Introduction To Python

Week 8: Program Dev:
Graphical Game of Life

Dr. Jim Lupo
Asst Dir Computational Enablement
LSU Center for Computation & Technology
Intro To Tkinter

- Tcl/Tk is a scripting command language with a graphical tool kit.
- Tk is embedded in Python to give it graphics capabilities - module Tkinter
- Complete programming language in it's own right - so very powerful.
- Approach by example.
Plotting With **graphics**

- **graphics** - Simplified module of plot commands based on Tkinter
  - **gfx.py** - draws two points on a surface using **graphics**, and reports some mouse events.
  - **Tconvert.py** - dialog that converts temperatures using **graphics**
  - **canvas_graph.py** - Function plotter using Tkinter directly.
  - **canvas_plot.py** - Grid plotter using Tkinter directly.
- It's often easiest to start with something that works and modify it to do something you want.
Down The Rabbit Hole

- Implemented a version with graphics module - agonizingly SLOWWWWWW!
- Works for one-off plots.
- gfx.py is an example of how to work with graphics module, but we'll ignored details.
Runtime Comments

- Messages appear in the console.
- When you click the mouse in the square, another point appears.
- The mouse point coordinates are reported in the console.
- Terminate program by hitting Enter.
Use **graphics** for Life

- Refer to life-gfx.py
- Sort of klutzy, but it works.
"Windows" Programming

- "Windows programming" - be it X-windows, Mac OS, or Microsoft - is an event driven enterprise.
- Mouse clicks are something new.
- Key strokes have always been events behind the curtains.
- graphics module gives you 'simplified' access to the event machinery.
from graphics import *

def main():
    win=GraphWin("Celsius Converter", 300, 200)
    win.setCoords(0.0,0.0,3.0,4.0)
    # Draw the interface
    Text(Point(1,3),"Celsius Temperature: ").draw(win)
    Text(Point(1,1),"Fahrenheit Temperature: ").draw(win)
    input = Entry(Point(2,3),5)
    input.setText("0.0")
    input.draw(win)
    output = Text(Point(2.25,1),"")
    output.draw(win)
    button = Text(Point(1.5,2.0),"Convert It")
    button.draw(win)
    Rectangle(Point(1,1.5),Point(2,2.5)).draw(win)
    # wait for a mouse click
    win.getMouse()
    # convert input
    celsius = eval(input.getText())
    fahrenheit = 9.0/5.0 * celsius + 32
    # display output and change button
    output.setText("%0.1f" % fahrenheit)
    button.setText("Quit")
    # wait for click and then quit
    win.getMouse()
    win.close()
main()
Tconvert
Use Tkinter Directly

- There are some limitations with `graphics` module.
- Does not rise that far above the use of Tkinter directly.
- `canvas_graph.py` gives an example.
- Re-implement `gfx.py` as `canvas_plot.py` using Tkinter.
canvas_graph.py

*Use pythonw.exe on Windows to suppress console.

16 Jul 2015, Page 12 of 27
canvas_plot.py

- Sort of replicate `gfx.py`.
- Use button controls.
- Plot 'points' using filled rectangles.
- Increment location with each button press.
- Provide a Quit button.
canvas_plot.py

3 colored cells. Appear after pushing Plot button. Will move (replot) with each push.
Important Tkinter (Tk) Terminology

- **root** - base object on which a graphical display is built.
- **frame** - a container to hold widgets.
- **widget** - a type of graphical object - various buttons, labels, entry fields, menus, etc.
- **pack()** - insert the widgets in the display area. One of 3 different methods possible.
Code Walk-Through

- Refer to canvas_plot.py
Better Way To Add Graphics?

- Use canvas_plot.py as a model?
  - cell() seems a natural fit.
- Where to create and destroy window?
- Mouse or console interaction?
  - How would user be told?
  - How to stop generations?
- Modify user input to indicate how many generations to run before pausing again?
Life with **Tkinter**: Phase 1

- Start by replacing printed output with graphical output.
  - Add module.
  - Decide where to create window, and close when done.
Code Walk-Through

• Refer to `canvas_plot.py`.
Life-Plot: Phase 1

Console generation and timing output.
Do Away with Console Output?

- Refer to `life_plot-2.py`
Life-Plot: Phase 2

*Use pythonw.exe on Windows to suppress console.
Life-Plot: Phase 3

- No output changes.
- An attempt at optimizing.
  - Use helper lists to reduce computation.

```python
cnt = 0
def handle_neighbor(i, j):
    for v in [-1, 0, 1]:
        for h in [-1, 0, 1]:
            if v == 0 and h == 0:
                continue
            x = i + h
            if x < 0:
                x = gridDims[0] - 1
            if x >= gridDims[0]:
                x = 0
            if y < 0:
                y = gridDims[1] - 1
            if y >= gridDims[1]:
                y = 0
            cnt += grids[g0][y][x]
```

```python
cnt = 0
for v in [0, 1, 2]:
    for h in [0, 1, 2]:
        if v == 0 and h == 0:
            continue
        x = i + h
        if x < 0:
            x = gridDims[0] - 1
        if x >= gridDims[0]:
            x = 0
        if y < 0:
            y = gridDims[1] - 1
        if y >= gridDims[1]:
            y = 0
        cnt += grids[g0][y][x]
```
Neighbor Indexing

- Look at the index pattern when checking for the neighbors of a cell:

<table>
<thead>
<tr>
<th>Neighbor indexing relative to one cell, say g(r,c):</th>
<th>g(r-1,c-1)</th>
<th>g(r-1,c)</th>
<th>g(r-1,c+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g(r,c-1)</td>
<td>g(r,c)</td>
<td>g(r,c+1)</td>
<td></td>
</tr>
<tr>
<td>g(r+1,c-1)</td>
<td>g(r+1,c)</td>
<td>g(r+1,c+1)</td>
<td></td>
</tr>
</tbody>
</table>

In terms of indices, the neighbors of a cell never change.
Look at One Corner

Assume the grid dimensions are $R_{\text{max}}$ rows by $C_{\text{max}}$ columns, and that we wrap around at the edges.

\[
\begin{array}{ccc}
g(R_{\text{max}}-1,C_{\text{max}}-1) & g(R_{\text{max}}-1,0) & g(R_{\text{max}}-1,1) \\
g(0,C_{\text{max}}-1) & g(0,0) & g(0,1) \\
g(1,C_{\text{max}}-1) & g(1,0) & g(1,1)
\end{array}
\]

The same sort of pattern happens at the other 3 corners.
Helper Lists

• Generate lists that include the wrapped ends.

rowNeighbors = [Rmax-1,0,1,2,...,Rmax-2,Rmax-1,0]
colNeighbors = [Cmax-1,0,1,2,...,Cmax-2,Cmax-1,0]

• For g(0,0), neighbor rows run from Rmax-1 to 1, neighbor columns run from Cmax-1 to 1.
• For g(1,1), neighbor rows run from 0 to 2, and neighbor columns run from 0 to 2.
• Trade off use of extra memory for reduced computation of not constantly checking for border wrapping.
Next Steps

- Allow to run continuously.
- Allow to advance specified number of generations.
- Use mouse to create Generation 0.
- Use Tkinter image file support to save copies at various generations.
- Etc, etc, etc.