COMP1005 Week 3 Cheat Sheet

Lisa Luff

8/17/2020

Contents

Arrays

- Hold an ordered sequence of values
- All values must be the **same** type

Array Implementation

- The total size of the array can be calculated because you know the size of each element given they're all the same type
- It can be stored as a single block of memory
- Simple maths can be used to find each element
- Moving from element to element is fast

Array Pros and Cons

- Pros:
  - They are fast
  - They make sense
  - Space efficient
  - can store a lot of useful data
- Cons
  - Not a part of "standard" Python
  - Need to use a package

Numpy

- Core library (package) for scientific computing in Python
- Provides high-performance for N-dimensional arrays (multi-dimensional)
- Includes:
  - Operations and functions to manipulate arrays
  - Sophisticated broadcasting functions, allowing a function to operate on an entire array at once without needing to loop
  - Tools to integrate C/C++ and Fortran
  - Good for linear algebra, Fourier transform and has random number capabilities

Numpy Arrays

- It is convention to import numpy as np
- To create an array:
  - Directly - np.array([x, y, z])
  - From a list - np.array(*list*)
    - Can use dtype=*type* to as an argument to specify what type of data you want the array to store the data as
- You can index, and slice arrays in the same way as other vectors
- You can make preset arrays of values:
  - np.zeros(x) - Array of x 0's
  - np.ones(x) - Array of x 1's
  - np.fill(x, y) - Array of size x, with each element containing y
  - np.random.random(x) - Array of x random numbers
  - np.arange(x, y, z) - Array created using values within a range, specified the same as range()
  - np.linspace(x, y, z) - Array of size z, of values between x and y inclusive, with each value evenly spread across the range
    - Be conscious of ranges for this one as it is inclusive
- You can loop through arrays the same as any other vector

Numpy Array Operations and Functions

- Element must be the same length
- Not using matrix rules
- Element-wise operations:

Command	Purpose
a + b	Adds elements of a to the associated element of b
a + n	Adds n to each element of a
a - b	Minus elements of a from the associated element of b
a * b	Multiply elements of a with associated element of b
a / b	Divide elements of a with associated element of b

- You can do an element-wise comparison resulting in a boolean array refering to the original array using:
  - a < or > b
  - a < or > n
  - a <= or >= b
  - a == b
- Element-wise functions:

Command	Purpose
np.sqrt()	Square root of each element
np.sin(), cos(), etc	Trig operation on each element
np.exp(), log(), etc	Mathematic operations on each element
np.add(), minus(), multiply(), divide(), etc.	Standard maths on each element

- Array-wise functions:

Command	Purpose
variable.sum()	Sum of array elements
variable.min()	Minimum value in the array
variable.max()	Maximum value in the array
variable.mean()	Mean of the array elements

- **Need to be careful, sometimes you might get an inexact value due to the translation of binary to decimal**

Matplotlib

- Matplotlib is the preferred package for 2D graphis in Python
- Includes:
  - Plots
  - Bar graphs
  - Histograms
  - Scatterplots
  - Power spectra
  - Error charts
  - And much more
- Based on Matlab
- Good enough to be published in research papers
- Created by John D. Hunter
- Has postscript output for inclusion with TeX documents

Pyplot

- matplotlib.pyplot is a collection of functions within matplotlib
- What we will be using
- Keeps track of the figure you are working
- Any function calls are used on the current figure
- Data needs to be stored to be plotted, or directly entered into plotting function
- Convention to import matplotlib.pyplot as plt
- Plot types:
  - plot(x, y) - Plot x on the x axis, and y on the y axis (default for single is y axis)
  - bar(x, y) - Plot a bar graph
    - width = n argument from 0 to 1 if you want some spacing between bars
  - hist(x) - Plot a histogram
    - Default breaks data into 10 bars, use bins=n to change to n bars
    - Cumulative = TRUE for cumulative histogram
    - histtype='step' to use a line instead of bars
    - normed=True to normalise the data
- Plotting tools:

Command	Purpose
title('*Title*')	Main title for graph
xlabel('*Label*')	Label for x axis
xaxis(x, y)	Sets start and end of x axis
ylabel('*Label*')	Label for y axis
yaxis('x, y)	Sets start and end of y axis
show()	The final command to print the graph with a collation of prior commands

- Multiples:
  - You can have multiple plots on the same graph, just plot them all before using show()
  - You can have multiple graphs together side by side using subplot() by using:
    - plt.figure(1) - Makes it one figure
    - plt.subplot(x, y, z)
      - This tells you where you want to place the graph in terms of a matrix
      - x is how many rows of subplots you want
      - y is how many columns of subplots
      - z is the position you want this subplot counting from 1 and 1,1 of the matrix of subplots, and counting from left to right, returning to the left as you move down a column

- Graph visuals:
  - You can use a variety of options to change the aesthetic or visibility of your plot
  - '*a* *b*' is the plottype argument to change to *a* coloured *b* shaped dots
    - Colour shorthand examples:
      - b - blue, g - green, r - red, etc
    - Marker shape examples:
      - +, „ ., 1, 2, s - square, ˆ - triangle, etc
    - Linestyles examples:
      - -, –, -., :, 'steps', . . . , etc
  - Or the argument color='*colour*'
    - 'blue', 'pink', etc
  - grid = TRUE - argument to have a grid
  - alpha=n - argument between 0 and 1 to set opacity if you want to overlay data