Practical 5
Grids and Files

Learning Objectives

1. Understand and use text files to store and load data
2. Develop simple grid-based simulations using 2-dimensional arrays: fire modelling, Game of Life
3. Apply list comprehensions to simplify code
4. Experiment with parameters to investigate how they alter the outcomes of simulations

Overview

In this practical you will read and write data using text files. You will also work with some grid-based algorithms – testing out different values to see how their parameters affect outcomes. We will also look at using list comprehensions to simplify our code.

Tasks

1. Reading a CSV file

Type in the following code, weather.py, for displaying the weather stored in a file:

```
#
# weather.py: Print min and max temps from a file
#             (source: http://www.bom.gov.au/climate/)

import matplotlib.pyplot as plt

fileobj = open(‘marchweather.csv’, ‘r’)

# add file reading code here

fileobj.close()


mins = # add splitting code here
maxs = # add splitting code here
dates = range(1,32)

plt.plot(dates, mins, dates, maxs)
plt.show()
```

Modify the code to read the data from the marchweather.csv file – available on Blackboard. You should download it to your Prac5 directory, look at its contents and format, then modify the code accordingly. **Hint:** look at split method in lecture slides.

2. Reading another CSV file

This time, go to the Bureau of Meteorology site and download the full list of weather data for March. This time we will plot the **min**, **max**, **9am** and **3pm** temperatures…
http://www.bom.gov.au/climate/dwo/201903/html/IDCJDW6111.201903.shtml

Save the data by scrolling down to the “Other Formats” section and right-clicking on the plain text version. Save it to your Prac5 directory as marchweatherfull.csv. If you open it in vim you can see all the data, but there are headers describing the data that we don’t need to read in. Remove the first header lines using ‘dd’ and then save the file. You now have your dataset.

Write a new program, **marchweather2.py** to read in the values and plot them. You will need to pick out columns from each line you read in from the file. First split it into a list, then pick out the values and assign them to the min, max, nine and three lists/arrays.

The code below will help start you off:

```
fileobj = open(‘marchweatherfull.csv’, ‘r’)
data = fileobj.readlines()
fileobj.close()

mins = []     # do the same for maxs, nines and threes

for line in data:
    splitline = line.split(‘,’)
    mins.append(splitline[2])
    maxs.append(splitline[3])
    nines.append(splitline[10])
    threes.append(splitline[16])
```

Then adjust your plt.plot call to plot mins, maxs, nines and threes. Make sure you set up the x values (dates) as in Task 1

3. Writing to a CSV file

Take your marchweather2.py and modify it to write the four lists of values into a csv file, four values per line.

```
file2 = open(‘marchout.csv’, ‘w’)
for i in range(len(mins)):
    file2.write(mins[i] + ‘,’ + maxs[i] + ‘,’ + nines[i] + ‘,’ +
                threes[i] + ‘\n’)
file2.close()
```

4. List comprehensions

Using list comprehensions can reduce and simplify your code. In the lecture, we saw some examples of using list comprehensions. Using the lecture slides as a guide, write code to do the following **using both loops and list comprehensions** for each:

1. Make a list, **numbers**, with the numbers from 1 to 5
2. Write a function triple() and use it to triple each number in **numbers**
3. Write code to read in a string and extract all of the numbers (Hint: isdigit())
4. Write code to capitalise the first letter of each word in a list of words (Hint: you can use use "+" to put the word back together)

5. Heat Diffusion

Download and run heat.py, available in the practical area on Blackboard:

heat.py

```
size = 20

curr = np.zeros((size,size))
print(curr)
for i in range(size):
    curr[i,0] = 10

next = np.zeros((size,size))

for timestep in range(5):
    for r in range(1, size-1):
        for c in range (1, size-1 ):
            next[r,c] = (curr[r-1,c-1]*0.1 + curr[r-1,c]*0.1
                + curr[r-1,c+1]*0.1 + curr[r,c-1]*0.1
                + curr[r,c]*0.2 + curr[r,c+1]*0.1
                + curr[r+1,c-1]*0.1 + curr[r+1,c]*0.1
                + curr[r+1,c+1]*0.1)

    for i in range(size):
        next[i,0] = 10

    print("Time step: ", timestep)
    print(next)
    curr = next.copy()


plt.imshow(curr, cmap=plt.cm.hot)
plt.show()
```

Modify the program to replace the highlighted code with the more readable code below:

```
next[r,c] = 0.1 * (curr[r-1:r+2,c-1:c+2].sum() + curr[r,c])
```

Make sure you understand what this code does. Re-run the program to see that it still works.

6. Heat diffusion with Functions

Our heat.py program has an ugly line of code to calculate the next values for each cell. Create a function “calcheat” to factor this out. You can then call the function as:

```
next[r,c] = calcheat(r, c)
```

The line to put in the function is:

```
   next[r,c] = = 0.1 * (curr[r-1:r+2,c-1:c+2].sum() + curr[r,c])
```

7. Reading (yet another) CSV file

Copy your heat.py and rename it to heatsource.py. This time we are going to read a heat source in from a file.

Create a file heatsource.csv to hold the heatsource:

```
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
10,0,0,0,0,0,0,0,0,0,0
```

(note, you can copy a line using ‘yy’ and ‘p’ in vim)

In our original program we had two loops to set up and maintain the heat source:

```
for i in range (size):
    b[i,0]=10
```

This could also have been done in a more Pythonic way with:

```
b[:,0] = 10
```

We are going to replace those lines with code to read the heat source from our file and update in each loop from our new h array to maintain the heat source.

Replace the first heat source code instance with the following to read data from a file:

```
# create heat source
hlist = []
fileobj = open('heatsource.csv','r')
for line in fileobj:
    line_s = line.strip()
    ints = [int(x) for x in line_s.split(',')]
    hlist.append(ints)
fileobj.close()

h = np.array(hlist, dtype=int)
b = h.copy()
```

list comprehension

And in the loop the heat source needs to be updated using the new h array…

```
# Calculate heat diffusion
for timestep in range(100):
    for r in range(1,size-1):
        for c in range (1, size-1):
            b2[r,c]=calcheat(r,c)

    for r in range(size):
        for c in range(size):
              if h[r,c]>b2[r,c]:
                  b2[r,c] = h[r,c]
    b=b2.copy()
```

Your code should now output the same information as it did before – test it and see.

In a similar way to list comprehensions, we can simplify the four lines of code above to one line using np.where(). This will overwrite values in b2 where the value in h is larger:

```
    b2 = np.where(h>b2, h, b2)
```

Try changing the values in heatsource.csv to see how it changes the output of the program.

8. Fireplan

Access the app at: http://www.shodor.org/interactivate/activities/FireAssessment/

Explore the use of the app and how it is making use of the grid and neighbours. Also look at the use of graphics to represent the different states of a cell. What different graphics are used for the states and what do they represent?

9. Game of Life

Have a read of http://web.stanford.edu/~cdebs/GameOfLife/ to see how the game of life works. Use your mouse to enter some life into the Game of Life Simulator, then click run to see the outcomes. How long does your population survive?

Submission

Create a README file for Prac5. Include the names and descriptions of all of your Python programs from today. All of your work for this week's practical should be submitted via Blackboard using the link in the assessments area. This should be done as a single "zipped" file.

Reflection

1. **Knowledge**: What are the three different read methods we can use on a file? What is the difference between them?
2. **Comprehension**: What does the line file2.write(…) do in Task 4?
3. **Application**: Given the Game of Life rules, what would happen to the centre cell in the following cases:

☛		☛
	☛	

☛		☛
	☛	
		☛

☛		
	☛	

☛		
	☛	☛
☛	☛	

☛		
	☛	

	☛	

4. **Analysis**: What variation of "neighbours" does heatsource.py use? How would the code change if it were to use the other neighbour approach?
5. **Synthesis**: How would you create a heat source input file with a 4x4 heat source in the centre of the 10x10 grid?
6. **Evaluation**: Name two advantages to reading initial data from a file as in the updated heatsource.py.

Challenge

- Follow the workflow from Task 3 to process and plot February weather data.
- Download the Game of Life code from Task 9 and get it running.