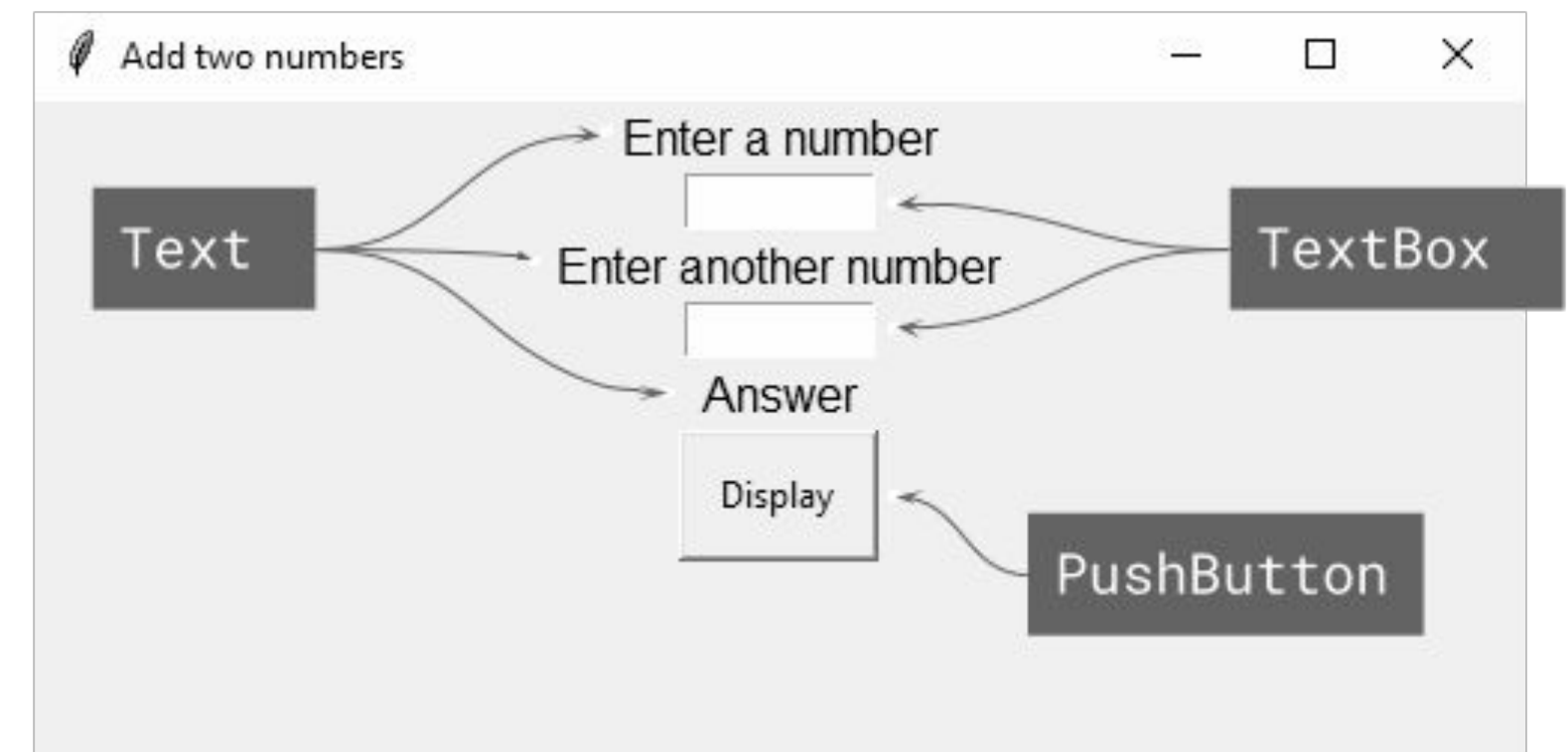
Step 4 - Build the app

Your code for adding the widgets to the app needs to be placed between the app creation line of code and the app display line of code. See below:

```
app = App(title="Add two numbers")  
# add your widget code here  
app.display()
```

Step 5

Take a look at the layout on the right for the app that you are going to build and note the required widgets.



Task 1 - adding two numbers

Step 6

You can be really specific about where your widgets are placed and you can explore these options later. In this project you will simply be placing one widget after the other in a sequence.

This sequence goes:

- Text
- TextBox
- Text
- TextBox
- Text
- PushButton

This means that you will need 6 lines of code to add all of the required widgets onto the app.

Each widget needs its own unique identifier, like a variable name.

On the next slide is a table to help you with the identifiers of each widget.



Task 1 - adding two numbers

Step 6 continued

Identifier	Widget
instructions	Text
enter_num1	TextBox
instructions2	Text
enter_num2	TextBox
display_answer	Text
display_number	PushButton



Task 1 - adding two numbers

Step 7

Below are code snippets for each widget with explanations of how they work. Use these to help you create the first 5 widgets in your app. The PushButton will be dealt with later.

```
identifier = Text(app, text="Text to display")
```

This code snippet can be used to display text. The identifier should be replaced with your unique identifier and the Text to display should be replaced with the text that you wish to display.

Note: the app part of this code instructs that this widget is controlled by the app. You can use other 'masters' here which you could explore later.



Task 1 - adding two numbers

Step 7 continued

Text box

```
identifier = TextBox(app)
```

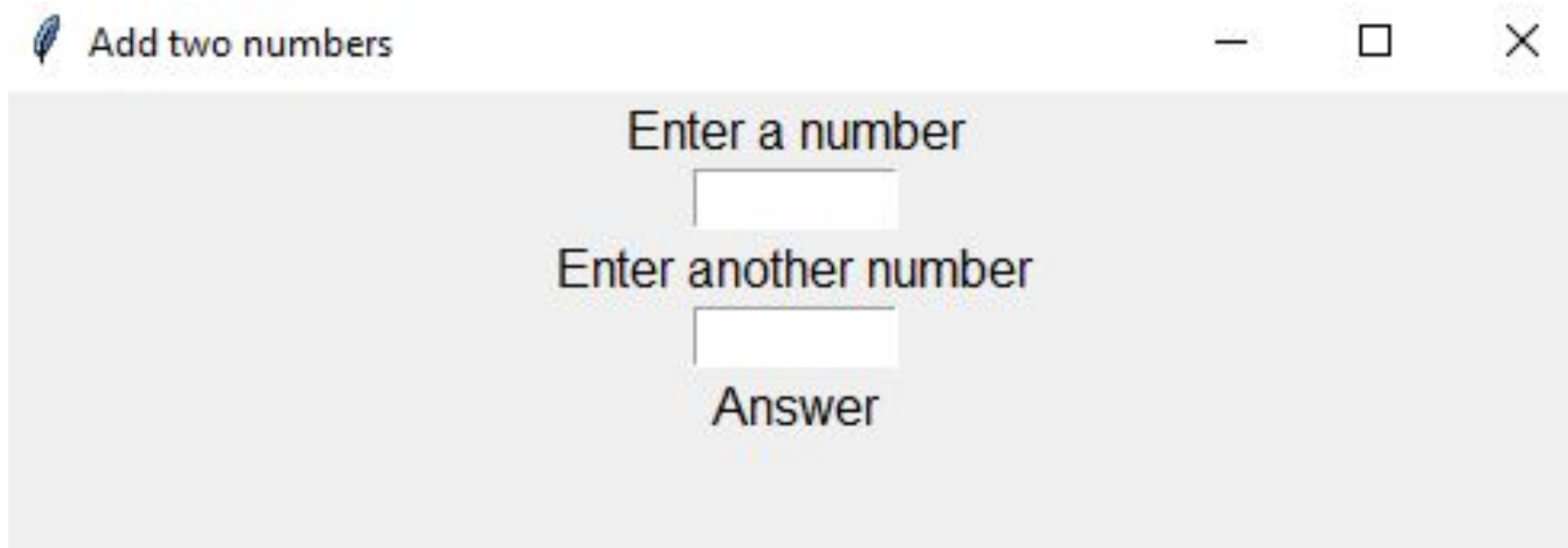
This code snippet can be used to add a text box. These work a little bit like the input() functions that you have used before. The identifier should be replaced with your unique identifier. The rest of the statement should stay the same.



Task 1 - adding two numbers

Step 8

Test your app by running it. If it works correctly it should load like this:



The screenshot shows a Java Swing window titled "Add two numbers". The window has a standard title bar with a minus sign, a maximize button, and a close button. The main content area is light gray and contains three text labels: "Enter a number", "Enter another number", and "Answer". Below "Enter a number" is a white text input field. Below "Enter another number" is another white text input field. Below "Answer" is a white text input field.

If your app doesn't look like the one above then double check that you have written your code in the correct order and that you have used the correct syntax for each widget.



Task 1 - adding two numbers

Step 9 - Adding a push button

Adding the PushButton is slightly different to the other widgets because you want the PushButton to call a subroutine when the button is pressed. The subroutine will add the two numbers together and display the answer in display_answer.

```
identifier = PushButton(app, command=subroutine, text="Button Text")
```

The identifier is the unique name for the widget. The subroutine is the subroutine that you wish to call when the button is pressed. The Button Text is what you want to appear on the button.

When the button is pressed the subroutine add is going to be called. We add the identifier for the subroutine here but we leave out the () part as including it will automatically call the subroutine before we want it to.

Create the line of code that will add the PushButton widget.

Note: if you try to test your code at this point then you will get an error because the add subroutine has not yet been defined.



Task 1 - adding two numbers

Step 10

The add subroutine needs to be defined at the beginning of the code. You should place this underneath the import statement.

```
from guizero import App, Text, TextBox, PushButton  
  
# define your subroutine here
```

To access and/or modify the value of a TextBox or Text widget you use this code snippet:

```
identifier.value
```

Below is all of the code required to create the add subroutine. Your job is to place the lines of code in the correct order and then add them to your program. Don't forget the indents!

```
display_answer.value = answer  
num2 = enter_num2.value  
def add():  
    answer = int(num1) + int(num2)  
    num1 = enter_num1.value
```

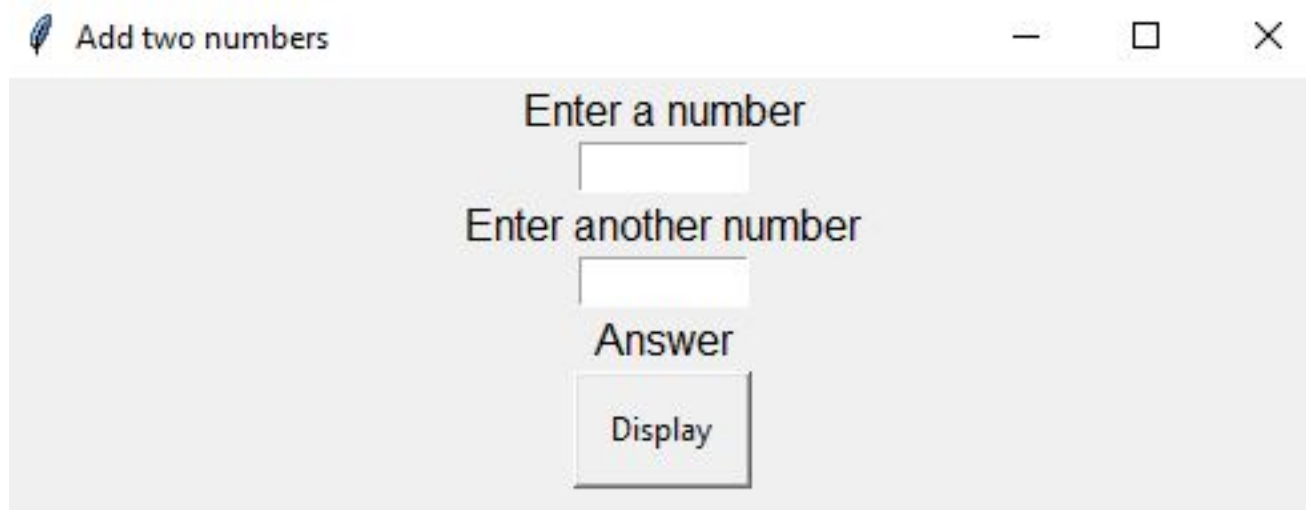


Task 1 - adding two numbers

Step 11

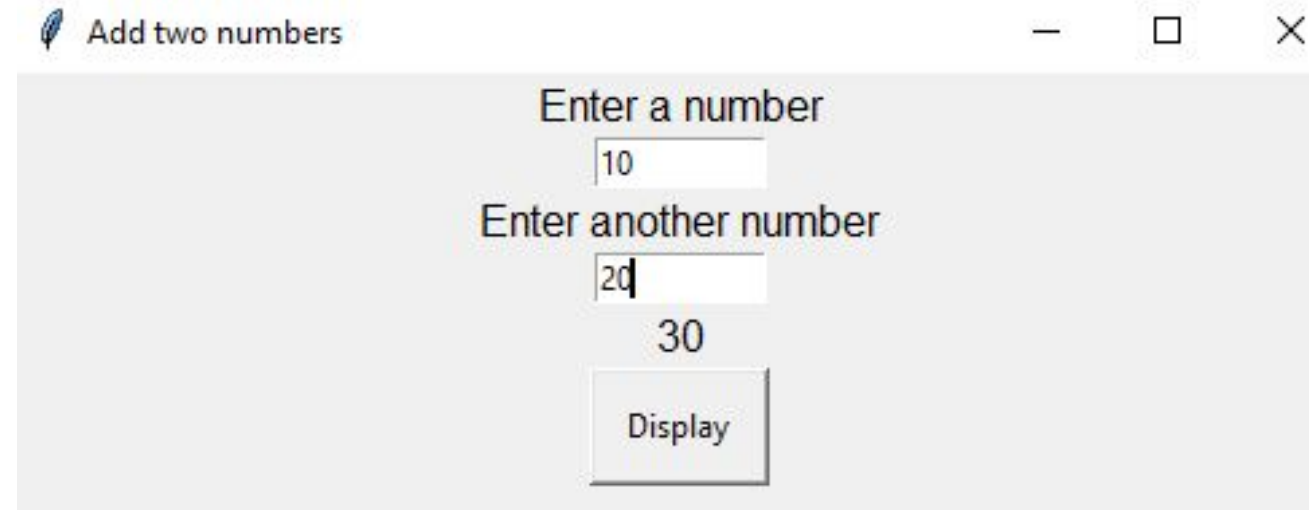
Now test your program. At this point it should be fully functional and should look like the screens below:

Screen on loading



A screenshot of a Java Swing window titled "Add two numbers". The window has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. The content is centered and consists of the following elements from top to bottom: a label "Enter a number", an empty text input field, a label "Enter another number", another empty text input field, a label "Answer", and a button labeled "Display".

Screen after entering numbers and pressing display

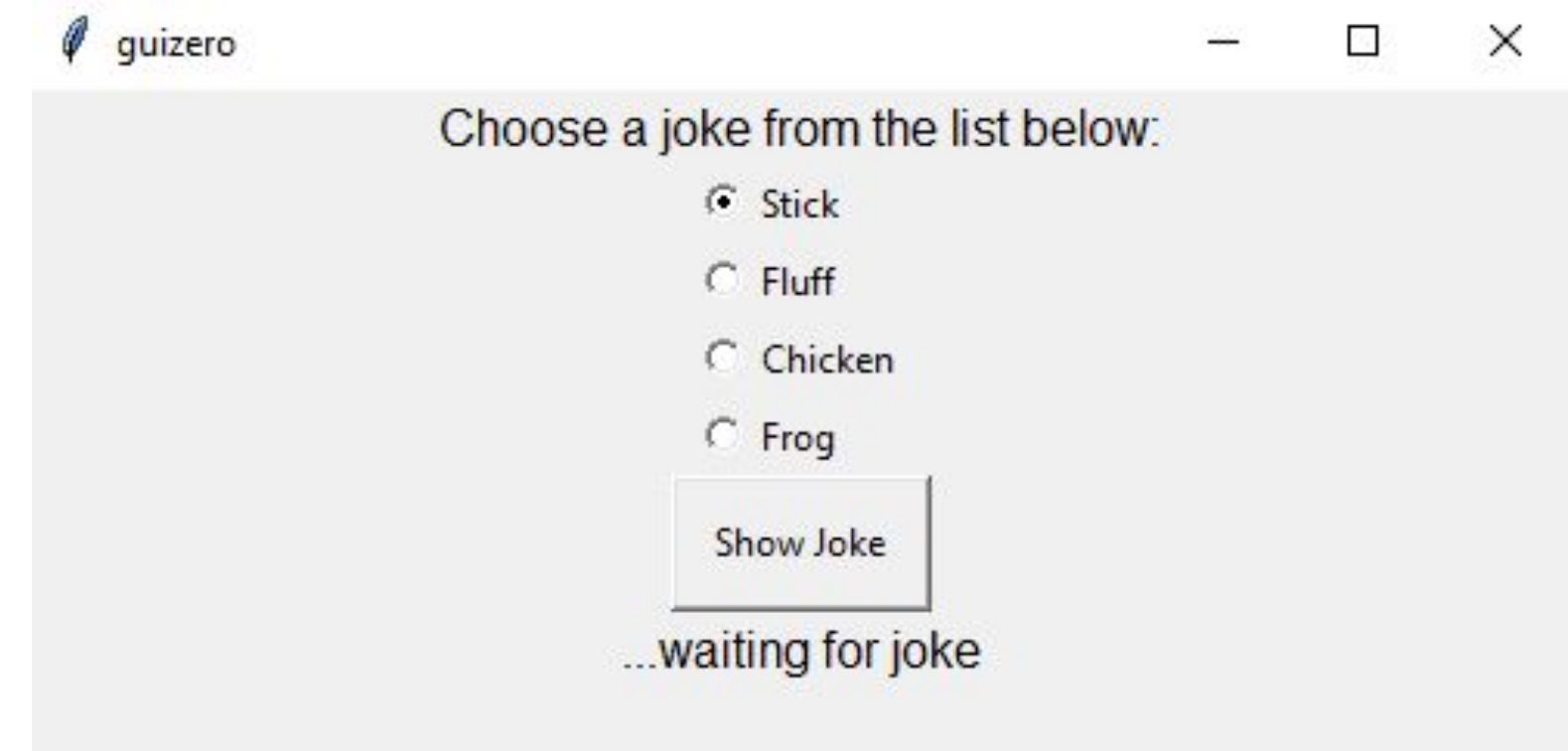


A screenshot of the same "Add two numbers" application window after user interaction. The first input field now contains the number "10" and the second input field contains the number "20". The "Display" button has been pressed, and the label "Answer" now displays the result "30".



Task 2- The joke machine

For this mini project you will be creating an app that allows the user to select a joke type and click the button to reveal the joke. It will look like the screen to the right.



Task 2- Setting up the app

Step 1

This program uses the following widgets:

App, ButtonGroup, Text, PushButton

Make sure that all of the widgets have been imported on the top line of code.

Step 2

Now make sure that you have included the line of code that creates the app and the final line of code that displays the app.

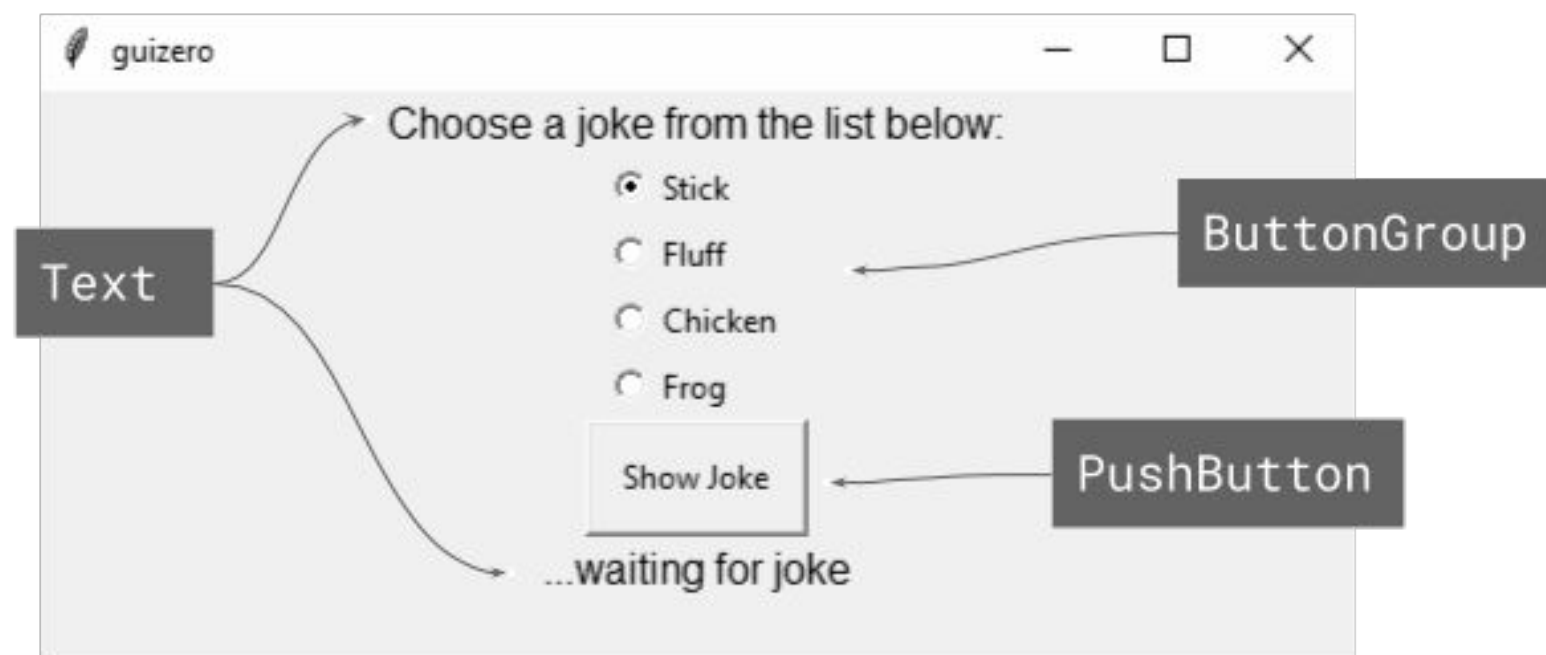
Tip: Look at your last project if you are unsure of what to do here



Task 2- Build the app

Step 3

Take a look at the layout for the app that you are going to build and note the required widgets and the order in which they appear.



Task 2- Build the app

Step 4

Make sure that each widget has its own unique identifier.

Here is a table to help you with the identifiers of each widget.

Identifier	Widget
instruction	Text
joke_choice	ButtonGroup
joke_button	PushButton
display_joke	Text



Task 2- Build the app

Step 5

The new widget that you need to use is the ButtonGroup widget. Here is a code snippet:

```
identifier = ButtonGroup(app, options=["op1", "op2"], selected="op1")
```

This code snippet can be used to create a list of options. The identifier should be replaced with your unique identifier. "op1","op2" are the list items. You can add more as long as you add a comma and surround the option with speech marks. The selected part is where you choose which option should initially be selected in the list.



Task 2- Create the jokes subroutine

Step 6

Make sure that your PushButton calls the jokes subroutine when it is pressed.

Tip: Look at your previous app to remember how to properly set up the PushButton.



Task 2- Create the jokes subroutine

Step 7

The jokes subroutine needs to be defined at the beginning of the code. You should place this underneath the import statement.

To access the selected option from the ButtonGroup you need to use:

```
identifier.value
```

Below is a code snippet to get you started with the jokes subroutine.

```
def jokes():  
    if joke_choice.value == "Stick":  
        display_joke.value = "What is brown and sticky? A stick!"
```

Complete the jokes subroutine so that it covers all the jokes that a user can select from the ButtonGroup.

- Joke 2: What is pink and fluffy?
Pink fluff!
- Joke 3: Why did the chicken cross the road? To buy some toilet paper!
- Joke 4: What happens to a frog's car when it breaks down? It gets toad away!

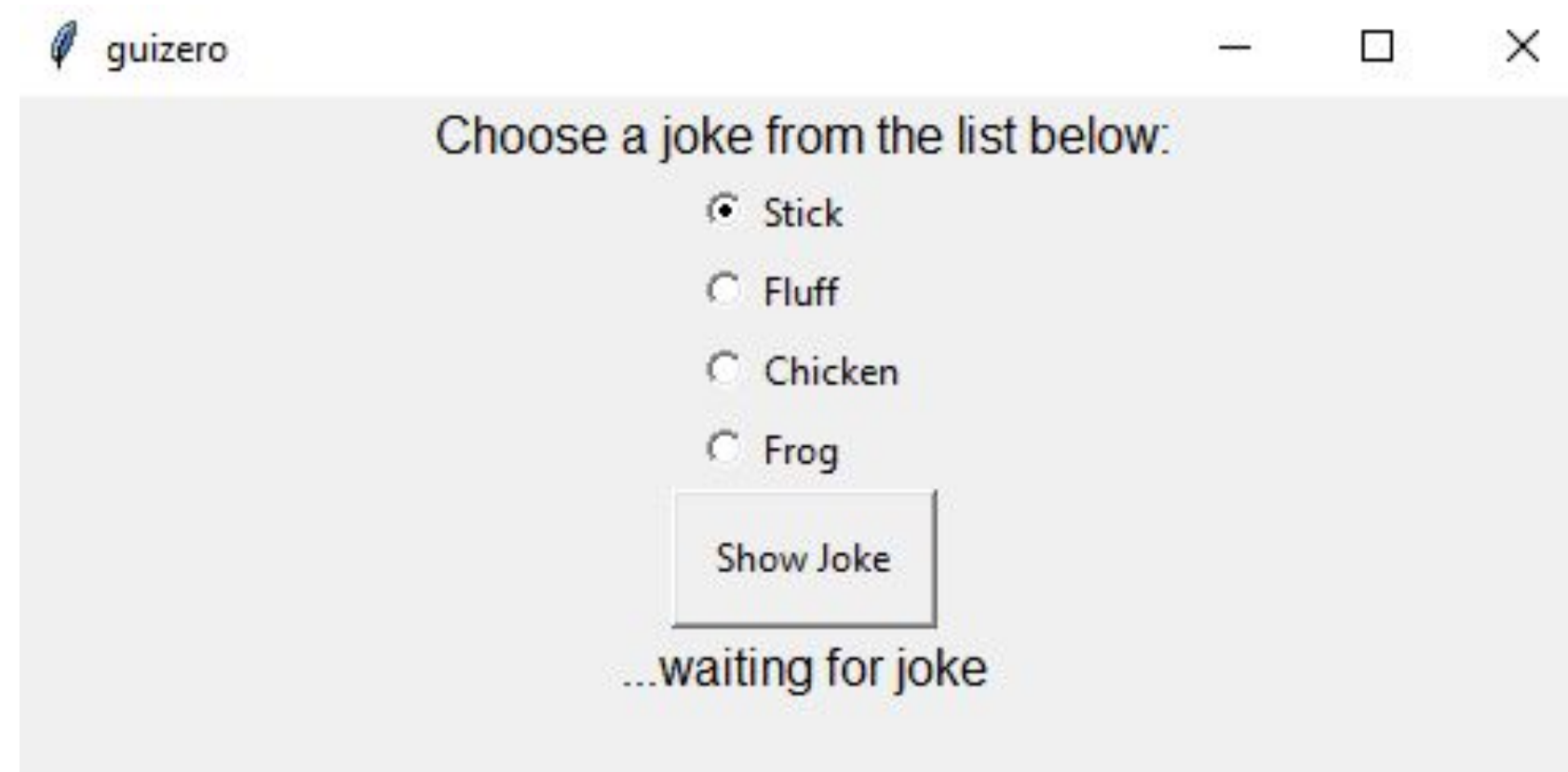


Task 2 - Testing

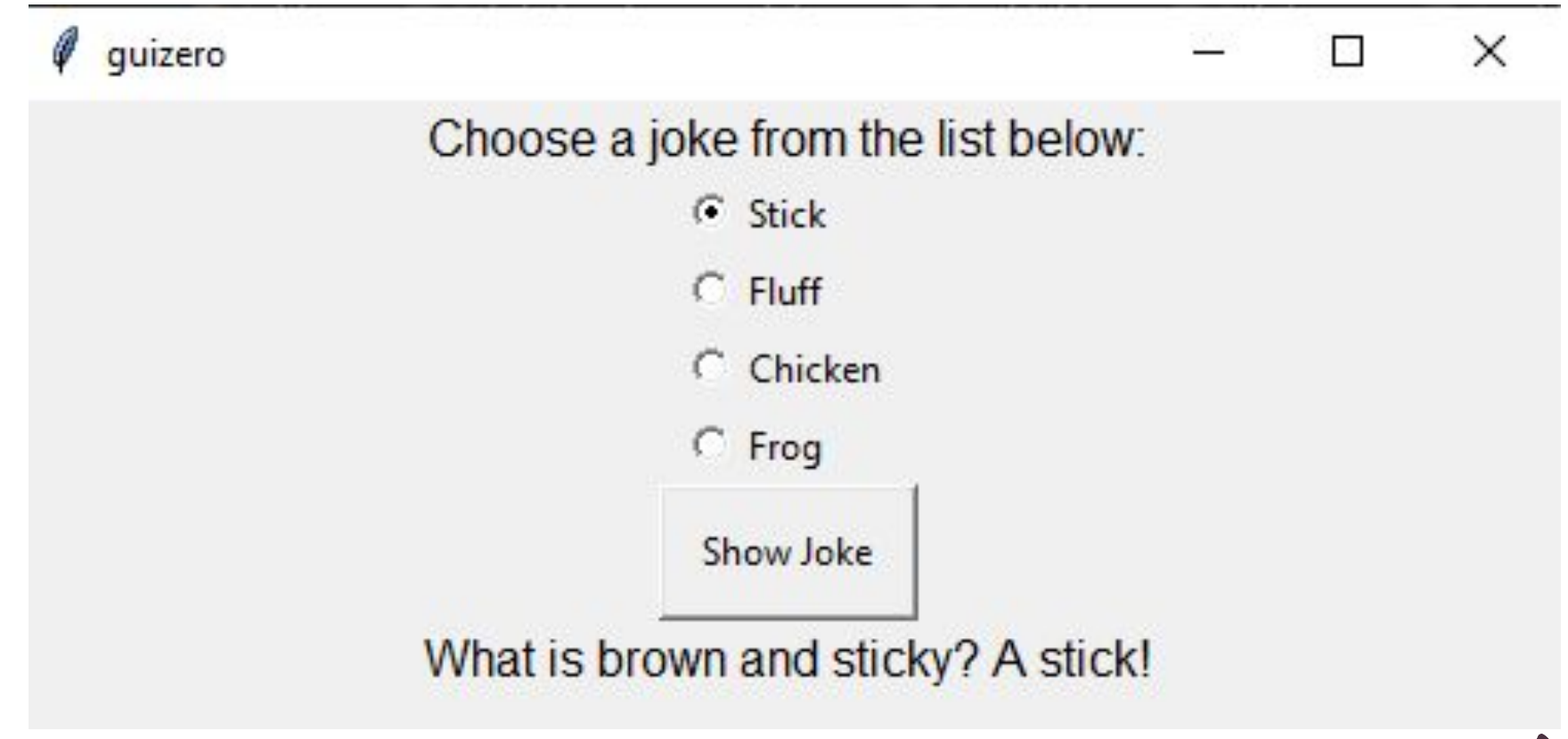
Step 8

Now test your program. At this point it should be fully functional and should look like the screens below:

Screen on loading



Screen after pressing “Show Joke”



Task 3 - Procedural vs event driven programming

Sort the statements below into either procedural or event driven programming:

- The flow of the program is determined by user actions, such as button clicks.
- The flow of control is executed in the order it is written.
- It is commonly used when creating graphical user interfaces.
- It is commonly used when the order of execution is known.
- The program waits for a user action before executing a block of statements.

Procedural programming	Event driven programming

