NumPy

INTRODUCTION TO PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
  - Mathematical operations over collections
  - Speed

Illustration

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
height
```

```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight
```

```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

```
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Solution: NumPy

- Numeric Python
- Alternative to Python List: NumPy Array
- Calculations over entire arrays

- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`

NumPy

```
import       as
np_height = np.array(height)
np_height
```

import numpy as np

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
np_weight = np.array(weight)
np_weight
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
bmi = np_weight / np_height ** 2
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

Comparison

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and
'int'
```

```
np_height = np.array(height)
np_weight = np.array(weight)
np_weight / np_height ** 2
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 ,
21.44127836])
```

NumPy: remarks

```
np.array([1.0, "is", True])
```

```
array(['1.0', 'is', 'True'], dtype='<U32')
```

- NumPy arrays: contain only one type

NumPy: remarks

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

```
python_list + python_list
```

```
[1, 2, 3, 1, 2, 3]
```

```
numpy_array + numpy_array
```

```
array([2, 4, 6])
```

- Different types: different behavior!

NumPy Subsetting

```
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

```
bmi[1]
```

```
20.975
```

```
bmi > 23
```

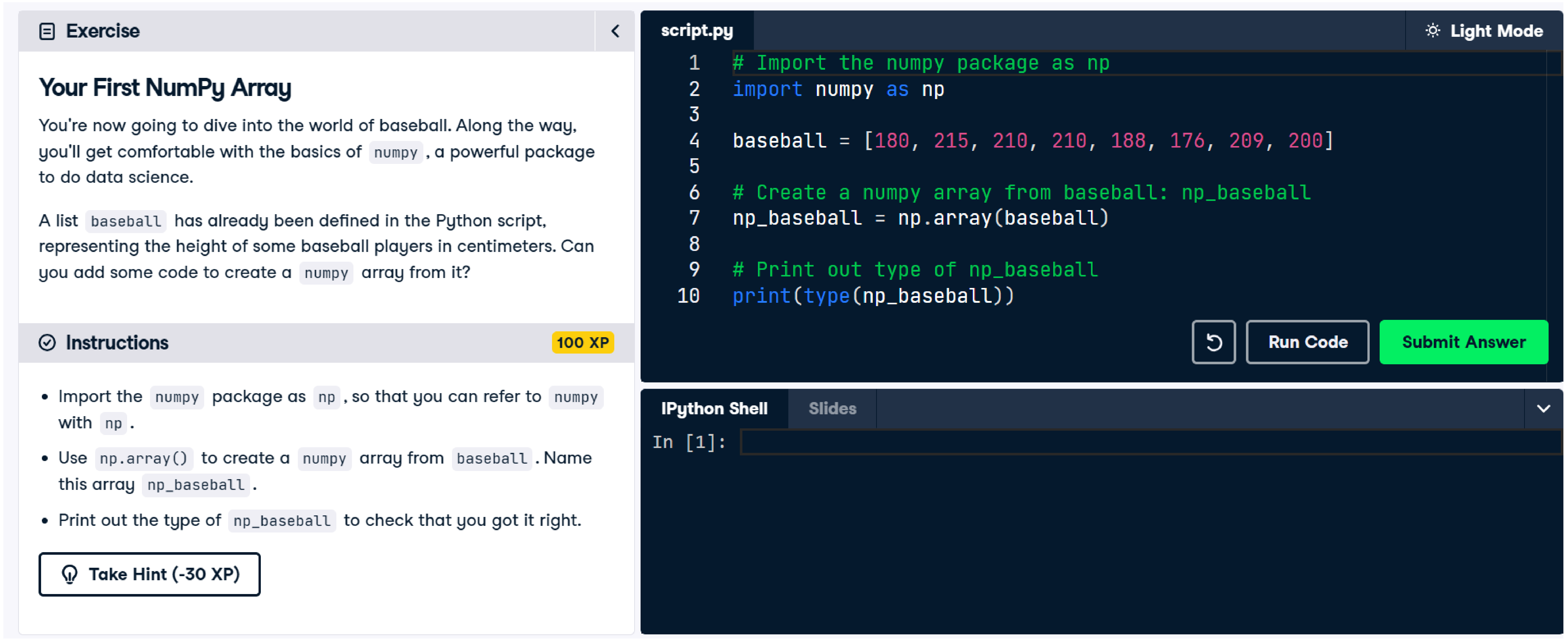
```
array([False, False, False,  True, False])
```

```
bmi[bmi > 23]
```

```
array([24.7473475])
```

Let's practice!

INTRODUCTION TO PYTHON



Your First NumPy Array

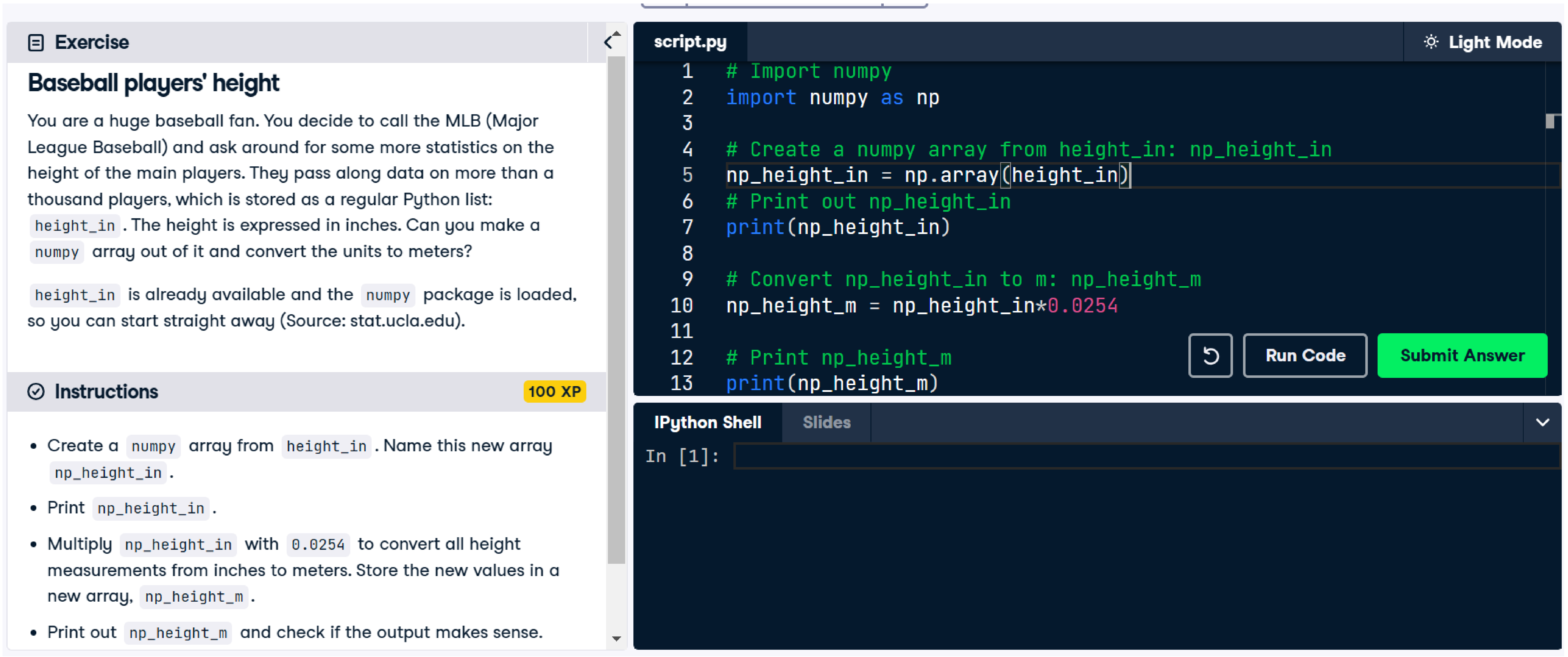
You're now going to dive into the world of baseball. Along the way, you'll get comfortable with the basics of `numpy`, a powerful package to do data science.

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?

- 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.

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

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

# Create a numpy array from baseball: np_baseball
np_baseball = np.array(baseball)

# Print out type of np_baseball
print(type(np_baseball))
```

```
In [1]:
```



Exercise

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?

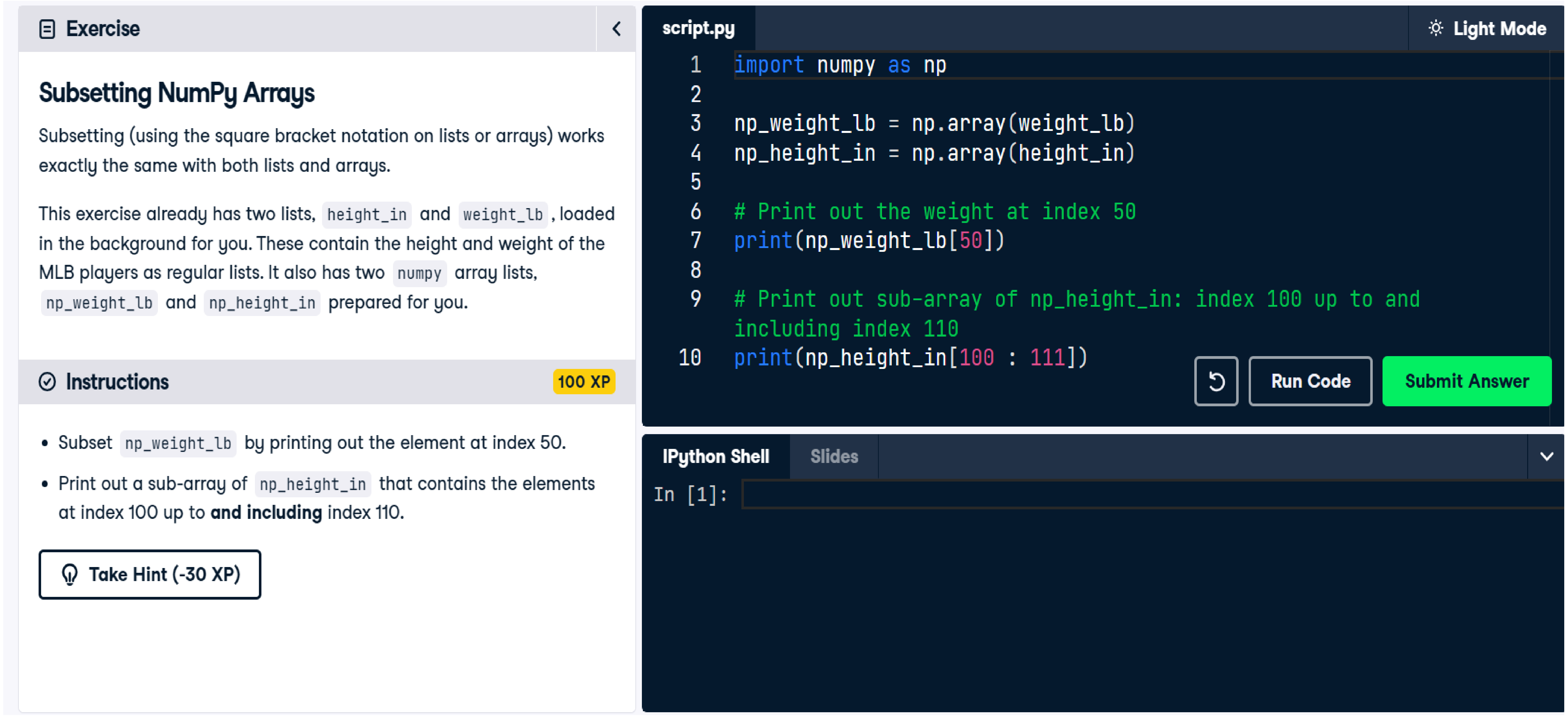
`height_in` is already available and the `numpy` package is loaded, so you can start straight away (Source: stat.ucla.edu).

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

```python
# 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)
```



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` array lists, `np_weight_lb` and `np_height_in` prepared for you.

- 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.

```python
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])
```

```
In [1]:
```

datacamp

2D NumPy Arrays

INTRODUCTION TO PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

Type of NumPy Arrays

```python
import numpy as np np_height = np.array([1.73, 1.68, 
1.71, 1.89, 1.79]) np_weight = np.array([65.4, 59.2, 
63.6, 88.4, 68.7])
```

```python
type(np_height)
```

```
numpy.ndarray
```

```python
type(np_weight)
```

```
numpy.ndarray
```

2D NumPy Arrays

```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, 68.7]])
np_2d
```

```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],
       [65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

```
np_2d.shape
```

```
(2, 5) # 2 rows, 5 columns
```

```
np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
          [65.4, 59.2, 63.6, 88.4, "68.7"]])
```

Subsetting

```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']], dtype='<U32')
```

```
             0        1        2        3        4
 array([[  1.73,    1.68,    1.71,    1.89,    1.79],
0         [  65.4,    59.2,    63.6,    88.4,    68.7]])
1
```

```
np_2d[0]
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
             0        1        2        3        4
 array([[  1.73,    1.68,    1.71,    1.89,    1.79],
0         [  65.4,    59.2,    63.6,    88.4,    68.7]])
1
```

```
np_2d[0][2]
```

Subsetting

```
1.71
```

```
np_2d[0, 2]
```

```
1.71
           0       1       2       3       4
 array([[  1.73,   1.68,   1.71,   1.89,   1.79],
0         [  65.4,    59.2,    63.6,    88.4,    68.7]])
1
```

```
np_2d[:, 1:3]
```

```
array([[ 1.68,  1.71],
       [59.2 , 63.6 ]])
```

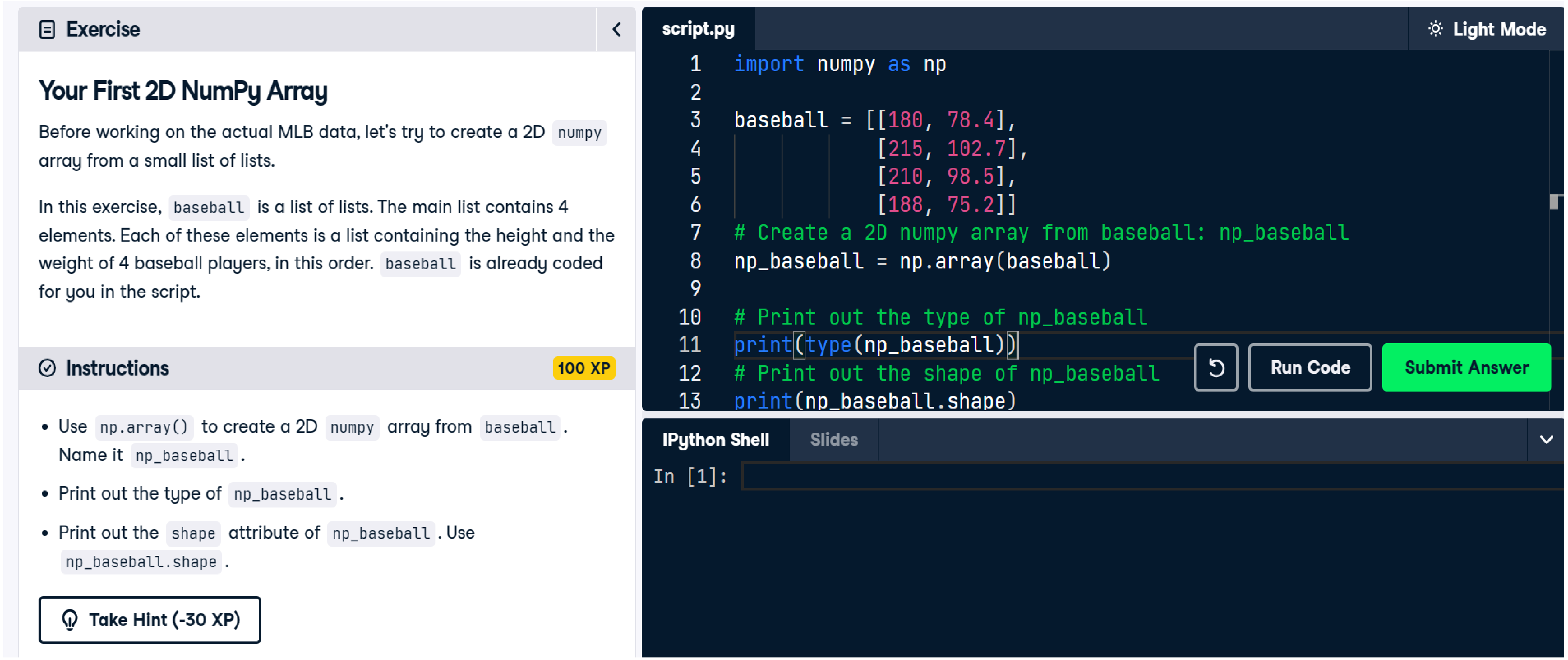
Subsetting

```
np_2d[1, :]
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

Let's practice!

INTRODUCTION TO PYTHON



Your First 2D NumPy Array

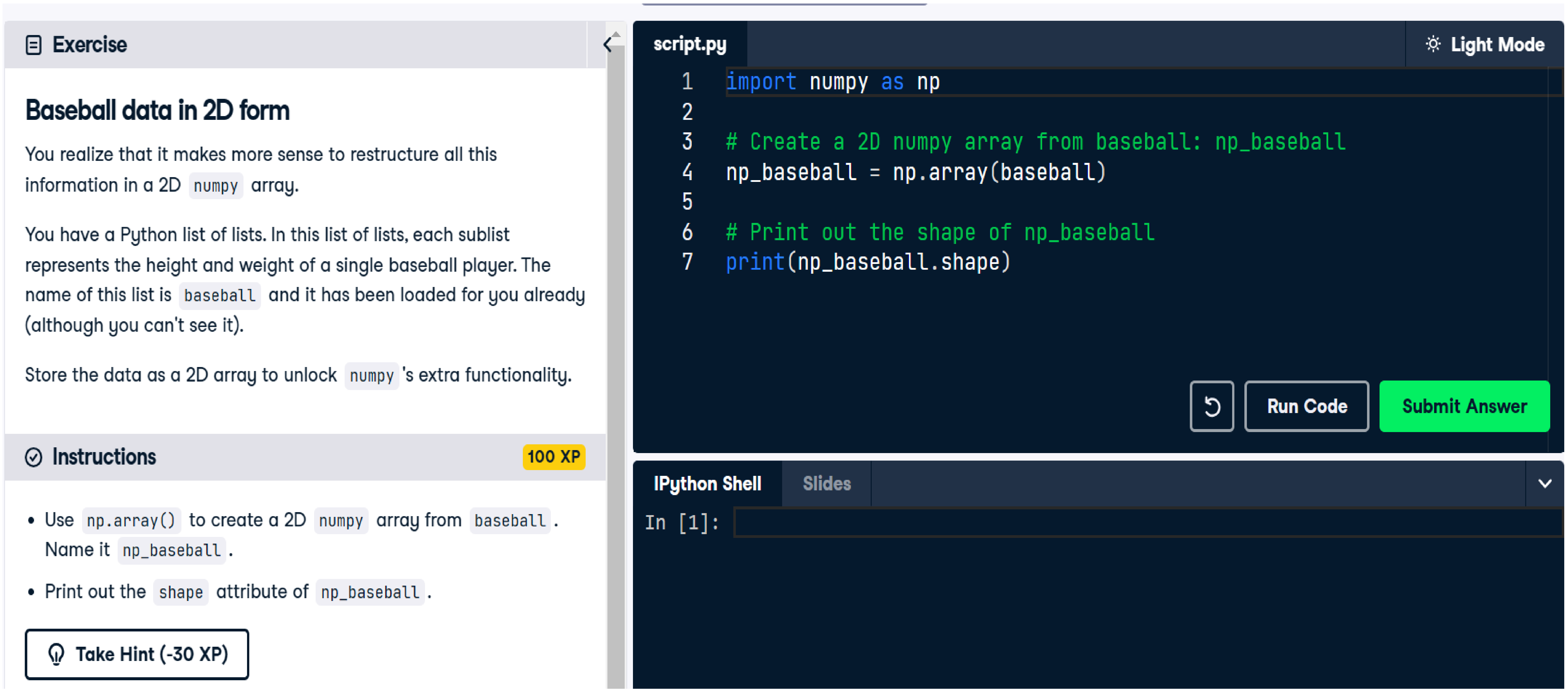
Before working on the actual MLB data, let's try to create a 2D `numpy` array from a small list of lists.

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.

- 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`.

```python
import numpy as np

baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]
# 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)
```

```
In [1]:
```



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 this list of lists, each sublist represents the height and weight of a single baseball player. The name of this list is `baseball` and it has been loaded for you already (although you can't see it).

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

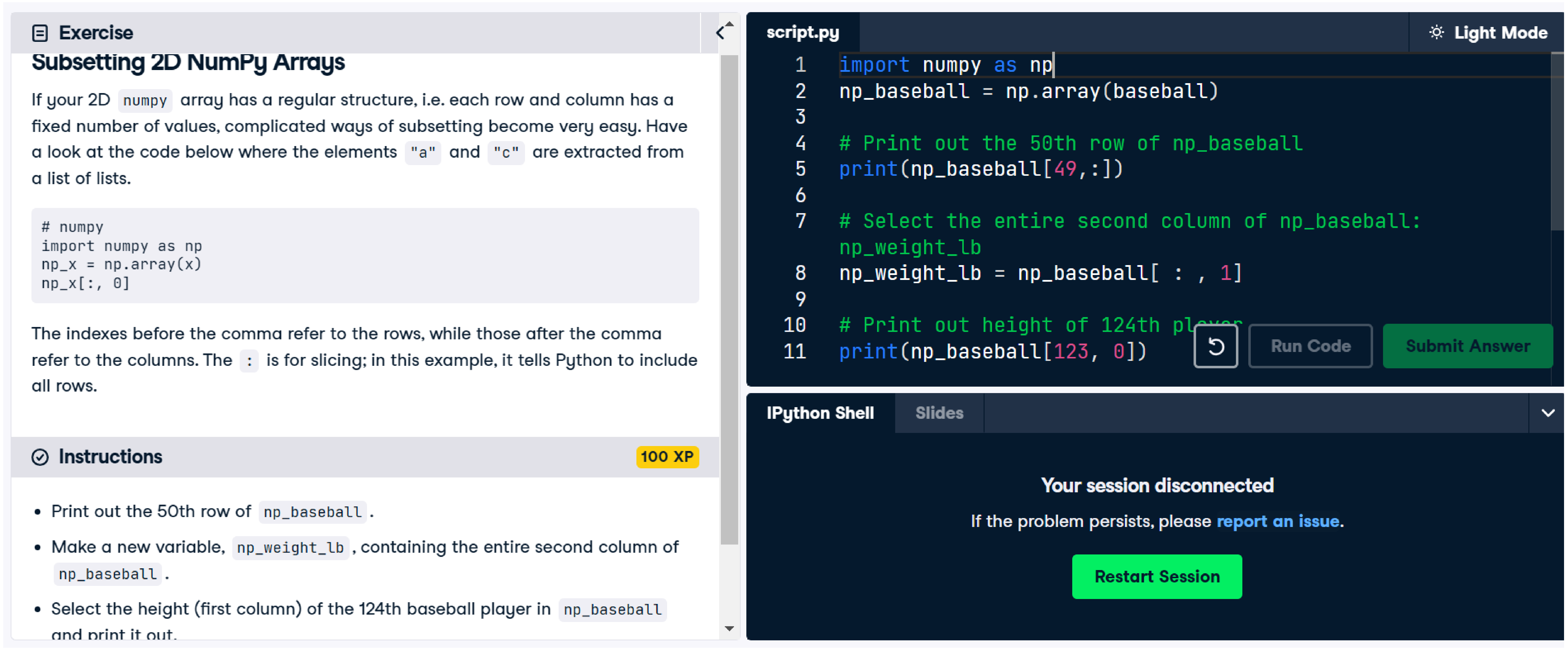
Instructions 100 XP

- 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)

script.py

```python
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)
```



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 `"a"` and `"c"` are extracted from a list of lists.

```
# numpy
import numpy as np
np_x = np.array(x)
np_x[:, 0]
```

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.

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.

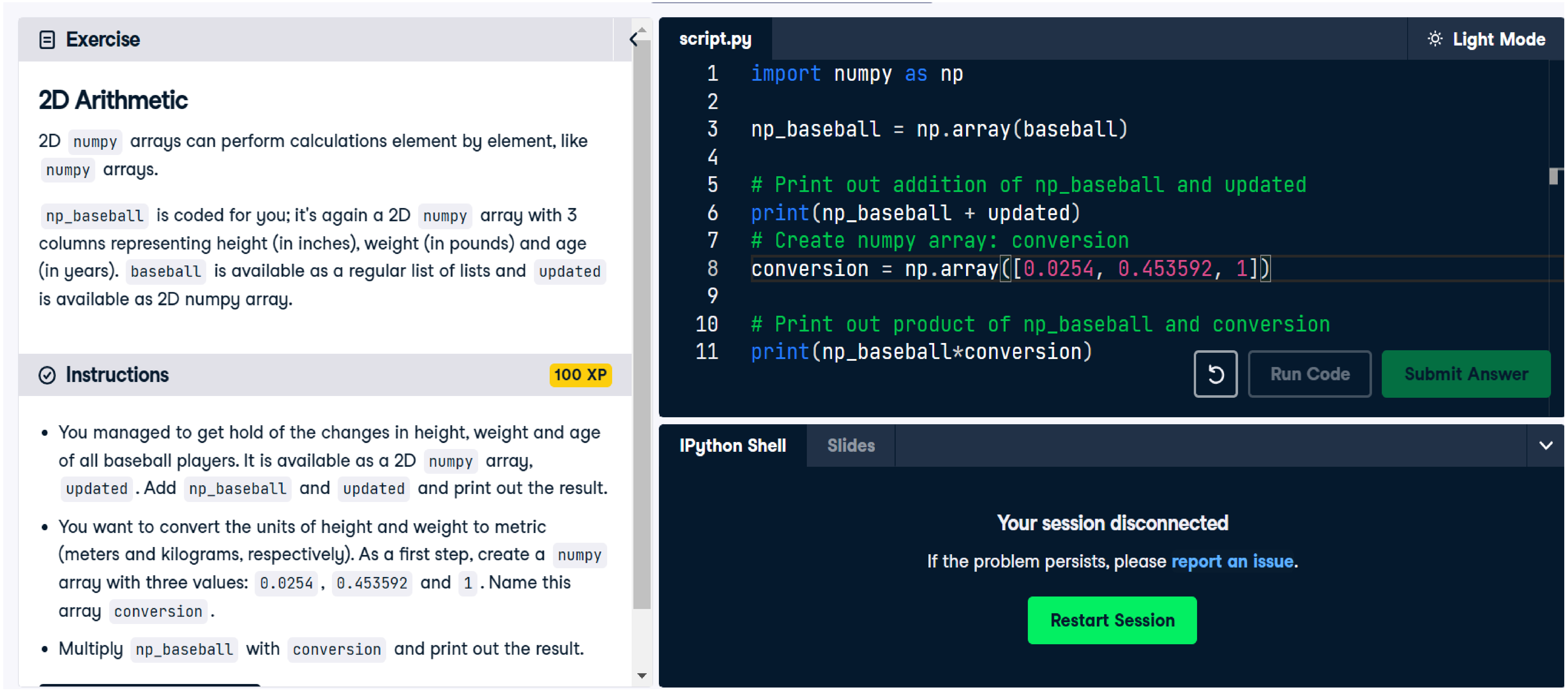
script.py

```
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 124th player
print(np_baseball[123, 0])
```

IPython Shell

Your session disconnected

If the problem persists, please report an issue.



Exercise

2D Arithmetic

2D `numpy` arrays can perform calculations element by element, like `numpy` arrays.

`np_baseball` is coded for you; it's again a 2D `numpy` array with 3 columns representing height (in inches), weight (in pounds) and age (in years). `baseball` is available as a regular list of lists and `updated` is available as 2D numpy array.

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

```python
import numpy as np

np_baseball = np.array(baseball)

# Print out addition of np_baseball and updated
print(np_baseball + updated)
# Create numpy array: conversion
conversion = np.array([0.0254, 0.453592, 1])

# Print out product of np_baseball and conversion
print(np_baseball*conversion)
```

Your session disconnected

If the problem persists, please report an issue.

NumPy: Basic Statistics

INTRODUCTION TO PYTHON

Hugo Bowne-Anderson

Data analysis

- Get to know your data
- Little data

-> simply look at it
- Big data -> ?

City-wide survey

```
import numpy as np
np_city = ... # Implementation left out
np_city
```

```
array([[1.64, 71.78],
       [1.37, 63.35],
       [1.6 , 55.09],
       ...,
       [2.04, 74.85],
       [2.04, 68.72],
       [2.01, 73.57]])
```

NumPy

```
np.mean(np_city[:, 0])
```

```
1.7472
```

```
np.median(np_city[:, 0])
```

```
1.75
```

NumPy

```
np.corrcoef(np_city[:, 0], np_city[:, 1])
```

```
array([[ 1.      , -0.01802],
       [-0.01803,  1.      ]])
```

```
np.std(np_city[:, 0])
```

```
0.1992
```

- sum(), sort(), ...
- Enforce single data type: speed!

Generate data

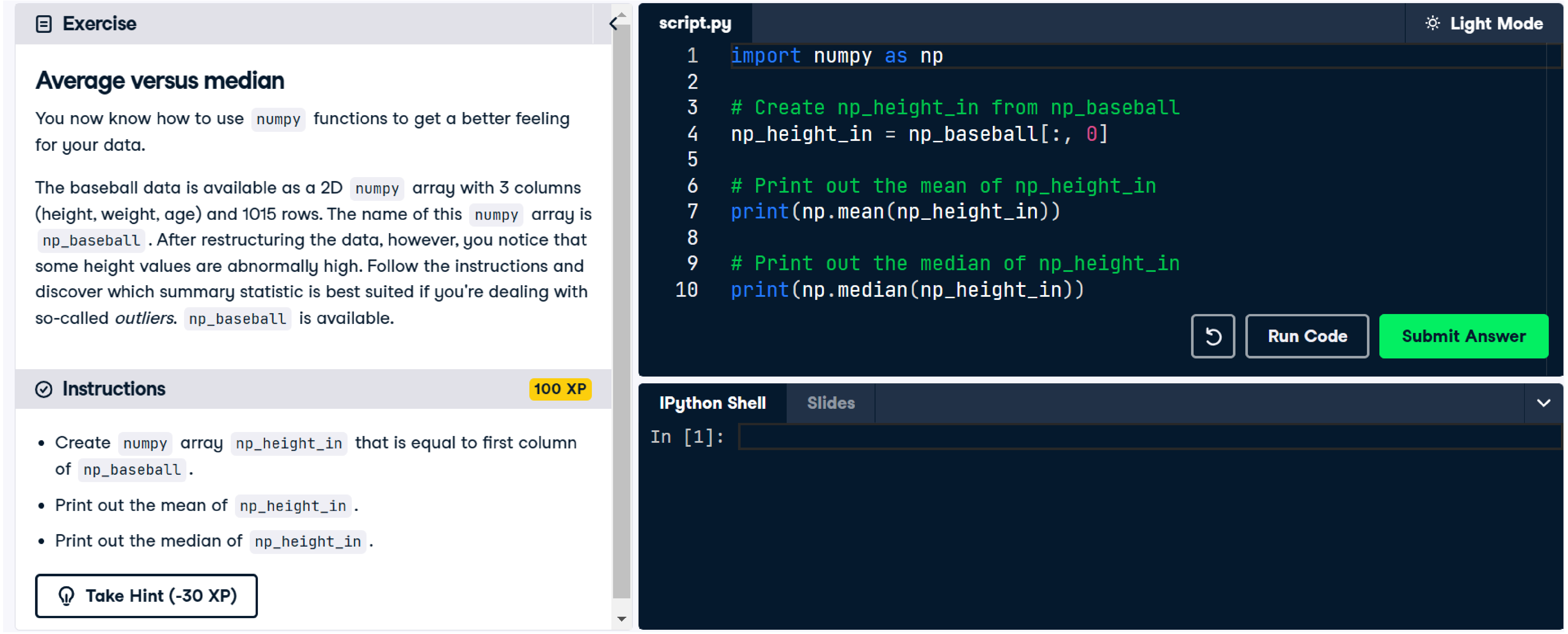
  - Arguments for

- distribution mean
- distribution standard deviation
- number of samples

```
height = np.round(np.random.normal(1.75, 0.20, 5000), 2)

weight = np.round(np.random.normal(60.32, 15, 5000), 2)

np_city = np.column_stack((height, weight))
```

Let's practice!

INTRODUCTION TO PYTHON



Exercise

Average versus median

You now know how to use `numpy` functions to get a better feeling for your data.

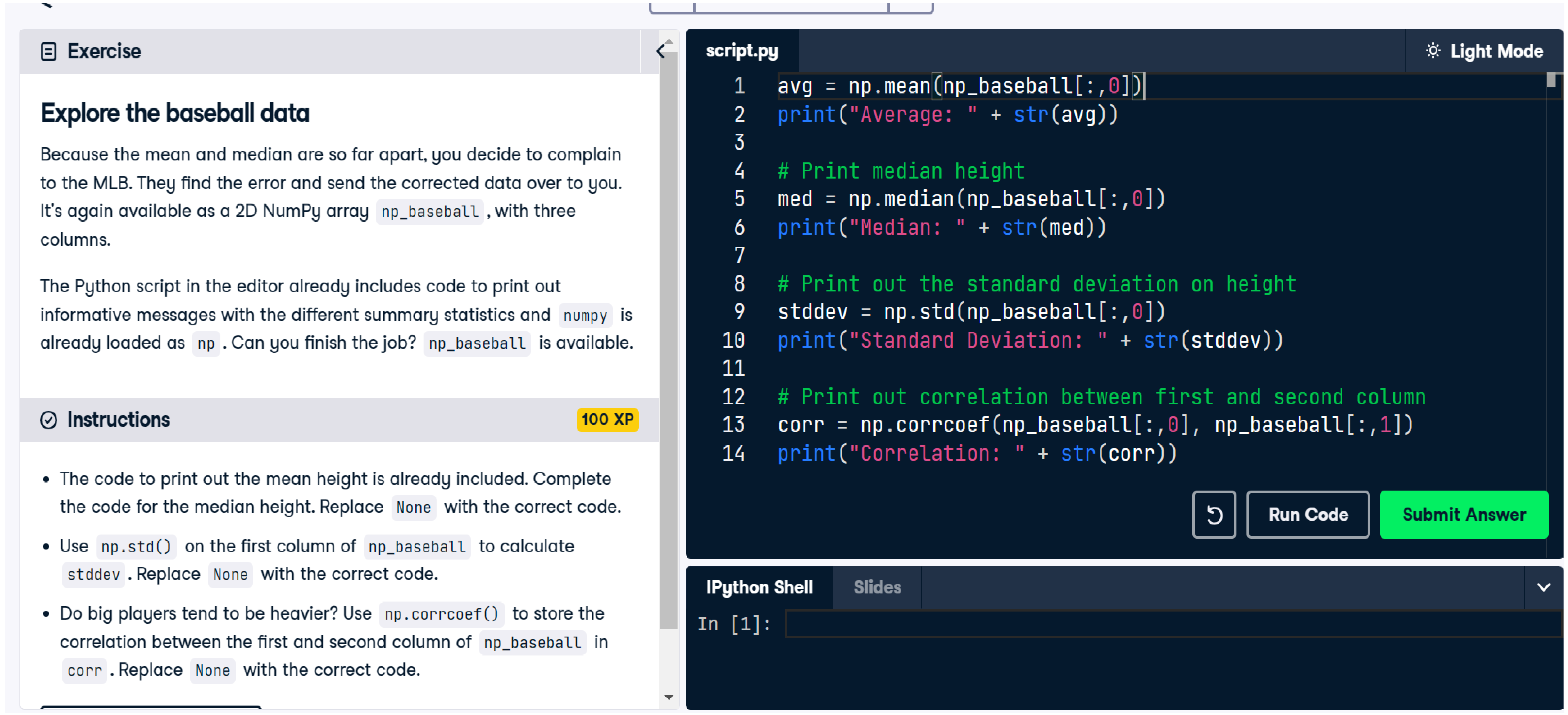
The baseball data is available as a 2D `numpy` array with 3 columns (height, weight, age) and 1015 rows. The name of this `numpy` array is `np_baseball`. After restructuring the data, however, you notice that some height values are abnormally high. Follow the instructions and discover which summary statistic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

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

```python
import numpy as np

# Create np_height_in from np_baseball
np_height_in = 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))
```



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 in the editor already includes code to print out informative messages with the different summary statistics and `numpy` is already loaded as `np`. Can you finish the job? `np_baseball` is available.

- The code to print out the mean height is already included. Complete the code for the median height. Replace `None` with the correct code.
- Use `np.std()` 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()` to store the correlation between the first and second column of `np_baseball` in `corr`. Replace `None` with the correct code.

```python
avg = np.mean(np_baseball[:,0])
print("Average: " + str(avg))

# Print median height
med = np.median(np_baseball[:,0])
print("Median: " + str(med))

# Print out the standard deviation on height
stddev = np.std(np_baseball[:,0])
print("Standard Deviation: " + str(stddev))

# Print out correlation between first and second column
corr = np.corrcoef(np_baseball[:,0], np_baseball[:,1])
print("Correlation: " + str(corr))
```

```
In [1]:
```