Name: ______________________________________

PIC 97, Winter 2016 – Midterm Exam 2

Administered 7:00 p.m. – 9:00 p.m. Wednesday, February 24, 2016. Corresponds with 4F-7F.
No outside resources other than features of Python software (Spyder, IPython, etc…), official Python, Plotly, SymPy, NumPy, SciPy, matplotlib, and PyQt4 documentation (found on their respective official websites) and the provided links (in the exam assignment page on CCLE) are allowed. You may use a search engine to find these pages, but visiting other websites (including those linked from the allowed websites, unless otherwise specified) is not acceptable.

Flip this sheet only when instructed.

(Until then, squint as much as you wish trying to read backwards through the page.)

The exam questions are contained in this document, but you are to submit your solution to Problem 1 in a Python module `pypaint.py` and to Problem 2 in a Jupyter notebook `bvp.ipynb`. For Problem 1, comments in your code indicating what part of the problem you are working on may help us assign partial credit. For Problem 2, include each part of the problem in a separate cell (labeled with the corresponding letter using a comment). Always include a print or similar statement that shows the requested results of your calculations. For instance, if the program asks you to produce an array, print some of it to show that it's correct. When you are done with the exam, submit `pypaint.py`, `bvp.ipynb`, and any other files (e.g. `.ui` file from Qt Designer, etc…) in a `.zip` file `exam2.zip` on CCLE. Of course, you are welcome to write code in any software installed on the PIC machines and copy it into the requested files when you're done. In fact, I recommend saving backups of all your code in a simple text document `backup.txt` and including that in your `.zip` file as well in case the primary files do not save properly.

If you can't do a problem, or you think it will take too long, find a way to move on. While the parts within a problem ideally work together to form a complete program, it is always possible to write correct code for subsequent parts and demonstrate its correctness even if previous parts are incorrect. Ask for suggestions during the exam for how to maximize partial credit if you find yourself stuck.

For now, log on to your computer, start Spyder and Jupyter, and get familiar with the allowed resources.

All parts of all problems are required, but partial credit is available. Read carefully to do the best you can!

The references linked from CCLE for the first midterm are still allowed and links are still available there. References more relevant to this exam are:

Plotly User Guide (https://plot.ly/python/user-guide/)[1]
Python Object Oriented (http://www.tutorialspoint.com/python/python_classes_objects.htm)
PyQt4 Tutorial (http://zetcode.com/gui/pyqt4/) – *direct* links from this page are OK
Let's Learn Python #24 (https://www.youtube.com/watch?v=GLqrzLIIW2E)
Getting Started with PyQt and Qt Designer (https://nikolak.com/pyqt-qt-designer-getting-started/)
SymPy Tutorial (http://docs.sympy.org/latest/tutorial/intro.html#what-is-symbolic-computation)[1]
SciPy Lectures (http://www.scipy-lectures.org/intro/index.html)[1]

Good luck!

[1] Any documents at plot.ly, sympy.org, and scipy-lectures.org are acceptable

Name: ______________________________________

1. Write a program `pypaint.py` in Spyder that behaves as demonstrated in this (silent) video[2]. For maximum credit, please attempt to complete the features in the following order:
    a. 10 pts - Create a white drawing area on the left and a vertical toolstrip on the right consisting of two buttons ("Clear" and "Select Color") and two radio buttons ("Square", which is selected by default, and "Circle"). When the user resizes the window, entitled "PyPaint", the buttons should stay centered vertically and aligned horizontally (with one another) toward the right of the window, but should not change size. Rather the white drawing area alone should resize to fill available space.
    b. 10 pts - Draw a black square 100px x 100px centered in the drawing area. If you can't do part c of this problem, turn in your program such that the centered black square is drawn when the program is run. If you can do part c, just *comment out* the code that draws this square.
    c. 20 pts - Clicking the mouse in the drawing area should make a black square (~10px) appear *centered* at the cursor location. Clicking and dragging the mouse in the drawing area should make black squares appear at each point the mouse passes through (provided that the mouse is not moving too fast. It is OK if points are skipped when the mouse moves quickly. We could fix this by drawing lines between the points, right? But the problem is solved in principle, so don't worry about it during the exam). Should the user resize the window, points still within the drawing area should remain visible. Those outside the drawing area should disappear, but should come back if the drawing area grows to encompass them again.
    d. 4 pts - Implement the functionality of the "Clear" button. The drawing area should be cleared completely, and the user should be able to draw again on the blank drawing area.
    e. 3 pts - Implement the functionality of the "Select Color" button. When clicked, the user should be prompted with a standard color selection dialog. The chosen color should not be applied to existing points, only those drawn *after* it is selected.
    f. 3 pts - Implement the functionality of the radio buttons. When the "square" radio button is selected, the mouse draws small squares (as before); when the "circle" radio button is selected, the mouse draws small circles.

2. Let's have some fun with the finite difference method of solving boundary value problems! No prior knowledge of this technique is needed (or helpful, really). This problem doesn't test the math, just the Python, so follow the steps[3]. Please heed the required variable names to maximize partial credit. Consider the function $f(x) = x \sin x^2$. In a Jupyter notebook `bvp.ipynb`,
    a. Evaluate the indefinite integral $F(x) = \int f(x)dx$ and store the resulting expression as `F`.
    b. Evaluate the derivative $f'(x) = \frac{df(x)}{dx}$ and store the resulting expression as `df`.
    c. "Lambdify" the expression for $F(x)$ to create a function `int_f(x)` that accepts a NumPy array as input and returns the corresponding function values for each value in the array. Likewise, lambdify $f'(x)$ to create a function `diff_f(x)`. Yes, this was in the tutorial.
    d. Generate a NumPy array `x` containing $N = 1000$ equally-spaced values on the interval $[0, 2\pi]$ (inclusive). Also, calculate the difference `delta` between any two successive values of the array `x`. I will refer to this value (a single number, not an array) using the symbol Δ in the following steps.

[2] Also available streaming from https://youtu.be/2u9712tdRi0.
[3] Each part is worth 4 pts up to a maximum of 50. If you can't do one part, find a way to move on. You can always write the correct code for subsequent parts even if the code for previous parts is incorrect. Most parts are actually independent, and with your help, we will do our best to grade this so that mistakes early on do not carry though.

Name: ______________________________________

e. (Preferably using your results from 2c. and 2d. above) Generate a NumPy array `df` of 1000 values $f'(x) = 2x^2 \cos x^2 + \sin x^2$ for x in the range $[0, 2\pi]$. Likewise, generate an array `F` of values $F(x) = \frac{-\cos x^2}{2}$ for x in the range $[0, 2\pi]$. (Yes, it's OK to re-assign `df` and `F`.)

f. (Preferably using your results from 2e. above) Plot $f'(x)$ and $F(x)$ from $x = 0$ to $x = 2\pi$. If you are using `matplotlib`, include the statement `%matplotlib inline` at the top of the cell to make the plot appear in the notebook.

g. *Modify* `df` as follows: subtract $\frac{F(-\Delta)}{\Delta^2}$ from the first element, and subtract $\frac{F(2\pi+\Delta)}{\Delta^2}$ from the last element. That is, *change* `df`; do not create a new array.

h. Write a function `diff2_mat(N)` that accepts a positive integer argument N and returns an $N \times N$ NumPy array like:

```
[[ 2. -1.  0. ...,  0.  0.  0.]
 [-1.  2. -1. ...,  0.  0.  0.]
 [ 0. -1.  2. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  2. -1.  0.]
 [ 0.  0.  0. ..., -1.  2. -1.]
 [ 0.  0.  0. ...,  0. -1.  2.]]
```

Show that your function works by printing the result of `diff2_mat(10)`.
For full credit, do this without any loops. There are multiple ways of doing this in a single statement involving calls to a few NumPy array generation functions covered in the readings.
I will refer to an $N \times N$ matrix of this form (with 2s on the main diagonal, -1s immediately above and below the main diagonal, and 0s elsewhere) as K_N in the following steps.

i. For $N = 10$, visualize the matrix produced by `diff2_mat` as an image (with any color scheme).

j. For $N = 1000$, create a matrix `D2` according to the formula $-K_N \frac{1}{\Delta^2}$.

k. Solve the linear equation `D2 F2 = df` for the unknown $N \times 1$ array `F2`. (Note that `D2 F2` means the product of the variables `D2` and `F2` in the linear algebra sense, not the element-wise array sense.)

l. If you've done everything above correctly, `F2` should nearly the same as `F`, and you've just solved a second-order boundary-value problem using the finite difference method! Show that `F` and `F2` are similar and explain why what you have done shows it, even if it is obvious to you. (There are several valid ways to approach this. Printing the arrays and asking the user to manually compare corresponding elements is not one of them.)

m. Because the diagonals of `D2` are constant, it is known as a Toeplitz matrix. Find a function for solving linear equations involving a Toeplitz matrix in the SciPy documentation, and solve the same equation as above (`D2 F2 = df`). On my machine this is about 20 times faster than multiplying the inverse of `D2` by `df`, and 9 times faster than solving the equation otherwise using more general solving functionality.