LECTURE 5
FILES AND GRIDS

Fundamentals of Programming - COMP1005

Department of Computing

Curtin University

Updated 26/3/18

Copyright Warning

Learning Outcomes

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

The story so far…

- We've gone from working with single variables…
  …through lists and strings…
  … then one dimensional arrays…
  … and 2-D arrays …
  … and N-D arrays …
- However, every piece of data we have used has been entered or generated within our programs.

The story so far…

- **EXCEPT…** the prettyface.py critter
- This was read in from a file within scipy.misc
- We have also saved our plots as files to use later
- Files can be read from (used as input) or written to (output)
- We'll learn how to use them this week

The story so far…

- Also, we have been using arrays to represent values
  growtharray.py for population values
- We can view an array as a grid, representing 2D or 3D space in the real (or imaginary) world
- Grids are an abstraction of the world, with each cell representing a 2D or 3D space
- We'll look at a few examples this week

FILES

Fundamentals of Programming

Lecture 5

Python input and output

- So far we've been printing to the screen and reading from the keyboard…

```
print('What... is your name?')
name = input()
# It is 'Arthur', King of the Britons.
print('What... is your quest?')
quest = input()
# To seek the Holy Grail.
print('What... is the air-speed velocity of
an unladen swallow?')
velocity = input()
# What do you mean? An African or European
swallow?
```

Files

- Files allow us to have **persistent** data – data that stays after our program has run
- Files are accessed via their name (and directory)
- Some files hold **text** (python code, csv files) and others may hold **binary** data (e.g. images)
- When we open a file, we can indicate if we want to read, write or append to the file
  - default is **'r'** - read

Working with Files

- Python provides access to file functions and methods through the file object
- To create a file object and have access to a file, we need to use the open() method:

```
spamfile = open('spam.txt')
# opens text file for reading
csvfile = open('newdata.csv', w)
# creates a new csv/text file for
# writing, overwrites it if exists
datafile = open('olddata.dat', ab)
# opens existing binary file and appends
```

File modes

Note: We will focus on text files

Read	Write	Append	Description
r	w	a	Read/write/append text files
rb	wb	ab	Read/write/append binary files
r+	w+	a+	Opens for reading and writing
rb+	wb+	ab+	Opens for reading and writing binary files
file pointer at **beginning**	file pointer at **beginning**	file pointer at **end** of file	***Note**: you can accidentally delete a file by opening it in the wrong mode!*

File locations

- To specify a path to the file, add the path to the name the file
- If you don’t specify a path, Python will look for the file in the current directory
- `path = '/home/12345678/FOP/spam.txt'`
- `spamfile = open(`**`path,`**` 'r')`

Closing Files

- A bit like ejecting a USB – we need to safely close our files
- Always call close() after you have finished working with files
- Closing flushes any unwritten information and closes the file object

```
spamfile.close()
csvfile.close()
datafile.close()
```

Reading files

- Python has three read methods to work with open files

```
pythonfile.read()
# reads entire contents of file
pythonfile.readline()
# reads one line of the file
#(to next newline)
pythonfile.readlines()
# reads entire contents of file as
# a list, one line per element
```

Example

names.py

```
names = open('names.txt')
thischunk = names.read()
print(thischunk)
names.close()

names = open('names.txt')
thischunk = names.readline()
print(thischunk)
names.close()

names = open('names.txt')
thischunk =
names.readlines()
print(thischunk)
names.close()
```

OUTPUT

```
Eric
John
Terry
Graham
Michael

Eric

['Eric\n', 'John\n', 'Terry
\n', 'Graham\n', 'Michael
\n']
```

Writing to files

- Use the returned **file object** to write to the file

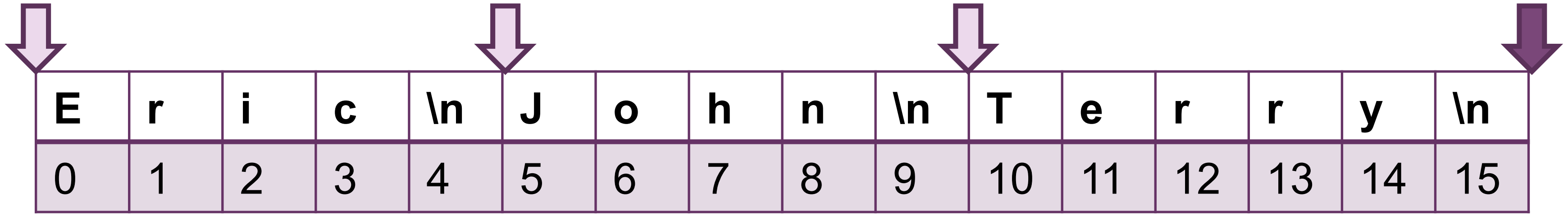
```
# Open a file
fo = open("grail.txt", "w")
fo.write( "Come back here and I'll bite your
legs off! \nBlack Knight\n");
# Close opened file
fo.close()
```

```
grail.txt:

Come back here and I'll bite your legs off!
Black Knight
```

Files are like arrays

- Our text files are very similar to 1D arrays
- They can be broken up by newline characters, but are really just long strings



E	r	i	c	\n	J	o	h	n	\n	T	e	r	r	y	\n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- You decide how you want to read and write them to suit the problem you are working on
- Python keeps track of where it is in the file with a pointer ⇩ that moves along with each read or write

CSV Files

- It's common to receive data as comma separated values (csv)
- There is a **csv package**, we'll look at that in a few weeks
- For now, read a csv file as text, then split on the commas:

```
namesfile = open('names.csv')
line = namesfile.readline()
linelist = line.split(',')
print(linelist)
```

- To write csv, convert the list to a comma separated string using the join() method:

```
newnames = open('names3.csv', 'w')
newline = ','.join(linelist)
print(newline)
newnames.write(newline)
```

Pythonic file handling

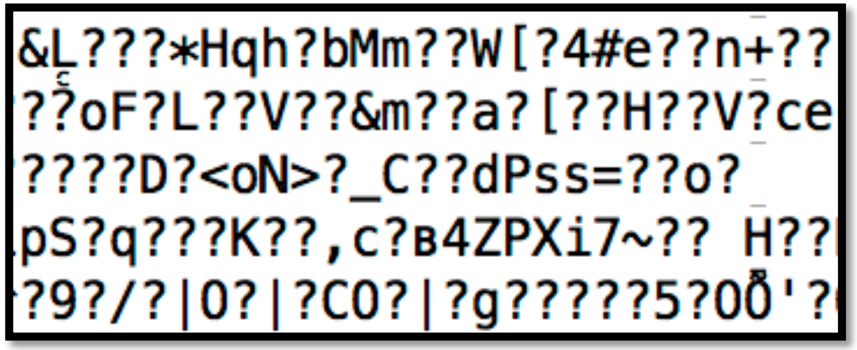
- It is good practice to use the with keyword when dealing with file objects.
- This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way.
- It is also much shorter than the other approach

```
with open('workfile', 'r') as f:
    read_data = f.read()
```

https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files

Binary Files

- We have focused on text files
- Using binary files is beyond the scope of this course
- With text files, data is stored as characters
- **Advantage**: binary files store the data as binary which is much more compact
- **Disadvantage**: we can't read them directly and are unlikely to be able to fix it if is corrupted
- The files look a bit like this when you open them
- See these docs for more information:



```
&L???*Hqh?bMm??W[?4#e??n+??
??oF?L??V??&m??a?[??H??V?ce
????D?<oN>?_C??dPss=??o?
pS?q???K??,c?B4ZPXi7~?? H??
?9?/?|O?|?CO?|?g?????5?OŐ'?
```

https://docs.python.org/3.3/tutorial/inputoutput.html#reading-and-writing-files

http://code.activestate.com/recipes/577610-decoding-binary-files/

GRIDS

Fundamentals of Programming

Lecture 5

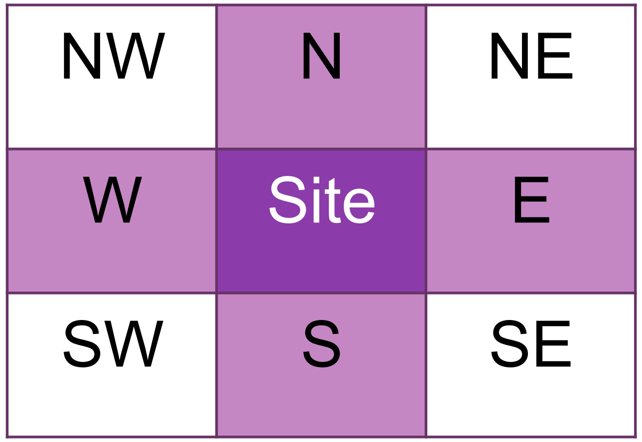
Grid Decomposition

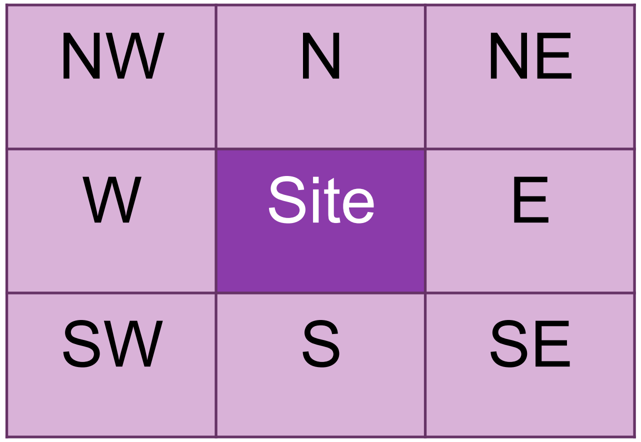
- This approach breaks up a space into a multidimensional grid
- Each cell on the grid has one or more associated values
- The cells impact on each other over time
- General algorithm:

```
For each time_step
    For each grid dimension
        For each cell
            Calculate the next value
```

Neighbourhoods

- In a 2-D grid, the **von Neumann** neighbourhood of a site is the set of cells directly North, South, East and West or the site, and the site itself
- The **Moore** neigbourhood adds NE, NW, SE and SW
- The four or eight cells, not including the site, are the site's **neighbours**



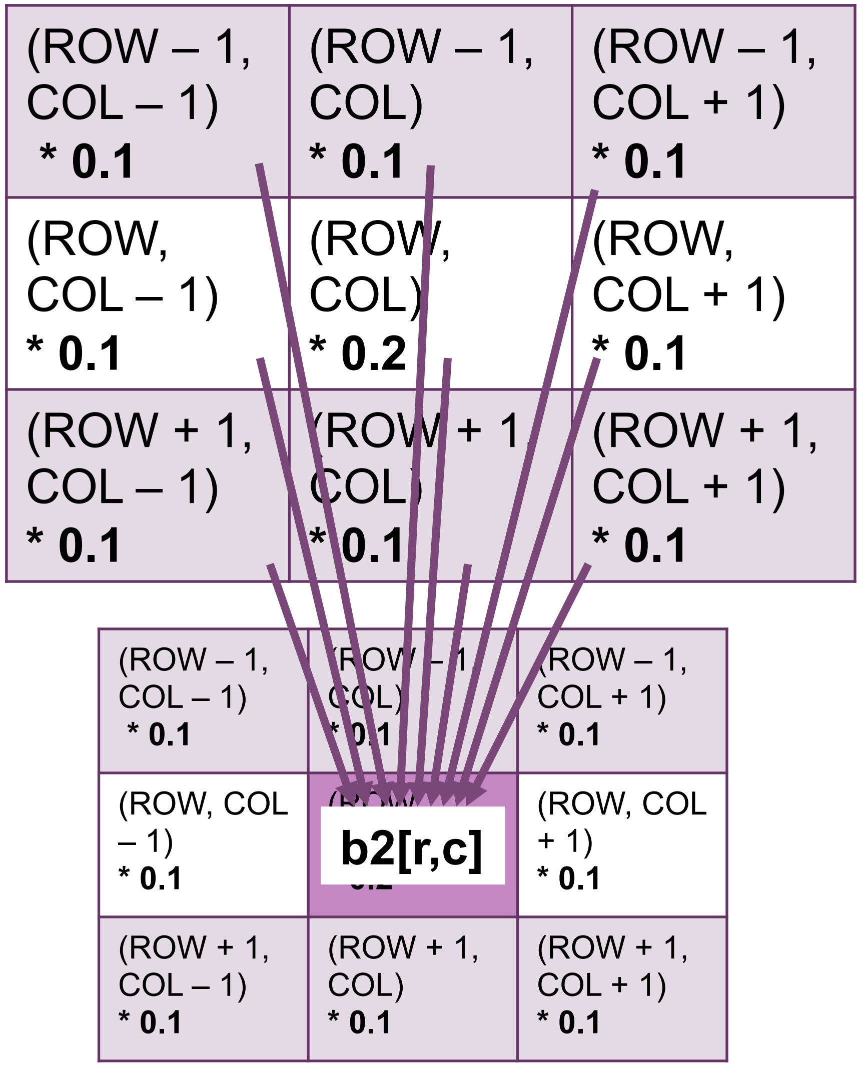


Heat diffusion

- In Practical 5, we will work with a grid-based program, heat.py
- The program is a simple model of heat diffusion
- This model is based on Newton's law of heating and cooling
  - *"Rate of change of temperature of an object is proportional to the difference between the objects' temperature and that of its surroundings" (Shiflet & Shiflet)*

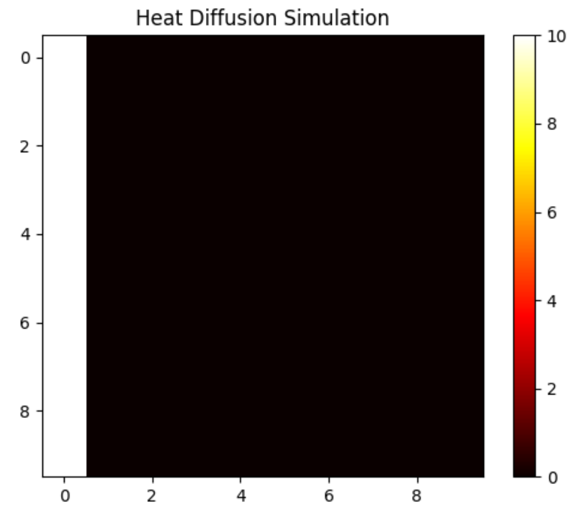
Heat diffusion formula

- We use a temporary array (b2) to store values for the next iteration
- The next (row,col) element b2[r,c] will be the result of the 3x3 calculation
- We do this for every row and column of the array

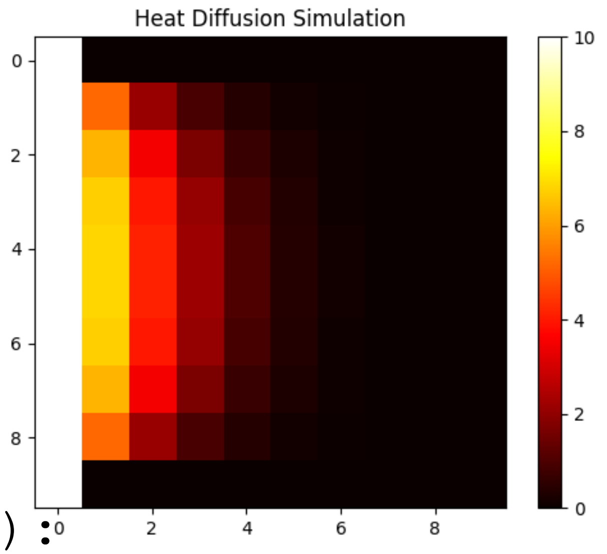


heat.py

```
#
# heat.py
#
import matplotlib.pyplot as plt
import numpy as np
size = 10
b = np.zeros((size,size))
# Create heat source
for i in range(size):
    b[i,0] = 10
print('\nHEAT DIFFUSION SIMULATION\n')
print('Initial array...')
print(b)
# Temp array for storing calculations
b2 = np.zeros((size,size))
```

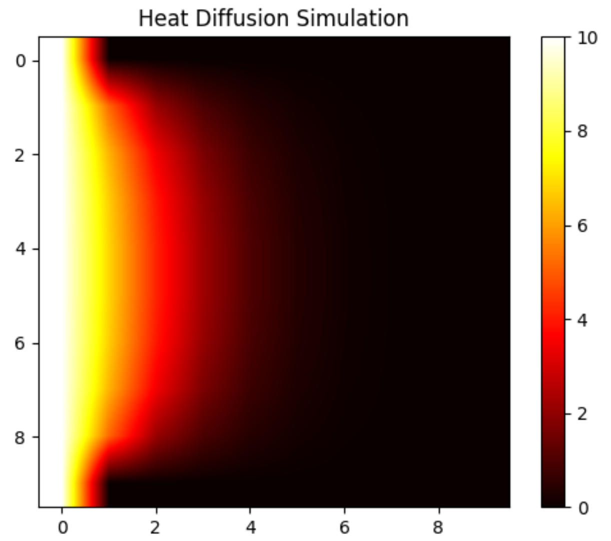


heat.py



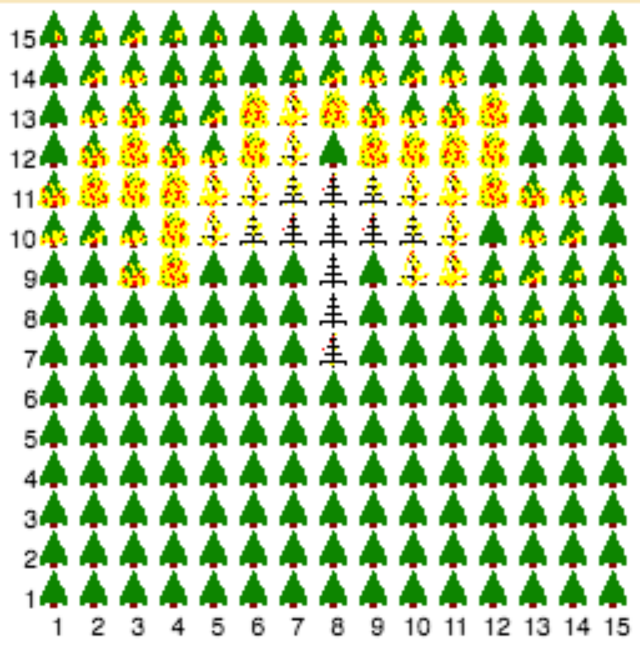
```
# Calculate heat diffusion
for timestep in range(5):
    for r in range(1, size-1):
        for c in range (1, size-1 ):
            b2[r,c] = (b[r-1,c-1]*0.1 + b[r-1,c]*0.1
                      + b[r-1,c+1]*0.1 + b[r,c-1]*0.1
                      + b[r,c]*0.2 + b[r,c+1]*0.1
                      + b[r+1,c-1]*0.1 + b[r+1,c]*0.1
                      + b[r+1,c+1]*0.1)
    for i in range(size):
        b2[i,0] = 10
    b = b2.copy()
plt.title('Heat Diffusion Simulation')
plt.imshow(b2, cmap=plt.cm.hot)
plt.colorbar()
plt.show()
```

heat.py



```
# Calculate heat diffusion
for timestep in range(5):
    for r in range(1, size-1):
        for c in range (1, size-1 ):
            b2[r,c] = (b[r-1,c-1]*0.1 + b[r-1,c]*0.1
                      + b[r-1,c+1]*0.1 + b[r,c-1]*0.1
                      + b[r,c]*0.2 + b[r,c+1]*0.1
                      + b[r+1,c-1]*0.1 + b[r+1,c]*0.1
                      + b[r+1,c+1]*0.1)
    for i in range(size):
        b2[i,0] = 10
    b = b2.copy()
plt.title('Heat Diffusion Simulation')
plt.imshow(b2, cmap=plt.cm.hot, interpolation='bilinear')
plt.colorbar()
plt.show()
```

Fireplan



- Another application of grid decomposition is fire modelling
- Each cell of the grid can include a value for fuel/vegetation, slope, how recently it's been burnt, moisture etc
- Putting these together in a calculation can help predict the movement of a fire
- Retired Curtin lecturer Steve Kessell commercialised a product FirePlan for this purpose
- We will do a simplifed version in the pracs

http://www.shodor.org/interactivate/activities/FireAssessment/

Game of Life

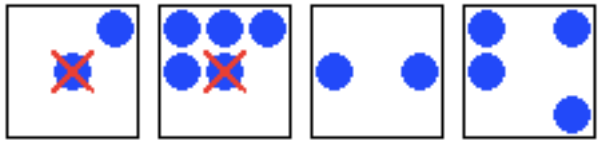
- The Game of Life was invented in 1970 by the British mathematician John Horton Conway
- The Game of Life was Conway's way of simplifying von Neumann's ideas
- It is the best-known example of a cellular automaton which is any system in which rules are applied to cells and their neighbours in a regular grid

http://web.stanford.edu/~cdebs/GameOfLife/

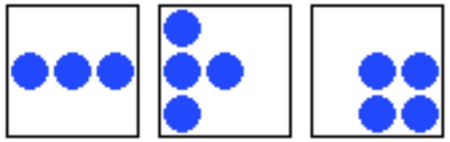
Game of Life - Rules

The game is designed around the following rules:

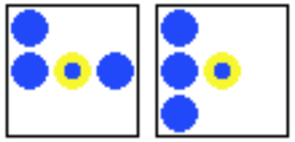
1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation
2. Any live cell with more than three live neighbors dies, as if by overcrowding



3. Any live cell with two or three live neighbors lives on to the next generation



4. Any dead cell with exactly three live neighbors becomes a live cell.

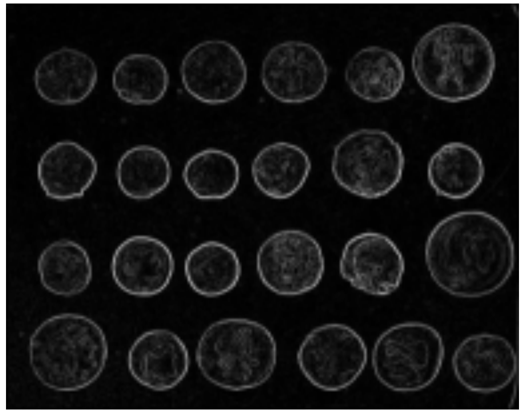


http://web.stanford.edu/~cdebs/GameOfLife/

Image Filters

- Many image processing techniques are based on small 2-D filters
- scikit-image provides these routines in Python
- They are making your plots pretty!
- To use a filter through scikit-image:

```
from skimage import data, io, filters
image = data.coins()
# ... or any other NumPy array!
edges = filters.sobel(image)
io.imshow(edges)
io.show()
```



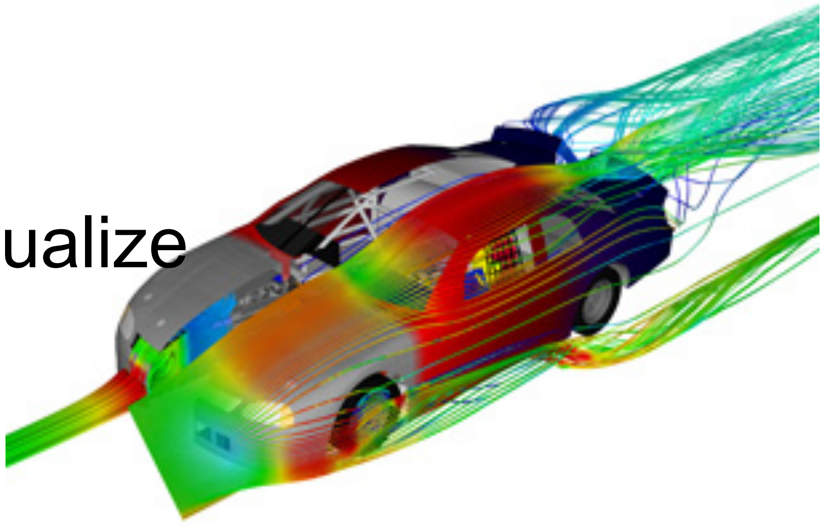
EXAMPLES

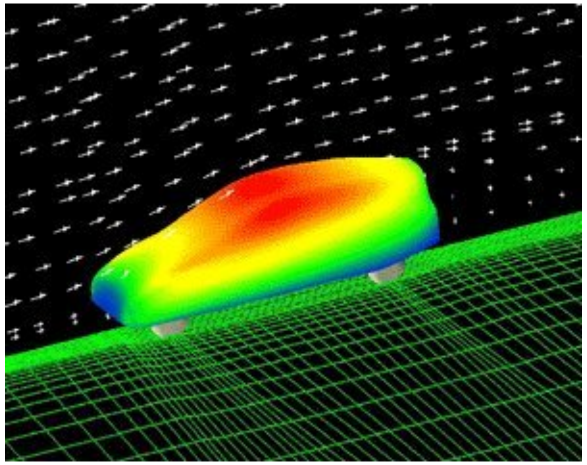
Fundamentals of Programming

Lecture 5

Computational Fluid Dynamics

- CFD is the use of applied mathematics, physics and computational software to visualize how fluids (gases or liquids) flow and how fluids affect objects as they flow past.
- CFD is based on Navier–Stokes equations. These equations describe how velocity, pressure, temperature and density of a moving fluid are related.

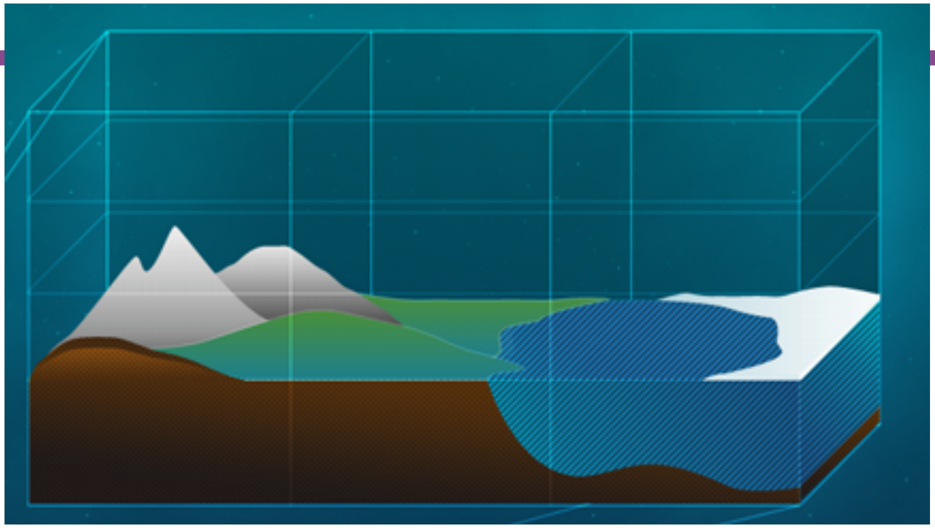




https://www.quora.com/What-is-a-brief-description-of-how-computational-fluid-dynamics-CFD-work

https://s1.cdn.autoevolution.com/images/news/how-does-cfd-computational-fluid-dynamics-work-6400.html

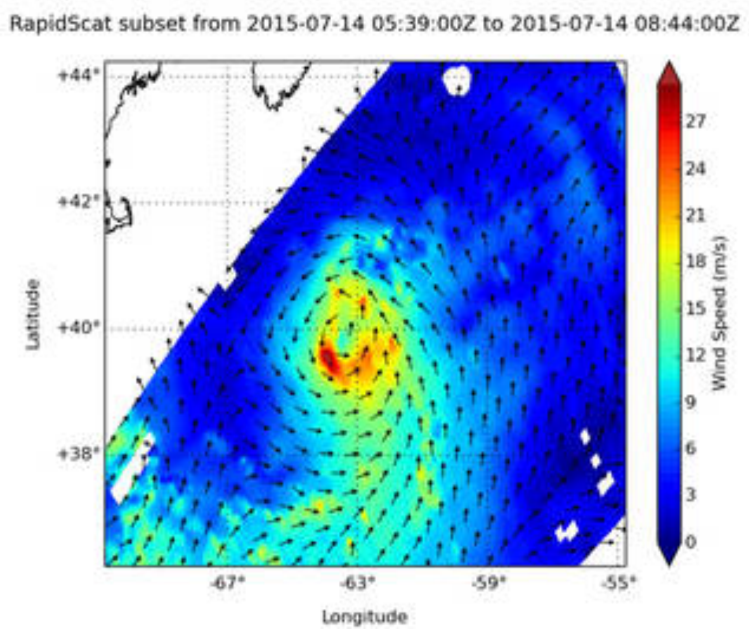
Climate models



- Some of the biggest and most complex computer models
- Made up of a 3-D grid with many, many variables being tracked and calculated
- We'll look at an online description of climate models:

  https://koshland-science-museum.org/explore-the-science/interactives/how-do-climate-models-work

Weather Forecasting



- Similar to climate modelling, but focussing on near predictions and local detail
- Allow for predictions and warnings as with Hurrican Harvey recently in the US
- NASA has a range of models for tracking hurricanes and wild weather: https://pmm.nasa.gov/articles/how-does-nasa-study-hurricanes
- Accurate predictions are vital for disaster planning

LIST COMPREHENSIONS

Fundamentals of Programming

Lecture 5

List Comprehensions

- A Pythonic approach to a frequent operation
- Turns a multi-line for-loop into a one liner
- Basic syntax:

```
[transformation iteration filter]    OR
[expression for item in list if conditional]
```

- Equivalent to:

```
    for item in list:
        if conditional:
            expression
```

http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/

Unconditional list comprehensions

```
numbers = [1, 2, 3, 4, 5]

doubled_numbers = []
for n in numbers:
    doubled_numbers.append(n * 2)
```

```
doubled_numbers = [n * 2 for n in numbers]
```

Conditional list comprehensions

```
numbers = [1, 2, 3, 4, 5]

doubled_odds = []
for n in numbers:
    if n % 2 == 1:
    doubled_odds.append(n * 2)
```

```
doubled_odds = [n * 2 for n in numbers
                if n % 2 == 1]
```

Practical 5: heatsource.py

```
# 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()
```

```
ints = []
for x in line_s.split(','):
    ints.append(int(x))
```

Practical 5: list of squares

```
# create list of first 10 square numbers
squares = []
for s in range(1, 11):
    squares.append(s * s)

numbers = list(range(1, 11))

squares = [s * s for s in numbers]

squares = [s * s for s in range(1, 11)]

print(numbers)

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

More examples:

```
list1 = [3,4,5]
multiplied = [item*3 for item in list1]
print(multiplied)

[9,12,15]

def double(x):
    return x*2

doubled = [double(x) for x in range(10)]

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

http://www.pythonforbeginners.com/basics/list-comprehensions-in-python

More examples:

```
fname = ["Arthur","King","Of","The","Britons"]
initials = [word[0] for word in fullname]
print(initials)

['A', 'K', 'O', 'T', 'B']

lowered = [x.lower() for x in ["F","O","P"] ]
print(lowered)

['f', 'o', 'p']
```

Summary

- We've looked at how we can use text files to store and load data
- We've explored and implemented simple grid-based simulations using 2-dimensional arrays: fire modelling, Game of Life
- We've seen how to apply list comprehensions to simplify code
- In the practicals we will:
  - Experiment with parameters to investigate how they alter the outcomes of simulations

Practical Sessions

- This week we will explore the heat diffusion simulation
- We will access and explore grid-based programs:
  - fireplan – modelling the movement of fire
  - game of life – modelling a population over time
- We will also explore reading from and saving to files, and using list comprehensions

Assessments

- The next assessment will be held during your assigned practical this week (Prac 5)
- It will be a short practical test using the lab computers
- Open book, open computer, help provided
- Everyone should be able to get 100%!

Practical Test 2

- Create files and directories as instructed
- Create Python program to match the description given – includes plotting
- Modify the code and the plot
- Capture your command history into a file within the PracTest2 directory
- Zip your files and submit them through the assessment page

References

- https://www.tutorialspoint.com/python/python_files_io.htm
- https://www.digitalocean.com/community/tutorials/how-to-handle-plain-text-files-in-python-3
- https://www.quora.com/What-is-a-brief-description-of-how-computational-fluid-dynamics-CFD-work
- https://www.autoevolution.com/news/how-does-cfd-computational-fluid-dynamics-work-6400.html
- https://pmm.nasa.gov/articles/how-does-nasa-study-hurricanes
- https://www.climatechangeinaustralia.gov.au/en/climate-campus/modelling-and-projections/climate-models/theory-and-physics/
- https://koshland-science-museum.org/explore-the-science/interactives/how-do-climate-models-work

Next week…

- Scripting
- Automation
- Data Wrangling (Lecture 7)