**Introduction to Python #4 Numpy**

**Your First NumPy Array**

In this chapter, we're going to dive into the world of baseball. Along the way, you'll get comfortable with the basics of numpy, a powerful package to do data science.

A list baseball has already been defined in the Python script, representing the height of some baseball players in centimeters. Can you add some code here and there to create a numpy array from it?

**Instructions**

**100 XP**

* Import the numpy package as np, so that you can refer to numpy with np.
* Use **[np.array()](http://docs.scipy.org/doc/numpy-1.10.0/glossary.html" \l "term-array" \t "_blank)** to create a numpy array from baseball. Name this array np\_baseball.
* Print out the type of np\_baseball to check that you got it right.

Script.py  
01 # Create list baseball

02 baseball = [180, 215, 210, 210, 188, 176, 209, 200]

03

04 # Import the numpy package as np

05 import numpy as np

06

07 # Create a numpy array from baseball: np\_baseball

08 np\_baseball = np.array(baseball)

09

10 # Print out type of np\_baseball

11 print(type(np\_baseball))

IPython Shell  
In [1]: # Create list baseball

baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Import the numpy package as np

import numpy as np

# Create a numpy array from baseball: np\_baseball

np\_baseball = np.array(baseball)

# Print out type of np\_baseball

print(type(np\_baseball))

<class 'numpy.ndarray'>

In [2]:

**Baseball players' height**

You are a huge baseball fan. You decide to call the MLB (Major League Baseball) and ask around for some more statistics on the height of the main players. They pass along data on more than a thousand players, which is stored as a regular Python list: height\_in. The height is expressed in inches. Can you make a numpy array out of it and convert the units to meters?

height\_in is already available and the numpy package is loaded, so you can start straight away (Source: [**stat.ucla.edu**](http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_MLB_HeightsWeights)).

**Instructions**

**100 XP**

* Create a numpy array from height\_in. Name this new array np\_height\_in.
* Print np\_height\_in.
* Multiply np\_height\_in with 0.0254 to convert all height measurements from inches to meters. Store the new values in a new array, np\_height\_m.
* Print out np\_height\_m and check if the output makes sense.

Script.py  
# height is available as a regular list

# Import numpy

import numpy as np

# Create a numpy array from height\_in: np\_height\_in

np\_height\_in = np.array(height\_in)

# Print out np\_height\_in

print(np\_height\_in)

# Convert np\_height\_in to m: np\_height\_m

np\_height\_m = np\_height\_in \* 0.0254

# Print np\_height\_m

print(np\_height\_m)

IPython Shell  
In [1]: # height is available as a regular list

# Import numpy

import numpy as np

# Create a numpy array from height\_in: np\_height\_in

np\_height\_in = np.array(height\_in)

# Print out np\_height\_in

print(np\_height\_in)

# Convert np\_height\_in to m: np\_height\_m

np\_height\_m = np\_height\_in \* 0.0254

# Print np\_height\_m

print(np\_height\_m)

[74 74 72 ..., 75 75 73]

[ 1.8796 1.8796 1.8288 ..., 1.905 1.905 1.8542]

In [2]:

**Baseball player's BMI**

The MLB also offers to let you analyze their weight data. Again, both are available as regular Python lists: height\_in and weight. height\_in is in inches and weight\_lb is in pounds.

It's now possible to calculate the BMI of each baseball player. Python code to convert height\_in to a numpyarray with the correct units is already available in the workspace. Follow the instructions step by step and finish the game!

**Instructions**

**100 XP**

* Create a numpy array from the weight\_lb list with the correct units. Multiply by 0.453592 to go from pounds to kilograms. Store the resulting numpy array as np\_weight\_kg.
* Use np\_height\_m and np\_weight\_kg to calculate the BMI of each player. Use the following equation:

BMI=weight(kg)height(m)2BMI=weight(kg)height(m)2

Save the resulting numpy array as bmi.

* Print out bmi.

Script.py  
01 # height and weight are available as regular lists

02

03 # Import numpy

04 import numpy as np

05

06 # Create array from height\_in with metric units: np\_height\_m

07 np\_height\_m = np.array(height\_in) \* 0.0254

08

09 # Create array from weight\_lb with metric units: np\_weight\_kg

10 np\_weight\_kg = np.array(weight\_lb) \* 0.453592

11

12 # Calculate the BMI: bmi

13 bmi = np\_weight\_kg / np\_height\_m \*\* 2

14

15 # Print out bmi

16 print(bmi)

IPython Shell  
In [1]: # height and weight are available as regular lists

# Import numpy

import numpy as np

# Create array from height\_in with metric units: np\_height\_m

np\_height\_m = np.array(height\_in) \* 0.0254

# Create array from weight\_lb with metric units: np\_weight\_kg

np\_weight\_kg = np.array(weight\_lb) \* 0.453592

# Calculate the BMI: bmi

bmi = np\_weight\_kg / np\_height\_m \*\* 2

# Print out bmi

print(bmi)

[ 23.11037639 27.60406069 28.48080465 ..., 25.62295933 23.74810865

25.72686361]

In [2]:

**Lightweight baseball players**

To subset both regular Python lists and numpy arrays, you can use square brackets:

x = [4 , 9 , 6, 3, 1]

x[1]

import numpy as np

y = np.array(x)

y[1]

For numpy specifically, you can also use boolean numpyarrays:

high = y > 5

y[high]

The code that calculates the BMI of all baseball players is already included. Follow the instructions and reveal interesting things from the data!

**Instructions**

**100 XP**

* Create a boolean numpy array: the element of the array should be True if the corresponding baseball player's BMI is below 21. You can use the < operator for this. Name the array light.
* Print the array light.
* Print out a numpy array with the BMIs of all baseball players whose BMI is below 21. Use light inside square brackets to do a selection on the bmi array.

Script.py

01 # height and weight are available as a regular lists

02

03 # Import numpy

04 import numpy as np

05

06 # Calculate the BMI: bmi

07 np\_height\_m = np.array(height\_in) \* 0.0254

08 np\_weight\_kg = np.array(weight\_lb) \* 0.453592

09 bmi = np\_weight\_kg / np\_height\_m \*\* 2

10

11 # Create the light array

12 light = np.array(bmi < 21)

13

14 # Print out light

15 print(light)

16

17 # Print out BMIs of all baseball players whose BMI is below 21

18 print(bmi[light])

IPython Shell  
In [1]: # height and weight are available as a regular lists

# Import numpy

import numpy as np

# Calculate the BMI: bmi

np\_height\_m = np.array(height\_in) \* 0.0254

np\_weight\_kg = np.array(weight\_lb) \* 0.453592

bmi = np\_weight\_kg / np\_height\_m \*\* 2

# Create the light array

light = np.array(bmi < 21)

# Print out light

print(light)

# Print out BMIs of all baseball players whose BMI is below 21

print(bmi[light])

[False False False ..., False False False]

[ 20.54255679 20.54255679 20.69282047 20.69282047 20.34343189

20.34343189 20.69282047 20.15883472 19.4984471 20.69282047

20.9205219 ]

In [2]:

**NumPy Side Effects**

As Filip explained before, numpy is great for doing vector arithmetic. If you compare its functionality with regular Python lists, however, some things have changed.

First of all, numpy arrays cannot contain elements with different types. If you try to build such a list, some of the elements' types are changed to end up with a homogeneous list. This is known as *type coercion*.

Second, the typical arithmetic operators, such as +, -, \* and / have a different meaning for regular Python lists and numpy arrays.

Have a look at this line of code:

np.array([True, 1, 2]) + np.array([3, 4, False])

Can you tell which code chunk builds the exact same Python object? The numpy package is already imported as np, so you can start experimenting in the IPython Shell straight away!

**Instructions**

**50 XP**

**Possible Answers**

* 

np.array([True, 1, 2, 3, 4, False])

* 

np.array([4, 3, 0]) + np.array([0, 2, 2])

* 

np.array([1, 1, 2]) + np.array([3, 4, -1])

* 

np.array([0, 1, 2, 3, 4, 5])

IPython Shell  
In [1]: np.array([True, 1, 2, 3, 4, False])

Out[1]: array([1, 1, 2, 3, 4, 0])

In [2]: np.array([4, 3, 0]) + np.array([0, 2, 2])

Out[2]: array([4, 5, 2])

In [3]: np.array([1, 1, 2]) + np.array([3, 4, -1])

Out[3]: array([4, 5, 1])

In [4]: np.array([0, 1, 2, 3, 4, 5])

Out[4]: array([0, 1, 2, 3, 4, 5])

In [5]: np.array([True, 1, 2]) + np.array([3, 4, False])

Out[5]: array([4, 5, 2])

In [6]:

**Subsetting NumPy Arrays**

You've seen it with your own eyes: Python lists and numpy arrays sometimes behave differently. Luckily, there are still certainties in this world. For example, subsetting (using the square bracket notation on lists or arrays) works exactly the same. To see this for yourself, try the following lines of code in the IPython Shell:

x = ["a", "b", "c"]

x[1]

np\_x = np.array(x)

np\_x[1]

The script on the right already contains code that imports numpy as np, and stores both the height and weight of the MLB players as numpy arrays.

**Instructions**

**100 XP**

* Subset np\_weight\_lb by printing out the element at index 50.
* Print out a sub-array of np\_height\_in that contains the elements at index 100 up to **and including** index 110

Script.py  
01 # height and weight are available as a regular lists

02

03 # Import numpy

04 import numpy as np

05

06 # Store weight and height lists as numpy arrays

07 np\_weight\_lb = np.array(weight\_lb)

08 np\_height\_in = np.array(height\_in)

09

10 # Print out the weight at index 50

11 print(np\_weight\_lb[50])

12

13 # Print out sub-array of np\_height\_in: index 100 up to and including index 110

14 print(np\_height\_in[100:111])

IPython Shell  
In [1]: # height and weight are available as a regular lists

# Import numpy

import numpy as np

# Store weight and height lists as numpy arrays

np\_weight\_lb = np.array(weight\_lb)

np\_height\_in = np.array(height\_in)

# Print out the weight at index 50

print(np\_weight\_lb[50])

# Print out sub-array of np\_height\_in: index 100 up to and including index 110

print(np\_height\_in[100:111])

200

[73 74 72 73 69 72 73 75 75 73 72]

In [2]:

**Your First 2D NumPy Array**

Before working on the actual MLB data, let's try to create a 2D numpy array from a small list of lists.

In this exercise, baseball is a list of lists. The main list contains 4 elements. Each of these elements is a list containing the height and the weight of 4 baseball players, in this order. baseball is already coded for you in the script.

**Instructions**

**100 XP**

* Use **[np.array()](http://docs.scipy.org/doc/numpy-1.10.0/glossary.html" \l "term-array" \t "_blank)** to create a 2D numpy array from baseball. Name it np\_baseball.
* Print out the type of np\_baseball.
* Print out the shape attribute of np\_baseball. Use np\_baseball.shape.

Script.py  
01 # Create baseball, a list of lists

02 baseball = [[180, 78.4],

03 [215, 102.7],

04 [210, 98.5],

05 [188, 75.2]]

06

07 # Import numpy

08 import numpy as np

09

10 # Create a 2D numpy array from baseball: np\_baseball

11 np\_baseball = np.array(baseball)

12

13 # Print out the type of np\_baseball

14 print(type(np\_baseball))

15

16 # Print out the shape of np\_baseball

17 print(np\_baseball.shape)

IPython Shell  
In [3]: # Create baseball, a list of lists

baseball = [[180, 78.4],

[215, 102.7],

[210, 98.5],

[188, 75.2]]

# Import numpy

import numpy as np

# Create a 2D numpy array from baseball: np\_baseball

np\_baseball = np.array(baseball)

# Print out the type of np\_baseball

print(type(np\_baseball))

# Print out the shape of np\_baseball

print(np\_baseball.shape)

<class 'numpy.ndarray'>

(4, 2)

In [4]:

**Baseball data in 2D form**

You have another look at the MLB data and realize that it makes more sense to restructure all this information in a 2D numpy array. This array should have 1015 rows, corresponding to the 1015 baseball players you have information on, and 2 columns (for height and weight).

The MLB was, again, very helpful and passed you the data in a different structure, a Python list of lists. In this list of lists, each sublist represents the height and weight of a single baseball player. The name of this embedded list is baseball.

Can you store the data as a 2D array to unlock numpy's extra functionality?

**Instructions**

**100 XP**

* Use **[np.array()](http://docs.scipy.org/doc/numpy-1.10.0/glossary.html" \l "term-array" \t "_blank)** to create a 2D numpy array from baseball. Name it np\_baseball.
* Print out the shape attribute of np\_baseball.

Script.py  
01 # baseball is available as a regular list of lists

02

03 # Import numpy package

04 import numpy as np

05

06 # Create a 2D numpy array from baseball: np\_baseball

07 np\_baseball = np.array(baseball)

08

09 # Print out the shape of np\_baseball

10 print(np\_baseball.shape)

IPython Shell  
In [1]: # baseball is available as a regular list of lists

# Import numpy package

import numpy as np

# Create a 2D numpy array from baseball: np\_baseball

np\_baseball = np.array(baseball)

# Print out the shape of np\_baseball

print(np\_baseball.shape)

(1015, 2)

In [2]:

**Subsetting 2D NumPy Arrays**

If your 2D numpy array has a regular structure, i.e. each row and column has a fixed number of values, complicated ways of subsetting become very easy. Have a look at the code below where the elements "a" and "c" are extracted from a list of lists.

# regular list of lists

x = [["a", "b"], ["c", "d"]]

[x[0][0], x[1][0]]

# numpy

import numpy as np

np\_x = np.array(x)

np\_x[:,0]

For regular Python lists, this is a real pain. For 2D numpyarrays, however, it's pretty intuitive! The indexes before the comma refer to the rows, while those after the comma refer to the columns. The : is for slicing; in this example, it tells Python to include all rows.

The code that converts the pre-loaded baseball list to a 2D numpy array is already in the script. The first column contains the players' height in inches and the second column holds player weight, in pounds. Add some lines to make the correct selections. Remember that in Python, the first element is at index 0!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Print out the 50th row of np\_baseball.
* Make a new variable, np\_weight\_lb, containing the entire second column of np\_baseball.
* Select the height (first column) of the 124th baseball player in np\_baseball and print it out.

Script.py  
01 # baseball is available as a regular list of lists

02

03 # Import numpy package

04 import numpy as np

05

06 # Create np\_baseball (2 cols)

07 np\_baseball = np.array(baseball)

08

09 # Print out the 50th row of np\_baseball

10 print(np\_baseball[49,:])

11

12 # Select the entire second column of np\_baseball: np\_weight\_lb

13 np\_weight\_lb = np\_baseball[:,1]

14

15 # Print out height of 124th player

16 print(np\_baseball[123,0])

IPython Shell  
In [1]: # baseball is available as a regular list of lists

# Import numpy package

import numpy as np

# Create np\_baseball (2 cols)

np\_baseball = np.array(baseball)

# Print out the 50th row of np\_baseball

print(np\_baseball[49,:])

# Select the entire second column of np\_baseball: np\_weight\_lb

np\_weight\_lb = np\_baseball[:,1]

# Print out height of 124th player

print(np\_baseball[123,0])

[ 70 195]

75

<script.py> output:

[ 70 195]

75

In [2]:

**2D Arithmetic**

Remember how you calculated the Body Mass Index for all baseball players? numpy was able to perform all calculations element-wise (i.e. element by element). For 2D numpy arrays this isn't any different! You can combine matrices with single numbers, with vectors, and with other matrices.

Execute the code below in the IPython shell and see if you understand:

import numpy as np

np\_mat = np.array([[1, 2],

[3, 4],

[5, 6]])

np\_mat \* 2

np\_mat + np.array([10, 10])

np\_mat + np\_mat

np\_baseball is coded for you; it's again a 2D numpyarray with 3 columns representing height (in inches), weight (in pounds) and age (in years).

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* You managed to get hold of the changes in height, weight and age of all baseball players. It is available as a 2D numpy array, updated. Add np\_baseball and updated and print out the result.
* You want to convert the units of height and weight to metric (meters and kilograms respectively). As a first step, create a numpy array with three values: 0.0254, 0.453592 and 1. Name this array conversion.
* Multiply np\_baseball with conversion and print out the result.

Script.py  
# baseball is available as a regular list of lists

# updated is available as 2D numpy array

# Import numpy package

import numpy as np

# Create np\_baseball (3 cols)

np\_baseball = np.array(baseball)

# Print out addition of np\_baseball and updated

print(np\_baseball + updated)

# Create numpy array: conversion

conversion = [0.0254, 0.453592, 1]

# Print out product of np\_baseball and conversion

print(np\_baseball \* conversion)

IPython Shell  
In [1]: # baseball is available as a regular list of lists

# updated is available as 2D numpy array

# Import numpy package

import numpy as np

# Create np\_baseball (3 cols)

np\_baseball = np.array(baseball)

# Print out addition of np\_baseball and updated

print(np\_baseball + updated)

# Create numpy array: conversion

conversion = [0.0254, 0.453592, 1]

# Print out product of np\_baseball and conversion

print(np\_baseball \* conversion)

[[ 75.2303559 168.83775102 23.99 ]

[ 75.02614252 231.09732309 35.69 ]

[ 73.1544228 215.08167641 31.78 ]

...,

[ 76.09349925 209.23890778 26.19 ]

[ 75.82285669 172.21799965 32.01 ]

[ 73.99484223 203.14402711 28.92 ]]

[[ 1.8796 81.64656 22.99 ]

[ 1.8796 97.52228 34.69 ]

[ 1.8288 95.25432 30.78 ]

...,

[ 1.905 92.98636 25.19 ]

[ 1.905 86.18248 31.01 ]

[ 1.8542 88.45044 27.92 ]]

<script.py> output:

[[ 75.2303559 168.83775102 23.99 ]

[ 75.02614252 231.09732309 35.69 ]

[ 73.1544228 215.08167641 31.78 ]

...,

[ 76.09349925 209.23890778 26.19 ]

[ 75.82285669 172.21799965 32.01 ]

[ 73.99484223 203.14402711 28.92 ]]

[[ 1.8796 81.64656 22.99 ]

[ 1.8796 97.52228 34.69 ]

[ 1.8288 95.25432 30.78 ]

...,

[ 1.905 92.98636 25.19 ]

[ 1.905 86.18248 31.01 ]

[ 1.8542 88.45044 27.92 ]]

**Average versus median**

You now know how to use numpy functions to get a better feeling for your data. It basically comes down to importing numpy and then calling several simple functions on the numpy arrays:

import numpy as np

x = [1, 4, 8, 10, 12]

np.mean(x)

np.median(x)

The baseball data is available as a 2D numpy array with 3 columns (height, weight, age) and 1015 rows. The name of this numpy array is np\_baseball. After restructuring the data, however, you notice that some height values are abnormally high. Follow the instructions and discover which summary statistic is best suited if you're dealing with so-called *outliers*.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Create numpy array np\_height\_in that is equal to first column of np\_baseball.
* Print out the mean of np\_height\_in.
* Print out the median of np\_height\_in.

Script.py  
01 # np\_baseball is available

02

03 # Import numpy

04 import numpy as np

05

06 # Create np\_height\_in from np\_baseball

07 np\_height\_in = np.array(np\_baseball[:,0])

08

09 # Print out the mean of np\_height\_in

10 print(np.mean(np\_height\_in))

11

12 # Print out the median of np\_height\_in

13 print(np.median(np\_height\_in))

IPython Shell  
In [3]: # np\_baseball is available

# Import numpy

import numpy as np

# Create np\_height\_in from np\_baseball

np\_height\_in = np.array(np\_baseball[:,0])

# Print out the mean of np\_height\_in

print(np.mean(np\_height\_in))

# Print out the median of np\_height\_in

print(np.median(np\_height\_in))

1586.46108374

74.0

<script.py> output:

1586.46108374

74.0

In [4]:

**Explore the baseball data**

Because the mean and median are so far apart, you decide to complain to the MLB. They find the error and send the corrected data over to you. It's again available as a 2D Numpy array np\_baseball, with three columns.

The Python script on the right already includes code to print out informative messages with the different summary statistics. Can you finish the job?

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* The code to print out the mean height is already included. Complete the code for the median height. Replace Nonewith the correct code.
* Use **[np.std()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.std.html" \t "_blank)** on the first column of np\_baseball to calculate stddev. Replace None with the correct code.
* Do big players tend to be heavier? Use **[np.corrcoef()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.corrcoef.html" \t "_blank)**to store the correlation between the first and second column of np\_baseball in corr. Replace Nonewith the correct code.

Script.py  
01 # np\_baseball is available

02

03 # Import numpy

04 import numpy as np

05

06 # Print mean height (first column)

07 avg = np.mean(np\_baseball[:,0])

08 print("Average: " + str(avg))

09

10 # Print median height. Replace 'None'

11 med = np.median(np\_baseball[:,0])

12 print("Median: " + str(med))

13

14 # Print out the standard deviation on height. Replace 'None'

15 stddev = np.std(np\_baseball[:,0])

16 print("Standard Deviation: " + str(stddev))

17

18 # Print out correlation between first and second column. Replace 'None'

19 corr = np.corrcoef(np\_baseball[:,0] , np\_baseball[:,1])

20 print("Correlation: " + str(corr))

IPython Shell  
In [1]: # np\_baseball is available

# Import numpy

import numpy as np

# Print mean height (first column)

avg = np.mean(np\_baseball[:,0])

print("Average: " + str(avg))

# Print median height. Replace 'None'

med = np.median(np\_baseball[:,0])

print("Median: " + str(med))

# Print out the standard deviation on height. Replace 'None'

stddev = np.std(np\_baseball[:,0])

print("Standard Deviation: " + str(stddev))

# Print out correlation between first and second column. Replace 'None'

corr = np.corrcoef(np\_baseball[:,0] , np\_baseball[:,1])

print("Correlation: " + str(corr))

Average: 73.6896551724

Median: 74.0

Standard Deviation: 2.31279188105

Correlation: [[ 1. 0.53153932]

[ 0.53153932 1. ]]

<script.py> output:

Average: 73.6896551724

Median: 74.0

Standard Deviation: 2.31279188105

Correlation: [[ 1. 0.53153932]

[ 0.53153932 1. ]]

In [2]:

**Blend it all together**

In the last few exercises you've learned everything there is to know about heights and weights of baseball players. Now it's time to dive into another sport: soccer.

You've contacted FIFA for some data and they handed you two lists. The lists are the following:

positions = ['GK', 'M', 'A', 'D', ...]

heights = [191, 184, 185, 180, ...]

Each element in the lists corresponds to a player. The first list, positions, contains strings representing each player's position. The possible positions are: 'GK'(goalkeeper), 'M' (midfield), 'A' (attack) and 'D'(defense). The second list, heights, contains integers representing the height of the player in cm. The first player in the lists is a goalkeeper and is pretty tall (191 cm).

You're fairly confident that the median height of goalkeepers is higher than that of other players on the soccer field. Some of your friends don't believe you, so you are determined to show them using the data you received from FIFA and your newly acquired Python skills.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Convert heights and positions, which are regular lists, to numpy arrays. Call them np\_heights and np\_positions.
* Extract all the heights of the goalkeepers. You can use a little trick here: use np\_positions == 'GK' as an index for np\_heights. Assign the result to gk\_heights.
* Extract all the heights of all the other players. This time use np\_positions != 'GK' as an index for np\_heights. Assign the result to other\_heights.
* Print out the median height of the goalkeepers using **[np.median()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.median.html" \t "_blank)**. Replace None with the correct code.
* Do the same for the other players. Print out their median height. Replace None with the correct code.

Script.py  
01 # heights and positions are available as lists

02

03 # Import numpy

04 import numpy as np

05

06 # Convert positions and heights to numpy arrays: np\_positions, np\_heights

07 np\_positions = np.array(positions)

08 np\_heights = np.array(heights)

09

10 # Heights of the goalkeepers: gk\_heights

11 gk\_heights = np\_heights[np\_positions == 'GK']

12

13 # Heights of the other players: other\_heights

14 other\_heights = np\_heights[np\_positions != 'GK']

15

16 # Print out the median height of goalkeepers. Replace 'None'

17 print("Median height of goalkeepers: " + str(np.median(gk\_heights)))

18

19 # Print out the median height of other players. Replace 'None'

20 print("Median height of other players: " + str(np.median(other\_heights)))

IPython Shell  
In [1]: # heights and positions are available as lists

# Import numpy

import numpy as np

# Convert positions and heights to numpy arrays: np\_positions, np\_heights

np\_positions = np.array(positions)

np\_heights = np.array(heights)

# Heights of the goalkeepers: gk\_heights

gk\_heights = np\_heights[np\_positions == 'GK']

# Heights of the other players: other\_heights

other\_heights = np\_heights[np\_positions != 'GK']

# Print out the median height of goalkeepers. Replace 'None'

print("Median height of goalkeepers: " + str(np.median(gk\_heights)))

# Print out the median height of other players. Replace 'None'

print("Median height of other players: " + str(np.median(other\_heights)))

Median height of goalkeepers: 188.0

Median height of other players: 181.0

<script.py> output:

Median height of goalkeepers: 188.0

Median height of other players: 181.0

In [2]: