CSSE1001

**Semester 1, 2011**

**Assignment 2**

**20 marks**

**Due Thursday 19 May, 2011, 5pm**

**A GUI for Plotting Functions**

# 1 Introduction

For this assignment you will write a GUI for plotting functions. The user will supply the definition of a function in the variable $x$, the range of $x$ and $y$ values to be plotted and the number of steps in the plot. The GUI will then display the function. The user can enter any number of functions. Each will be plotted in the chosen colour. The plot information can also be changed - i.e. the $x$ and $y$ ranges and the number of steps. All the functions can then be redrawn using the new setup.

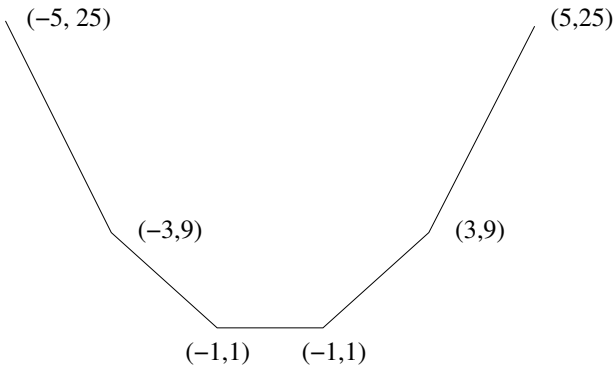
One way to plot a function is to construct a sequence

$< (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) >$

where $x_1$ is the smallest value of $x$, $x_n$ is the largest value of $x$, $n$ is the number of steps, each $y_i$ is the result of applying the function to $x_i$ and all the $x_i$ are equally spaced. The lines from $(x_i, y_i)$ to $(x_{i+1}, y_{i+1})$ are then drawn to produce an approximation to the function. The larger the number of steps, the better the approximation.

Typically though, the sequence is not really constructed. All that is needed is the pairs of adjacent points so that a line can be drawn between them. The assignment support file defines an iterator that generates the required sequence.

Below is an example of plotting the function $x^2$ from -5 to 5 in 6 steps. First a line from $(-5, 25)$ to $(-3, 9)$ is drawn followed by the line from $(-3, 9)$ to $(-1, 1)$, followed by the line from $(-1, 1)$ to $(1, 1)$, followed by the line from $(1, 1)$ to $(3, 9)$, followed by the line from $(3, 9)$ to $(5, 25)$.




For the actual plotting you will be using a canvas widget. The canvas has a given width and height while the plotting parameters specify ranges for $x$ and $y$. Therefore a conversion is required that map 'real world' coordinates (i.e. the $x$,$y$ values) to screen (canvas) coordinates. Remember that in the 'real world' $y$ values increase as you move up while on a canvas they decrease as you move up. The assignment support file defines a coordinates conversion class that does this job.

# 2 Assignment Tasks

For each class and method that you write you need to provide a suitable comment giving a description and where necessary the type and any preconditions. You should use the triple-quote commenting style.

## 2.1 Download file

The file `assign2.py` is for your assignment. Add your name and student number in the space provided. When you have completed your assignment you will submit the file `assign2.py` containing your solution to the assignment.

The file already contains some code (in two parts). Do not modify any of this code! The first part should be left at the beginning of the file and consists of an import statement. The second part should appear at the end of your code. **NOTE:** You will lose marks if you don't follow these instructions.

## 2.2 Read the support file carefully

We have supplied you with some code in `assign2_support.py` to be used in your assignment. You need to understand how to make use of this code.

## 2.3 Write the code

Finally, write your solution to the assignment making sure you have included suitable comments. Your solution should include at least the following classes.

## 2.4 PointFrame Class

This class defines a display widget for showing the cursor position of the last left mouse click as a point and the current cursor position as a point, both in real-world coordinates.

## 2.5 FunctionFrame Class

The required GUI has several parts. This class, which inherits from the Frame class, defines a widget for entering the function definition and choosing the plot colour for the function. This class should define a method that returns the function (as a function not a string) and the colour used to draw the function. This method should report an error in an error message box (`tkMessageBox.showerror`) if either no function is entered or the function entered is not a valid function of $x$. You also need to deal with invalid colours by producing an error message. The easiest way to do this is to trap errors when trying to draw a line.

## 2.6 PlotFrame Class

This class defines a widget for entering the plot information for all the functions that are to be drawn. It also inherits from the Frame class. This class should define a method that returns the plot information as a tuple to be used

for plotting the function. Error checking should be carried out and any error should be reported in an error message box (`tkMessageBox.showerror`). The possible errors are:

- An $x$ or $y$ value is not a float.
- The number of steps is not an integer.
- The minimum $x$ is bigger than the maximum $x$.
- The minimum $y$ is bigger than the maximum $y$.
- The number of steps is not positive.

## 2.7 ButtonFrame Class

This class, inheriting from the Frame class, defines a widget that contains a collection of buttons. The buttons are:

- `Add Function`: add the given function to the collection of functions and redraw all the functions (should do nothing if there is an error).
- `Redraw All Functions`: redraw all the functions (do nothing if there is an error).
- `Remove Last Function`: remove the last entered function and redraw all remaining functions. Do nothing if there are no functions to remove.
- `Remove All Functions`: remove all functions and clear the canvas.
- `Exit`: exit the application.

## 2.8 PlotApp Class

This is the top-level plot application class. It is responsible for interacting with the user and plotting the functions.

When the application is resized, all functions should be redrawn using the new width and height of the canvas. As a hint for managing the application window try something like

```
master.minsize(700,480)
master.geometry("800x600")
self.canvas.bind("<Configure>", self.resize)
```

When at least one function is drawn then the `PointFrame` should display the point information.

## 2.9 Examples

The course web page contains the following examples.

- A screenshot showing a plot of the function $x^2$ in red (entered as `x**2` or `x*x`) and $sin(x)$ in blue (entered as `sin(x)`).
- A screencast showing the application in action.

## 2.10 Look and Feel

GUIs look different on different operating systems but you should at least satisfy the following criteria.

- The widgets should be in the following top-to-bottom order: the point frame, the canvas, the function frame, the plot frame, and the button frame.
- The background for the Function frame and the Plot frame should be the same colour and different from the overall background colour.
- All the buttons should have the same colour and be different from the other colours.
- the background of the canvas should be white.
- The canvas and the Function and Plot frames should be sunken.
- There should be some space to the sides of the canvas.

- There should be some vertical spacing between the widgets.
- The buttons should be centered in the Button frame.
- The function colour and select button should be right justified in the function frame while the function part should be left justified.

### 2.11 Hints

For colour selection consider `tkColorChooser`.

If something changes and so some sort of redrawing is required, the simplest thing to do is to delete everything and then redraw everything.

When drawing a function, the function iterator will return the sequence of points. You need to draw a line between two adjacent points so you need to 'get things started'. One way to do this is to start with some point to the left of the canvas and draw the first line between that point and the first point of the sequence. Another way is to use a boolean flag to avoid drawing a line until you have the second point.

For getting the frames to look sunken you will need to set the border size (`bd=2`). It seems that, because of the border setting, when you resize the application and want to recompute the new canvas size you will end up writing something like the following. Some experimentation might be required on your operating system in order to get the right values.

```
self.width = e.width
self.height = e.height-2
```

To exit the application you can use something like `master.destroy()`.

## 3 Marking Criteria

We will mark your assignment according to the following criteria.

| Criteria | Mark |
|---|---|
| Comments: | 3 |
| - comments that are complete, consistent, clear and succinct | 3 |
| - comments with some minor problems | 2 |
| - comments with major problems | 1 |
| - work with little or no academic merit | 0 |
| Code: | 17 |
| - code that is mostly complete, correct, clear and succinct | 12 - 17 |
| - code with a number of problems | 6 - 11 |
| - code that is clearly incomplete, incorrect, too complex or hard to understand | 1 - 5 |
| - work with little or no academic merit | 0 |
| Total marks | 20 |

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out that code and we will mark that.

Please read the section in the course profile about plagiarism.

# 4 Assignment Submission

You must submit your completed assignment electronically through the website: `http://submit.itee.uq.edu.au`

Please read `http://submit.itee.uq.edu.au/student-guide.pdf` for information on using electronic submission.

You should electronically submit your copy of the file `assign2.py` (use this name - all lower case).

You may submit your assignment multiple times before the deadline - only the last submission will be marked.

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on-time, you should contact the lecturer in charge and be prepared to supply appropriate documentary evidence. You should be prepared to submit whatever work you have completed at the deadline, if

required. Requests for extensions should be made as soon as possible, and preferably before the assignment due date.