Creating a Graphical User Interface (GUI) for your Python program

This brief guide will show you how to create a simple graphical user interface (GUI) for your Python program. This is a very basic introduction to GUI programming in Python, and you can find more advanced tutorials and resources online.

Why create a GUI?

As the name implies, a GUI allows users to interact with your program using graphical elements such as buttons, text boxes, and menus. This can make your program more user-friendly and intuitive, especially for users who are not familiar with the command line.

Do I need a GUI for my program?

It depends! You need to think about who the target audience for your program is and what their needs are. If your program is intended for users who are comfortable with the command line, a GUI may not be necessary, and might even add unnecessary complexity. On the other hand, if your program is intended for a general audience, a GUI can make it more accessible and user-friendly.

Which GUI library should I use?

As with many things in Python, there are several options for creating GUIs. Some of the most popular libraries include:

- **Tkinter**: This is the standard GUI library that comes with Python. It is relatively easy to use and is a good choice for simple GUIs. Installation is not required as it comes with Python.
- **PyQt**: This is a more advanced GUI library that is based on the Qt framework. It is more powerful and flexible than Tkinter, but also more complex. Install via `pip install pyqt5`.
- **wxPython**: This is another popular GUI library that is based on the wxWidgets toolkit. It is similar in complexity to PyQt but has a different look and feel. Install via `pip install wxPython`.

There are many other GUI libraries available for Python, so you may want to explore other options as well. See for example this list.

In the past I would have recommended PySimpleGUI, but its author has recently changed the licensing terms to more restrictive ones; while still free to use for personal projects, it would not be my first choice.

How to create a simple GUI with Tkinter

In this simple example we will create a GUI with a button and a counter that displays the number of times the button has been clicked. The counter will change color every 10 clicks and the font size will increase with each click. (OK, this is a bit silly, but it's just an example!)

This works by generating a main window, and then "packing" (i.e. placing) widgets (buttons, labels, lists, etc.) into the window. The `root.mainloop()` command starts the main event loop, which listens for user input and updates the GUI accordingly.

```
import tkinter as tk
from random import choice

# Define the function that will be called when the button is clicked
def update_counter():
    count = int(label.cget("text")) + 1
    label.config(text=str(count))
    if count % 10 == 0:
        # Every 10 clicks change the color of the label to a random color
        colours = ["red", "blue", "green", "yellow", "purple", "orange"]

        label.config(fg=colours[choice(range(len(colours)))])
    # Make the font size proportional to the count
    label.config(font=("Arial", 40 + count))

# Create the main window, 600x400 pixels
root = tk.Tk()
root.geometry("600x400")
root.title("Update the counter!")

# Create a label to display the counter
label = tk.Label(root, text="0")
# Make the font bigger.
# The config method is used to change the properties of the widget,
# such as the text, font, color, etc.
label.config(font=("Arial", 40))
label.pack()

# Create a button.
# When the button is clicked, the update_counter function will be called
# Note that we don't use parentheses when specifying the function name!
# This is because we are passing the function itself,
# not the result of calling the function.
button = tk.Button(root, text="Click me!", command=update_counter)
button.pack()

# Start the main event loop
root.mainloop()
```

`tkinter` can do much more! You can create more complex layouts, add images, create menus, and much more. You can find more information in the official documentation and in many tutorials online.

Creating a GUI with PyQt

We will now rewrite the previous example using PyQt. The code is a bit more complex, but PyQt offers more flexibility and power than Tkinter.
In this case we derive a child class from `QWidget` (the main PyQt widget class) and define the GUI elements in the `initUI` method. We then create an instance of this class and start the application event loop.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QPushButton
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt
from random import choice

class CounterApp(QWidget):
    def __init__(self):
        super().__init__()
        self.count = 0
        self.initUI()

    def initUI(self):
        self.setGeometry(100, 100, 600, 400)
        self.setWindowTitle("Update the counter!")

        self.label = QLabel(self)
        self.label.setText("0")
        self.label.setFont(QFont("Arial", 40))
        self.label.setAlignment(Qt.AlignCenter)
        self.label.setGeometry(200, 100, 200, 100)

        self.button = QPushButton("Click me!", self)
        self.button.setGeometry(250, 200, 100, 50)
        self.button.clicked.connect(self.update_counter)

        self.show()

    def update_counter(self):
        self.count += 1
        self.label.setText(str(self.count))
        if self.count % 10 == 0:
            colours = ["red", "blue", "green", "yellow", "purple", "orange"]
            self.label.setStyleSheet(f"color: {choice(colours)}")
        self.label.setFont(QFont("Arial", 40 + self.count))

# Create a QApplication instance (required for PyQt)
# We pass the command line arguments to the QApplication constructor
# through sys.argv. This is not necessary for this example, but it is good
practice.
# Alternatively, you can use an empty list: app = QApplication([])
app = QApplication(sys.argv)
# Create an instance of the CounterApp class (our main window)
ex = CounterApp()
```

```
# Start the application event loop
app.exec_()
```