Mr. Powell

Chapter 8 Exercises

**Section 8-1**

**Page 253**

**1. Describe two fundamental differences between terminal-based user interfaces**

**and GUIs**.

A terminal-based program maintains constant control over the interactions with the user. Put differently, a terminal-based program prompts users to enter successive inputs, whereas a GUI program puts users in change, allowing them to enter inputs in any order and waiting for them to press a command button or select a menu option.

**2. Give an example of one application for which a terminal-based user interface is**

**adequate and one example that lends itself best to a GUI**.

A terminal based user interface is good for text messages, while a GUI is good for images.

**Section 8-2**

**Page 253**

**1. Describe what usually happens in the \_\_init\_\_ method of a main window class**.

This method is automatically run when the window is created. The \_\_init\_\_ method runs a method

with the same name on the EasyFrame class and then sets up any window components to display in the window

**2. Explain why it’s a good idea to make a new class a subclass of an existing class**.

When you make a new class a subclass of another class, your new class inherits and thereby acquires the attributes and behavior defined by its parent class, and any of its ancestor classes, for free. Subclassing and inheritance are thus useful abstraction mechanisms, in that you do not have to reinvent the entire wheel when defining a new class of objects, but only customize it a bit.

**Section 8-3**

**Page 259**

**1. Write a code segment that centers the labels RED, WHITE, and BLUE vertically in a GUI window. The text of each label should have the color that it names, and the window’s background color should be green. The background color of each label should also be green**.

#------------- gui.py ----------

import tkinter as tk

class GUI:

def \_\_init\_\_(self,master):

self.master = master

#create 3 labels

#tkinter by default centers the compoenents.

self.red = tk.Label(self.master,text = "RED",fg="red",bg='green')

self.white = tk.Label(self.master,text = "WHITE",fg="white",bg='green')

self.blue = tk.Label(self.master,text = "BLUE",fg="blue",bg='green')

self.red.config(font=("Courier", 18))

self.white.config(font=("Courier", 18))

self.blue.config(font=("Courier", 18))

self.red.pack()

self.white.pack()

self.blue.pack()

root = tk.Tk()

gui = GUI(root)

root.configure(background='green')

root.mainloop()

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()

A picture containing drawing

Description automatically generated

**2. Run the demo program fontdemo.py to explore the font families available on your system. Then write a code segment that centers the labels COURIER, HELVETICA, and TIMES horizontally in a GUI window. The text of each label should be the name of a font family. Substitute a different font family if necessary**

. from tkinter import \*

from tkinter import font

window = Tk()

window.title("Font-Family")

window.geometry("450x450")

# Prints all the font families available with the system

print(font.families())

# Label 1 for Courier font family

font\_label1 = Label(window, text="COURIER", fg="black", font="Courier")

font\_label1.config(anchor=CENTER)

font\_label1.pack()

# Label 2 for Helvetica font family

font\_label2 = Label(window, text="HELVETICA", fg="black", font="Helvetica")

font\_label2.config(anchor=CENTER)

font\_label2.pack()

# Label 3 for Times font family

font\_label3 = Label(window, text="TIMES", fg="black", font="Times")

font\_label3.config(anchor=CENTER)

font\_label3.pack()

window.mainloop()

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()

**3. Write a code segment that uses a loop to create and place nine labels into a 3-by-3 grid. The text of each label should be its coordinates in the grid, starting with (0, 0) in the upper left corner. Each label should be centered in its grid cell. You should use a nested for loop in your code**.

# List to store text values

grid=list()

# LOOP FOR 3 ROWS

for i in range(3):

# LIST TO STORE COLUMN

row = list()

# LOOP FOR 3 COLUMNS

for j in range(3):

# CREATING TEXT WITH

# COORDINATES FORMAT

text = "({},{})".format(i,j)

# APPEND TEXT TO ROW

row.append(text)

# APPEND ROW TO GRID

grid.append(row)

# AFTER THE ABOVE LOOP

# GRID WILL BE CREATED IN TO

# 3 X 3 GRID

# PRINTING GRID

# FOR ALL ROWS

for i in grid:

# FOR ALL COLUMNS

for j in i:

# PRINTING VALUE AT

# ROW I, COLUMN J

print(j,end=" ")

print()

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a cell phone

Description automatically generated]()

**4. Jill has a plan for a window layout with two rows of widgets. The first row contains two widgets, and the second row contains four widgets. Describe how she can align the widgets so that they are evenly spaced in each row**.

Grid will be 2 rows and 4 columns. She should center the widgets with a Sticky attribute (NSEW) and then add the widgets in the first row to columns 0 and 2 with columnspan of 2

**5. Describe the procedure for setting up the display of an image in a window**.

A PhotoImage object (imported from tkinter.font). Must be loaded from a GIF file.

The image label is first added to the window with an empty text string. The program then creates a PhotoImage object from an image file and sets the image attribute of the image label to this object. Note that the variable used to hold the reference to the image must be an instance variable (prefixed by self), rather than a temporary variable. The image file must be in GIF format. Lastly, the program creates a Font object with a non-standard font and resets the text label’s font and foreground attributes to obtain the caption. The window is shrink-wrapped around the two labels and its dimensions are fixed.

**Section 8-4**

**Page 262**

**1. Explain what happens when a user clicks a command button in a fully functioning GUI program**.

The event handler method associated with that button is initialized. The code in that method is executed

**2. Why is it a good idea to write and test the code for laying out a window’s components before you add the methods that perform computations in response to events**?

To see what it will look like and get to test the GUI. Testing it will make sure it is working properly and if need be add, remove or change code until it is functioning the way you want it.

**Section 8-5**

**Page 267**

**1. Explain why you would not use a text field to perform input and output of numbers**.

Because you would have to change the data from text to numbers and you input it and vice versa for output.

**2. Write a line of code that adds a FloatField to a window, at position (1, 1) in the grid, with an initial value of 0.0, a width of 15, and a precision of 2**.

**addFloatField(value = 0.0, row = 1, column = 1, width = 15, precision = 2)**

**3. What happens when you enter a number with a decimal point into an IntegerField**?

An exception is raised when you do that in a FloatField. You’ll have errors when you run the getNumber

**4. When would you make a data field read-only, and how would you do this**?

You would make a data field read only for output. Set the field’s state attribute to “readonly”

**5. Explain what happens when a program receives a non-numeric string when a number is expected as input, and explain how the try-except statement can be of use in this situation**.

If the input text is not an integer or is a negative integer, Python raises an exception and, if the program is terminal based, it crashes (you learned about exceptions, like dividing by zero and using an index out of range, in earlier chapters). However, when a GUI-based program raises an exception, the GUI stays alive, allowing the user to edit the input and continue, but a stack trace appears in the terminal window. We next examine how to trap such errors and respond gracefully with error messages

**Section 8-6**

**Page 269**

**1. What is meant by the state of an object, and how does the programmer access and manipulate it**?

The values of an object’s instance variables make up its **state**. The state of a given window, for example, includes its title, background color, and dimensions, among other things. You have seen that a dictionary maintains these data within the window object. The window class’s \_\_init\_\_ method establishes the initial state of a window object when it is created, and other methods within that class are run to access or modify this state (to make the window larger, change its title, or respond to an event). These basic elements of a window’s state are defined and managed in the classes breezypythongui.EasyFrame and tkinter.frame.

**2. Explain the differences between instance variables and temporary variables. Focus on their visibility in a class definition, and on their roles in managing data for an object of that class**.

When you customize an existing class, you can add to the state of its objects by including new instance variables. You define these new variables, which must begin with the name self, within the class’s \_\_init\_\_ method. They then become visible to other methods throughout the class definition.

**3. Explain the purpose of the variable self in a Python class definition**.

A memory location, referenced by an identifier, whose value can be changed during execution of a program.