LECTURE 3
ARRAYS AND PLOTS

Fundamentals of Programming - COMP1005

Department of Computing

Curtin University

Updated 16/8/2019

Copyright Warning

Learning Outcomes

- Understand and use Python arrays implemented in NumPy
- Understand and use simple plotting techniques using matplotlib
- Apply arrays and plotting to more complex systems dynamics problems

The story so far…

- We have looked at **simple data types**:
  integers, **float**ing point
  complex numbers,
  bool (True/False) values

- We can assign these values to **variables**
- We can use **operations** on variables and values and combine them together into **expressions**
- **Control structures** provide choice and repetition (if/elif/else, for and while loops)

The story so far…

- We’ve also looked at more **complex datatypes**: strings and lists
  greeting = ‘hello’
  bucket = ['Visit Stonehenge', 'Skydiving']
- Strings and lists are **ordered sequences**
- **Strings** hold characters and are **immutable**
- **Lists** can hold anything (including sub-lists) and can be updated/changed (**mutable**)
- We can access elements in strings and lists using **indexes**, e.g. instring[0], bucket[5]

The story so far…

- Strings and lists have operators and functions, they include:
- String operators: +, *, <, >, ==, in…
- String functions: len, upper(), lower(), min(), max(), count()…
- List operators: +, *, <, >, in
- List functions: len, append(), extend(), del
- Select parts of a string using slicing [start: stop: step]

ARRAYS

Fundamentals of Programming

Lecture 3

Arrays

- Arrays are common across almost all programming languages
- They hold an ordered **sequence** of values

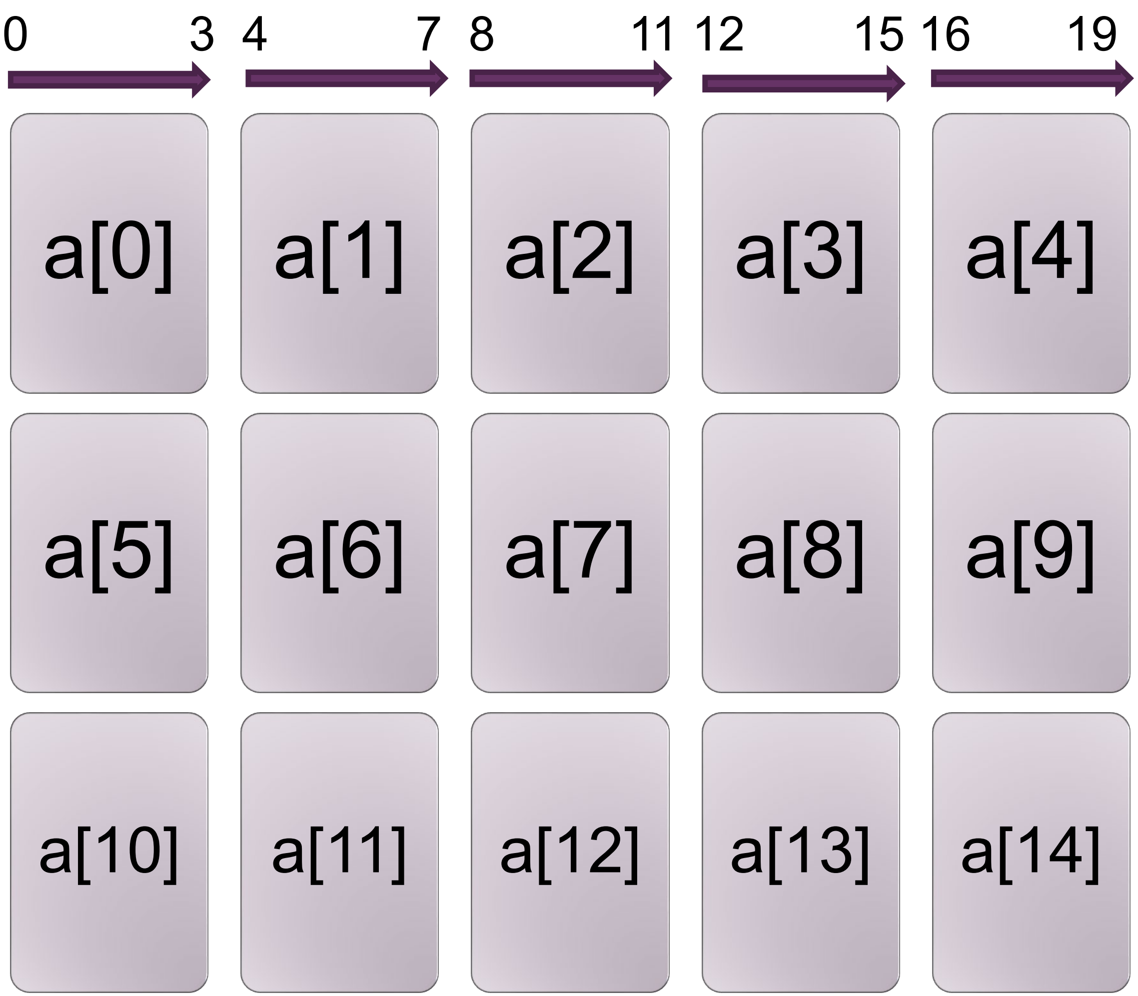
- All values must be of the same type, e.g.:
  - an array of temperatures, stored as floats
  - an array of attendance values, stored as ints
  - an array of responses, stored as bools (True/False)
  - an array of names, stored as strings

Array implementation

- If you know the size of each element, and how many elements you have, then:
  1. The total size of the array can be calculated
  2. The array can be stored as a single block in memory
  3. Simple maths can be used to find each element
  4. Moving from element to element will be fast

- Compare this to a dynamic **list** where you add and delete elements of different types

Array implementation



- If each element is 4 bytes long…
  - a[0] is at offset 0
  - a[1] is at offset 4
  - a[n] is at offset n*4
- Size in bytes is size * # elements = 4 * 15 =60 bytes

Arrays are Awesome!

- They are fast
- They make sense
- They don't take any more space than they need
- They can store lots of useful data

BUT

- They are not part of "standard" Python
- We need to use a package…

NumPy

- Pronounced "num-pai"
- This is the core library for scientific computing in Python – everything else builds upon it
- Provides high-performance N-dimensional arrays
- Includes:
  - Operations and functions to manipulate arrays
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities

NUMPY ARRAYS

Fundamentals of Programming

Lecture 3

NumPy arrays

- To use an external package, we have to import it
- The convention with NumPy is to give it a new, shorter name np

```
import numpy as np
```

- Then we can refer to NumPy functions and create arrays with np

```
myarray = np.array([1, 2, 3])
```

- np.array is a function to create an array

Creating arrays

- Directly:

```
array1 = np.array([1, 2, 3, 4, 5])
                    [1, 2, 3, 4]
```

- From a list:

```
templist = [1.0, 2.0, 3.0, 4.0]
array2 = np.array( templist )
                    [1.0, 2.0, 3.0, 4.0]

templist2 = [1, 2, 3, 4, 5]
array2 = np.array(templist2, dtype=float)
                    [1.0, 2.0, 3.0, 4.0]
```

Accessing/updating elements

- As with lists and strings, we use an **index** to access array elements

```
e.g.
print('Element zero is: ', array1[0])
array1[2] = 5
sumoftwo = array1[1] + array1[2]
```

- Access parts of the array with **slicing**:

```
array1 = np.array([1, 2, 3, 4, 5])
print(array1[1:3])
```

```
[2, 3]
```

Preset arrays

- Create an array of 100 float (default), all = 0

```
zeroarray = np.zeros(100)
```

- Create an array of 100 integers, all set to 1

```
onesarray = np.ones(100, dtype=int)
```

- Array of 50 floats, each set to 100

```
hundreds = np.fill(50,100)
```

- Array of 50 random numbers

```
randarray = np.random.random(50)
```

```
   array([ 0.41976329,  0.52865412,  0.55652662, ...,
0.55957403, 0.03146015,  0.49496509])
```

Arrays with ranges of numbers

- Similar to using range to create a sequence for a for loop…
- Create an array of integers, going from 0 to 50, increasing by 5 each time

```
fives = np.arange(0,55,5)
print(fives)
     array([ 0,  5, 10, ..., 40, 45, 50])

tens = np.arange(0, 55, 10)
print(tens)
     array([0, 10, 20, 30, 40, 50])
```

Arrays with ranges of numbers

- An alternative is to give the start and stop values, and indicate how many values should be in between
- Create an array of integers, going from 0 to 50 (inclusive), increasing by 5 each time

```
tofifty = np.linspace(0, 50, 5)
print(tofifty)
          array([  0. ,  12.5,  25. ,  37.5,  50. ])

toten = np.linspace(0, 10, 10)
print(toten)
array([  0., 1.11111111, 2.22222222,  3.33333333,
4.44444444, 5.55555556,  6.66666667, 7.77777778,
8.88888889, 10. ])
```

Looping through arrays

- As with strings, there are multiple ways to go through the elements of an array…

```
index = 0
while index < len(tofifty):
    print('Element ', index,  ' is: ', tofifty[index])
    index = index + 1
```

Or…

```
for index in range(0, len(tofifty) ):
    print('Element ', index,  ' is: ', tofifty[index])
```

Or…

```
for item in tofifty:
    print('Element is: ', item)
```

Slicing arrays

- Arrays can be sliced in the same way as strings and lists: `myarray[start:stop:step]`
- If a slice term is missing, it will default to:
  - start = 0
  - stop = len(myarray)
  - step = 1
- `myarray[1:3]` will give elements 1 and 2
- `myarray[:3]` will give elements 0, 1 and 2
- `myarray[3:]` will give elements 3, 4, 5, …
- `myarray[3::2]` will give elements 3, 5, 7, …
- `myarray[::2]` will give elements 0, 2, 4, …

Slicing to make new arrays

- The result of a sliced array is an array, so we can use slicing to make new arrays

```
tofifty = np.linspace(0, 50, 5)
print(tofifty)
      array([  0. ,  12.5,  25. ,  37.5,  50. ])

tofifty2 = tofifty[1:3]
print(tofifty2)
                        array([ 12.5,  25. ])

tofifty3 = tofifty[::2]
print(tofifty3)
                     array([  0.,  25.,  50.])
```

MANIPULATING ARRAYS

Fundamentals of Programming

Lecture 3

Manipulating arrays

- So far arrays look pretty good – but not that different to strings and lists
- It's the tools for manipulation of arrays that make them powerful
- Many maths and science applications use matrix operations
- NumPy provides fast and easy implementation of matrix algebra

Operations

- Arithmetic operations are carried out on **each element** of an array

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = a + b             array([5, 7, 9])
c = a + 1             array([2, 3, 4])
c = a - b             array([-3, -3, -3])
c = a * b             array([4, 10, 16])
c = a / b             array([0.25, 0.4, 0.5])
```

Comparisons

- Can do element-wise comparisons of values using <, <=, >, >=, ==, !=
- Result is an array

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.array([6, 5, 4])

d = a < b       array([True,True,True], dtype=bool)
d = a < 2       array([True,False,False], dtype=bool)
d = b == c      array([False,True,False], dtype=bool)
d = b <= c      array([True, True,False], dtype=bool)
```

Element-wise Functions

- These functions are carried out on **each element** of an array

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.sqrt(a)
          array([ 1., 1.4142, 1.7320])
c = np.sin(a)
          array([ 0.8414, 0.9092, 0.1411])
```

`Also… exp(), cos(), log(), add(), multiply() etc.`

Array-wise Functions

- These functions return a single result across the array (or a dimension of the array)

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])


a.sum()        6
b.min()        4
b.max()        6
a.mean()       2.0
```

EXAMPLES

Fundamentals of Programming

Lecture 3

Exchange rates

```
usdprices = np.array([10, 20, 30 , 40, 50])
exchrate = 1.3
audprices = usdprices * 1.3
print(audprices)

[ 13.,  26.,  39.,  52.,  65.]
```

Adding GST

```
prices = np.array([10, 20, 30, 40, 50])
gstprices = prices * 1.1
print(gstprices)

[ 11.,  22.,  33.,  44.,  55.]
```

Conversion miles to km

```
distmiles = np.array([100, 150, 200, 250])
mileskm = 1.60934
distkm = distmiles * mileskm
print(distkm)

[ 160.934,  241.401,  321.868,  402.335]
```

Temperatures

```
march2017 = np.array([37.7, 29.9, 35.2, 36.1,
36.2, 34.7, 33.8, 34.5, 31.9, 29.9, 30.9,
24.8, 24.2, 24.1, 24.0])

print(march2017.min())
24.0

print(march2017.max())
37.700000000000003

print(march2017.mean())
31.193333333333332
```

http://www.bom.gov.au/climate/dwo/201703/html/IDCJDW6111.201703.shtml

PLOTTING WITH MATPLOTLIB

Fundamentals of Programming

Lecture 3

Plotting with Matplotlib

- Visualising data is one of the best tools for gaining insight and understanding
- Matplotlib is the preferred package for 2D graphics in Python
- Matplotlib tries to "make easy things easy and hard things possible"
- Includes plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.,
- Originally aligned to MATLAB plotting
- Plots are production-quality, so can be included in research papers
- http://matplotlib.org/

Matplotlib, origins

- When I went searching for a Python plotting package, I had several requirements:
  - Plots should look great - publication quality. One important requirement for me is that the text looks good (antialiased, etc.)
  - Postscript output for inclusion with TeX documents
  - Embeddable in a graphical user interface for application development
  - Code should be easy enough that I can understand it and extend it
  - Making plots should be easy
- Finding no package that suited me just right, I did what any self-respecting Python programmer would do: rolled up my sleeves and dived in.

John D. Hunter

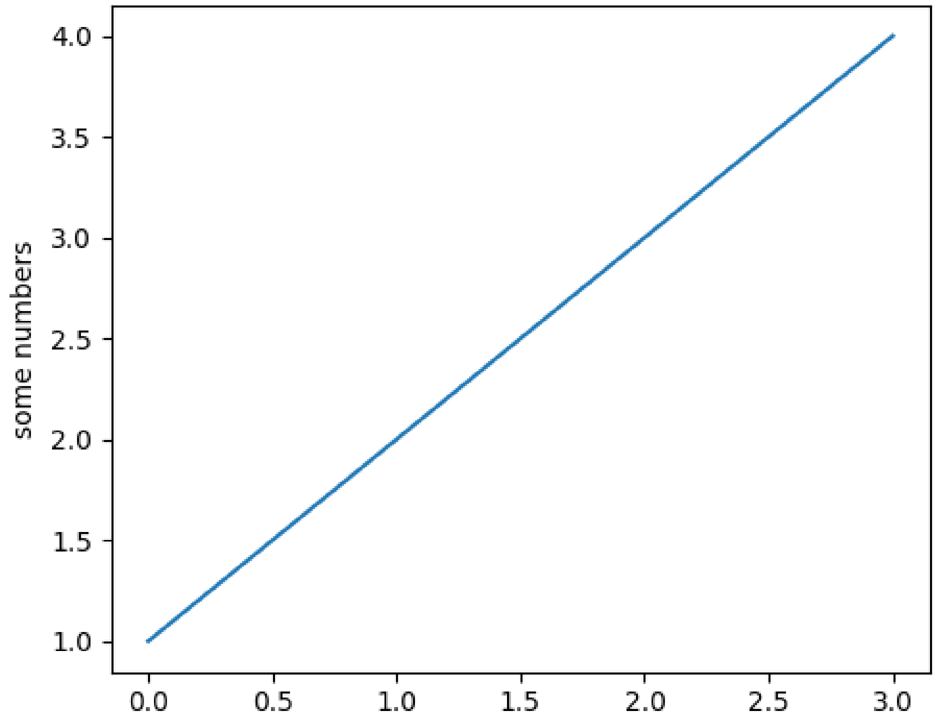
matplotlib.pyplot

- This is a collection of functions inside matplotlib that make it work like MATLAB
- Each pyplot function makes some change to a figure, e.g.
  - create a figure, create a plotting area, plot lines
- matplotlib.pyplot keeps track of the figure you are working on
- functions calls are directed to the current figure

Line Plot

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('Some Numbers')
plt.show()
```

- Assumes supplied values are y-values
- Uses default values for x-values



Anatomy of a plot

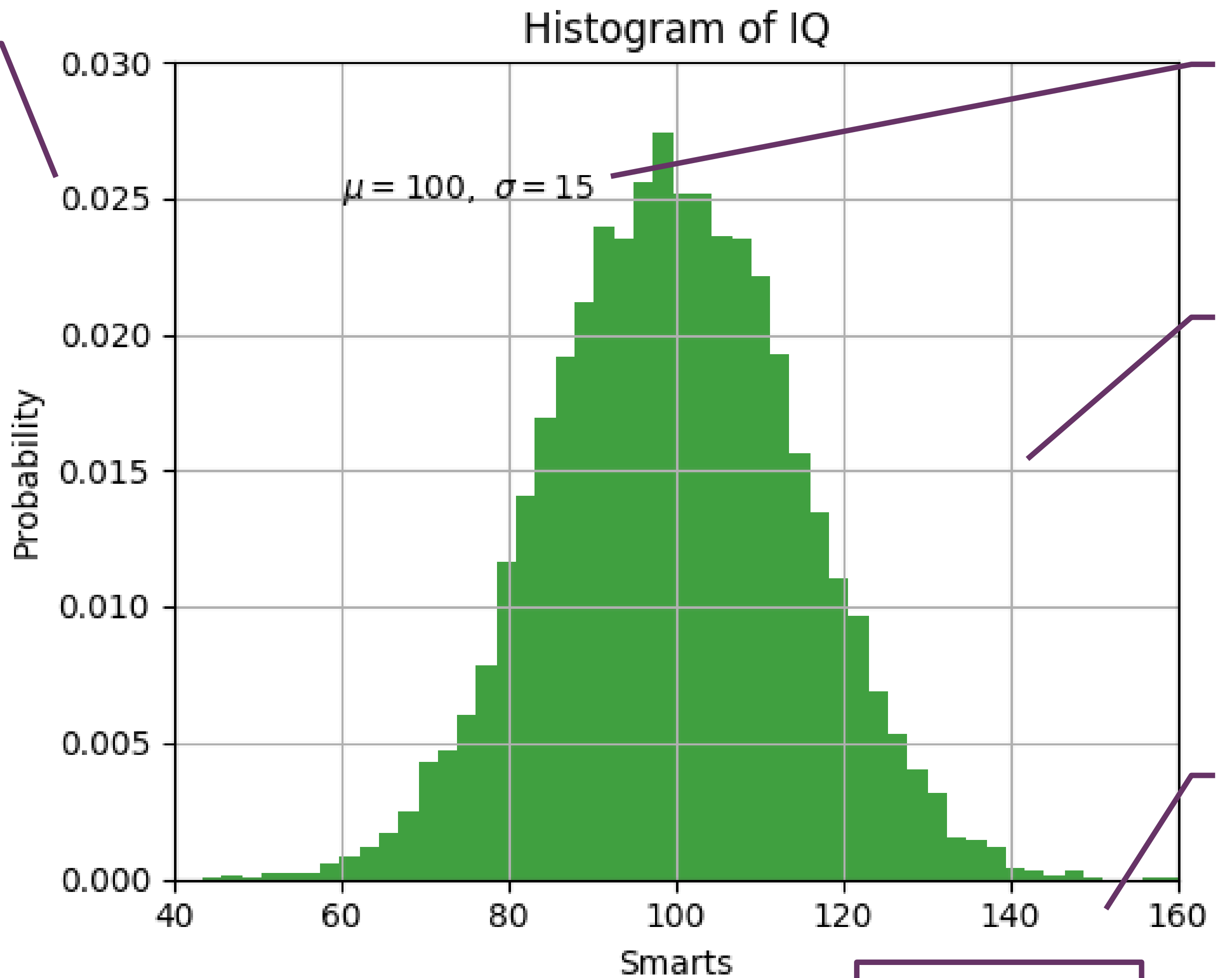
Title

y axis
0-0.3

text at
60, 0.025

ylabel

grid = True

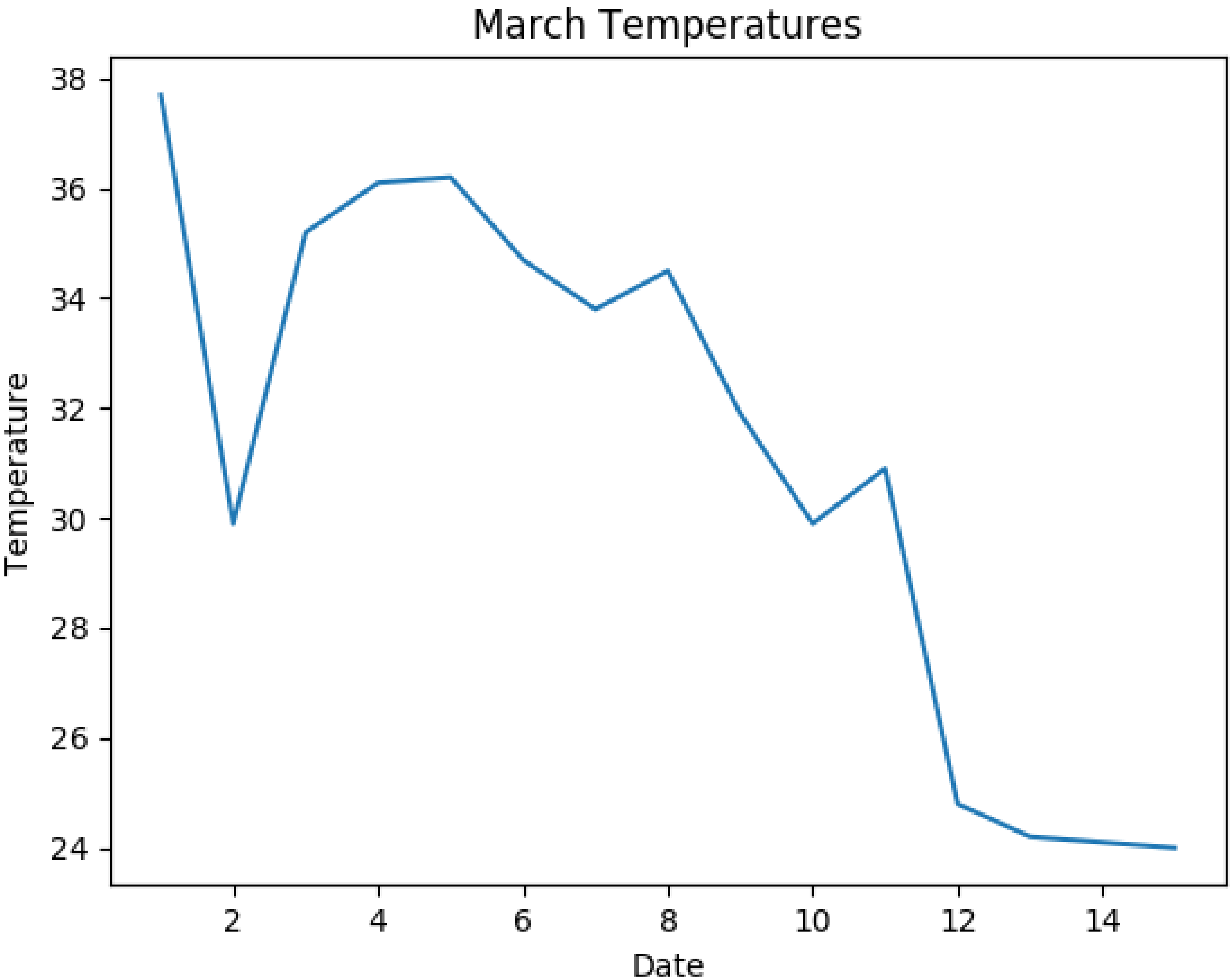


x axis
40-160

xlabel

March Temperatures

```
import matplotlib.pyplot as plt
import numpy as np
march2017 = np.array([37.7, 29.9, 35.2,
36.1, 36.2, 34.7, 33.8, 34.5, 31.9, 29.9,
30.9, 24.8, 24.2, 24.1, 24.0])
dates = range(1, len(march2017)+1)
plt.plot(dates, march2017)
plt.title('March Temperatures')
plt.ylabel('Temperature')
plt.xlabel('Date')
plt.show()
.
```

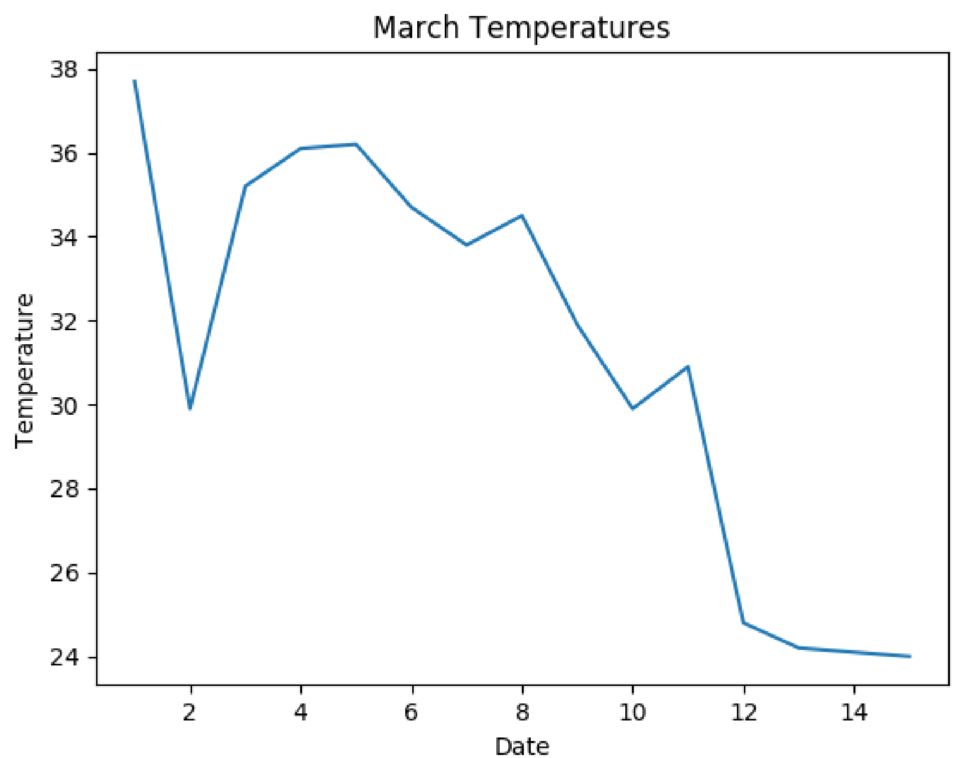


March Temperatures
Temperature
Date
38
36
34
32
30
28
26
24
2
4
6
8
10
12
14

March Temperatures

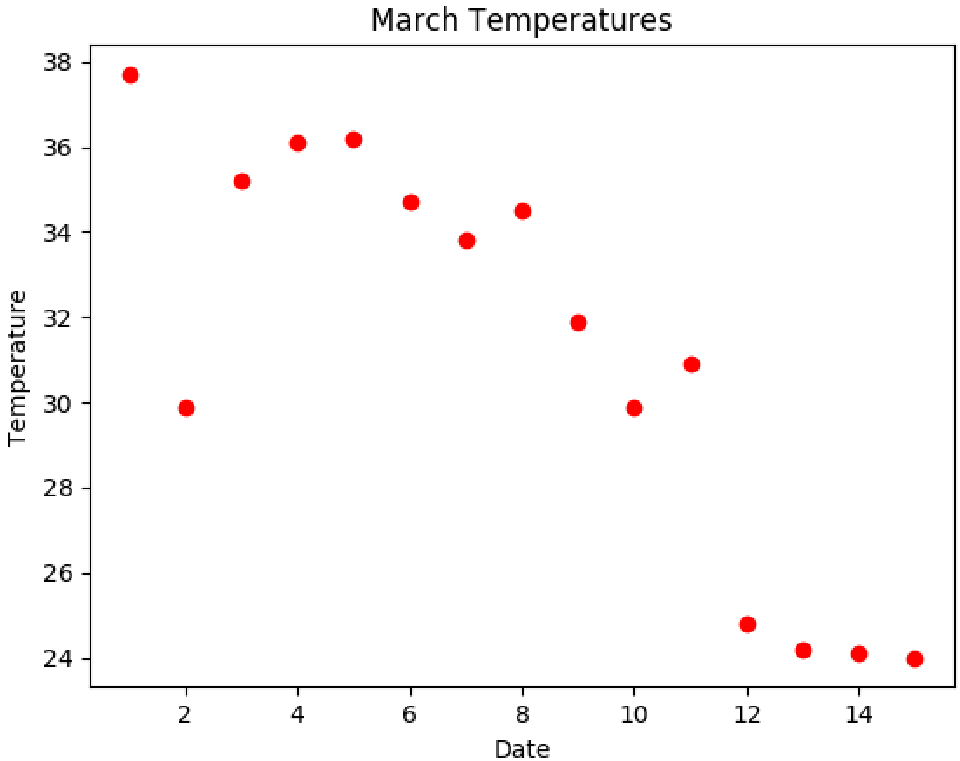
```
import matplotlib.pyplot as plt
import numpy as np
march2017 = np.array([37.7, 29.9, 35.2,
36.1, 36.2, 34.7, 33.8, 34.5, 31.9, 29.9,
30.9, 24.8, 24.2, 2
dates = range(1, le
plt.plot(dates, mar
plt.title('March Te
plt.ylabel('Tempera
plt.xlabel('Date')
plt.show()
```

•



March Temperatures – red dots

```
dates = range(1, len(march2017)+1)
plt.plot(dates, march2017, 'ro')
plt.title('March Temperatures')
plt.ylabel('Temperature')
plt.xlabel('Date')
plt.show()
```

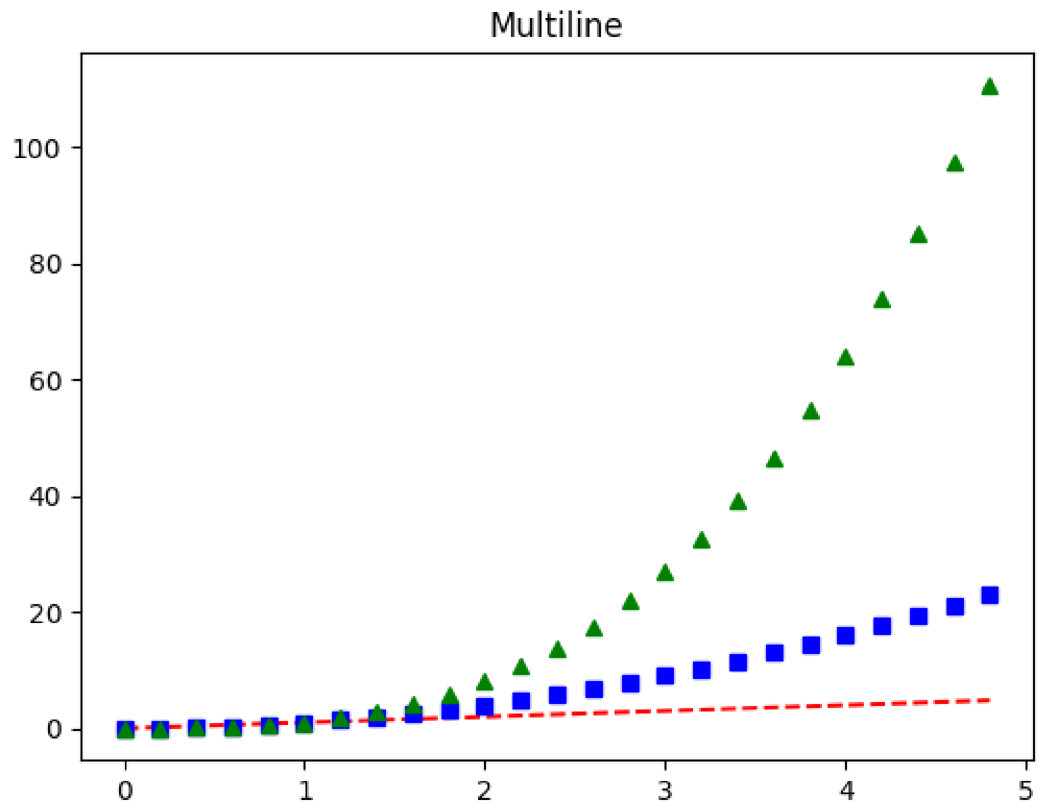


Multiple plots on an axis

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
t2 = t**2
t3 = t**3

# red dashes, blue squares and green triangles
plt.title('Multiline')
plt.plot(t, t, 'r--', t, t2, 'bs', t, t3, 'g^')
plt.show()
```

Multiple plots

```
import numpy as n
import matplotlik

# evenly sampled
t = np.arange(0.,
t2 = t**2
t3 = t**3
```




```
# red dashes, blu
plt.title('Multiline')
plt.plot(t, t, 'r--', t, t2, 'bs', t, t3, 'g^')
plt.show()
```

Subplots

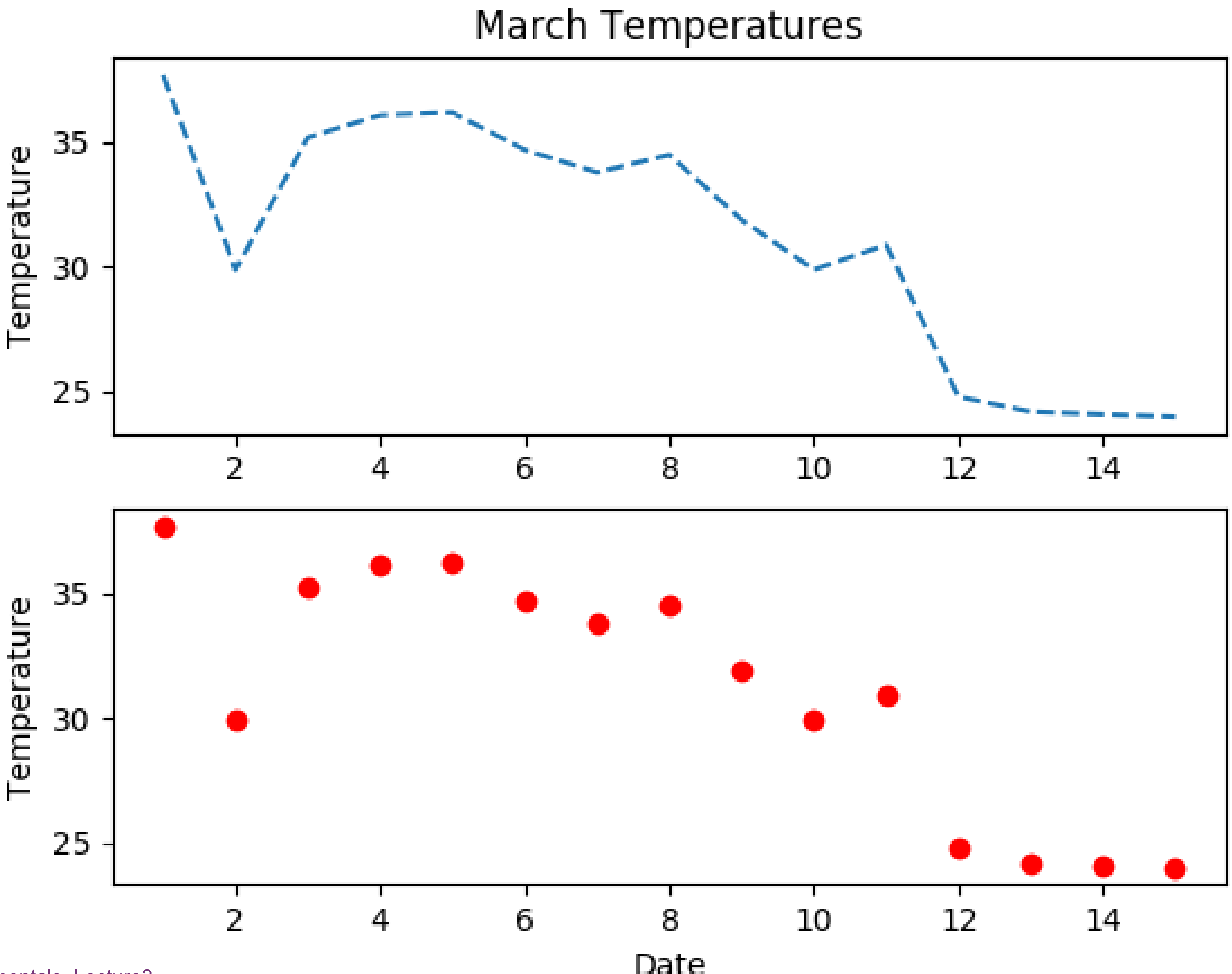
```
import matplotlib.pyplot as plt
import numpy as np
march2017 = np.array([37.7, 29.9, 35.2, 36.1, 36.2, 34.7,
33.8, 34.5, 31.9, 29.9, 30.9, 24.8, 24.2, 24.1, 24.0])
dates = range(1, len(march2017)+1)

plt.figure(1)
plt.subplot(211)
plt.plot(dates, march2017, '--')
plt.title('March Temperatures')
plt.ylabel('Temperature')

plt.subplot(212)
plt.plot(dates, march2017, 'ro')
plt.ylabel('Temperature')
plt.xlabel('Date')

plt.show()
```

plt.subplot(211)

numrows

numcols

subplot #

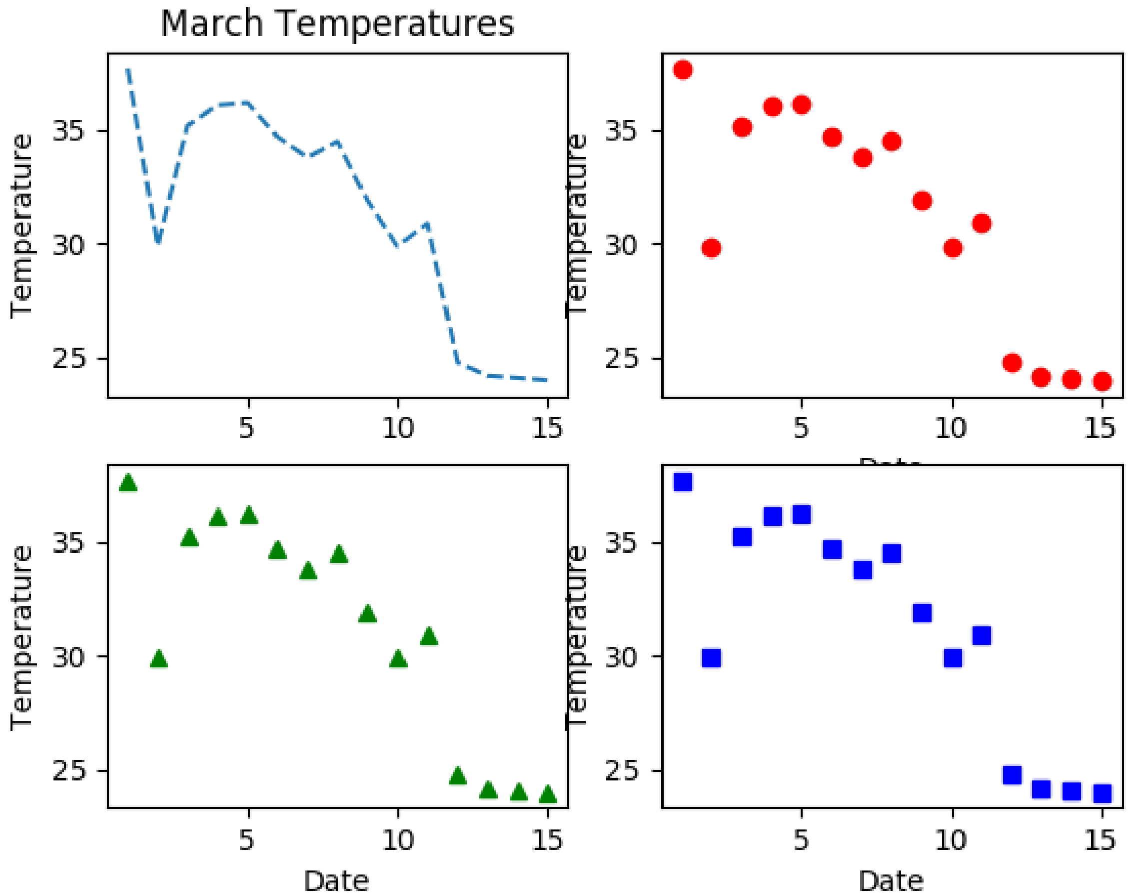


March Temperatures
Temperature
35
30
25
2
4
6
8
10
12
14
Temperature
35
30
25
2
4
6
8
10
12
14
Date

```
plt.figure(1)
plt.subplot(221)
plt.plot(dates, march2017, '--')
plt.title('March Temperatures')
plt.ylabel('Temperature')

plt.subplot(222)
plt.plot(dates, march2017, 'ro')
plt.ylabel('Temperature')
plt.xlabel('Date')

plt.subplot(223)
plt.plot(dates, march2017, 'g^')
plt.ylabel('Temperature')
plt.xlabel('Date')

plt.subplot(224)
plt.plot(dates, march2017, 'bs')
plt.ylabel('Temperature')
plt.xlabel('Date')

plt.show()
```

Subplots 2x2

```
plt.figure(1)
plt.subplot(221)
plt.plot(dates, m
plt.title('March
plt.ylabel('Tempe

plt.subplot(222)
plt.plot(dates, m
plt.ylabel('Tempe
plt.xlabel('Date'

plt.subplot(223)
plt.plot(dates, m
plt.ylabel('Tempe
plt.xlabel('Date'

plt.subplot(224)
plt.plot(dates, march2017, 'bs')
plt.ylabel('Temperature')
plt.xlabel('Date')

plt.show()
```



Subplots 2x2

Line Colours, Styles and Markers

```
plt.plot(t,t, 'r--', t,t2, 'bs', t, t3, 'g^')
```

- linestyles or ls ['-' | '--' | '-.' | ':' | 'steps' | ...]
- markers ['+' | ',' | '.' | '1' | '2' | '3' | '4'] s=square ^=triangle
- colours (short names) ['b' | 'g' | 'r' | 'c' | 'm' | 'y' | 'k' |'w']
- color='blue'
- color='pink'
- Lots of flexibility on colours – see http://matplotlib.org/users/colors.html for more information

Named Colours

Bar Charts

- Uses plt.bar() instead of plt.plot()

```
plt.bar(x_values,
           y_values,
           width,
           other arguments)
```

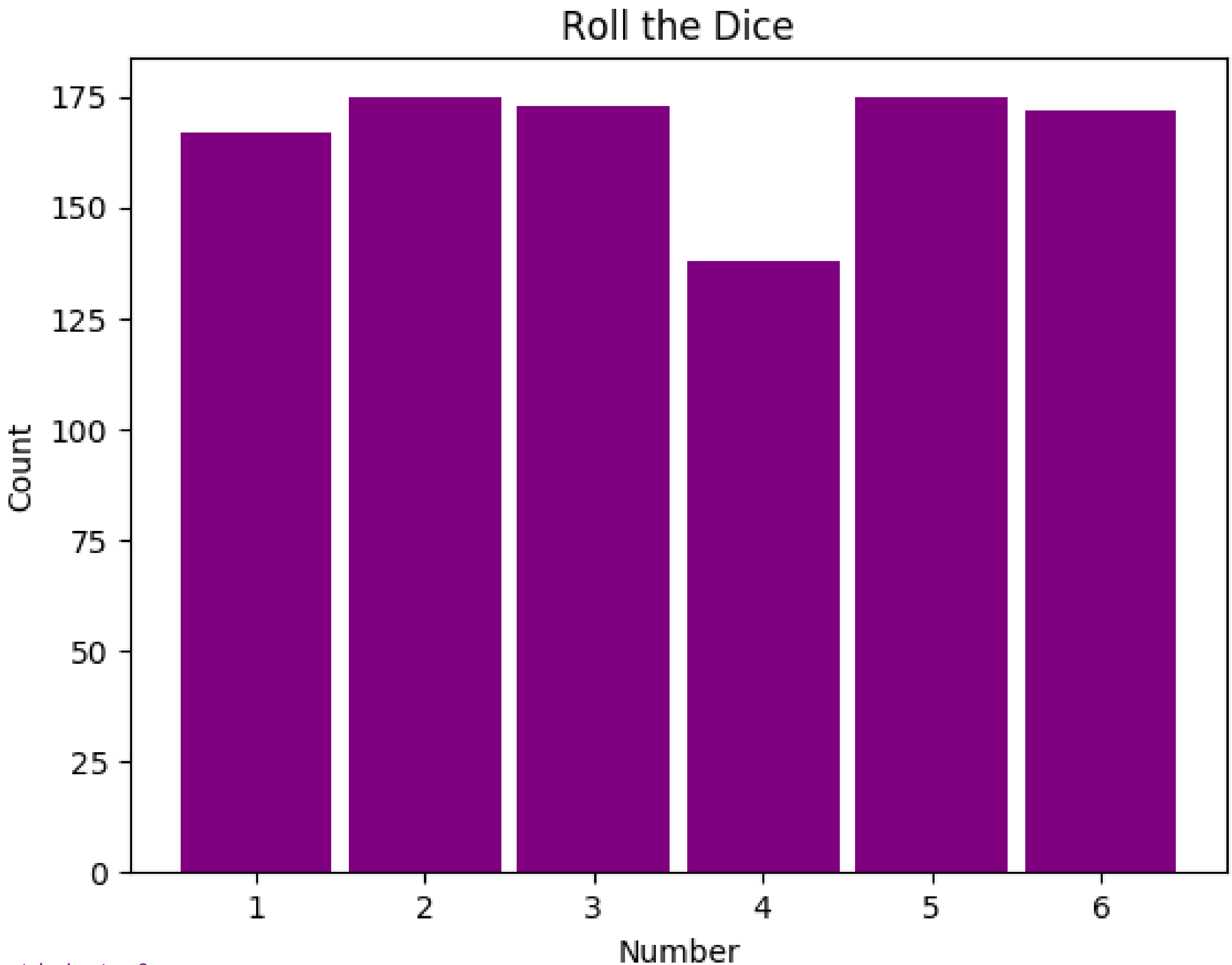
- Can also add series' of bars in different colours

Roll the Dice

```
import matplotlib.pyplot as plt
import random
dice = ['one','two','three','four','five','six']
one, two, three, four, five, six = 0,0,0,0,0,0

trials = 1000
print('\nDICE TOSS\n')
for index in range(trials):
   choice = random.choice(dice)
   if choice == 'one':
       one += 1
   elif choice == 'two':
       two += 1
   elif choice == 'three':
       three += 1
   elif choice == 'four':
       four += 1
   elif choice == 'five':
       five += 1
   else:
       six += 1
```

Roll the Dice (continued)

```
print('\nRESULTS\n')
print('1: ', one)
print('2: ', two)
print('3: ', three)
print('4: ', four)
print('5: ', five)
print('6: ', six)

plt.title('Roll the Dice')
plt.xlabel('Number')
plt.ylabel('Count')

plt.bar([1, 2, 3, 4, 5, 6],[one, two, three, four,
        five, six], 0.9, color='purple')
plt.show()
```


Roll the Dice
Count
175
150
125
100
75
50
25
0
1
2
3
4
5
6
Number

Histograms

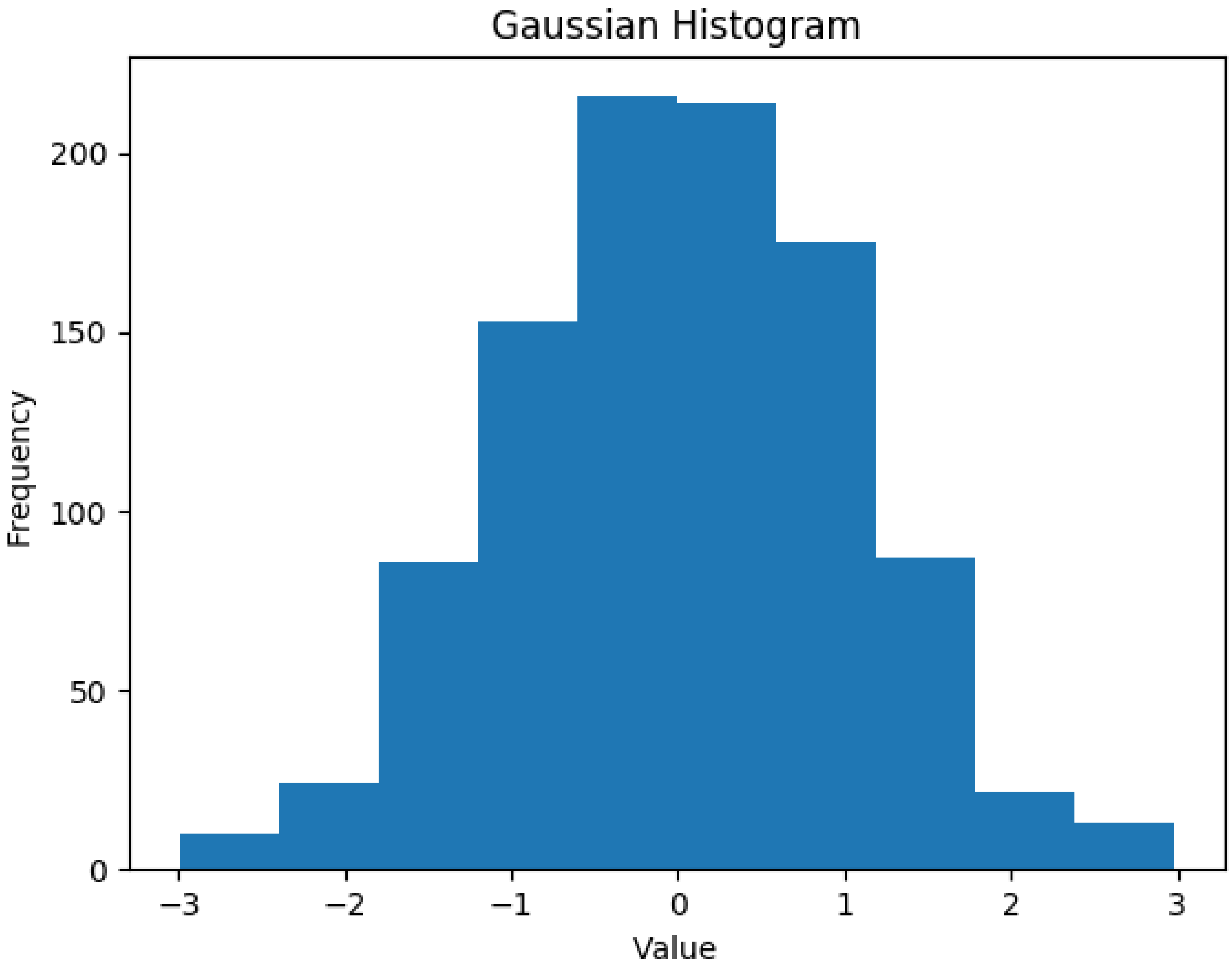
- Histograms give us a plot of the frequency of a value or event across a range of possible values
- For example, we could keep track of how many infinite loops occur per practical ☺
- Instead of plt.plot(), we use plt.hist()

```
plt.hist(data)
```

- Default number of bins is 10

Basic Histogram

```
import matplotlib.pyplot as plt
from numpy.random import normal

gaussian_numbers = normal(size=1000)

plt.hist(gaussian_numbers)
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```


Gaussian Histogram
Frequency
Value
200
150
100
50
0
−3
−2
−1
0
1
2
3

Histograms

- Can normalize to convert to probabilities

```
plt.hist(data, bins=20, normed=True)
```

- Can make the graph cumulative, so then the probability of the values up to that value

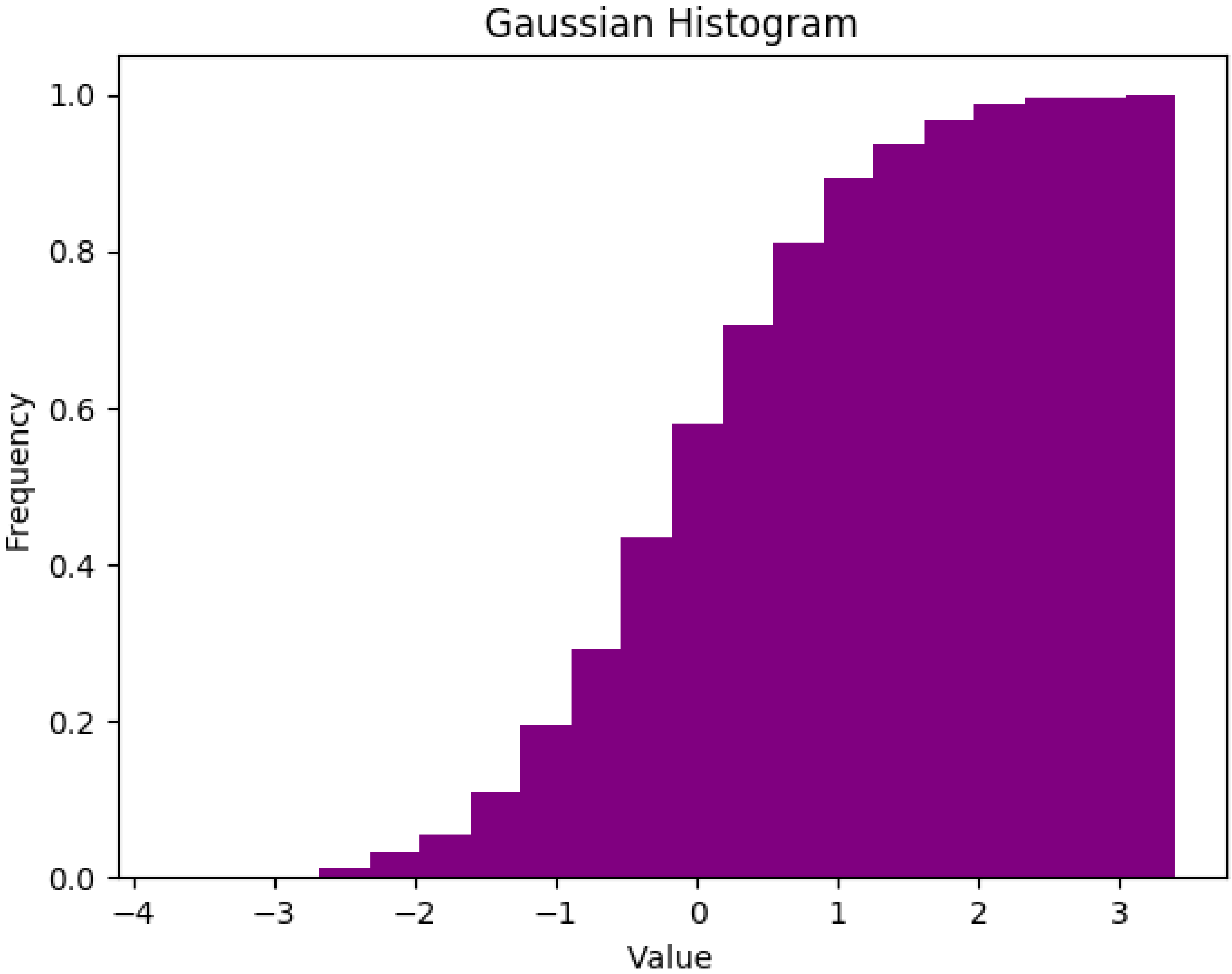
```
plt.hist(data, bins=20, cumulative = True)
```

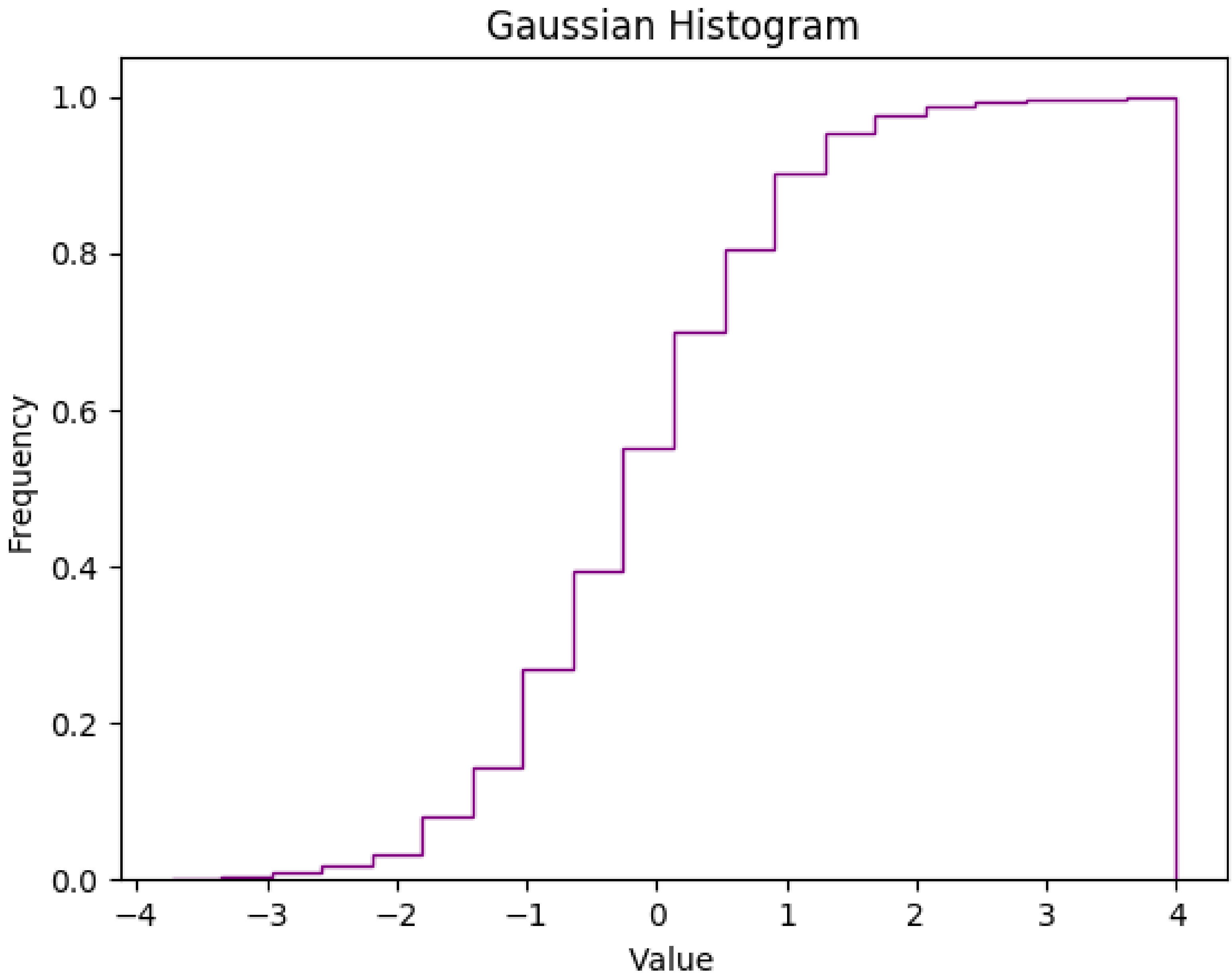
- Change the graph type the be a step outline rather than filled in

```
plt.hist(data, histtype='step', alpha=0.5)
```

- And, of course, change the colour

```
plt.hist(gaussian_numbers, bins=20,
         normed=True, cumulative=True,
         color=['purple'])
```


Gaussian Histogram
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
−4
−3
−2
−1
0
1
2
3
Value


Gaussian Histogram
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4
Value

EXAMPLE

Fundamentals of Programming

Lecture 3

growth.py

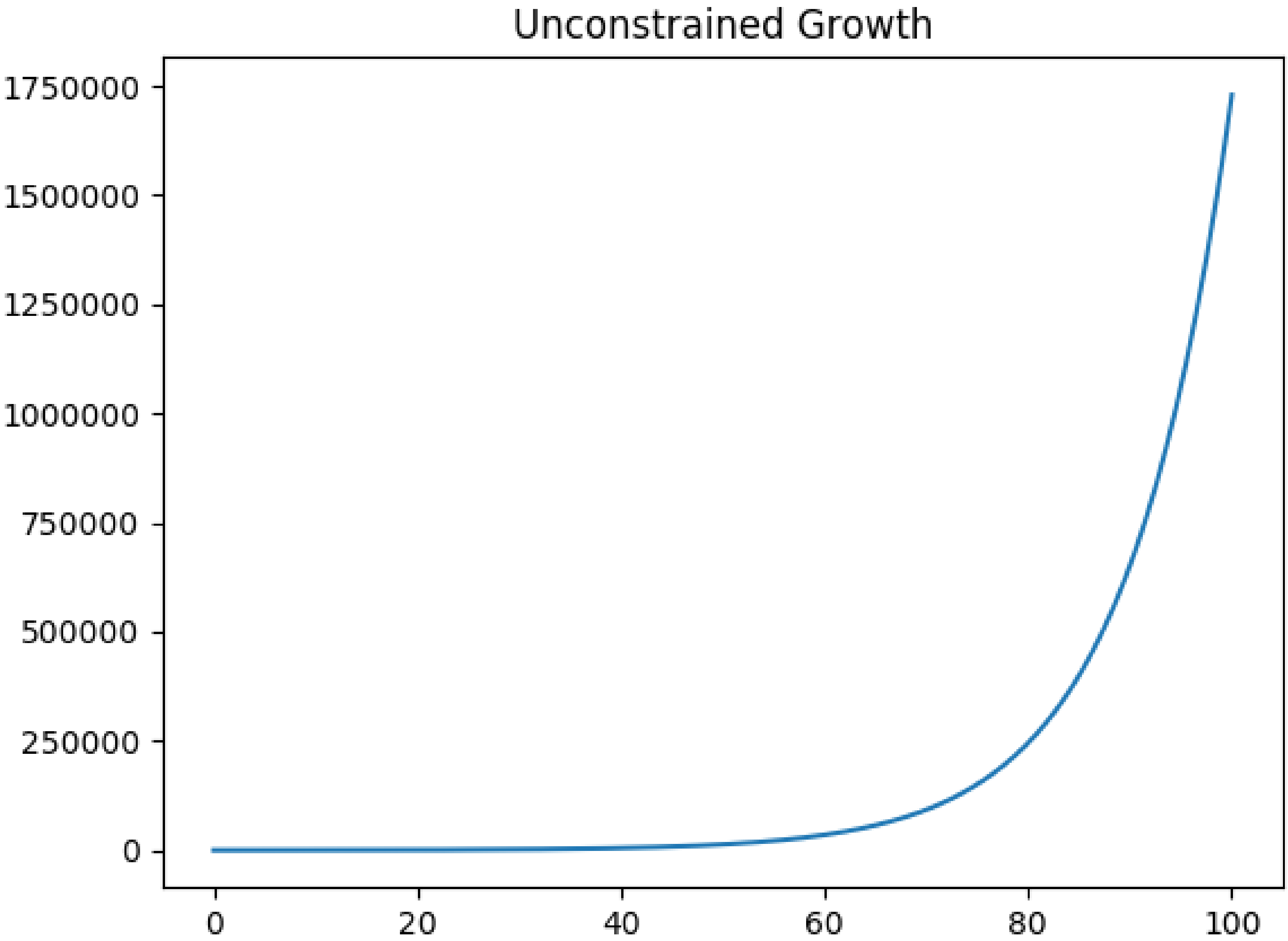
```
#
# growth.py - simulation of unconstrained growth
#
import matplotlib.pyplot as plt

print("\nSIMULATION - Unconstrained Growth\n")
length = 100
population = 100
growth_rate = 0.1
time_step = 0.5
num_iter = length / time_step
growth_step = growth_rate * time_step
```

growth.py

```
print("INITIAL VALUES:\n")
print("Simulation Length (hours): ", length)
print("Initial Population: ", population)
print("Growth Rate (per hour): ", growth_rate)
print("Time Step (part hour per step): ", time_step)
print("Num iterations (length * timestep / hour):",num_iter)
print("Growth step (growthrate per timestep):",growth_step)

print("\nRESULTS:\n")
print("Time: ", 0, " \tGrowth: ", 0, " \tPopulation: ", 100)
times=[0]
pops=[100]
```

growth.py

```
for i in range(1, int(num_iter) + 1 ):
    growth = growth_step * population
    population = population + growth
    time = i * time_step
    times.append(time)
    pops.append(population)
    print("Time: ", time, " \tGrowth: ", \
          growth," \tPopulation: ", population)
print("\nPROCESSING COMPLETE.\n")

plt.title('Unconstrained Growth')
plt.plot(times, pops)
plt.show()
```


Unconstrained Growth
1750000
1500000
1250000
1000000
750000
500000
250000
0
0
20
40
60
80
100

Summary

- We’ve looked at how to use Python arrays as implemented in the Numpy package
- We’ve looked at how to use simple plotting techniques from the matplotlib package
- We have applied arrays and plotting to more complex systems dynamics problems … and we’ll do more in the practicals

Practical Sessions

- Practical 2 was quite challenging!
- You will need to spend time on the pracs beyond the 2 hour class

- Labs 218, 219, 220 and 221 are all available to you and have Linux/Python
- Take note of when the FOP practicals are on – you can come in and ask questions
- The Senior Tutors are available 3 hours per day to help students – office near rm 218

Assessments

- The next assessment will be held during your assigned practical in Week 3/4 (Practical 3)
- It will be a short practical test using the lab computers
- Everyone should be able to get 100%!

Practical Test 1 - Instructions

- Create files and directories as instructed
  - mkdir, cd, vim
- Create Python program to match the description given
  - e.g. vim test.py
- Capture your command history into a file within the PracTest1 directory
  - e.g. history > hist.txt
- Zip your files and submit them through the assessment page
  - zip PracTest1_12345678 *

References

- Weather data : http://www.bom.gov.au/climate/dwo/201703/html/IDCJDW6111.201703.shtml
- Retail trade figures: http://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/8501.0Jan%202017?OpenDocument
- Pyplot tutorial: http://matplotlib.org/users/pyplot_tutorial.html
- Histograms: https://bespokeblog.wordpress.com/2011/07/11/basic-data-plotting-with-matplotlib-part-3-histograms/

Next week…

- Multi-dimensional arrays
- More plotting