Unit 18 Exercises

Aims and objectives

- Further practice in planning and program development
- Further practice in GUI programming

Summary

This exercise is based on an audience response system similar to the one used in CS1P lectures. With this system, each student has a handset that can be used to select one from up to 10 possibilities in order to answer a multiple-choice question. During a lecture, a number of questions may be asked. The responses to each question are stored in a file for later analysis. The aim of this exercise is to write a program that manipulates response data gathered during a lecture.

The student responses for a sequence of questions are stored in one file, and details such as the correct answer are stored in a second file. The program should read in the contents of these files and then display summary data for each question, using both textual and graphical output.

Detailed explanation

The two files `Responses.txt` and `QuestionInfo.txt` hold data for any number of questions, but are consistent in that if the former contains responses for five questions, then the latter will contain information about five questions also.

The Responses file

A text file named `Responses.txt` contains a log of handset responses for a given lecture. Each line of the file represents a single response from a student and is formatted as follows:

```
<Question_No><space><Handset_ID><space><Response>
```

`Question_No` holds the number of the question for which this is a response. `Handset_ID` uniquely identifies the handset that generated this response. `Response` is the number of the button that was pressed to generate the response.

Responses are in the range 0-9. A response of 0 means 10. A question can have up to 10 possible answers, but the handsets use 0 to represent 10. You will see in the example output below that answers should be shown from 1 to 9 and then 0. Valid handset IDs lie in the range 1-160. Questions are numbered from 1, and all the responses for one question will appear before the responses for the next question.

Not all handsets need to provide a response for every question – a student may have chosen not to answer a question. A given handset can generate more than one response for a given question, *but only the last response should be counted*.

You can assume that the data in this file is correctly formatted.

A typical (albeit very small!) responses file might look as follows:

```
1 23 1
1 18 1
1 28 2
1 24 1
1 12 3
1 26 3
1 29 1
1 23 5
2 23 3
3 17 1
```

The Question Information file

A second text file named `QuestionInfo.txt` contains details about each question asked in the lecture. Each line of this file represents data about one question and is formatted as follows:

```
<Correct_Answer><space><Number_Answers>
```

The questions are numbered implicitly according to where they appear in the file. Hence the data on the first line is for question 1, on the second line for question 2, and so on. `Correct_Answer` holds the number of the correct answer for this question. `Number_Answers` gives the number of valid answer options for this question – remember that most questions have less than the maximum 10 possible answer options.

A sample question information file corresponding with the sample responses file given above is as follows:

```
2 5
3 3
0 10
```

Calculations and output

The program should read in and process the data stored in the two files in order to produce the following output.

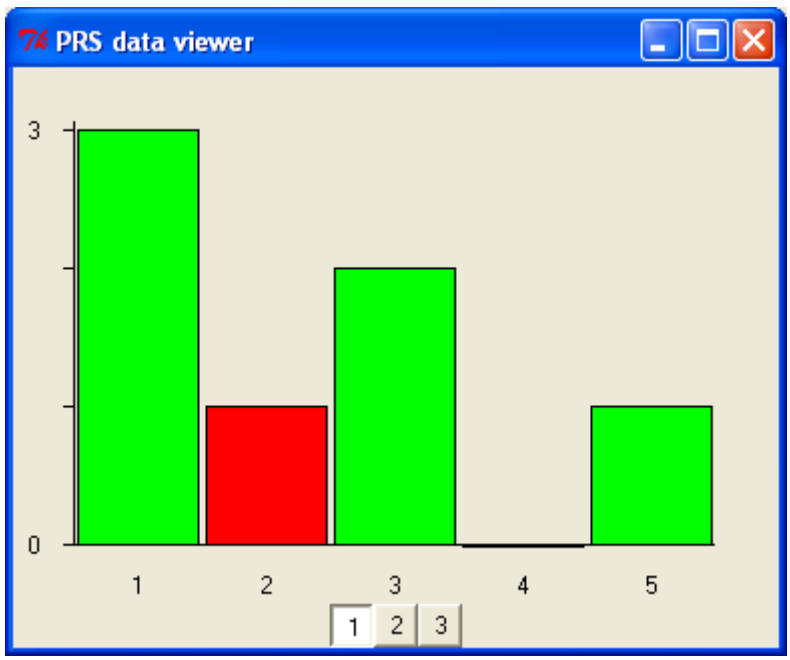
Task 1

Write out summary data indicating how many responses were given for each valid answer, ignoring invalid answers. The correct answer should be highlighted. The format to be used should be as follows, using the sample data given above:

```
Question 1
1:3 **2:1** 3:2 4:0 5:1
Question 2
1:0 2:0 **3:1**
Question 3
1:1 2:0 3:0 4:0 5:0 6:0 7:0 8:0 9:0 **0:0**
```

Task 2

By extending the code provided in the file `GUI.py`, you should extend your program to display a window that allows the data for each question to be displayed as a bar chart. For example:



Note that the set of radio buttons at the bottom allows the particular question to be viewed to be chosen – in the picture, the first button is selected and so a chart for the first question is displayed. Also, the correct answer is shown in a different colour.

ProcessResponses.py, is a skeleton program provided for this exercise containing only the relevant import statement and program heading. In addition, GUI.py is provided, along with the sample input files given here.

Note that in order to do Task 2, you will need to understand `GUI.py` thoroughly. The modifications you make to `GUI.py` will probably include some or all of: adding code to existing functions; adding extra parameters to existing functions; defining new functions.

ProcessResponses.py skeleton program

```
# Read in and display voting responses
# Name:
# Date:

from GUI import *

# For Part 1, add code here to process the files and produce the
# textual output.

########

# For Part 2, uncomment the line below (with suitable modifications) to
# start the graphical user interface.

# application(...)    # The "..." indicates that parameters will be needed
```

GUI.py program

```
from Tkinter import *

# First a function to draw a chart of data onto the supplied canvas, can
# labels is a list of strings for the x-axis labels
# data is a list of values to be used for the bars on the chart

def chart( can, labels, data ):
    # Delete all before starting
    l = can.find_all()
    for i in l:
        can.delete( i )

    # Calculate the size for the barchart
    maxX = can.winfo_width()
    maxY = can.winfo_height()
    xOrig = 30
    yOrig = maxY - 30
    xSize = maxX - 2*30
    ySize = maxY - 2*30

    # Find the largest data value, to calculate scaling
    maxV = data[ 0 ]
    for i in data:
        if i > maxV:
            maxV = i

    yScale = ySize / maxV
    xScale = xSize / len( data )

    # Draw the axes
    can.create_line( xOrig, yOrig, xOrig+xScale*len(data),yOrig )
    can.create_line( xOrig, yOrig, xOrig,yOrig-yScale*maxV-5 )
```

```
    for i in range( len( data ) ):
        can.create_text( xOrig+xScale*i+xScale/2,yOrig+20,text=labels[ i ])

    for i in range( maxV + 1 ):
        can.create_line(xOrig-5,yOrig-yScale*i,xOrig,yOrig-yScale*i )

    can.create_text(xOrig-20,yOrig,text="0")
    can.create_text(xOrig-20,yOrig-yScale*maxV,text=str(maxV))

    # Draw the bars
    for i in range( len(data) ):
        can.create_rectangle(xOrig+xScale*i+2,yOrig-yScale*data[i],
                             xOrig+xScale*i+xScale-2,yOrig,fill="green")


# This sets up a window with a canvas for your bar chart to be drawn in,
# and just now a single radio button

def application():
    # Graphical user interface
    root = Tk()                     # Top level window
    root.title("PRS data viewer")

    ### Now create a frame to hold the various parts of the application
    grapher = Frame( root )
    grapher.grid()

    # Put a canvas into it to draw on
    graphCan = Canvas( grapher )    # Note, this is a Tkinter canvas!!
    graphCan.grid()

    ### Now create a set of radio buttons along the bottom
    ### for choosing which question to view
    ### Put them in a new frame to get a tidy layout
    buttons = Frame( root )  # This is to hold the buttons
    buttons.grid()           # Display it under the canvas
    iv = IntVar()            # To hold the value representing which radio
                             # button is pressed

    def rbAction():
        q = iv.get()
        graphCan.create_text( 100,100, text=str( q ) )

    # Make one button
    r = Radiobutton( buttons, value=0, variable=iv, indicatoron=0,
                     text=' '+str( 1 )+' ', command=rbAction )
    r.grid()

    root.mainloop()
```