Week 2 Assignment

July 20, 2025

1 SOFTWARE CARPENTRY FOR PYTHON

Variables

Any Python interpreter can be used as a calculator:

[3]:
```
3 + 5 * 4
```

[3]:
```
23
```

Types of data

Python knows various types of data. Three common ones are:

1. integer numbers
2. floating point numbers, and
3. strings.

In the example above, variable weight_kg has an integer value of 60. If we want to more precisely track the weight of our patient, we can use a floating point value by executing:

[5]:
```
weight_kg = 60.3
patient_id = '001'
weight_lb = 2.2 * weight_kg
patient_id = 'inflam_' + patient_id
```

2 Built-in Python functions

To carry out common tasks with data and variables in Python, the language provides us with several built-in functions.

[6]:
```
print(weight_lb)
print(patient_id)
```

```
132.66
inflam_001
```

We can display multiple things at once using only one print call:

[7]:
```
print(patient_id, 'weight in kilograms:', weight_kg)
```

```
inflam_001 weight in kilograms: 60.3
```

We can also call a function inside of another function call. For example, Python has a built-in function called type that tells you a value's data type:

[8]:
```
print(type(60.3))
print(type(patient_id))
```

```
<class 'float'>
<class 'str'>
```

3 Analyzing Patient Data

Loading data into Python To begin processing the clinical trial inflammation data, we need to load it into Python. We can do that using a library called NumPy, which stands for Numerical Python. In general, you should use this library when you want to do fancy things with lots of numbers, especially if you have matrices or arrays. To tell Python that we'd like to start using NumPy, we need to import it:

[9]:
```
import numpy
```

[10]:
```
numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
```

[10]:
```
array([[0., 0., 1., …, 3., 0., 0.],
       [0., 1., 2., …, 1., 0., 1.],
       [0., 1., 1., …, 2., 1., 1.],
       …,
       [0., 1., 1., …, 1., 1., 1.],
       [0., 0., 0., …, 0., 2., 0.],
       [0., 0., 1., …, 1., 1., 0.]], shape=(60, 40))
```

so we can do the same for all 12 inflammation data

Our call to numpy.loadtxt read our file but didn't save the data in memory. To do that, we need to assign the array to a variable. In a similar manner to how we assign a single value to a variable, we can also assign an array of values to a variable using the same syntax. Let's re-run numpy.loadtxt and save the returned data:

[11]:
```
data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
```

[12]:
```
print(data)
```

```
[[0. 0. 1. … 3. 0. 0.]
 [0. 1. 2. … 1. 0. 1.]
 [0. 1. 1. … 2. 1. 1.]
 …
 [0. 1. 1. … 1. 1. 1.]
 [0. 0. 0. … 0. 2. 0.]
 [0. 0. 1. … 1. 1. 0.]]
```

[13]:
```
print(type(data))
```

```
<class 'numpy.ndarray'>
```

[14]:
```
print(data.shape)
```

```
(60, 40)
```

[15]:
```
print('first value in data:', data[0, 0])
```

```
first value in data: 0.0
```

4 Slicing data

An index like [30, 20] selects a single element of an array, but we can select whole sections as well. For example, we can select the first ten days (columns) of values for the first four patients (rows) like this:

[16]:
```
print(data[0:4, 0:10])
```

```
[[0. 0. 1. 3. 1. 2. 4. 7. 8. 3.]
 [0. 1. 2. 1. 2. 1. 3. 2. 2. 6.]
 [0. 1. 1. 3. 3. 2. 6. 2. 5. 9.]
 [0. 0. 2. 0. 4. 2. 2. 1. 6. 7.]]
```

[17]:
```
print(data[5:10, 0:10])
```

```
[[0. 0. 1. 2. 2. 4. 2. 1. 6. 4.]
 [0. 0. 2. 2. 4. 2. 2. 5. 5. 8.]
 [0. 0. 1. 2. 3. 1. 2. 3. 5. 3.]
 [0. 0. 0. 3. 1. 5. 6. 5. 5. 8.]
 [0. 1. 1. 2. 1. 3. 5. 3. 5. 8.]]
```

[18]:
```
small = data[:3, 36:]
print('small is:')
print(small)
```

```
small is:
[[2. 3. 0. 0.]
 [1. 1. 0. 1.]
 [2. 2. 1. 1.]]
```

5 Analyzing data

NumPy has several useful functions that take an array as input to perform operations on its values. If we want to find the average inflammation for all patients on all days

Let's use three other NumPy functions to get some descriptive values about the dataset. We'll also use multiple assignment, a convenient Python feature that will enable us to do this all in one line.

[19]:
```
maxval, minval, stdval = numpy.amax(data), numpy.amin(data), numpy.std(data)
```

```
print('maximum inflammation:', maxval)
print('minimum inflammation:', minval)
print('standard deviation:', stdval)
```

```
maximum inflammation: 20.0
minimum inflammation: 0.0
standard deviation: 4.613833197118566
```

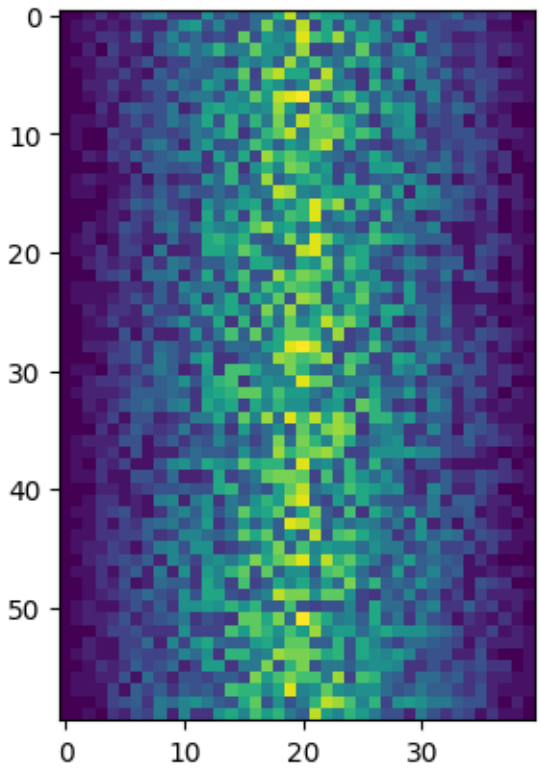
[20]:
```
patient_0 = data[0, :] # 0 on the first axis (rows), everything on the second␣
↪(columns)
print('maximum inflammation for patient 0:', numpy.amax(patient_0))
```

```
maximum inflammation for patient 0: 18.0
```

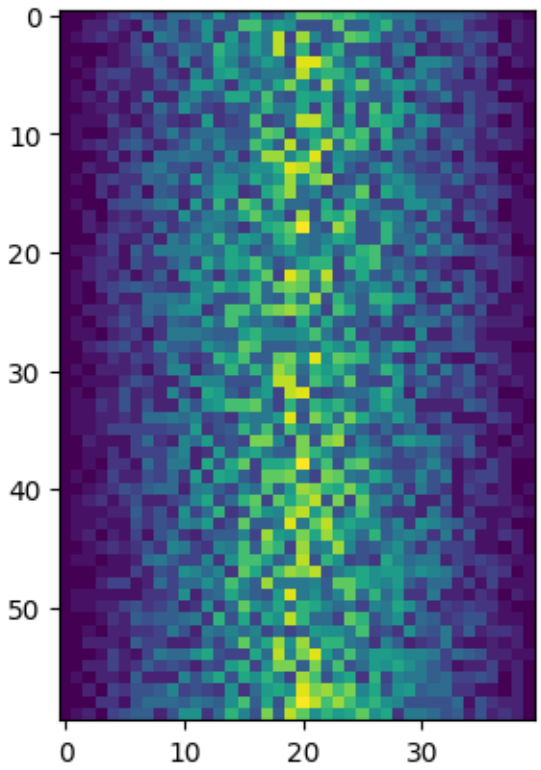
6 Visualizing data

The mathematician Richard Hamming once said, "The purpose of computing is insight, not numbers," and the best way to develop insight is often to visualize data. Visualization deserves an entire lecture of its own, but we can explore a few features of Python's matplotlib library here. While there is no official plotting library, matplotlib is the de facto standard. First, we will import the pyplot module from matplotlib and use two of its functions to create and display a heat map of our data:

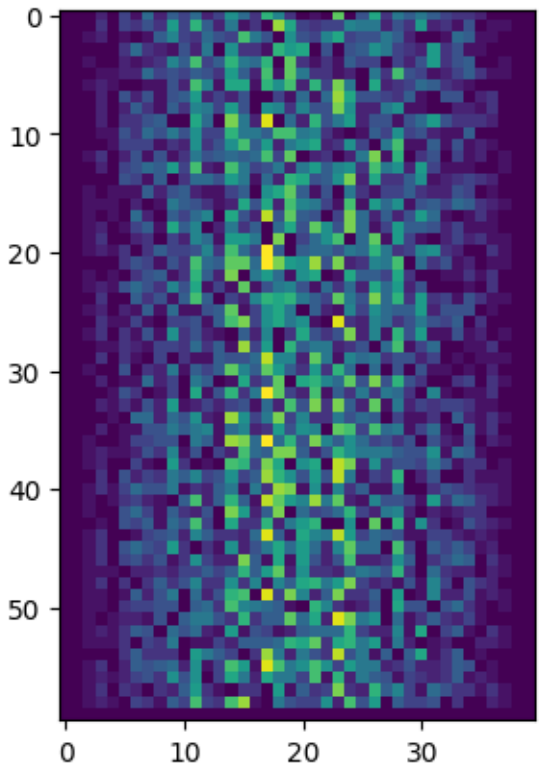
[21]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
import matplotlib.pyplot
image = matplotlib.pyplot.imshow(data)
matplotlib.pyplot.show()
```



[22]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-02.csv', delimiter=',')
import matplotlib.pyplot
image = matplotlib.pyplot.imshow(data)
matplotlib.pyplot.show()
```



[23]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-03.csv', delimiter=',')
import matplotlib.pyplot
image = matplotlib.pyplot.imshow(data)
matplotlib.pyplot.show()
```



we can go on but i wanna cut short it to 11 and 12 .

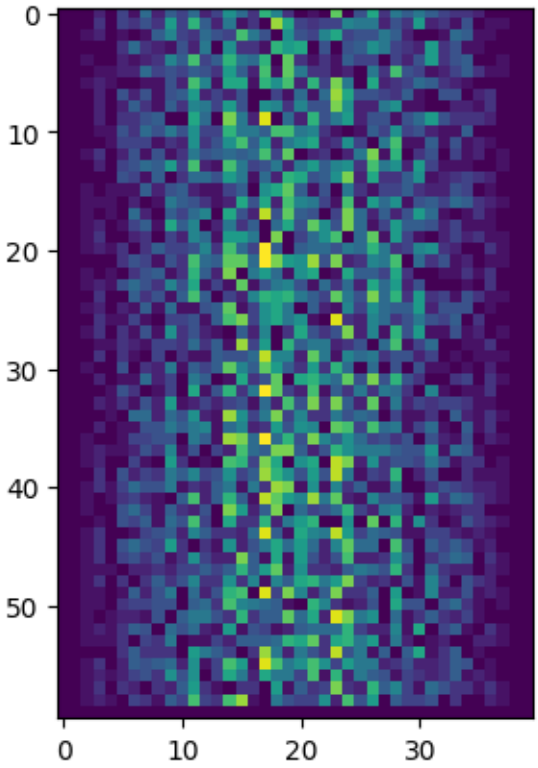
.

.

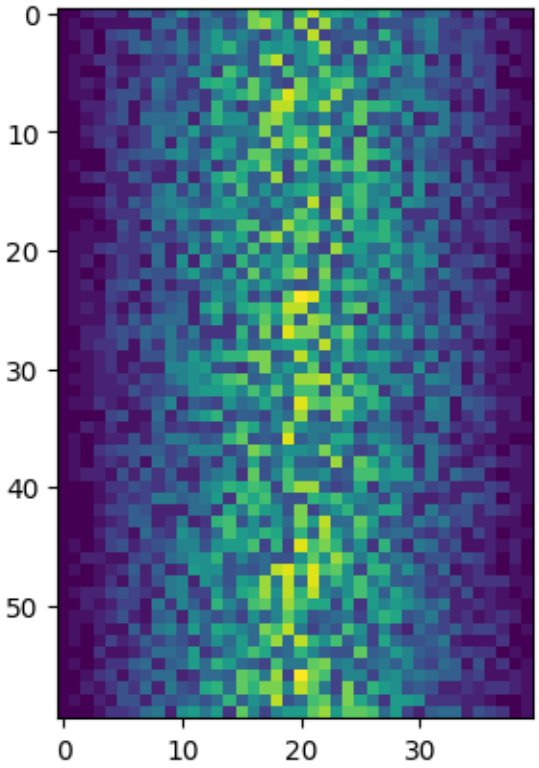
.

.

[24]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-11.csv', delimiter=',')
import matplotlib.pyplot
image = matplotlib.pyplot.imshow(data)
matplotlib.pyplot.show()
```



[26]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-12.csv', delimiter=',')
import matplotlib.pyplot
image = matplotlib.pyplot.imshow(data)
matplotlib.pyplot.show()
```

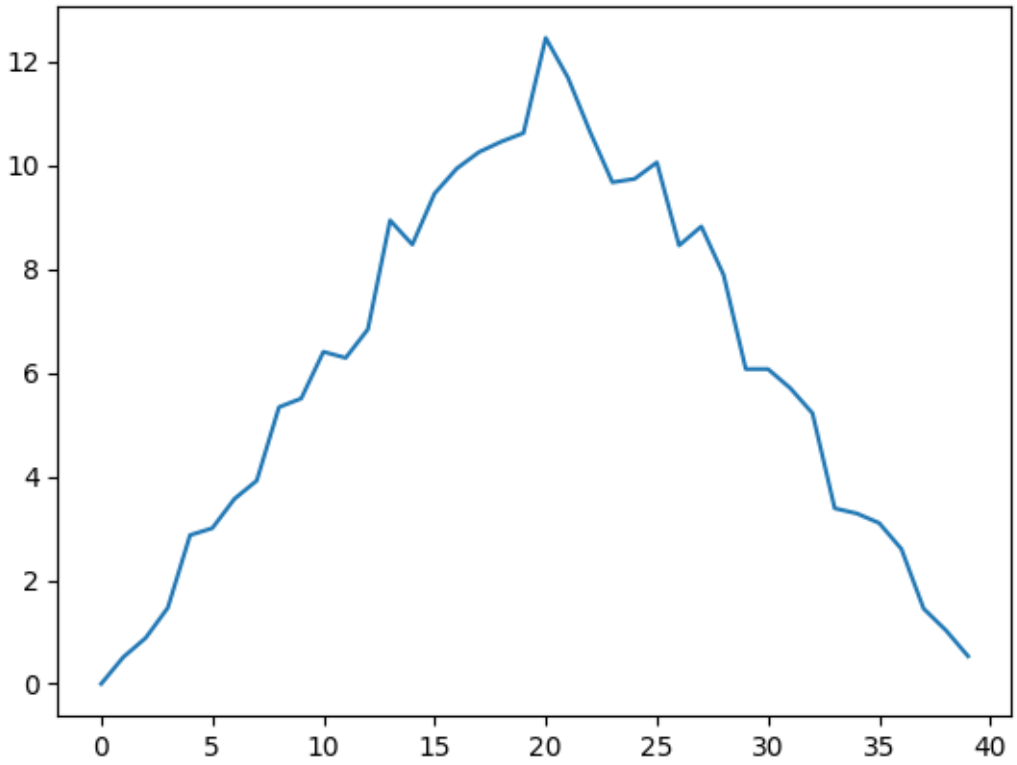


So far so good as this is in line with our knowledge of the clinical trial and Dr. Maverick's claims:

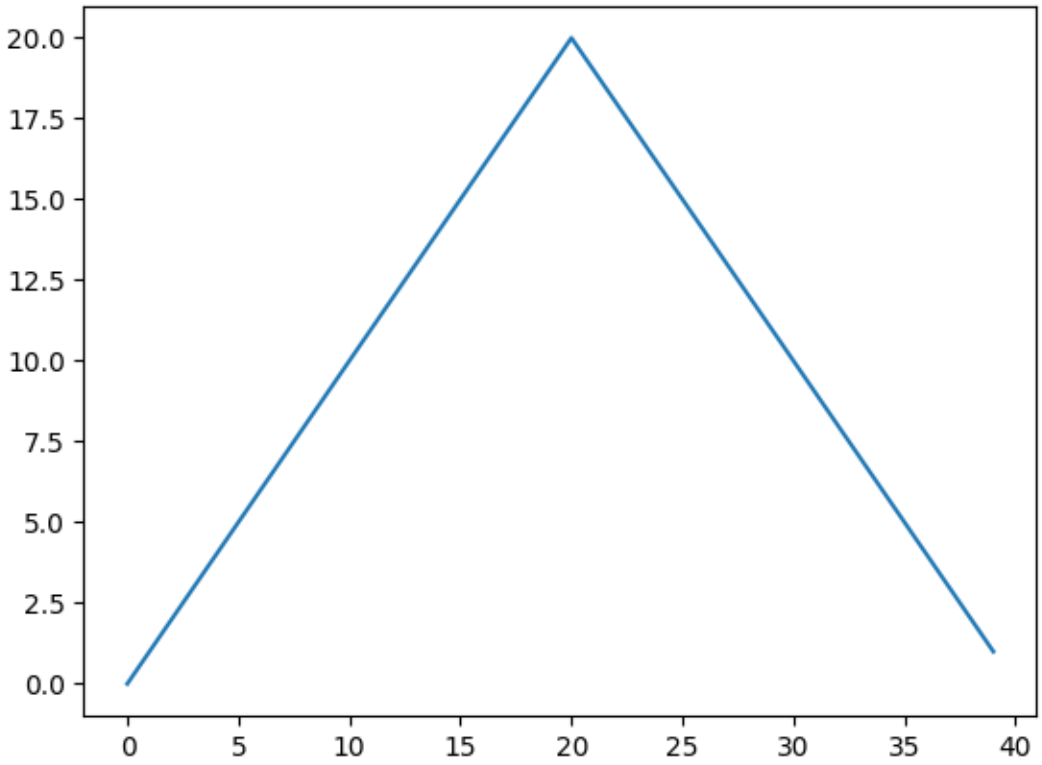
- the patients take their medication once their inflammation flare-ups begin
- it takes around 3 weeks for the medication to take effect and begin reducing flare-ups
- and flare-ups appear to drop to zero by the end of the clinical trial.

Now let's take a look at the average inflammation over time:

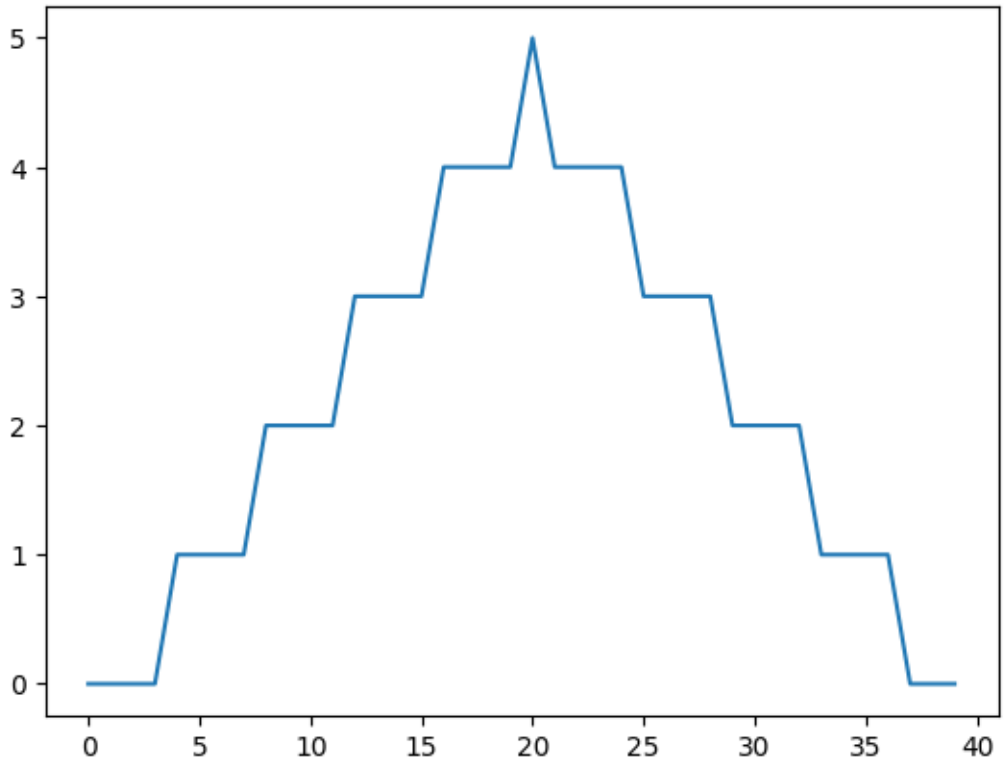
[27]:
```
ave_inflammation = numpy.mean(data, axis=0)
ave_plot = matplotlib.pyplot.plot(ave_inflammation)
matplotlib.pyplot.show()
```



[28]:
```
max_plot = matplotlib.pyplot.plot(numpy.amax(data, axis=0))
matplotlib.pyplot.show()
```



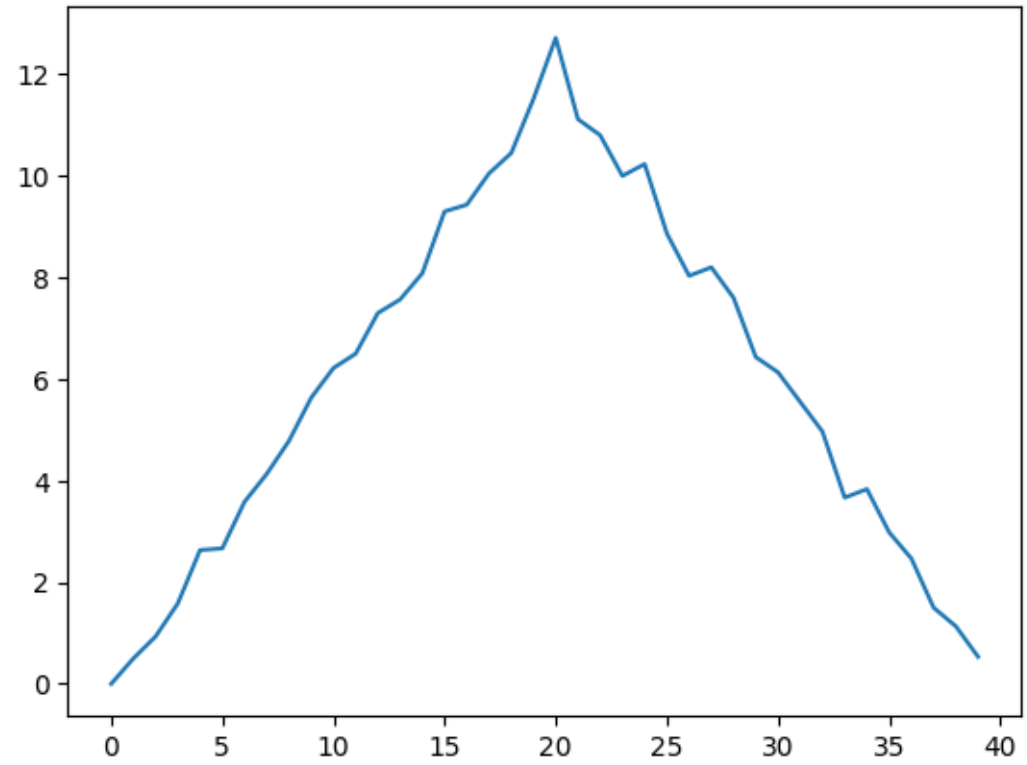
[29]:
```
min_plot = matplotlib.pyplot.plot(numpy.amin(data, axis=0))
matplotlib.pyplot.show()
```

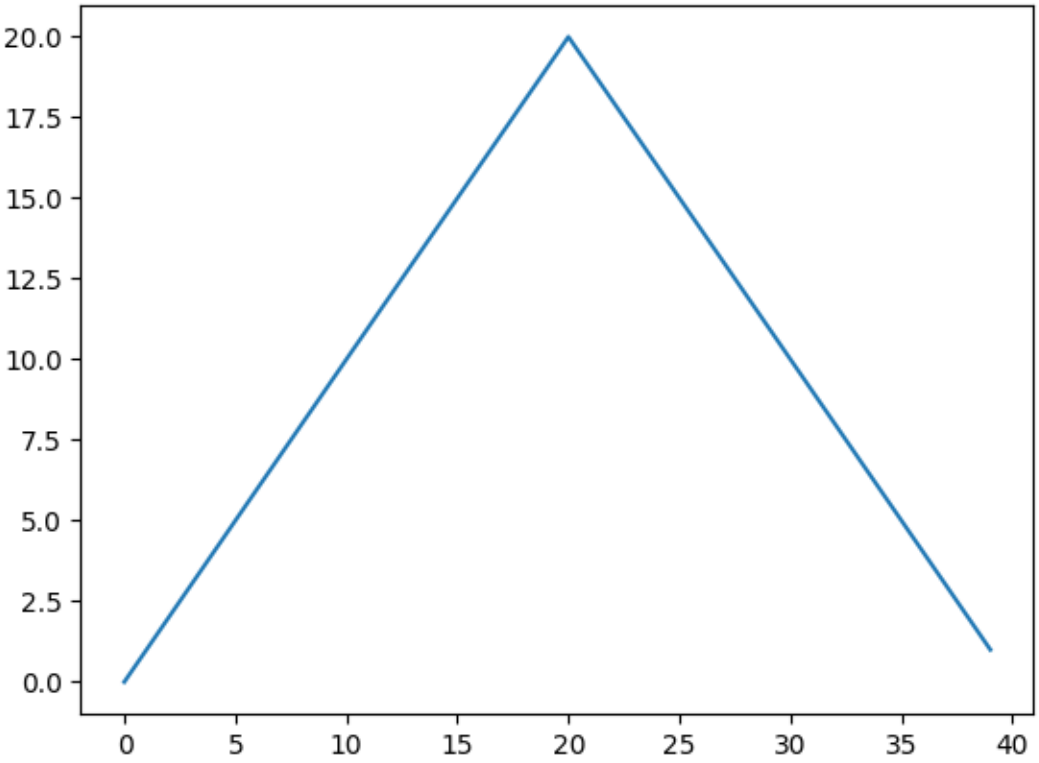


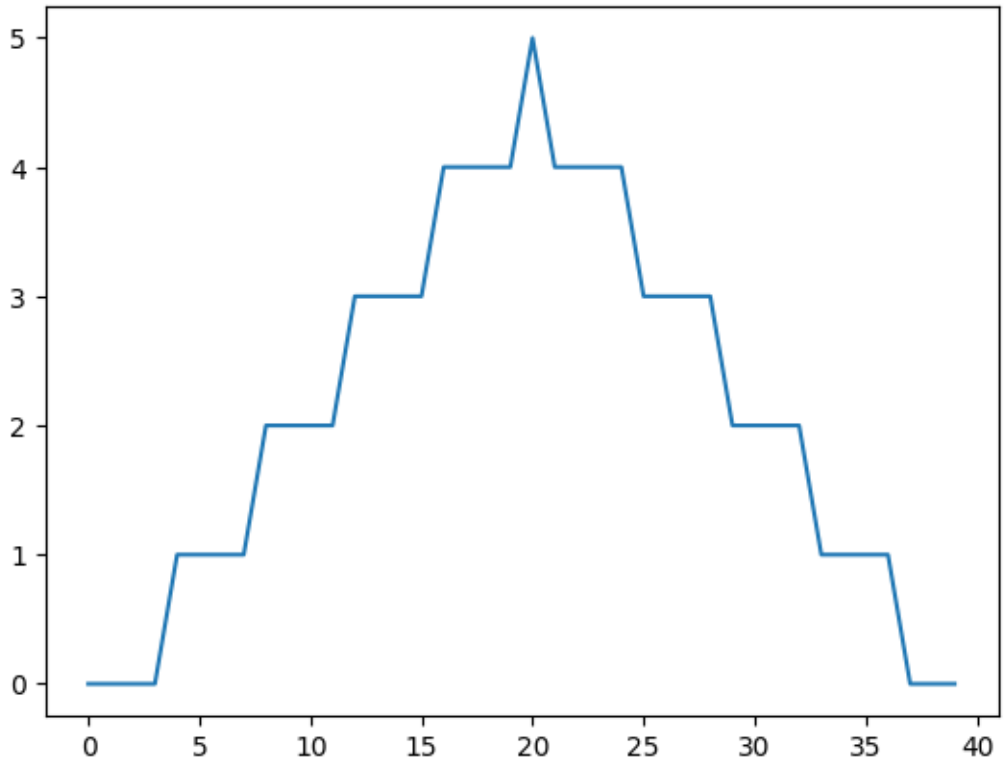
The above 3 graphs are from the “inflammation 12.csv” file because the last code run was from that file

If u want inflammation 02.csv

[40]:
```
import numpy
data = numpy.loadtxt(fname='inflammation-02.csv', delimiter=',')
import matplotlib.pyplot

ave_inflammation = numpy.mean(data, axis=0)
ave_plot = matplotlib.pyplot.plot(ave_inflammation)
matplotlib.pyplot.show()

max_plot = matplotlib.pyplot.plot(numpy.amax(data, axis=0))
matplotlib.pyplot.show()

min_plot = matplotlib.pyplot.plot(numpy.amin(data, axis=0))
matplotlib.pyplot.show()
```


0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
5
10
15
20
25
30
35
40

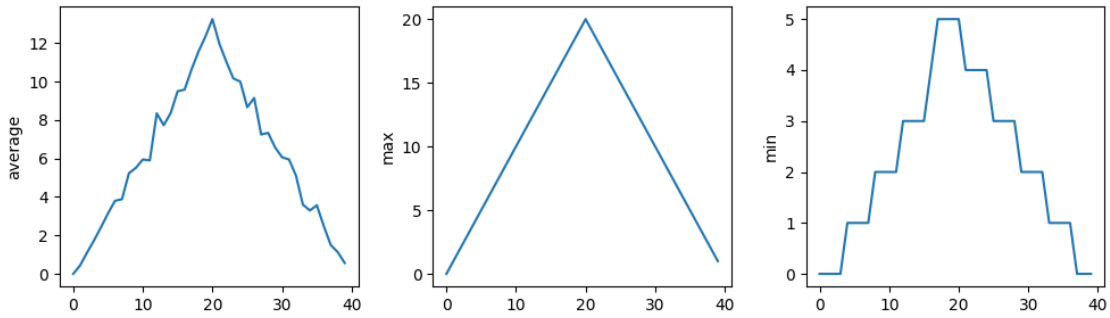


Grouping plots

You can group similar plots in a single figure using subplots. This script below uses a number of new commands.

[30]:
```
import numpy
import matplotlib.pyplot

data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')

fig = matplotlib.pyplot.figure(figsize=(10.0, 3.0))

axes1 = fig.add_subplot(1, 3, 1)
axes2 = fig.add_subplot(1, 3, 2)
axes3 = fig.add_subplot(1, 3, 3)

axes1.set_ylabel('average')
axes1.plot(numpy.mean(data, axis=0))

axes2.set_ylabel('max')
```

```
axes2.plot(numpy.amax(data, axis=0))

axes3.set_ylabel('min')
axes3.plot(numpy.amin(data, axis=0))

fig.tight_layout()

matplotlib.pyplot.savefig('inflammation.png')
matplotlib.pyplot.show()
```



7 Analyzing Data from Multiple Files

As a final piece to processing our inflammation data, we need a way to get a list of all the files in our data directory whose names start with inflammation- and end with .csv. The following library will help us to achieve this:

[31]:
```
import glob
```

[32]:
```
print(glob.glob('inflammation*.csv'))
```

```
['inflammation-01.csv', 'inflammation-02.csv', 'inflammation-03.csv',
'inflammation-04.csv', 'inflammation-05.csv', 'inflammation-06.csv',
'inflammation-07.csv', 'inflammation-08.csv', 'inflammation-09.csv',
'inflammation-10.csv', 'inflammation-11.csv', 'inflammation-12.csv']
```

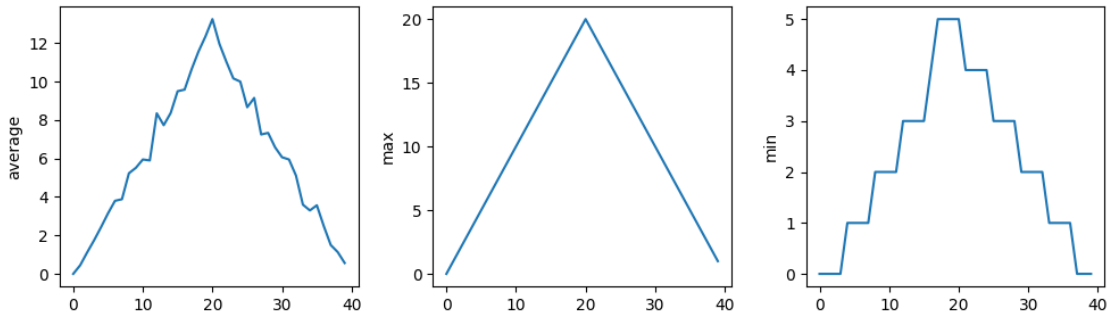
As these examples show, glob.glob's result is a list of file and directory paths in arbitrary order. This means we can loop over it to do something with each filename in turn. In our case, the "something" we want to do is generate a set of plots for each file in our inflammation dataset.

If we want to start by analyzing just the first three files in alphabetical order, we can use the sorted built-in function to generate a new sorted list from the glob.glob output:

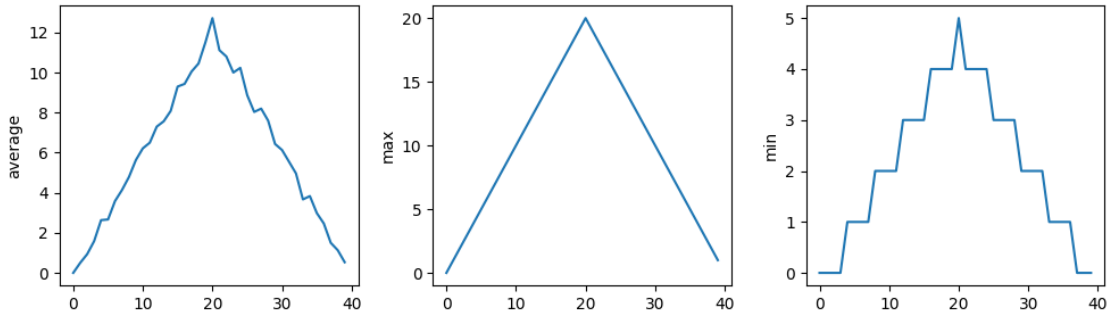
[33]:
```
import glob
import numpy
import matplotlib.pyplot
```

```
filenames = sorted(glob.glob('inflammation*.csv'))
filenames = filenames[0:3]
for filename in filenames:
    print(filename)

    data = numpy.loadtxt(fname=filename, delimiter=',')

    fig = matplotlib.pyplot.figure(figsize=(10.0, 3.0))

    axes1 = fig.add_subplot(1, 3, 1)
    axes2 = fig.add_subplot(1, 3, 2)
    axes3 = fig.add_subplot(1, 3, 3)

    axes1.set_ylabel('average')
    axes1.plot(numpy.mean(data, axis=0))

    axes2.set_ylabel('max')
    axes2.plot(numpy.amax(data, axis=0))

    axes3.set_ylabel('min')
    axes3.plot(numpy.amin(data, axis=0))

    fig.tight_layout()
    matplotlib.pyplot.show()
```

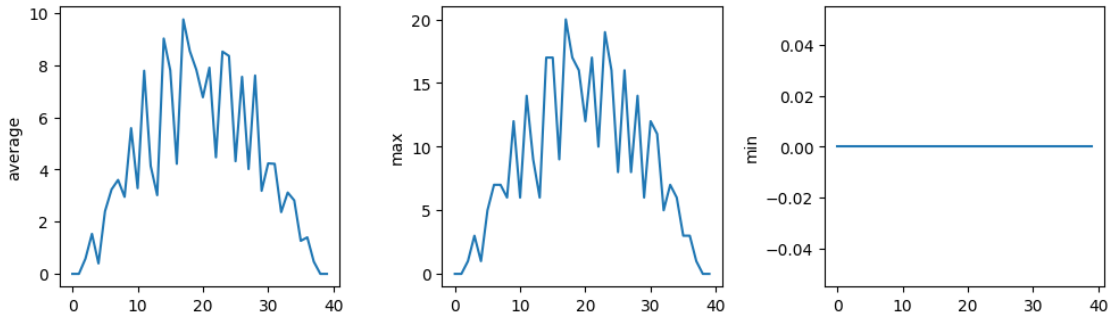
inflammation-01.csv



inflammation-02.csv



inflammation-03.csv



8 Checking our Data

Now that we've seen how conditionals work, we can use them to check for the suspicious features we saw in our inflammation data. We are about to use functions provided by the numpy module again. Therefore, if you're working in a new Python session, make sure to load the module and data with:

[39]:
```
import numpy as np
data = numpy.loadtxt(fname='inflammation-11.csv', delimiter=',')

max_inflammation_0 = numpy.amax(data, axis=0)[0]
max_inflammation_20 = numpy.amax(data, axis=0)[20]

if max_inflammation_0 == 0 and max_inflammation_20 == 20:
    print('Suspicious looking maxima!')
elif numpy.sum(numpy.amin(data, axis=0)) == 0:
    print('Minima add up to zero!')
else:
    print('Seems OK!')
```

```
Minima add up to zero!
```

[36]:
```
data = numpy.loadtxt(fname='inflammation-07.csv', delimiter=',')

max_inflammation_0 = numpy.amax(data, axis=0)[0]
max_inflammation_20 = numpy.amax(data, axis=0)[20]

if max_inflammation_0 == 0 and max_inflammation_20 == 20:
    print('Suspicious looking maxima!')
elif numpy.sum(numpy.amin(data, axis=0)) == 0:
    print('Minima add up to zero!')
else:
    print('Seems OK!')
```

```
Suspicious looking maxima!
```

THIS IS ALL LEARNT FROM THE CARPENTRY