

Lesson 8 - Numerical Python & Matplotlib

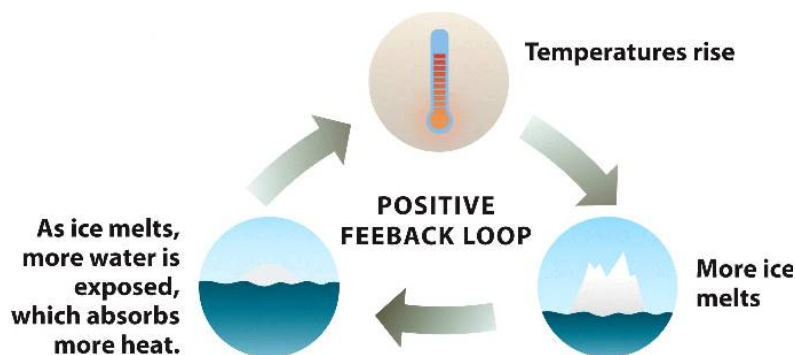
Manipulating and visualizing data

The Albedo Effect

Different surfaces on the Earth reflect or absorb the sun's radiation in different proportions - what is referred to as a surface's **albedo**. As demonstrated by the data below, the smaller the albedo, the lower the correlating temperatures. For example, forest has an albedo of 10%. Since it absorbs more of the sun's radiation than fresh snow (an albedo of 85%) it would be much warmer.

Temperatures (F)	265	255	245	235	225	215
Albedo	0.15	0.25	0.35	0.45	0.55	0.65

Albedo has a significant impact on the warming effect of the Earth. For example, [sea ice located in the Arctic](#) has a high albedo as it reflects approximately 80% (or 0.8 as expressed as a ratio) of sunlight back into space. If sea ice is melting, and the darker surface of the ocean absorbs more light than was reflected by the ice (approx. 8-10%), then this would further cause more ice to melt. This warming effect, in which the effect further increases the original cause, is an example of a **positive feedback loop**.



To learn more about the albedo effect, you can watch this NASA [video](#), or use this NatGeo [activity](#).

Source for albedo data: An online course taught by Dr. David Archer called, "Global Warming II: Create Your Own Models in Python."

Image source: [Bio1110](#)

Many different libraries were used to process data for major scientific breakthroughs, including the [first image of a black hole](#), which used a combination of NumPy, Matplotlib, as well as other libraries. NumPy is a library that contains multi-dimensional arrays (rather than 1D lists), as well as arithmetic functions to operate on these arrays. As demonstrated below, arrays are a different data type than a list (part 1), and since they use less memory, they are much faster to execute.

Part 1: Import the NumPy library and use the array() function

1. Import the library as np so when referring to it later you can just type “np”.

```
import numpy as np
```

2. Create two lists and pass them through your NumPy array by using a dot to access np (similarly to how a dot was used to access an object) and the array() function.

```
a = np.array([1,3,5])
```

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

As you can see, if you tried to multiply these two lists without passing them through a NumPy array, you would not be able to multiply them (an error message would be generated):

```
a = [1,3,5]
b = [1,2,3]
print(a*b)
# error
```

```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    print(a*b)
TypeError: can't multiply sequence by non-int of type 'list'
```

However, once passing the lists through the NumPy array, it is possible to do execute this function:

```
main.py
12 import numpy as np
13 a = np.array([1,3,5])
14 b = np.array([1,2,3])
15
16 print(a*b)
17
```

Console Shell

```
[ 1  6 15]
>
```

NumPy arrays can also have multiple dimensions. The examples above are both 1D arrays. A 2D array would have two arrays, a 3D array would have three arrays, and so on.

3. Create a 2D array with the arrays [1,2,3] and [4,5,6].

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

4. Check the number of dimensions by using a dot to access the variable, followed by ndim for both the variables a and c.

```
print(a.ndim)
```

```
print(c.ndim)
```

5. Classify the variable b using type().

```
print(type(b))
```

Example:

```
20 import numpy as np
21 a = np.array([1,3,5])
22 b = np.array([1,2,3])
23
24 print(a.ndim)
25 # # one dimension
26
27 c= np.array([[1,2,3],[4,5,6]])
28 print(c.ndim)
29 # # two dimensions
30
31 print(type(b))
```

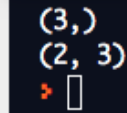
```
1
2
<class 'numpy.ndarray'>
> []
```

NumPy arrays also have a shape - the number of dimensions, and the number of elements. The shape attribute can be determined by placing a dot after the variable and the word “shape”.

6. Print the shape for variables b and c.

Example:

```
37 print(b.shape)
38 # it's only printing the number of
   # elements since it's 1D
39
40 print(c.shape)
--
```



```
(3,)
(2, 3)
> |
```

Part 2: Introduction to Matplotlib

Matplotlib is a graphing library that allows the development of plots and interactive figures, with many customizable features. To illustrate how to use matplotlib, we will be using data collected from the [Mauna Loa Observatory](#) in Hawaii:

Year	Atmospheric CO ₂ concentrations (ppm)
2017	406.36
2018	408.15
2019	411.03
2020	413.61
2021	415.52

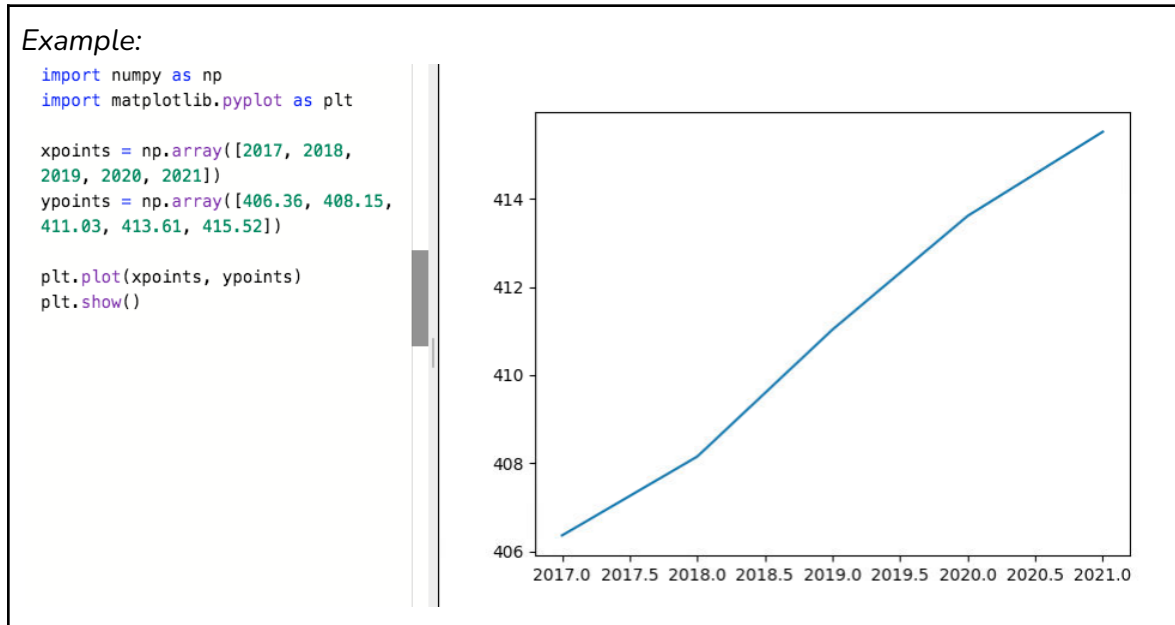
1. Import both numpy (as np) and matplotlib.pyplot (as plt).

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

2. Create the variables “xpoints” and “ypoints” and pass the years and the CO₂ concentrations as NumPy arrays.

```
xpoints = np.array([2017, 2018, 2019, 2020, 2021])
ypoints = np.array([406.36, 408.15, 411.03, 413.61, 415.52])
```

3. Plot the points by using the plot() function and accessing the library with a dot.
`plt.plot(xpoints, ypoints)`
4. Show the plot by using the show() function and accessing the library with a dot.
`plt.show()`



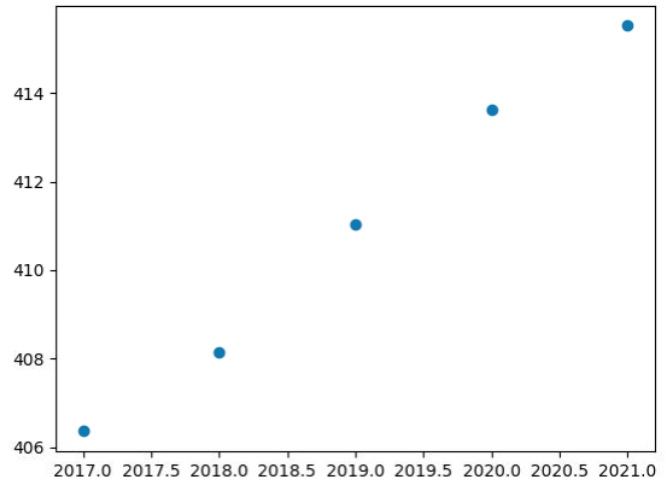
There are a few ways to customize the graph. For example, using 'o' prints just the points of the plot and using marker= 'o' prints both the points and the line.

Examples:

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

xpoints = np.array([2017, 2018,
2019, 2020, 2021])
ypoints = np.array([406.36, 408.15,
411.03, 413.61, 415.52])

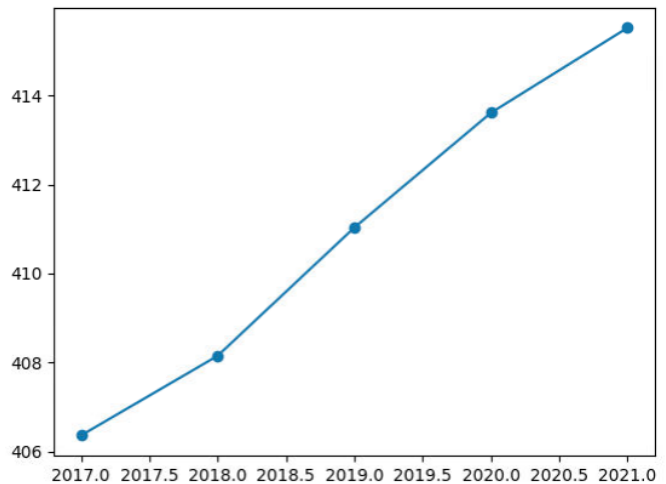
# plotting without a line
plt.plot(xpoints, ypoints, 'o')
plt.show()
```



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

xpoints = np.array([2017, 2018,
2019, 2020, 2021])
ypoints = np.array([406.36, 408.15,
411.03, 413.61, 415.52])

# plotting with both points and a
line
plt.plot(xpoints, ypoints,
marker='o')
plt.show()
```



5. Create an x-axis and y-axis label by accessing the library with a dot, followed by `xlabel()`, `ylabel()`, and `title()`

```
plt.xlabel("Time (year)")
```

```
plt.ylabel("CO2 (ppm)")
```

```
plt.title("CO2 atmospheric concentrations at the Mauna Loa Observatory")
```

Example:

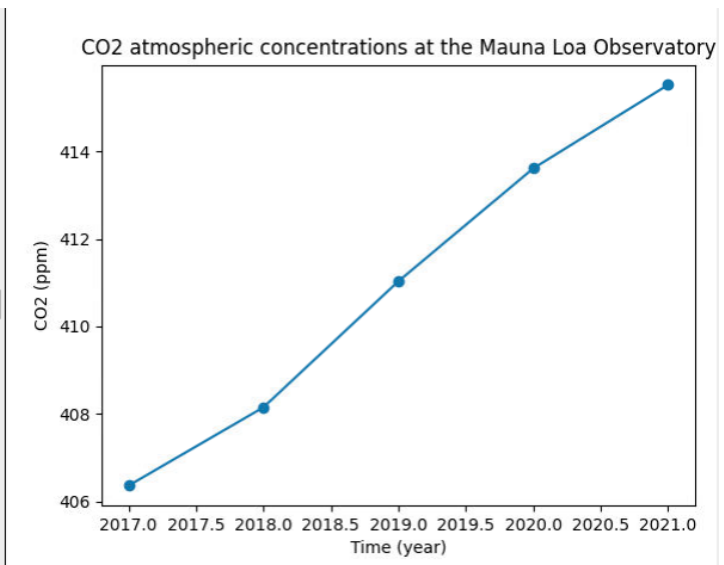
```
import numpy as np
import matplotlib.pyplot as plt

xpoints = np.array([2017, 2018,
2019, 2020, 2021])
ypoints = np.array([406.36, 408.15,
411.03, 413.61, 415.52])

# plotting with both points and a
line
plt.plot(xpoints, ypoints,
marker='o')

## Creating labels for x-axis and
y-axis
plt.xlabel("Time (year)")
plt.ylabel("CO2 (ppm)")
plt.title("CO2 atmospheric
concentrations at the Mauna Loa
Observatory")

plt.show()
```



*Don't forget to place `plt.show()` at the very end to show the graph with all the new changes.

✓ **Task 1:** Create a graph for the temperature and albedo data at the beginning of the chapter. Make the line blue and dashed, and include the points as well. Label the axes and include a title. For more information on how to customize the data visualization using matplotlib, visit this [website](#).

✓ **Task 2:** Create a function that calculates the slope using the first and last points and prints a statement that states the calculated slope. If you want to learn how to embed variables into a string, visit [this website](#) to learn about f-String literals.

