**Chapter 8 – Week 11 – Exercises**

Exercises #1 – page 249

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

Two fundamental differences between terminal-based applications and GUI base applications are as follows:

1-Terminal based applications are nearly always linear. This means that actions must occur in a certain order and user input goes in a specified order. GUI applications, on the other hand, are event-driven. These events are often supplied by the user. This means the user may update the fields on the screen in the order s/he sees fit. It also means that application code is often started by a user mouse click allowing the user to cause code to be executed in the order s/he desires.

2-Terminal based applications are text-based applications. All information from the program is in the form text. GUI applications on the other hand are graphical in nature. This allows images to be part of the user interface. These images are often small like icons but can be of any size.

1. **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.**

The unix utility grep is an example of a terminal-based user interface. The grep command will search a section of text (from a file of from the standard input) for a supplied pattern. When a match is found, it is displayed to the console for the user to see.

A word processor is an example of an application that lends itself to a graphical interface. Even before graphical interfaces were possible, word processors did their best to emulate them. This is because the user must be able to distinguish things like **bold** and *italic* text from plain text. A word processor is not much use without such functionality.

Exercises #2 – page 253

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

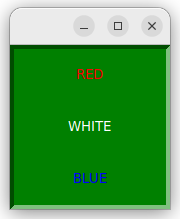
The \_\_init\_\_ method of a main window usually initializes and populates the window. This means it sets the size, color scheme, window title, etc. for the window and then it adds the widgets that will appear in the window such as labels, text boxes, buttons, and other similar items.

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

While it not always a good idea to make a new class a subclass of an already existing class because it can to lead to complicated inheritance structures and code that is more difficult to debug, it often is. The advantage of sub-classing an existing class is that all of its state and behavior are available to the subclass. This prevents code duplication and speeds the development process along.

Exercises #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.**



from breezypythongui import EasyFrame  
  
class PatrioticLabels(EasyFrame):  
 *""" Display a greeting in a window. """*  
  
def \_\_init\_\_(self):  
 *""" Sets up the window and the label. """*  
EasyFrame.\_\_init\_\_(self)  
 self["background"]="green"  
 self.addLabel(text="RED",  
 row=0,  
 column=0,  
 sticky="NSEW",  
 foreground="red",  
 background="green"  
 )  
 self.addLabel(text="WHITE",  
 row=1,  
 column=0,  
 sticky="NSEW",  
 foreground="white",  
 background="green"  
 )  
 self.addLabel(text="BLUE",  
 row=2,  
 column=0,  
 sticky="NSEW",  
 foreground="blue",  
 background="green"  
 )  
  
def main():  
 *""" Instantiate and pop up window """*  
PatrioticLabels().mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

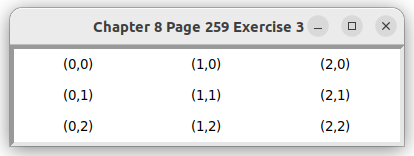
1. **Run the demo program fontdemo.py to explore the font families available on your system. Then write a code segment that centers the labels COURIER, HELVETICA, and TIMES horizontally in a GUI window. The text of each label should be the name of a font family. Substitute a different font family if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

IMPORTANT NOTE: I could find no file fontdemo.py in the data-files from this course. I also did an Internet search which returned some fontdemo.py files, but none related to our text.



from breezypythongui import EasyFrame  
from tkinter.font import Font  
  
class ImageDemo(EasyFrame):  
 *"""Displays an image and a caption."""*  
  
def \_\_init\_\_(self):  
 *"""Sets up the window and widgets."""*  
EasyFrame.\_\_init\_\_(self, title = "Chapter 8 Exercise 2")  
  
 # Rasa text label  
 textLabel = self.addLabel(text = "Rasa",  
 row = 0, column = 0,  
 sticky = "NSEW")  
 font = Font(family = "Courier", size = 20)  
 textLabel["font"] = font  
 textLabel["foreground"] = "black"  
  
 # FreeSerif text label  
 textLabel = self.addLabel(text = "FreeSerif",  
 row = 0, column = 1,  
 sticky = "NSEW")  
 font = Font(family = "FreeSerif", size = 20)  
 textLabel["font"] = font  
 textLabel["foreground"] = "blue"  
  
 # FreeSans text label  
 textLabel = self.addLabel(text = "FreeSans",  
 row = 0, column = 2,  
 sticky = "NSEW")  
 font = Font(family = "FreeSans", size = 20)  
 textLabel["font"] = font  
 textLabel["foreground"] = "purple"  
  
def main():  
 *"""Instantiates and pops up the window."""*  
ImageDemo().mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **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.**



from breezypythongui import EasyFrame  
from tkinter.font import Font  
  
class NineLabelsFrame(EasyFrame):  
 def \_\_init\_\_(self):  
 *"""Sets up the window and widgets."""*  
EasyFrame.\_\_init\_\_(self, title = "Chapter 8 Page 259 Exercise 3")  
  
 for row in range(3):  
 for col in range(3):  
 txt = "(" + str(col) + "," + str(row) + ")"  
  
 self.addLabel(text = txt,  
 row = row,  
 column = col,  
 sticky = "NSEW"  
 )  
def main():  
 *"""Instantiates and pops up the window."""*  
NineLabelsFrame().mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **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.**

Jill can use colspan=2 for the items in the first row to allow them to span 2 columns each allowing the first row to be evenly spaced.

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

To setup to display an image in a window the programmer must follow this 3 step process:

* + Create a label
  + Load the image from the disk into an image object
  + Set the image attribute of the label the loaded image object

Exercises #4 – page 262

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

When a user clicks a command button in a working GUI program, the Operating System receives the click as an interrupt from the mouse. It determines which window should receive the click based on the location of the pointer and the visibility of the windows. It then sends a click message to the Window/Frame in question. This Window will handle the event according to its programming. Generally, this means calling the method associated with the button’s click event.

1. **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?**

It is a good idea to examine your interface before adding code because you may see missing or extra behavior necessary to implementing the interface. In such a case, you could end up doing unnecessary coding or end up with extraneous or confusing code.

Exercises #5 – page 267

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

Using a text field for the input and output of numbers is not appropriate because the text must be converted into a numeric value. Additionally, the text could be set to a non-numeric value which would be a confusing user experience.

1. **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.**

self.addFloatField(value=0.0,  
 row = 1,  
 column = 1,  
 precision=2  
 )

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

When a decimal point is entered into an IntegerField, an error such as this one will be received when a call to getNumber is made:

ValueError: invalid literal for int() with base 10: '5.0'

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

A data field should be set to read-only when the user should not be able to edit the value stored in the field. This can be done by including a state parameter with a value of “readonly” when creating the field.

1. **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.**

As discussed in question 3, when a program receives a non-numeric string when a number is expected an exception is thrown.

The try … except statement can be used to react to these exceptions in a way that is appropriate under the circumstances. For example, a message box could be displayed stating the fact that the user entered incorrect data.

Exercises #6 – page 269

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

The state of an object is the collection of its instance variables. (See pg 267 of text)

1. **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.**

Temporary variables exist (have a lifetime) during a single function execution while instance variables continue to exist as long as the object containing it exists.

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

The purpose of self is to allow a method to access the instance variables associated with a particular instance of a class. This instance is known as an object, and the state of each object must be maintained independently of each other. When a method is called, it is called “on” a particular object and **self** allows Python to know which set of instance variables to use.