

# **HW 0-A Prerequisite for Beginners:**

## Introduction to Python Course – DataCamp

James Nicolas Recasens  
CSCI 7090 – Data Science & Machine Learning  
Georgia Southern University

January 21, 2026

# Chapter 1: Python Basics

**Your first Python code**

It's time to run your first Python code!

Head to the code and hit the run code button to see the output.

**Instructions** 100 XP

- Hit the run code button to see the output of `print(5 / 8)`.

[Take Hint \(-30 XP\)](#)

IPython Shell	Slides
<pre># Hit run code to see the output! print(5 / 8)  0.625</pre>	In [1]:

**Python as a calculator**

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples.

Now it's your turn to practice by writing some code yourself.

**Instructions** 100 XP

- Print the result of subtracting 5 from 5 under `# Subtraction using print()`.
- Print the result of multiplying 3 by 5 under `# Multiplication`.

[Take Hint \(-30 XP\)](#)

IPython Shell	Slides
<pre># Addition and division print(4 + 5) print(10 / 2)  # Subtraction print(5-5)  # Multiplication print(3*5)  9 5.0 0 15</pre>	In [1]:

**Variable Assignment**

In Python, a variable allows you to refer to a value with a name. To create a variable `x` with a value of `5`, you use `=`, like this example:

```
x = 5
```

You can now use the name of this variable, `x`, instead of the actual value, `5`.

Remember, `=` in Python means assignment, it doesn't test equality! Try it in the exercise by replacing `_____` with your code.

**Instructions** 100 XP

- Create a variable `savings` with the value of `100`.
- Check out this variable by typing `print(savings)`.

[Take Hint \(-30 XP\)](#)

IPython Shell	Slides
<pre># Create a variable savings savings = 100 # Print out savings print(savings)  100</pre>	In [1]:

**Calculations with variables**

You've now created a `savings` variable, so let's start saving!

Instead of calculating with the actual values, you can use variables instead.

How much money would you have saved four months from now, if you saved \$10 each month?

**Instructions** 100 XP

- Create a variable `monthly_savings`, equal to `10` and `num_months`, equal to `4`.
- Multiply `monthly_savings` by `num_months` and assign it to `new_savings`.
- Print the value of `new_savings`.

[Take Hint \(-30 XP\)](#)

IPython Shell	Slides
<pre># Create the variables monthly_savings and num_months monthly_savings = 10 num_months = 4  # Multiply monthly_savings and num_months new_savings = monthly_savings * num_months  # Print new_savings print(new_savings)  40</pre>	In [1]:

The figure consists of three screenshots from a Python learning environment:

- Left Screenshot:** Shows a code editor window titled "script.py" with the following code:
 

```
1 # Create a variable half
2 half = 0.5
3
4 # Create a variable intro
5 intro = "Hello! How are you?"
6
7 # Create a variable is_good
8 is_good = True
```

Below the code editor is a "Instructions" section with 100 XP available. It contains three tasks:
 
  - Create a new float, `half`, with the value `0.5`.
  - Create a new string, `intro`, with the value `"Hello! How are you?"`.
  - Create a new boolean, `is_good`, with the value `True`.
 There is also a "Take Hint (30 XP)" button.
- Middle Screenshot:** Shows a code editor window titled "script.py" with the following code:
 

```
1 savings = 100
2 new_savings = 40
3 total_savings = savings + new_savings
4 # Calculate total_savings using savings and new_savings
5
6 print(total_savings)
7
8 # Print the type of total_savings
9 print(type(total_savings))
```

Below the code editor is an "Instructions 1/2" section with 50 XP available. It contains two tasks:
 
  - Add `savings` and `new_savings` and assign it to `total_savings`. Use `type()` to print the resulting type of `total_savings`.
  - Calculate the sum of `intro` and `intro` and assign the result to `doubleintro`. Did you expect this?
 There is also a "Run Code" and "Submit Answer" button.
- Bottom Screenshot:** Shows a code editor window titled "script.py" with the following code:
 

```
savings = 100
new_savings = 40
total_savings = savings + new_savings
# Calculate total_savings using savings and new_savings
print(total_savings)

# Print the type of total_savings
print(type(total_savings))

140
<class 'int'>
```

Below the code editor is an "Instructions 1/2" section with 50 XP available. It contains two tasks:
 
  - Add `savings` and `new_savings` and assign it to `total_savings`. Use `type()` to print the resulting type of `total_savings`.
  - Calculate the sum of `intro` and `intro` and assign the result to `doubleintro`. Did you expect this?
 There is also a "Run Code" and "Submit Answer" button.

**Course Outline (Bottom Screenshot):**

## 1 Python Basics

An introduction to the basic concepts of Python. Learn how to use Python interactively and by using a script. Create your first variables and acquaint yourself with Python's basic data types.

▷ Hello Python!	✓ 50 XP
◁ Your first Python code	✓ 100 XP
◁ Python as a calculator	✓ 100 XP
▷ Variables and Types	✓ 50 XP
◁ Variable Assignment	✓ 100 XP
◁ Calculations with variables	✓ 100 XP
◁ Other variable types	✓ 100 XP
◁ Operations with other types	✓ 100 XP

[Hide Details ▾](#)

Figure 1: Chapter 1 Completion – XP Summary

## Chapter 2: Python Lists

**Exercise**

Create a list.

A list is a **compound data type**; you can group values together, like this:

```
a = "list"
b = "list"
my_list = ["a", "list", a, b]
```

After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are stored in separate variables in the exercise.

**Instructions** 100 XP

- Create a list, `areas`, that contains the area of the hallway (`hall`), kitchen (`kit`), living room (`liv`), bedroom (`bed`) and bathroom (`bath`), in this order. Use the predefined variables.
- Print `areas` with the `print()` function.

**Take Hint (30 XP)**

**IPython Shell** Slides

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Create list areas
areas = [
    hall, kit, liv, bed, bath
]

# Print areas
print(areas)

[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [1]:

**Exercise**

Create lists with different types

Although it's not really common, a list can also contain a mix of Python types including strings, floats, and booleans.

You're now going to add the room names to your list so you can easily see both the room name and size together.

Some of the code has been provided for you to get started. Pay attention here: “`hallway`” is a string, while `hall` is a variable that represents the float `11.25` you specified earlier.

**Instructions** 100 XP

- Finish the code that creates the `areas` list. Build the list so that the first list contains the name of each room as a string and the second list contains the corresponding float (“`hall`”, “`kitchen`”, “`bedroom`”, and “`bathroom`” at the appropriate locations).
- Print `areas` again; is the printout more informative this time?

**Take Hint (30 XP)**

**IPython Shell** Slides

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# Adapt list areas
areas = ["hallway", hall, "Kitchen", kit, "Living room", liv, "bedroom", bed,
         "bathroom", bath]
# Print areas
print(areas)

['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75, 'bathroom', 9.5]
```

In [1]:

**Exercise**

List of lists

As a data scientist, you'll often be dealing with a lot of data, and it will make sense to group some of this data.

Instead of creating a single list of strings and floats, representing the names and areas of the rooms in your house, you can create a list of lists.

Remember: “`hallway`” is a string, while `hall` is a variable that represents the float `11.25` you specified earlier.

**Instructions** 100 XP

- Finish the list of lists so that it also contains the bedroom and bathroom data. Make sure you enter these in order.
- Print out `house`; does this way of structuring your data make more sense?

**Take Hint (30 XP)**

**IPython Shell** Slides

```
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50

# House information as list of lists
house = [["hallway", hall],
          ["kitchen", kit],
          ["Living room", liv],
          ["bedroom", bed],
          ["bathroom", bath]]
```

# Print out house

```
print(house)

[[hallway, 11.25], [kitchen, 18.0], ['Living room', 20.0], ['bedroom', 10.75], ['bathroom', 9.5]]
```

In [1]:

**Exercise**

Subset and conquer

Subsetting Python lists is a piece of cake. Take the code sample below which creates a list `x` and then selects “`y`” from it. Remember that this is the second element, so it has index `1`. You can also use negative indexing:

```
x = ["a", "b", "c", "d"]
y = x[1] # same result!
```

Remember that `areas` list from before, containing both strings and floats? Its definition is already in the script. Can you add the correct code to do some Python subsetting?

**Instructions** 100 XP

- Print out the second element from the `areas` list (it has the value `11.25`).
- Subset and print out the last element of `areas`, being `9.50`. Using a negative index makes sense here!
- Select the number representing the area of the living room (`10.75`) and print it out.

**Take Hint (30 XP)**

**IPython Shell** Slides

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75,
         "bathroom", 9.50]

# Print out second element from areas
print(areas[1])

# Print out last element from areas
print(areas[-1])

# Print out the area of the living room
print(areas[3])
```

In [1]:

**Exercise**

Slicing and dicing

Selecting single values from a list is just one part of the story. It's also possible to slice your list, which means selecting multiple elements from your list. Use the following syntax:

```
my_list[start:end]
```

The `start` index will be included, while the `end` index is not. However, it's also possible not to specify these indices. If you don't specify the `start` index, Python figures out that you want to start your slice at the beginning of the list.

**Instructions** 100 XP

- Use slicing to create a list, `downstairs`, that contains the first two elements of `areas`.
- Create `upstairs`, on the last 3 elements of `areas`. This time, simply the slicing by omitting the `end` index.
- Print both `downstairs` and `upstairs` using `print()`.

**Take Hint (30 XP)**

**IPython Shell** Slides

```
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "Living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Use slicing to create downstairs
downstairs = areas[:2]

# Use slicing to create upstairs
upstairs = areas[2:]

# Print out downstairs and upstairs
print(downstairs)
print(upstairs)

[["hallway", 11.25], ['kitchen', 18.0], ['Living room', 20.0], ['bedroom', 10.75], ['bathroom', 9.50]]
```

In [1]:

**Exercise**

Subsetting lists of lists

A Python list can also contain other lists. To subset lists of lists, you can use the same technique as before: square brackets. This would look something like this for a list, `house`:

```
house[2][0]
```

**Instructions** 100 XP

- Subset the `house` list to get the float `9.5`.

**Take Hint (30 XP)**

**IPython Shell** Slides

```
house = [{"hallway": 11.25, "kitchen": 18.0, ["Living room": 20.0, "bedroom": 10.75, "bathroom": 9.50]}]

# Subset the house list
house[0][1]
```

In [1]:

The image shows four screenshots of a Python exercise interface from a learning platform. Each screenshot displays a code editor, instructions, and a command-line interface (IPython Shell or Python Shell).

- Screenshot 1:** Shows code to replace list elements. The code creates a list `areas` with room names and areas, then changes "living room" to "chill zone". Instructions ask to update the bathroom area to 10.50 and change "living room" to "chill zone".
- Screenshot 2:** Shows code to extend a list. It adds a poolhouse to the `areas` list and a garage to `areas\_1`. Instructions ask to add a garage to `areas\_1`.
- Screenshot 3:** Shows code to delete list elements. It removes the poolhouse from the `areas` list. Instructions ask to delete the string and float for the "poolhouse" from the `areas` list.
- Screenshot 4:** Shows code to copy lists. It creates a copy of `areas` called `areas\_copy`, changes an element in `areas\_copy`, and prints both lists. Instructions ask to change the second command to create an explicit copy of `areas`.

Figure 2: Chapter 2 Completion – XP Summary

## Chapter 3: Functions and Packages

The screenshot shows a Jupyter Notebook interface with the following components:

- Left Panel:** Shows the course outline and navigation buttons.
- Header:** Displays "Learn / Courses / Introduction to Python" and "Course Outline".
- Exercise Section:** Contains the title "Functions" and a note about built-in functions.
- Code Cell:** A snippet of Python code named "script.py" that creates variables `var1` and `var2`, prints their types, and converts `var2` to an integer.
- Instructions:** A list of tasks:
  - Use `print()` in combination with `type()` to print the type of `var1`.
  - Use `len()` to get the length of the list `var1`. Wrap it in a `print()` call to directly print it out.
  - Use `int()` to convert `var2` to an `integer`. Store the output as `out2`.
- Buttons:** "Run Here (80 XP)" button and a progress bar indicating 80% completion.
- Python Shell:** Shows the executed code and its output, including the creation of variables, printing their types, calculating the length of the list, and converting the integer back to a string.
- Bottom Bar:** Buttons for "Run Code" and "Submit Answer".

Learn / Courses / Introduction to Python

Exercise

## Multiple arguments

In the previous exercise, you identified optional arguments by viewing the documentation with `help(sorted)`. Now you'll apply this to change the behavior of the `sorted()` function.

Now look at the documentation of `sorted()`, by typing `help(sorted)` in the Python Shell.

You'll see that `sorted()` takes three arguments: `iterable`, `key`, and `reverse`. In this exercise, you'll only have to specify `iterable` and `reverse`, not `key`.

Two lists have been created for you.

Can you paste them together and sort them in descending order?

**Instructions**

Use `=` to merge the contents of `first` and `second` into a new list: `full`.  
Call `sorted()` on `full` and specify the `reverse` argument to be `True`. Save the sorted list as `full_sorted`.  
Finish off by printing out `full_sorted`.

[Take Hint \(+30 XP\)](#)

```
script.py
1 # Create lists first and second
2 first = [11.25, 18.0, 20.0]
3 second = [10.75, 9.50]
4
5 # Paste together first and second: full
6 full = first + second
7
8 # Sort full in descending order: full_sorted
9 full_sorted = sorted(full, reverse=True)
10
11 # Print out full_sorted
12 print(full_sorted)
```

Run Code

Submit Answer

---

IPython Shell

```
[In 1]: first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.5]

# Paste together first and second: full
full = first + second

# Sort full in descending order: full_sorted
full_sorted = sorted(full, reverse=True)

# Print out full_sorted
print(full_sorted)

[20.0, 18.0, 11.25, 10.75, 9.5]

In [1]:
```

The screenshot shows a Jupyter Notebook interface with two tabs: "Exercise" and "Introduction to Python". The "Exercise" tab is active, displaying a code cell with the following content:

```
script.py
1 # string to experiment with: place
2 place = "posthouse"
3
4 # use upper() on place
5 place.upper()
6
7 # Print out place and place.upper
8 print(place)
9 print(place.upper())
10
11 # Print out the number of o's in place
12 print(place.count('o'))
```

Below the code cell, there is a "Take Hint (+30 XP)" button. To the right of the code cell are three buttons: "Run Code", "Submit Answer", and "Close".

At the bottom of the screen, there is an "IPython Shell" tab and a "Slides" tab. The IPython Shell tab contains the same code as the code cell above, with the output:

```
place
POSTHOUSE
In [1]:
```

Learn / Courses / Introduction to Python

Exercise

### List Methods

String is one the only Python types that have methods associated with them. Lists, floats, integers and booleans are also types that come packaged with a bunch of useful methods. In this exercise, you'll be experimenting with:

- `.index()`, to get the index of the first element of a list that matches its input and
- `.count()`, to get the number of times an element appears in a list.

You'll be working on the list with the area of different parts of a house: `areas`.

Instructions (60 XP)

- Use the `.index()` method to get the index of the element in `areas` that has value `20.0`. Print out this index.
- Call `.count()` on `areas`, to find out how many times `9.50` appears in the list. Again, simply print out this number.

Take Hint (30 XP)

Daily XP 750 EN ⓘ

Light Mode

```
script.py
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0
print(areas.index(20.0))

# Print out how often 9.50 appears in areas
print(areas.count(9.50))
```

IPython Shell Slides

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0
print(areas.index(20.0))

# Print out how often 9.50 appears in areas
print(areas.count(9.50))

2
3

In [3]:
```

Run Code Submit Answer

The screenshot shows a Jupyter Notebook interface with the following content:

**Learn / Courses / Introduction to Python**

**Exercise**

**List Methods (2)**

Most list methods will change the list they're called on. Examples are:

- `.append()`, that adds an element to the list it is called on,
- `.remove()`, that removes the first element of a list that matches the input, and
- `.reverse()`, that reverses the order of the elements in the list it is called on.

You'll be working on the list with the area of different parts of the house: `areas`.

**Instructions** 90 XP

- Use `.append()` twice to add the size of the poolhouse and the garage again: `24.5` and `15.45`, respectively. Make sure to add them in this order.
- Print `areas`.
- Use the `.reverse()` method to reverse the order of the elements in `areas`.
- Print `areas` once more.

**Q Take Hint (30 XP)**

**script.py**

```
# Create list areas
areas = [11.25, 18.0, 28.0, 10.75, 9.50]
# Append twice to add poolhouse and garage size
areas.append(24.5)
areas.append(15.45)
# Print out areas
print(areas)
# Reverse the orders of the elements in areas
areas.reverse()
# Print out areas
print(areas)
```

**Python Shell**

```
# Create list areas
areas = [11.25, 18.0, 28.0, 10.75, 9.50]

# Append twice to add poolhouse and garage size
areas.append(24.5)
areas.append(15.45)

# Print out areas
print(areas)

# Reverse the orders of the elements in areas
areas.reverse()

# Print out areas
print(areas)

[11.25, 18.0, 28.0, 10.75, 9.5, 24.5, 15.45]
[15.45, 24.5, 9.5, 10.75, 28.0, 18.0, 11.25]
```

18 [1]:

**Daily XP 1750** EN

**Light Mode**

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Learn / Courses / Introduction to Python
- Exercise Section:** Import package
- Text:** Let's say you wanted to calculate the circumference and area of a circle. Here's what those formulas look like.
- Equation:**  $C = 2\pi r$   
 $A = \pi r^2$
- Note:** Rather than typing the number for  $\pi$ , you can use the `math` package that contains the number
- Text:** For reference,  $\approx$  is the symbol for exponentiation. For example  $3\approx4$  is 3 to the power of 4 and will give 81.
- Section:** Instructions (30 XP)
- List:**
  - Import the `math` package.
  - Use `math.pi` to calculate the circumference of the circle and store it in `C`.
  - Use `math.pi` to calculate the area of the circle and store it in `A`.
- Buttons:** Take Hint (30 XP), Run Code, Submit Answer

**Code Cell:**

```
script.pyt
1 # Import the math package
2 import math
3 # Calculate C
4 C = 2 * math.pi * 0.43
5
6 # Calculate A
7 A = math.pi * 0.43 ** 2
8
9 print("Circumference: " + str(C))
10 print("Area: " + str(A))
```

**Python Shell:**

```
# Python Shell Slides
# Import the math package
import math
# calculate C
C = 2 * math.pi * 0.43
# calculate A
A = math.pi * 0.43 ** 2

print("Circumference: " + str(C))
print("Area: " + str(A))

Circumference: 2.70794902987222
Area: 0.59888040148727
```

In [1]:

The screenshot shows a learning environment with the following components:

- Code Editor:** Displays a Python script named `script.py` containing code to calculate the circumference and area of a circle. It includes imports from the `math` module, calculations for circumference (`c`) and area (`A`), and print statements for both.
- IPython Shell:** Shows the execution of the script. The output includes the calculated circumference (2.78549200722) and area (0.588804816467527).
- XP Summary Table:** A table showing completion status for various topics in Chapter 3. Topics include Functions, Familiar functions, Help!, Multiple arguments, Methods, String Methods, List Methods, List Methods (2), Packages, Import package, Selective import, and Different ways of importing. Most topics have been completed with 50 XP, except for Familiar functions (100 XP) and Import package (100 XP).

Topic	XP
Functions	✓ 50 XP
Familiar functions	✓ 100 XP
Help!	✓ 50 XP
Multiple arguments	✓ 100 XP
Methods	✓ 50 XP
String Methods	✓ 100 XP
List Methods	✓ 100 XP
List Methods (2)	✓ 100 XP
Packages	✓ 50 XP
Import package	✓ 100 XP
Selective import	✓ 100 XP
Different ways of importing	✓ 50 XP

Figure 3: Chapter 3 Completion – XP Summary

# Chapter 4: NumPy

**Exercise**

**Your First NumPy Array**

You'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 to create a `numpy` array from it?

**Instructions**

- Import the `numpy` package as `np`, so that you can refer to `numpy` with `np`.
- Use `np.array()` 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.

**Take Hint (-30 XP)**

**IPython Shell**

```
# Import the numpy package as np
import numpy as np

baseball = [180, 215, 210, 210, 188, 176, 209, 208]

# 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 [1]:
```

**Submit**

**Exercise**

**Baseball player's height**

You are a huge baseball fan! You decide to call the MLB Major League Baseball API to look 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 values to meters?

`height_in` is already available and the `numpy` package is loaded, so you can start straight away! (Source: statdducks).

**Instructions**

- 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.

**Take Hint (-30 XP)**

**IPython Shell**

```
# 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)

[1.75 1.72 ... 1.75 1.73]
[1.8796 1.8796 1.8288 ... 1.905 1.905 1.8542]

In [1]:
```

**Submit**

**Exercise**

**Subsetting NumPy Arrays**

Subsetting (using the square bracket notation on lists or arrays) works exactly the same with both lists and arrays.

This exercise already has two lists, `height_in` and `weight_lb`, loaded in the background for you. These contain the height and weight of the MLB players as regular lists. It also has two `numpy` arrays, `np_weight_lb` and `np_height_in`, prepared for you.

**Instructions**

- 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.

**Take Hint (-30 XP)**

**IPython Shell**

```
import numpy as np

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

In [1]:
```

**Submit**

**Exercise**

**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. `savemat` is already coded for you in the script.

**Instructions**

- Use `np.array()` 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`.

**Take Hint (-30 XP)**

**IPython Shell**

```
import numpy as np

baseball = [[180, 78.4], [215, 102.7], [210, 98.0], [188, 75.3]]

# 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 [1]:
```

**Submit**

**Exercise**

**Baseball data in 2D form**

You realize that it makes more sense to restructure all this information in a 2D numpy array.

You have a Python list of lists, in the list of lists, each sublist represents the height and weight of a single baseball player. The name of this list is `baseball`. It has been loaded for you directly (although you can see it).

Store the data as a 2D array to unlock `numpy`'s extra functionality.

**Instructions**

- Use `np.array()` to create a 2D `numpy` array from `baseball`. Name it `np_baseball`.
- Print out the `shape` attribute of `np_baseball`.

**Take Hint (-30 XP)**

**IPython Shell**

```
# Import numpy as np
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)

(100, 2)

In [1]:
```

**Submit**

**Exercise**

**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 "`<: ->`" are extracted from a list of lists.

**Instructions**

- 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 24th baseball player in `np_baseball`, and print it out.

**Take Hint (-30 XP)**

**IPython Shell**

```
import numpy as np

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 24th player
print(np_baseball[23,0])

120
[79 199]

In [1]:
```

**Submit**

The figure displays four panels of a DataCamp Python course interface, showing the completion of Chapter 4. Each panel includes a code editor, a terminal-like IPython Shell, and a Slides section.

- Panel 1 (Top Left):** Exercises on 2D Arithmetic. The code calculates the product of np\_baseball and conversion.
- Panel 2 (Top Right):** Exercises on Average versus median. The code prints the mean and median of np\_height\_in.
- Panel 3 (Bottom Left):** Exercises on Exploring the baseball data. The code prints various statistics (mean, median, standard deviation, correlation) for np\_baseball.
- Panel 4 (Bottom Right):** A summary of completed XP tasks for Chapter 4, including NumPy, Your First NumPy Array, and various sub-sections of NumPy.

```

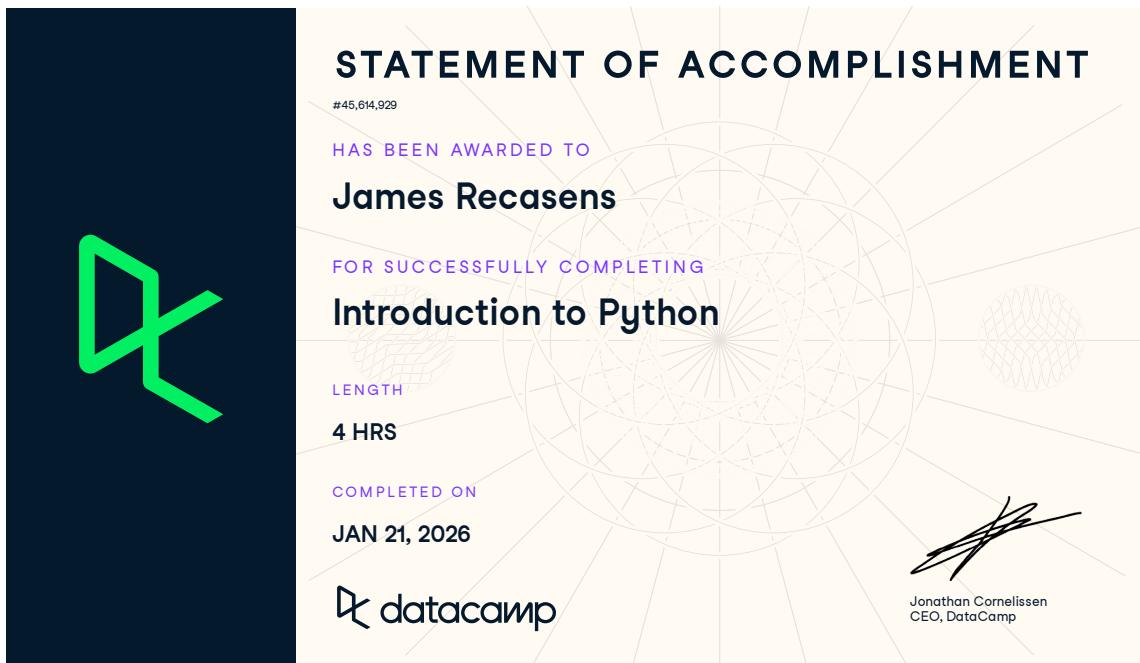
script.py
1 import numpy as np
2 np_baseball = np.array(baseball)
3
4 # Print out addition of np_baseball and updated
5 print(np_baseball + updated)
6
7 # Create numpy array: conversion
8 conversion = [0.0254, 0.45392, 1]
9
10 # Print out product of np_baseball and conversion
11 print(np_baseball*conversion)
12
13 print(np.sum(np_baseball*conversion))
14
15 print(np.mean(np_baseball[:,0]))
16 print("Average: " + str(avgr))
17
18 print("Median: " + str(med))
19
20 print("Standard Deviation: " + str(stdev))
21
22 print("Correlation: " + str(corr))
23
24 print("Correlation: " + str(corr))
25
26 print("Correlation: " + str(corr))

In [1]:
```

XP Task	XP Points
NumPy	50 XP
Your First NumPy Array	100 XP
Baseball players' height	100 XP
NumPy Side Effects	50 XP
Subsetting NumPy Arrays	100 XP
2D NumPy Arrays	50 XP
Your First 2D NumPy Array	100 XP
Baseball data in 2D form	100 XP
Subsetting 2D NumPy Arrays	100 XP
2D Arithmetic	100 XP
NumPy: Basic Statistics	50 XP
Average versus median	100 XP
Explore the baseball data	100 XP

Figure 4: Chapter 4 Completion – XP Summary

## Certificate of Completion



## Reflection

The introduction to Python course from DataCamp was a nice refresher on using Python as a tool for data science. I have used Python in previous courses, but my main language has always been Java, so it was nice to gain fluency in using Python because the tools available are very intuitive and simple. The ease by which arrays can be scaled, added together, etc. seems way easier than any data cleaning I have done in MATLAB or R. Overall, this brief course was a good experience because I am walking away from it feeling more familiar with Python, and most importantly, perhaps, I am not frustrated or discouraged by the exercises so far.

---

*Disclosure: This document was formatted with the assistance of an AI tool (Claude). All coursework, code submissions, and written reflections are entirely my own original work. The AI was used solely for document compilation and formatting purposes.*