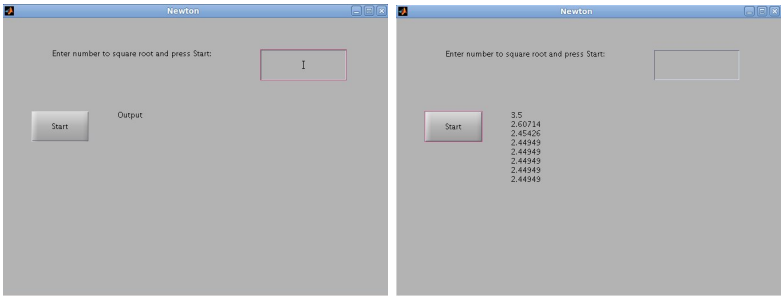
University of Pennsylvania
EAS 105: Fall 2011
MATLAB PROJECT: Perpetual Calendar
Instructor: Donna Dietz

In this project, you will build a perpetual calendar which is accurate between 1800 and 2099. In other words, you will be able to give the correct day of the week for any date of any year in that range. Techniques will be discussed in class.

This project is the second of three progressive projects you will do this semester. In this part of the project, you will use GUIDE get a gui (graphical user interface) up and running. (GUIDE is the GUI Development Environment.)

Before you start work on this project, take some time to play with the **Newton.m** and **Newton.fig** files which are included with the p-code files for this project. The explanation for this code begins on p.297 of your text. You need to be following along in your text anyhow, so while you're there, you should fix this typo: On page 299, the second last line should read *str{i}=sprintf('%g',x);* rather than using parenthesis around the *i*. This gui closes properly when the "x" in the upper right corner is clicked.

To build this Newton gui, open guide, and accept the defaults presented. (Do this in a new folder where the Newton I've provided does not exist.) I'm giving two views of Newton here, the first one is how Newton looks before the user touches it, and the second shows Newton after calculations have been made.



Design the pushbutton, two static text controls and the edit text control within guide. Then, right click anywhere in guide to get Property Inspector for a particular control (or for the main gui body). By adjusting the various items inside, you can adjust all sorts of things such as color, initial string to display, and so forth. You can make the main gui resizable by selecting Tools in the Layout Editor's toolbar, and then GUI Options.

There are a few ways you can get information into and out of the gui. To get variables into the gui, you can pass them when the gui is called. For example, in QuizMe, the auto-generated code will produce this line: function varargout = QuizMe(varargin) And later, if you had sent any data structures in, you could retrieve them this way: [handles.setA,handles.setB,handles.scorecard]... =SetupTest(varargin{1},varargin{2}); Which is a line I put in my

QuizMe_OpeningFcn (after the initial auto-generation, of course). This, of course, brings us to the second main way to pass information, which is the *handles* structure. The name *handles* is special in Matlab, so don't use it unless you have to. In this case, it is the name of a cell-array which holds all the information you want to pass around between various parts of QuizMe. You can just keep growing this thing and keep passing it around. The auto-generated code brings it into each subfunction of QuizMe, and then you are supposed to send it back to the rest of QuizMe using this command: guidata(hObject, handles);

Last, but certainly not least, are the ever-present *get* and *set* commands! When the gui is running, all the properties which are listed in the Property Inspector can be retrieved or adjusted to your heart's content. So, for example,
set(handles.edit1,'String',' '); means to erase the default string on the edit1 text box. But, what if you wanted to see what the user put in that box? In that case, you'd use get(handles.edit1,'String'). (If it's a textbox, it will give you back a string.)

For this project, the following functions are required:

function [string] = IntToDayOfWeek(n) This function turns a number into a string. For example, 0 becomes 'Saturday', 1 becomes 'Sunday', and 6 becomes 'Friday'. This should be no longer than 30 lines long. (Wraparound lines should not be used for this project.)

function [boolean] = IsValidDate(year, month, date) returns 0 for invalid dates, and 1 for valid dates. It is only required to work between 1800 and 2099. Note that it is of utmost importance to figure out whether or not the year is a leap year! You are permitted to write a separate function to determine if a year is a leap year, if you desire. (If you do, it should not be longer than 20 lines long.) IsValidDate should not exceed 50 lines.

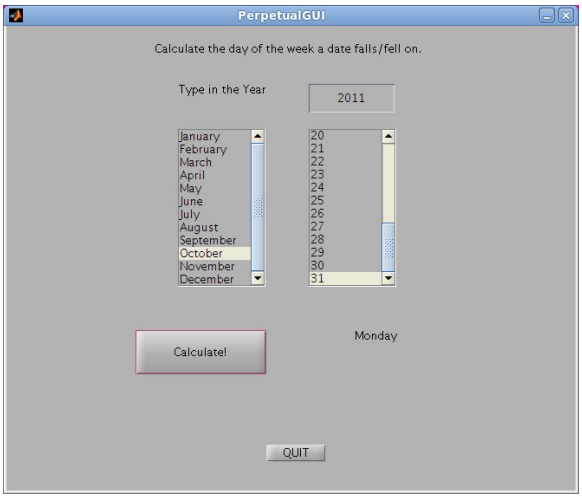
function [dayOfWeek] = ComputeDayOfWeek(year, month, date) returns the proper day of the week as a number. Again, you (probably) need to first determine whether or not the year is a leap year. This should in no case be longer than 50 lines long.

function varargout = PerpetualGUI(varargin) will be autogenerated by GUIDE, then edited by you. PerpetualGUI.fig will also be generated at the same time.

The tester only tests the first three functions. Your GUI will be tested by a TA with a sense of humor. Make the experience fun for your TA, who will otherwise be very bored!

This is a screenshot showing the components you will be required to have in your gui. You are encouraged to have fun beyond these requirements. For example, you may offer to find the next year after the chosen year in which a date occurs on a

given day of the week.



The required components (shown) are:

- One or more static text boxes for prompts or other instructions.
- A "Calculate" pushbutton
- A static text box for displaying day of week or error message
- Two Listboxes, one for the months and one for the dates
- A "Quit" pushbutton (You are also welcome to make this gui close smoothly upon clicking the upper right "x", but you are not obligated to do this.)

To begin your graphical layout, type *guide* in your command window, then accept the default options by clicking *ok*. The process has the following cycle: You arrange, organize, and basically do all your artistic work in this window, and finally you save your work. When your work is saved, various names become 'hardcoded' into your project, so pick carefully. Once you save your work, a **.m** file and a **.fig** file are created with the same name, and they must both be present for your project to work. (But, you will note that pcode files still replace m-files nicely!) To fix names of variable prior to this saving point, you can right-click on any areas in your composition and choose **Inspect Properties**. To save your work, click **Run Figure**[1], which is a small green triangle at the right-end of the Layout Editor toolbar.

Most of what you need to know about *guide* should be in your text on pages 292 - 303, however, there have been many changes to Matlab since this book was drafted! The thing to keep in mind is that if there is a block of code you don't understand, it's probably autogenerated code you are not supposed to understand. If Dr. Hahn says there is some line of code which was auto-generated and it's not in your code, don't worry. It's just the new version! For example, the statements to display 'Callback not implemented yet.' have been removed.

A couple things to note: *Application Options* is now called **GUI Options**. Also, the auto-generated code uses a protocol for functions we have not used, which is

[1]Formerly, *Activate Figure*

to leave the final 'end' off each function in a file. You can't mix the two protocols iwithin a single file, but it's fine to use our old protocol in all our other files, and just let the autogenerated code do its own thing for **QuizMe.m**. If you are trying to follow the examples in the text, they are very good!

Once you save your artwork, you will edit things within the **PerpetualGUI.m** file.Here, I will summarize the adjustments I made to the auto-generated code.
In PerpetualGUI_OpeningFcn, I uncommented out the uiwait line. All variables are of the form *handles.whatever*, so they can be passed to the next subfunction.
In PerpetualGUI_OutputFcn, I commented out the only actual line of code.

I put most of my acutal code into *calculateRequest_Callback*, excpet that you do need to actually close the gui. So, in quit_Callback I added close(gcf); to the end of the subfunction.