Chapter 7 - NumPy Arrays - 1D

Goals

- Create, use and modify NumPy arrays
- Understand the similarities and differences between NumPy arrays and Python lists
- Know when to use NumPy arrays and when to use Lists
- Become proficient using NumPy arrays, functions and methods to perform mathematical calculations

7.0 Similarities and differences between NumPy Arrays and Python Lists

Similarities:

- Both can hold ordered sequences of elements
- Both are indexable and sliceable: `x[0]`, `x[1:5]`, etc.
- Both can be iterated over with loops

Differences:

- NumPy arrays can only hold one data type (i.e. floats or ints), whereas Lists can hold mixed data types (floats, ints, strings, etc.) in the same list. This fact means NumPy arrays use less computer memory.
- NumPy arrays cannot be extended after they are created. Their length is fixed.
- They have built-in vectorized operations that allow calculations on all elements without for loops (see below)
- NumPy arrays contain lots of built-in mathematical functions written in C (and are thus fast) for linear algebra, statistics, reshaping, etc.
- NumPy arrays can be 2D, 3D, 4D, etc. for higher-dimensional calculations

Bottom Line: Use NumPy arrays instead of Python Lists when you need to perform mathematical calculations, need multidimensional support or want access to NumPy merthods and functionality.

NumPy arrays can be used to store any sequence of numbers. Here are some examples of the many ways they might be used:

- To store vector components in 2D, 3D, 4D, etc. space. Each element of the array represents a vector component in sp
- To store multiple measurements of the same variable

- To store measurements made over time

7.1 - Use the NumPy `.array()` method to create a NumPy array from a list

- The NumPy library has lots of methods for creating, modifying and analyzing arrays.
- Before using NumPy arrays, we need to import the NumPy library and (as usual) call it by its nickname, `np`. The import command only needs to be run once per Jypyter notebook.
- We'll then pass a list of numbers to the `.array()` method and assign it to a variable:

```
import numpy as np                                  # import the NumPy library

my_array = np.array([10, 20, 30, 40, 50])           # create a NumPy array from
print("my_array has data type = ",type(my_array))
print("first element of my_array (index = 0) is ",my_array[0])
```

```
my_array has data type =  <class 'numpy.ndarray'>
first element of my_array (index = 0) is  10
```

- Like Python lists, NumPy arrays start at index 0
- Notice that NumPy arrays are a new data type called `numpy.ndarray`, which stands for a N-dimensional array (NumPy arrays can have any number of dimensions).
- Also notice that we use square brackets (same used for Python lists) to access elements of a NumPy Array

7.2 Use `.copy()` to copy NumPy arrays

- Like Python Lists, the assignment operator only creates an alias to a NumPy array.
- To make a true copy (also called a "deep copy"), you need to use the `.copy()` method:

```
#####  Shallow Copy Example
A = np.array([1, 1, 1, 1])
B = A                          # B is an alias of A
B[-1] = 2                      # modifying B affects A
print()
print("B is alias of A:")
print("A = ",A)                # A will show changes applied to B
print("B = ",B)

#####  Deep Copy Example
A = np.array([1, 1, 1, 1])
B = A.copy()                   # B and A are independent arrays (a deep copy)
```

```
B[-1] = 2                    # modifying B does not affect A
print()
print("Deep copy:")
print("A = ",A)              # A will not show changes applied to B
print("B = ",B)
```

```
B is alias of A:
A =  [1 1 1 2]
B =  [1 1 1 2]

Deep copy:
A =  [1 1 1 1]
B =  [1 1 1 2]
```

7.3 - NumPy array elements can be accessed and modified using square brackets and slicing

```
# display the first n elements of my_array
my_array = np.array([10, 20, 30, 40, 50])
n = 3
print("first",n,"elements of my_array are",my_array[:n])
```

```
first 3 elements of my_array are [10 20 30]
```

- NumPy arrays are mutable, so you can modify their elements:

```
# reassign the first element of my_array
my_array[0] = 100
print("my_array is now",my_array)
```

```
my_array is now [100  20  30  40  50]
```

✅ Skill Check 1

Given the following list, do the following:

- Convert it to a numpy array and name it `x`
- Create a deep copy of `x` and name it `y`
- Set all the elements of `y` to 0
- Print the first 3 elements of both `x` and `y`, labeling the arrays accordingly.

```

```

```
my_list = [3,6,9,12]

# Enter your code here
```

7.4 Basic operations on NumPy arrays will be applied to all elements of the array

- The following examples are called "**element-wise**" operations since a given operator acts on each array, element by element.
- They are also sometimes called "**vectorized**" operations, since you can think of the array elements as components of a vector and operations on them apply to all their vector components.
- Let's look at a simple example of squaring each element of a NumPy array:

```
A = np.array([3, 6, 9, 12])   # define a NumPy array from a list
A2 = A**2                     # square each element
print("square of A = ",A2)    # display the result
```

```
square of A =  [  9  36  81 144]
```

- We see that the `A2` array contains the square of each element of `A`.
- The above code is equiavlant to squaring each element in the array in a for loop. However, for loops can be much slower than the vectorized operations (especially when applied to large arrays) because vectorized operations are written in C, instead of Python.

🔆 Example: Squaring elements in an array

- The following two examples give the same result, but the first, vectorized, method is more computationally efficient:

```
##### The following two code snippets perform the same operation:

# define a NumPy array from a list
A = np.array([1, 2, 3, 4])

#####  vectorized method:  square each term in A with single command. FASTER
A2 = A**2
print("method 1:  A^2 = ",A2)

#####  Python loop:  each element squared "by hand". SLOWER
for i in range(len(A)):
    A[i] = A[i]**2
```

```
print("method 2:  A^2 = ",A)
```

```
method 1:  A^2 =  [ 1  4  9 16]
method 2:  A^2 =  [ 1  4  9 16]
```

🔆 Example: Multiplying NumPy arrays

If `x` and `y` are two equal-length NumPy arrays, we can calculate the element-wise product by simply multiplying the arrays:

```
x = np.array([1,2,4,8])  # NumPy array (length 4)
y = np.array([3,0,1,1])  # NumPy array (length 4)

z = x*y                  # element-wise product of x and y
print("x*y = ",z)        # result is also length 4
```

```
x*y =  [3 0 4 8]
```

- This operation multiplies the first element of each array and stores it in the first element of `z`, and so on.
- Examine the x,y and z arrays to make sure the result makes sense.
- We can also multiply an array by a float (or integer) variable like this:

```
x = np.array([1,2,4,8])  # NumPy array (length 4)
a = 10.                  # variable containing a floating-point number

z = a*x                  # multiply each element of x by a
print("x*y = ",z)        # result is also length 4
```

```
x*y =  [10. 20. 40. 80.]
```

Note: Multiplying two arrays that have different lengths will produce an error:

```
x = np.array([1, 2, 0, 2])   # array containing 4 elements
y = np.array([1, 3])         # array containing 2 elements

z = x*y                      # attempting to multiply x*y gives an error!!!
```

✅ Skill Check 2

Without running the following code, try to predict what the print statement will display. Enter your answer in a Text Cell below. If you want, you can copy and paste the code in a Code Cell to see if your calculations are correct:

```
x = np.array([1, 2, 1, 2])
y = np.array([1, 0, 2, 0])
a1 = 2
a2 = 4
z = a1*x**2 + a2*x*y
print("The array z = ",z)
```

Enter your response here

7.5 Common NumPy Math Functions

The following list of mathematical NumPy functions were given in Chapter 3. We repeat the list here because they are vectorized and operation on NumPy arrays in additon to scalar numbers.

```
x = [.1, .2, .3, .4, .5]   # define a NumPy array for x

# constants
np.pi          # pi
np.e           # e
np.inf         # infinity
np.nan         # not a number

# logarithmic and exponential functions
np.sqrt(x)     # square root(x)
np.exp(x)      # e^x
np.log(x)      # ln(x)
np.log10(x)    # log base 10(x)
np.log2(x)     # log base 2(x)

# trigonometric functions
np.sin(x)      # sin(x)
np.cos(x)      # cos(x)
np.tan(x)      # tan(x)

# degree-radian conversions
np.deg2rad(x)  # converts degrees to radians
np.rad2deg(x)  # converts radians to degrees

# inverse trigonometric functions
np.asin(x)     # np.arcsin(x) also works for the arcsin() of a value
np.acos(x)     # np.arccos(x) also works
np.atan(x)     # np.arctan(x) also works
np.atan2(y,x)  # passing the x and y vector components lets the atan2()
               # function return an angle in the correct quadrant
```

```
# hyperbolic functions
np.sinh(x)     # hyperbolic sin
np.cosh(x)     # hyperbolic cos
np.tanh(x)     # hyperbolic tan
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-184441502.py in <cell line: 0>()
     27 np.acos(x)     # np.arccos(x) also works
     28 np.atan(x)     # np.arctan(x) also works
---> 29 np.atan2(y,x)  # passing the x and y vector components lets the
atan2()
     30                # function return an angle in the correct quadrant
     31

ValueError: operands could not be broadcast together with shapes (4,) (5,)
```

Next steps: Explain error

🔆 Example: Applying the sin() function to a NumPy array

- Most NumPy functions will operate on the elements of NumPy arrays.
- For example, if we have some angles stored in an array, we can take the `sin()` of each angle using a single statement:

```
theta_deg = np.array([0, 10, 20, 30, 40, 50])   # angles in degrees
theta_rad = np.deg2rad(theta_deg)               # convert the anges to radians
y_val = np.sin(theta_rad)                       # take the sine of each angle

for th,y in zip(theta_deg,y_val):               # print results
    print(f"sin({th:2.0f}) = {y:6.4f}")
```

✅ Skill Check 3

Vector components.

- The following Code Cell has two NumPy arrays `vx` and `vy` that store the x and y components of 5 vectors.
- Calculate the magnitude and direction (measured in degrees from the +x axis) for each of the 5 vectors.
- Use "vectorized" operators in your calculation. In other words, don't use a for loop
- print and label your results

```
vx = np.array([ 5.1, 4.0,  2.9, 0.1, 3.2])   # x measurements (meters)
vy = np.array([-4.5, 1.6, -3.7, 4.2, 4.3])   # y measurements (meters)

# Enter your code here
```

7.4 NumPy Statistics

NumPy arrays have both methods and attributes defined:

- Methods are functions that are specific to a particular object (in this case NumPy arrays).
- They are accessed by adding ".method_name()" at the end of the NumPy array name. The parentheses are required
- Attributes are properties of the array. They are accessed by adding ".attribute_name" at the end of the NumPy array name. they do not take parentheses.
- The following is a list of some of the more useful methods and attributes for numerical computation and physics:

```
A = np.array([1,9,2,8,3,7,4,6,5])

##### Attributes do not take parentheses
A.size        # number of elements
A.dtype       # data type of the array

##### Methods require parentheses
A.sum()       # sum of the elements
A.prod()      # product of the elements
A.mean()      # mean of the elements
A.std()       # standard deviation
A.var()       # variance
A.min()       # minimum value
A.max()       # maximum value
A.cumsum()    # cumulative sum (result will have same length as A)
A.cumprod()   # cumulative product (result will have same length as A)
```

🔆 Example: Array stats

- Here's an example of the stats in action.

```
A = np.array([1,9,2,8,3,7,4,6,5])

N = A.size
print(f"A contains {N} elements")

A_mean = A.mean()
A_std  = A.std()
```

```
print(f"The mean and standard deviation of A are {A_mean:.2f} ± {A_std:.2f}")

A_min = A.min()
A_max = A.max()
print(f"The min and max values are {A_min:.2f} and {A_max:.2f}")
```

✅ Skill Check 4

The following Code Cell contains a NumPy array with multiple temperature readings of a liquid nitrogen cryostat (in K). Calculate and display the following (with labels):

- number of data points
- mean and standard deviation
- minimum and maximum temperature
- peak-to-peak value of the temperature fluctuations (i.e. max-min)
- the number of data points that lie outside the 1 sigma limit. In other words, temperatures > mean + 1 standard deviation

```
T = np.array([ 77.1, 77.2, 77.0, 77.8, 77.1, 77.0, 77.9, 77.1])   # x measurem

# Enter your code here
```

7.4 Creating NumPy arrays

7.4.1 Use np.linspace() to generate N evenly spaced numbers

use `.linspace()` when you know the total **number** elements

- `linspace` stands for "linearly spaced"
- The `np.linspace(start, stop, N)` method creates an array with `N` elements equally spaced from `start` to `stop`, **inclusive** (i.e. the stop value will be the last element of the array).
- The following example creates an array filled with 10 numbers starting at 100 and going to 1000:

```
my_array = np.linspace(100,1000,10)
print("my_array is",my_array)
```

- Notice that 1000 is included in the list as the last array element.

7.4.2 Use np.arange() to generate evenly spaced numbers with a known spacing

- Sometimes, you don't know exactly how many data points you want, you just know their spacing. In this situation use `np.arange()`
- `arange` stands for "array range". It's not a misspelling of "arrange" as in to "arrange flowers".
- The `np.arange(start, stop, interval)` method creates an array with elements equally spaced from `start` to `stop` **exclusive** (i.e. the stop value will **not** be in the array) with spacing = `interval`.
- The following example creates an array filled with evenly spaced numbers from 0 to 5 spaced by 0.5:

```
my_array = np.arange(0,5.5,0.5)
print("my_array is",my_array)
```

Notice that we had to set the `stop` value to 5.5 for the 5.0 value to be included in the list. In general the the `stop` value must be set to the desired max value + interval value, i.e. $5.0 + 0.5 = 5.5$ in this example.

🔆 Example: Falling rock

A common application of `np.linspace()` and `np.arange()` is to create an array of evenly spaced time or space values.

In this example, we calculate the distance a rock has fallen every 0.5 seconds up to a total of 3 seconds. The distance fallen is given by $d = \frac{1}{2}gt^2$, where $g = 9.8$ m/s^2.

Solution

- We use the the `np.arange()` method to create an array of times equally space by 0.5 seconds. We call this array `times`
- Note: we need to set the `stop` value in `np.arange(start,stop,step)` to 3.5 (instead of 3.0) since `np.arange()` goes up to but does not include the `stop` value
- Next define a variable `g` to store the acceleration of gravity
- We calculate the distance fallen at every time in the `times` array and store these values in a new array called `dist`.
- Finally, we print the `times` and `dist` arrays to display our results

```
times = np.arange(0,3.5,0.5)   # create equally spaced times every 0.5 seconds
g = 9.8                        # acceleration of gravity
```

```
dist = 0.5 * g * times**2      # calculate distance fallen for every time in t

for t,d in zip(times,dist):    # loop over the time and distance arrays to pri
    print(f"t = {t:3.1f} s    d = {d:5.2f} m")
```

We see that at time $t = 0$, the rock hasn't moved, which gives us some confidence that our code is working at least for this simple case. At time $t = 3$ s, the rock has fallen 44.1 meters.

✅ Skill Check 5

The international space station orbits the Earth approximately 400 km above the Earth's surface. Calculate the acceleration of gravity (g) every 100 km from the sea level up to the space station.

- The acceleration of gravity is given by $g(h) = (9.8 \text{ m/s}^2) * R^2/(R + h)^2$, where $R = 6378$ km = radius of the Earth and h = altitude above the Earth (in km).
- Use `np.arange()` to create an array to store the altitude h values from 0, 100... 400 km.
- print a nicely formatted list of the altitudes and the acceleration of gravity in two columns.

```
# Enter your code here
```

7.4.3 Use np.zeros() or np.ones() to generate a uniform array filled with zeros or ones

- Because NumPy arrays cannot be extended after they are created, it is common to preallocate memory for the array using `np.zeros()` (see below)
- The functions `np.zeros(N)` and `np.ones(N)` create an array of length N filled with zeros or ones respectively.
- To create an array filled with any other number simply multiply `np.ones(N)` by that number.

```
A = np.zeros(5)     # create an NumPy array of length 5 filled with 0's
print("A = ",A)

B = np.ones(5)      # create an NumPy array of length 5 filled with 1's
print("B = ",B)

C = 9*np.ones(5)    # create an NumPy array of length 5 filled with 9's
print("C = ",C)
```

🔆 Example

Let's create a sequence of numbers where each element equals the square of the previous element. This is equivalent to repeatedly pressing the `^2` button on your calculator.

- We'll start with the number 2 and calculate a total of 10 terms in the sequence
- We preallocate our array using `np.zeros()` before starting our `for` loop to do the calculation

```
N = 10                  # number of elements in our sequence (i.e. array)
y0 = 2                  # initial number

y = np.zeros(N)         # preallocate array to hold the sequence
y[0] = y0               # set first number in the sequence

for n in range(1,N):    # calculate next element in series based on previous el
    y[n] = y[n-1]**2

for n in range(N):      # print result
    print(f"{y[n]:.10g}")
```

- This series gets large fast!
- You might try and rerun it to include 11 terms instead of 10. If you do, you'll find that the next number in the series exceeds Python's limits and you'll get an error 😳 . The largest number standard NumPy floats can handle is around 1.8×10^{308}.

✅ Skill Check 6

Back in Chapter 5, you wrote some code to produce the Fibonacci sequence using lists. In this Skill Check, you will rewrite your code using a NumPy array.

- Because NumPy arrays cannot be extended using the `append()` command, you'll need to preallocate an array to store the sequence.
- Write some code to produce the first 15 terms of the Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8...

- The Fibonacci sequence is defined as $x_n = x_{n-1} + x_{n-2}$, i.e. each term is the sum of the previous two terms.
- Initialize your array with the first two terms in the series: `[0,1]`
- Print your list when you are done

```
# Enter your code here
```

7.4.4 Create NumPy arrays filled with random numbers

- The NumPy library has its own functions to generate arrays of random numbers,
- In order to use these random number generators, an "instance" of the random number generator ogject must first be created (more on object-oriented programming in a future chapter). T do this we use the command:

`rng = np.random.default_rng()`

- Once the `rng` object is created, we can use it to produce a variety of random arrays
- Here are some examples:

```
import numpy as np

rng = np.random.default_rng()        # create random number Generator
N = 5                                # size of random array

# Create array of random integers between 0 <= z < 10
z = rng.integers(0, 10, size=N)
print("random integers:       ",z)
print()

# Create array of random integers between 0 <= z <= 10 (includes 10 endpoint)
z = rng.integers(0, 10, size=N, endpoint=True)
print("random integers (v2):  ",z)
print()

# Create array of random numbers drawn from Gaussian distribution with mean 0
z = rng.normal(size=N)
print("Gaussian distribution: ",z)
print()

# Create array of random numbers drawn from a uniform distribution with 1 <= z
z = rng.uniform(1,2, size=N)
print("Uniform distribution:  ",z)
print()

# Choose a random sampling from a list or array of items
my_list = ['red', 'green', 'blue']
z = rng.choice(my_list, size=N)
```

```
print("Uniform Sampling:      ",z)
print()

# Choose 5 random numbers from 0-9, with no duplicates
z = np.random.choice(10,size=5,replace=False)
print("No duplicates:         ",z)
```

- The `rng` object can be used to produce random number sequences drawn from many more distributions as well. See the documentation or your favorite AI tool for more info.

✅ Skill Check 7

Use the random choice number generator to draw 5 unique random cards from a 52-card card deck and display them.

- The following code uses a Python Dictionary to translate a given number (0-3) into the card's suit
- It uses a second dictionary to translate a second number (0-12) to the card's face value (2-Ace)
- We haven't talked about dictionaries yet, but the example shows you how to use them

Run the following code to see how it works. It defines numbers representing the suit and face value of a card, uses the dictionaries to translate them to actual face value and suit, and prints the result.

- Adapt this code to display 5 unique cards drawn from a deck
- Use the following trick: first generate a single number between 1-52, and then use that number to assign the card's face value and suit.
- Hint: you might consider using the `%` and `//` operators.
- Print the 5 cards following the example code.

```
# Dictionary defining suit
# example:  0 is clubs, 1 is diamonds, etc
suit_lookup = {
    0:"♣",
    1:"♦",
    2:"❤️",
    3:"♠"
}

# dictionary defining face value
# example:  0 has face value 2, 1 has face value 3,... 11 is a king, 12 is an
card_lookup = {
    0:" 2",
```

```
    1:" 3",
    2:" 4",
    3:" 5",
    4:" 6",
    5:" 7",
    6:" 8",
    7:" 9",
    8:"10",
    9:" J",
    10:" Q",
    11:" K",
    12:" A"
}

# suit is a number between 0-3
suit = 2
# face value is a number between 0-12
face_value = 12

my_card = card_lookup[face_value] + suit_lookup[suit]

print("my card is ",my_card)
```

7.5 Logical Indexing

logical indexing (also called boolean indexing) is one of NumPy's most powerful features. It lets you select elements from an array using conditions rather than explicit indices.

7.5.1 Using a mask to select array elements

- To select array elements based on their value, specify a condition inside the square brackets like this: `A[A>50]`. This will select only those values > 50

```
# Create an array of squares from 1 to 100
A = np.linspace(1,10,10)**2
print("A = ",A)

# select only the array elements > 50
A_sample = A[A>50]

print()
print("A values > 50", A_sample)
```

🔆 Example: Use a mask with multiple arrays

- You can use a mask to select elements from multiple arrays if the arrays

```
# Create an array of squares from 1 to 10^2
A = np.linspace(1,10,10)**2
print("A = ",A)

# Create an array of cubes from 1 to 10^3
B = np.linspace(1,10,10)**3
print("B = ",B)

mask = A > 50    # list of indices where array A is > 50

print("A values where A > 50", A[mask])
print("B values where A > 50", B[mask])
```

7.5.2 Using a mask with multiple conditions

To combine multiple conditions in the mask, we must use "bit-wise" operators:

- `&` is bit-wise "and"
- `\` is bit-wise "or"

```
# select the array elements with values > 10 and < 50
A_sample = A[(A>10) & (A<50)]

print("A values > 10 and < 50:  ", A_sample)
```

7.5.3 Use logical indexing to assign values in an array

- The following example replaces all negative values in an array with zeros

```
A = np.array([1,3,-1,3,9,-2,3,5])
print("A before:  ",A)

A[A<0] = 0
print("A after:   ",A)
```

7.5.4 Use `where()` to assign elements based on a condition

The statement `np.where(condition, x, y)` returns an array choosing elements from `x` where `condition` is True and frpm `y` where `condition` is False.

- The following example replaces all positive numbers in an array with +1 and all negative numbers with -1

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

result = np.where(A > 0, 1, -1)

print("A      = ",A)
print("result = ",result)
```

✅ Skill Check 8

The following Code Cell contains an array of `x` values. Missing data points have been set to 999.

- Write a single line of code using logical indexing to replace all 999 values with 0's

```
x = np.array([1,2,3,999,5,6,7,999])

# Enter your code here
```

✅ Skill Check 9

Flipping a coin

- Use the NumPy random number generator to simulate a series of coin flips.
- Define the total number of coin flips as `N`. Set `N` to some value, say 10.
- Use the `rng.integers()` function described above to produce an array of N random integers. You could use 1 for heads and 2 for tails, or whatever you choose.
- Use logical indexing to count the number of heads and calculate the fraction compared with the total number of coin flips. Express your result as a percentage.
- Calculate the percent error compared to a perfectly "fair" coin that produces heads 50% of the time: $err = |(percent - 50)/50| \times 100$
- Print the number of flips `N`, the number of heads and the percent error on a single line.

```
# Enter your code here
```

✅ Skill Check 10

Modify your coin flip code as follows:

- Copy and paste your code in the Code Cell below.

- Place your code in a `for` loop for the following number of flips: 10, 10^2, 10^3, 10^4, 10^5, 10^6.
- When counting random events, the counting error is the expected variation due to random statistics. The random variation is expected to be around $\sqrt{N}$, the corresponding relative error is $(\sqrt{N})/N = 1/\sqrt{N}$ and the percent error is $100/\sqrt{N}$.
- Print the statistical percent counting error on the same line as the other info
- Create nicely formatted columns and label the columns (i.e. have a print statement before the for loop)
- Run your code a few times and observe how the percent errors vary. Your simulated coin flips should fluctuate around the expected percent counting error. Do they?

```
# Enter your code here
```

Key Points

- NumPy arrays are similar to Python Lists, but they are faster and have lots of built-in mathematical functions
- The `np.array()` function converts a list to a NumPy array
- Like Python lists, elements of NumPy arrays are accessed and sliced using square brackets
- Basic operations on NumPy arrays are applied to all elements of the array, making them ideal for processing large datasets
- NumPy arrays can be used like normal (scalar) variables in equations
- A variety of functions are available to create evenly spaced data points, uniform distributions and random number sequences
- Logical indexing can be used to select array elements based on their value. It can also be used to reassign the values of elements.

This tutorial is an adaptation of "Python for Physicists" © Software Carpentry