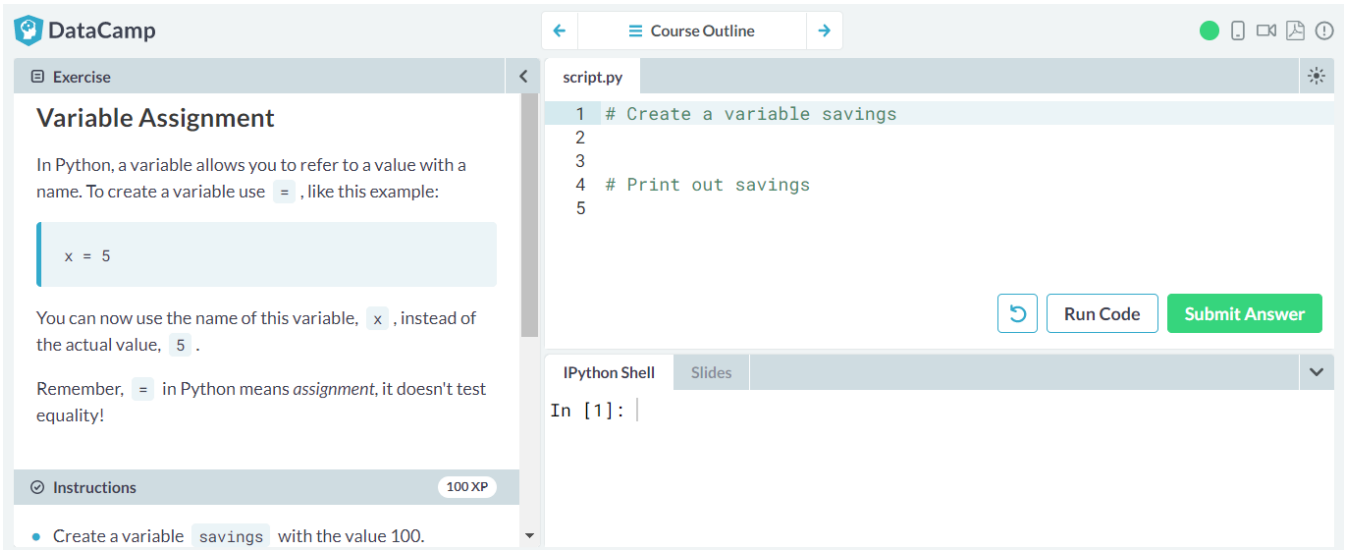
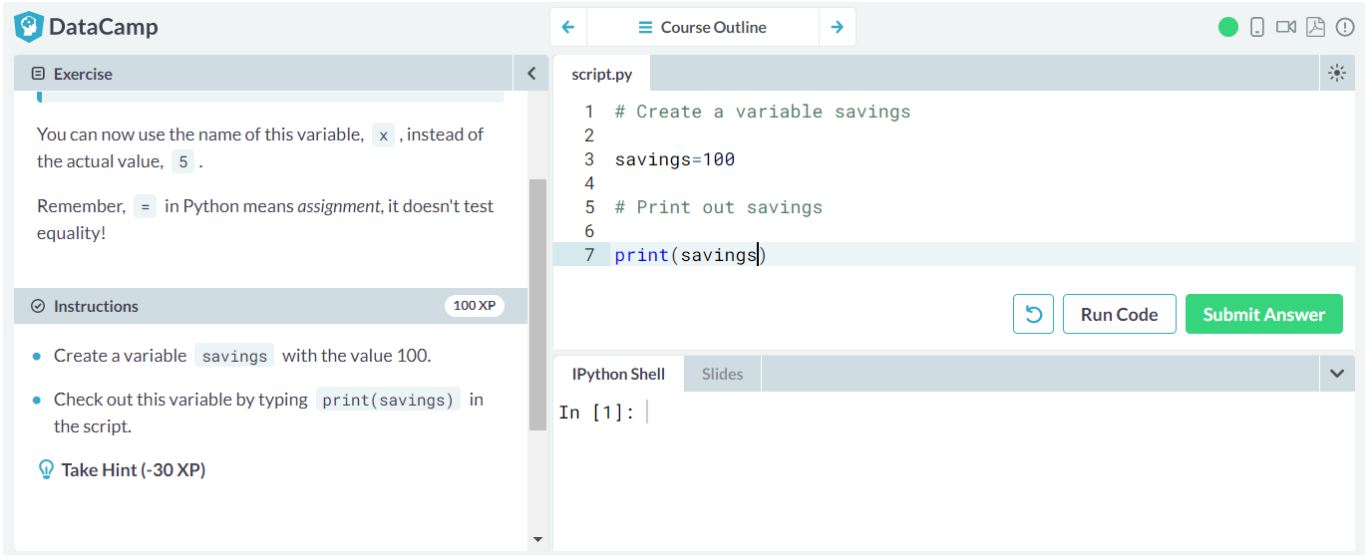
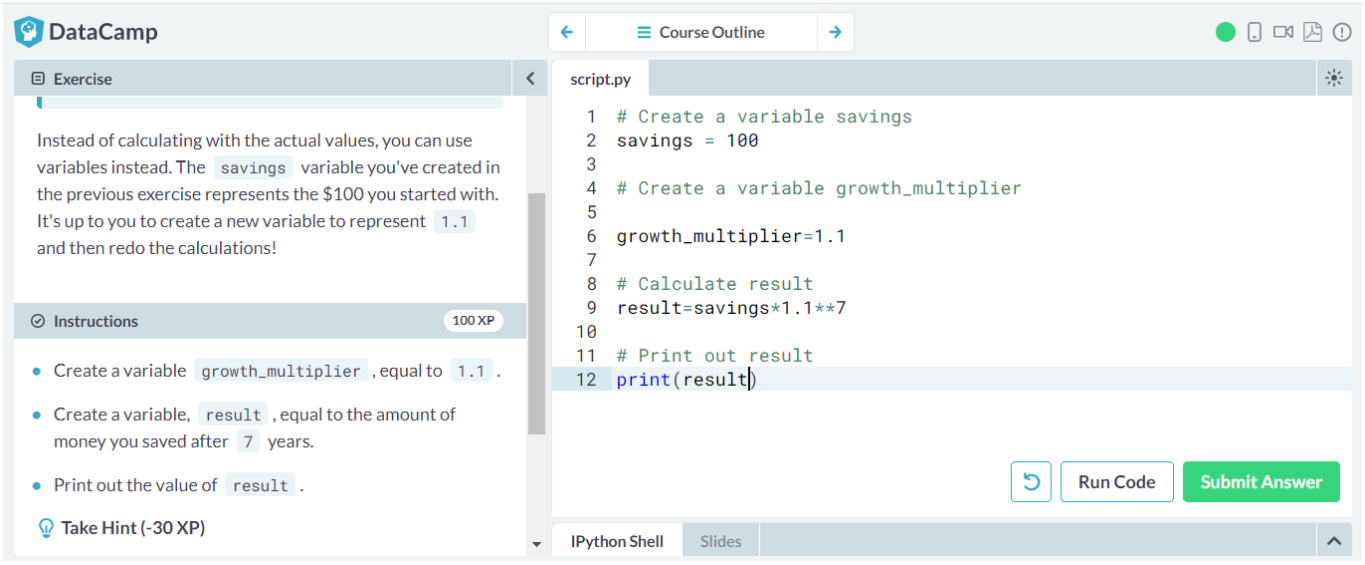
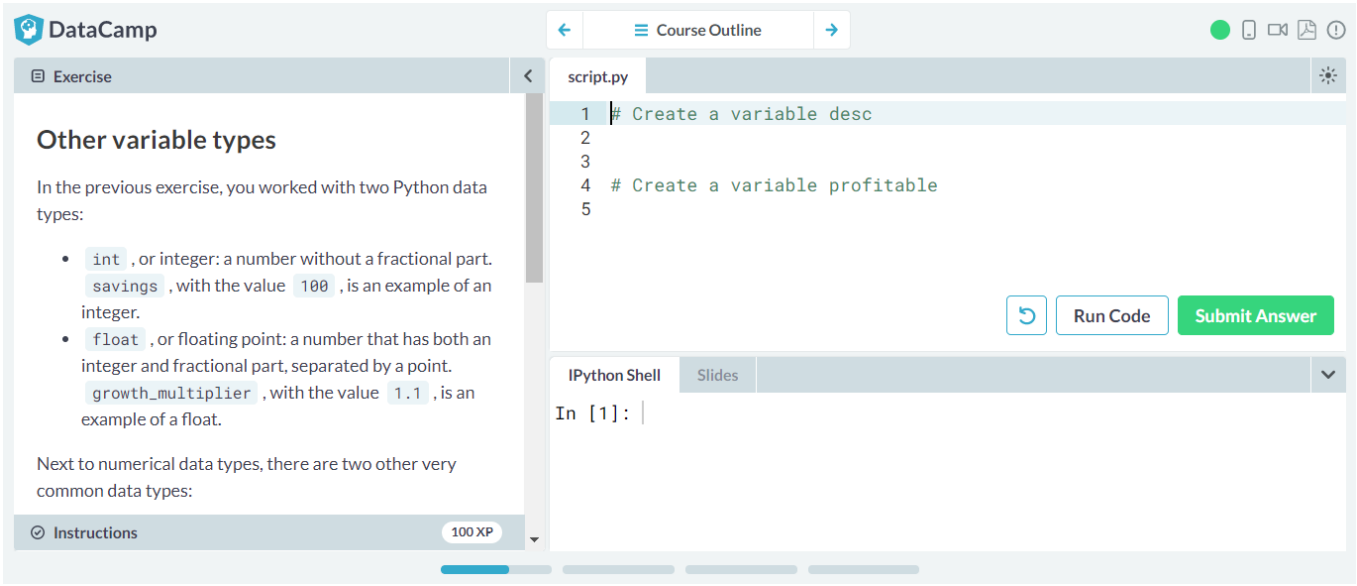
# INTRODUCTION TO PYTHON

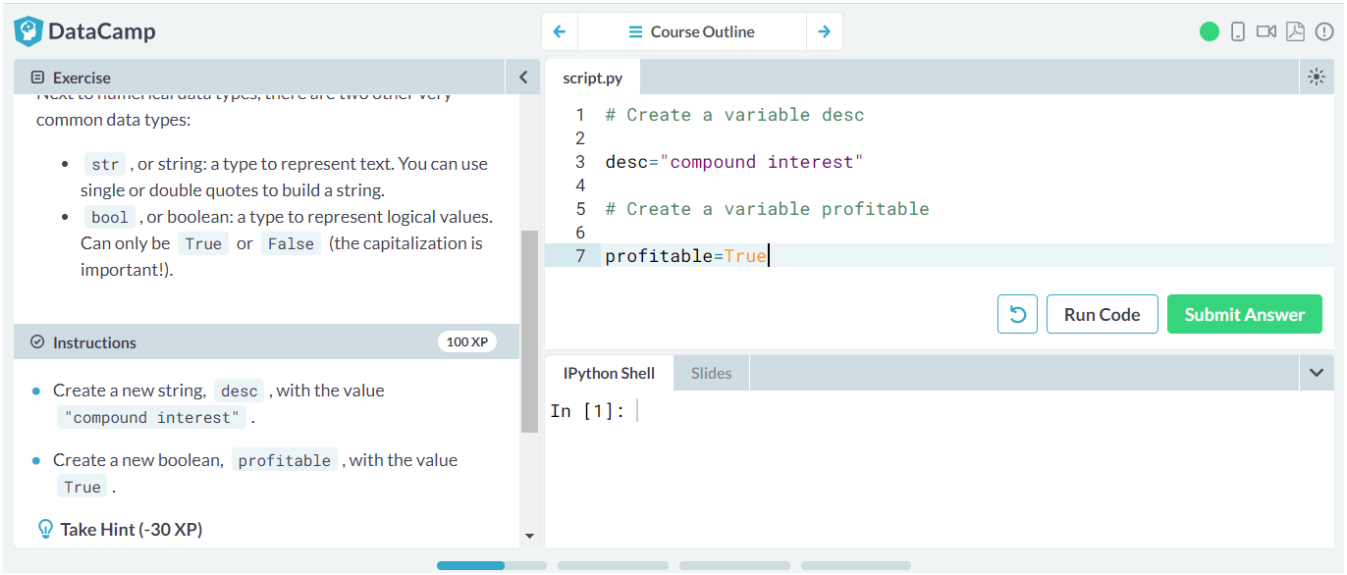
# DataCamp

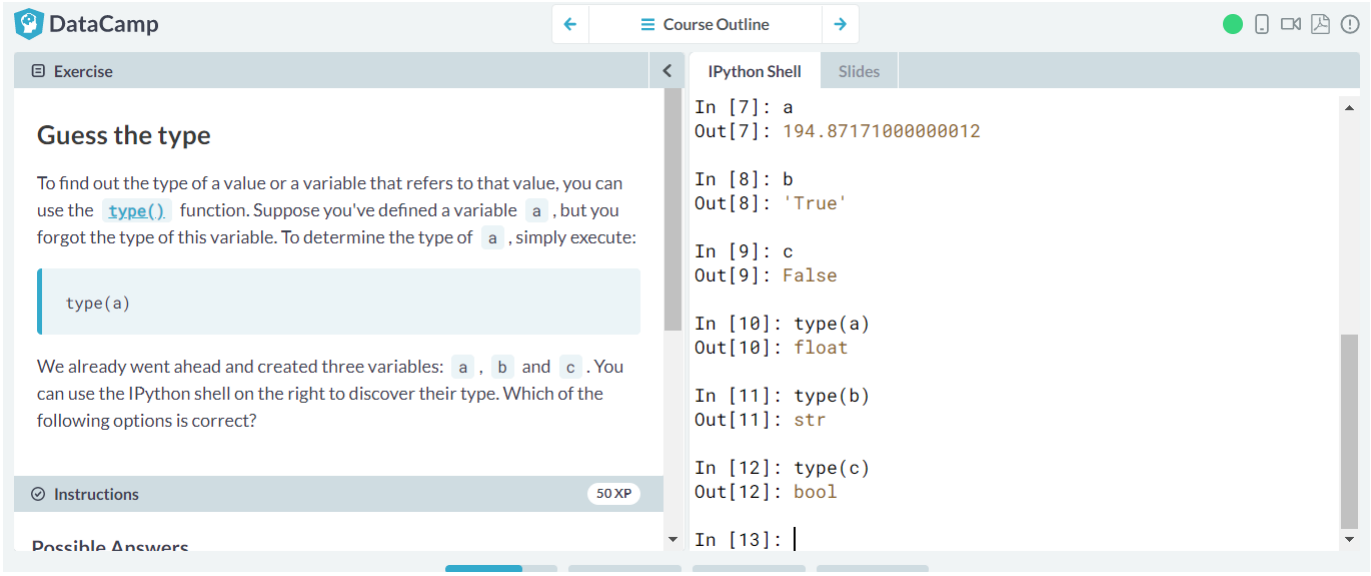


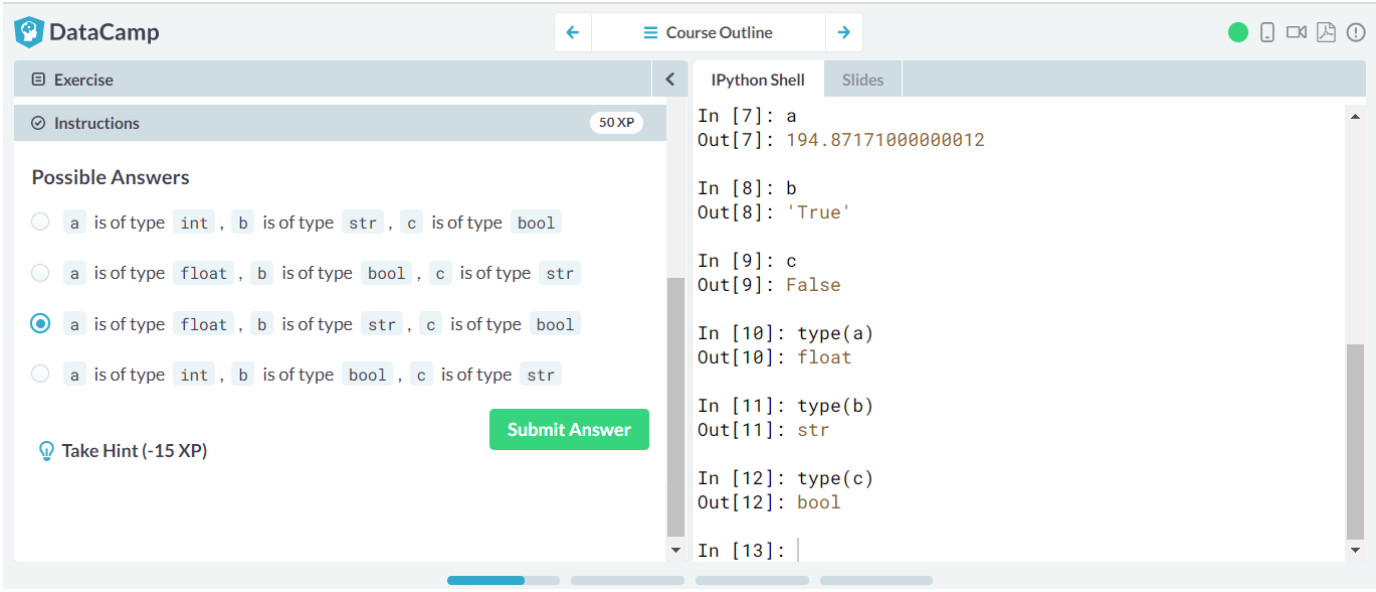



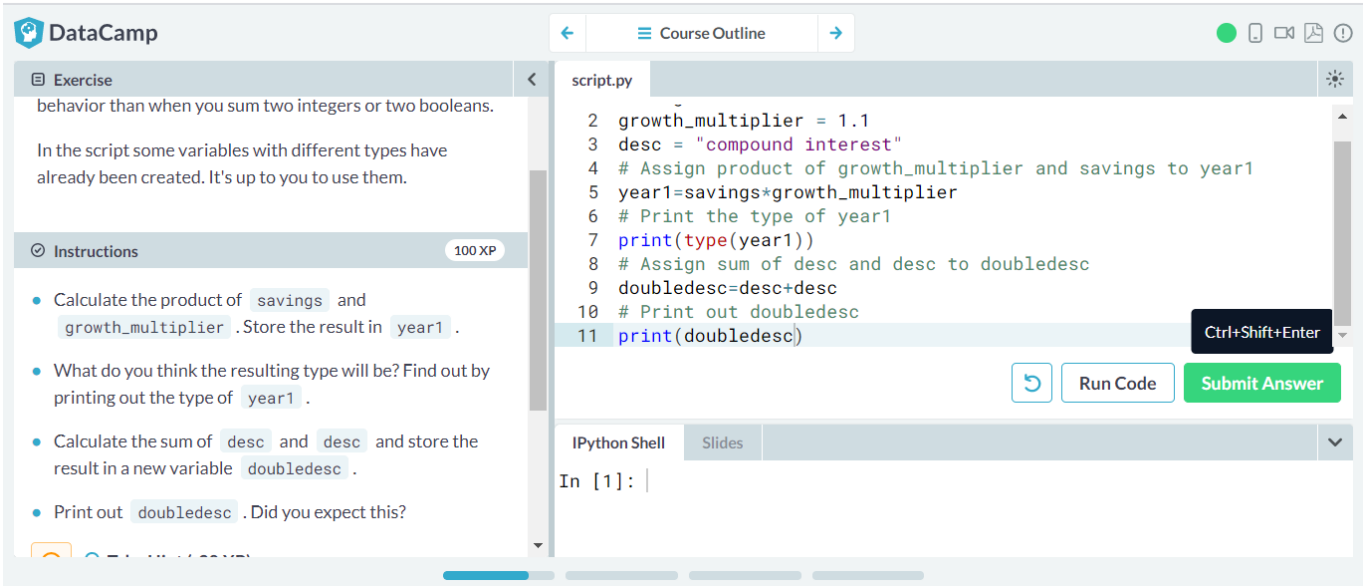

DataCamp
Course Outline
Exercise
Instead of calculating with the actual values, you can use variables instead. The savings variable you've created in the previous exercise represents the $100 you started with. It's up to you to create a new variable to represent 1.1 and then redo the calculations!
Instructions
100 XP
Create a variable growth_multiplier, equal to 1.1.
Create a variable, result, equal to the amount of money you saved after 7 years.
Print out the value of result.
Take Hint (-30 XP)
script.py
# Create a variable savings
savings = 100
# Create a variable growth_multiplier
growth_multiplier=1.1
# Calculate result
result=savings*1.1**7
# Print out result
print(result)
Run Code
Submit Answer
IPython Shell
Slides

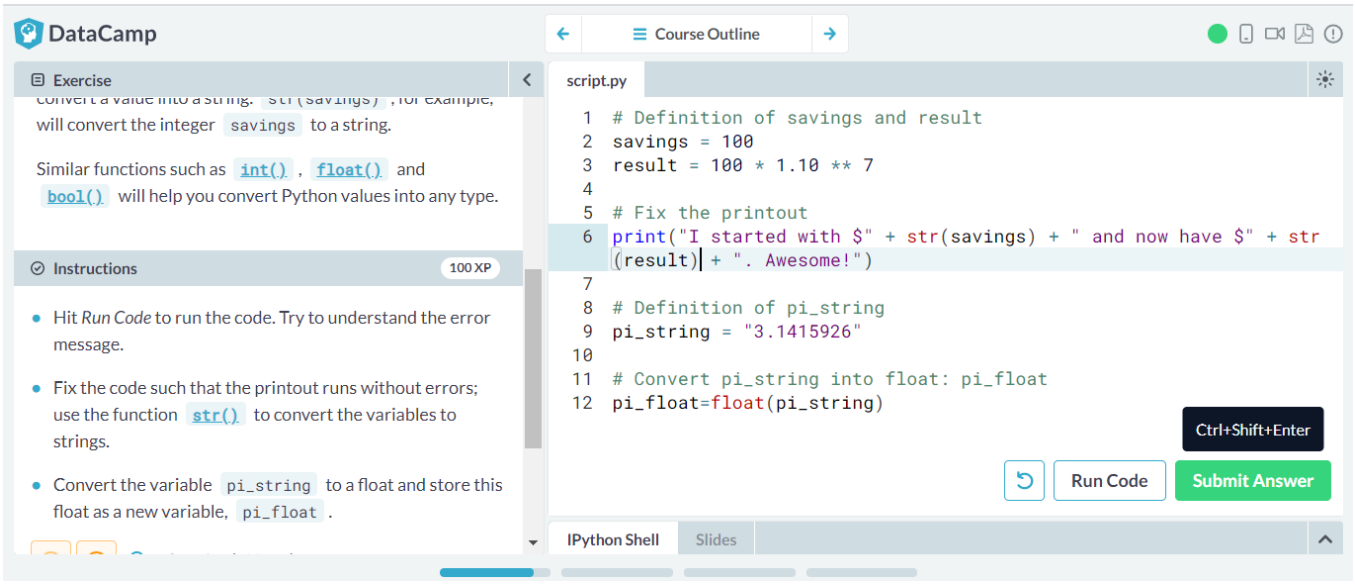

DataCamp
Course Outline
Exercise
Other variable types
In the previous exercise, you worked with two Python data types:
int, or integer: a number without a fractional part. savings, with the value 100, is an example of an integer.
float, or floating point: a number that has both an integer and fractional part, separated by a point. growth_multiplier, with the value 1.1, is an example of a float.
Next to numerical data types, there are two other very common data types:
Instructions
100 XP
script.py
# Create a variable desc
# Create a variable profitable
Run Code
Submit Answer
IPython Shell
Slides
In [1]:

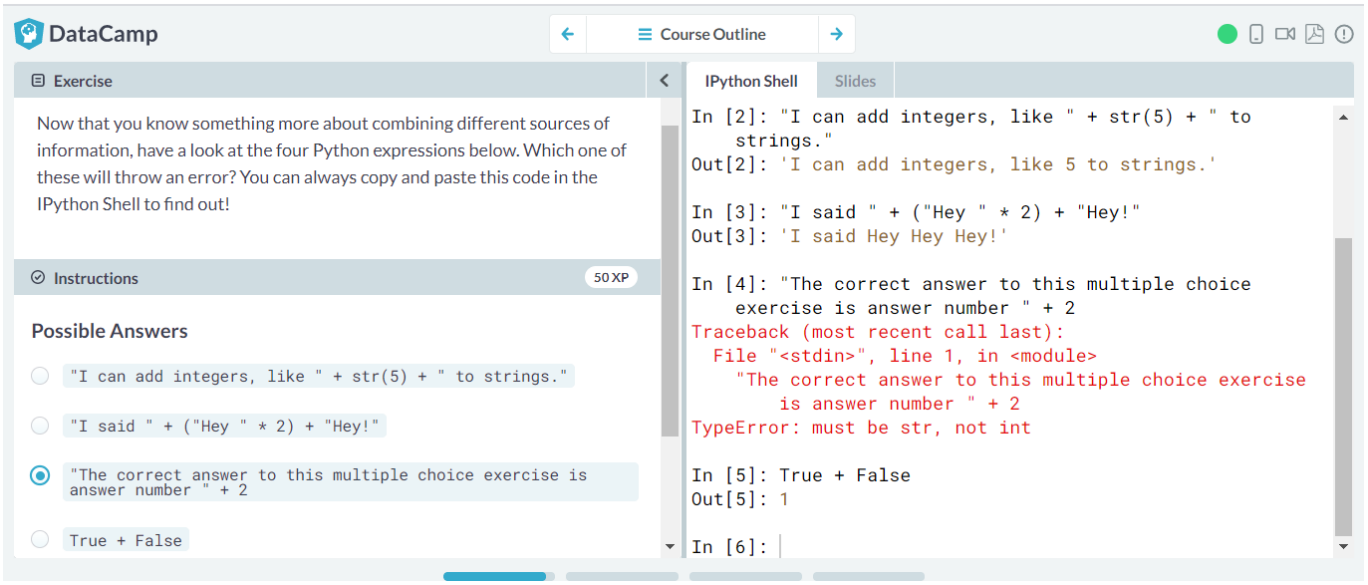

DataCamp
Course Outline
Exercise
common data types:
str , or string: a type to represent text. You can use single or double quotes to build a string.
bool , or boolean: a type to represent logical values. Can only be True or False (the capitalization is important!).
Instructions
100 XP
Create a new string, desc , with the value "compound interest" .
Create a new boolean, profitable , with the value True .
Take Hint (-30 XP)
script.py
1 # Create a variable desc
2
3 desc="compound interest"
4
5 # Create a variable profitable
6
7 profitable=True
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
Guess the type
To find out the type of a value or a variable that refers to that value, you can use the type() function. Suppose you've defined a variable a , but you forgot the type of this variable. To determine the type of a , simply execute:
type(a)
We already went ahead and created three variables: a , b and c . You can use the IPython shell on the right to discover their type. Which of the following options is correct?
Instructions
50 XP
Possible Answers
IPython Shell
Slides
In [7]: a
Out[7]: 194.87171000000012
In [8]: b
Out[8]: 'True'
In [9]: c
Out[9]: False
In [10]: type(a)
Out[10]: float
In [11]: type(b)
Out[11]: str
In [12]: type(c)
Out[12]: bool
In [13]:


DataCamp
Course Outline
Exercise
Instructions
50 XP
Possible Answers
a is of type int , b is of type str , c is of type bool
a is of type float , b is of type bool , c is of type str
a is of type float , b is of type str , c is of type bool
a is of type int , b is of type bool , c is of type str
Submit Answer
Take Hint (-15 XP)
IPython Shell
Slides
In [7]: a
Out[7]: 194.87171000000012
In [8]: b
Out[8]: 'True'
In [9]: c
Out[9]: False
In [10]: type(a)
Out[10]: float
In [11]: type(b)
Out[11]: str
In [12]: type(c)
Out[12]: bool
In [13]:


DataCamp
Course Outline
Exercise
behavior than when you sum two integers or two booleans.
In the script some variables with different types have already been created. It's up to you to use them.
Instructions
100 XP
Calculate the product of savings and growth_multiplier . Store the result in year1 .
What do you think the resulting type will be? Find out by printing out the type of year1 .
Calculate the sum of desc and desc and store the result in a new variable doubledesc .
Print out doubledesc . Did you expect this?
script.py
growth_multiplier = 1.1
desc = "compound interest"
# Assign product of growth_multiplier and savings to year1
year1=savings*growth_multiplier
# Print the type of year1
print(type(year1))
# Assign sum of desc and desc to doubledesc
doubledesc=desc+desc
# Print out doubledesc
print(doubledesc)
Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
will convert the integer savings to a string.
Similar functions such as int() , float() and bool() will help you convert Python values into any type.
Instructions
100 XP
Hit Run Code to run the code. Try to understand the error message.
Fix the code such that the printout runs without errors; use the function str() to convert the variables to strings.
Convert the variable pi_string to a float and store this float as a new variable, pi_float .
script.py
1 # Definition of savings and result
2 savings = 100
3 result = 100 * 1.10 ** 7
4
5 # Fix the printout
6 print("I started with $" + str(savings) + " and now have $" + str(result) + ". Awesome!")
7
8 # Definition of pi_string
9 pi_string = "3.1415926"
10
11 # Convert pi_string into float: pi_float
12 pi_float=float(pi_string)
Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides


DataCamp
Course Outline
Exercise
Now that you know something more about combining different sources of information, have a look at the four Python expressions below. Which one of these will throw an error? You can always copy and paste this code in the IPython Shell to find out!
Instructions
50 XP
Possible Answers
"I can add integers, like " + str(5) + " to strings."
"I said " + ("Hey " * 2) + "Hey!"
"The correct answer to this multiple choice exercise is answer number " + 2
True + False
IPython Shell
Slides
In [2]: "I can add integers, like " + str(5) + " to strings."
Out[2]: 'I can add integers, like 5 to strings.'
In [3]: "I said " + ("Hey " * 2) + "Hey!"
Out[3]: 'I said Hey Hey Hey!'
In [4]: "The correct answer to this multiple choice exercise is answer number " + 2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
"The correct answer to this multiple choice exercise is answer number " + 2
TypeError: must be str, not int
In [5]: True + False
Out[5]: 1
In [6]:

# Python Data Types

- float - real numbers
- int - integer numbers
- str - string, text
- bool - True, False

```
height = 1.73
tall = True
```

- Each variable represents single value


# Problem

- Data Science: many data points
- Height of entire family

```
height1 = 1.73
height2 = 1.68
height3 = 1.71
height4 = 1.89
```

- Inconvenient

# Python List

- `[a, b, c]`

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

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

```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

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

- Name a collection of values
- Contain any type
- Contain different types

integer, booleans, strings, but also more advanced Python types, even lists.



# Python List

- `[a, b, c]`

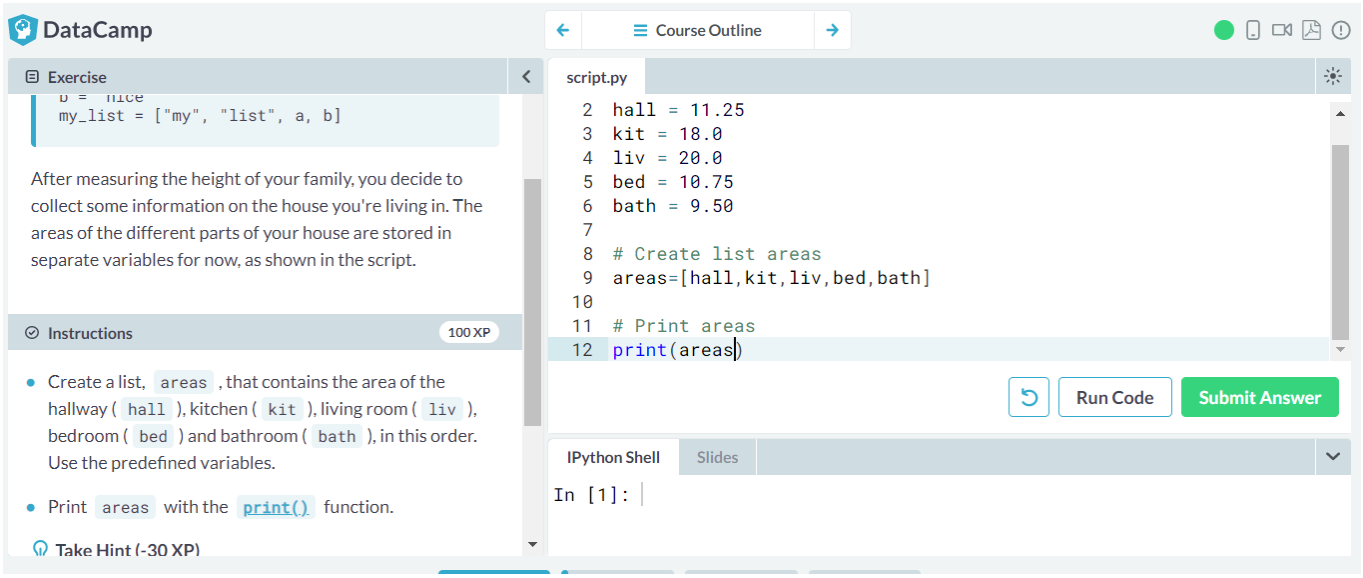
```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
fam
```

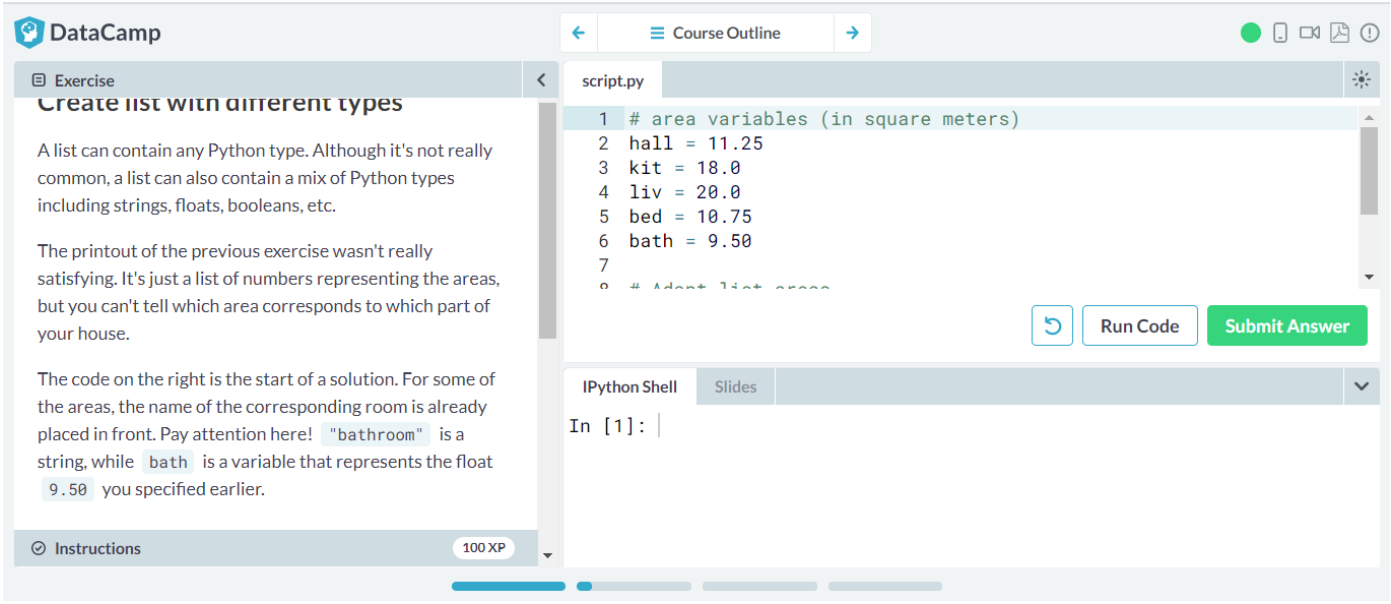
```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

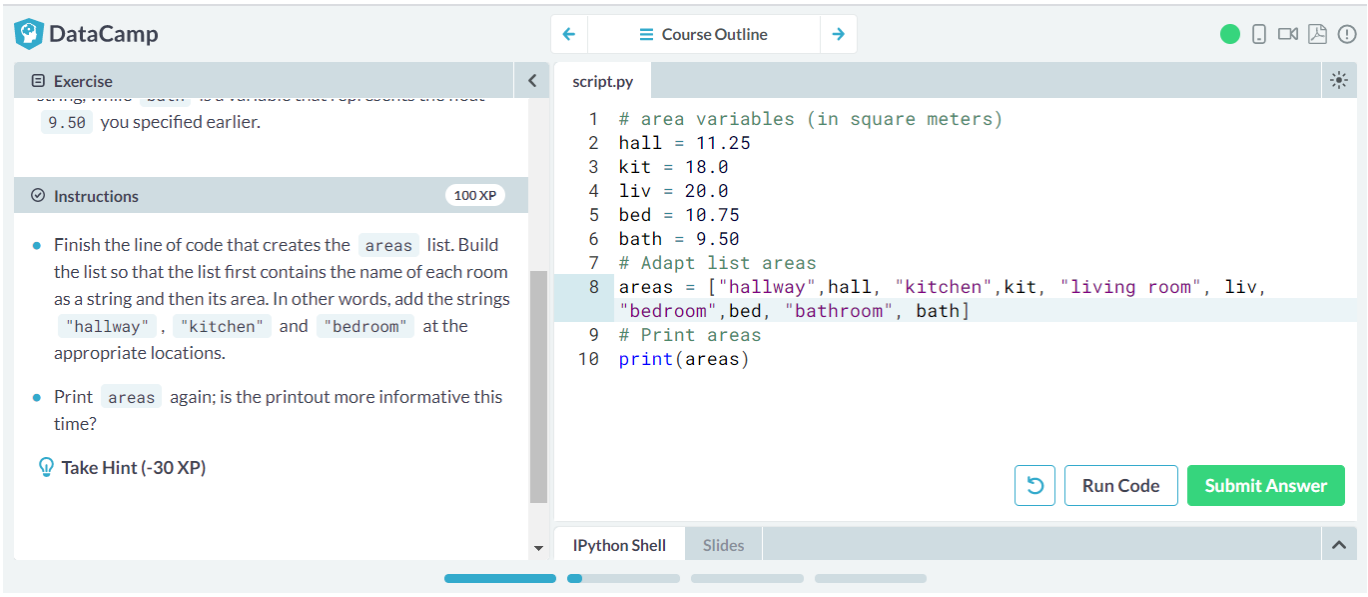
```
fam2 = [["liz", 1.73],
        ["emma", 1.68],
        ["mom", 1.71],
        ["dad", 1.89]]
fam2
```

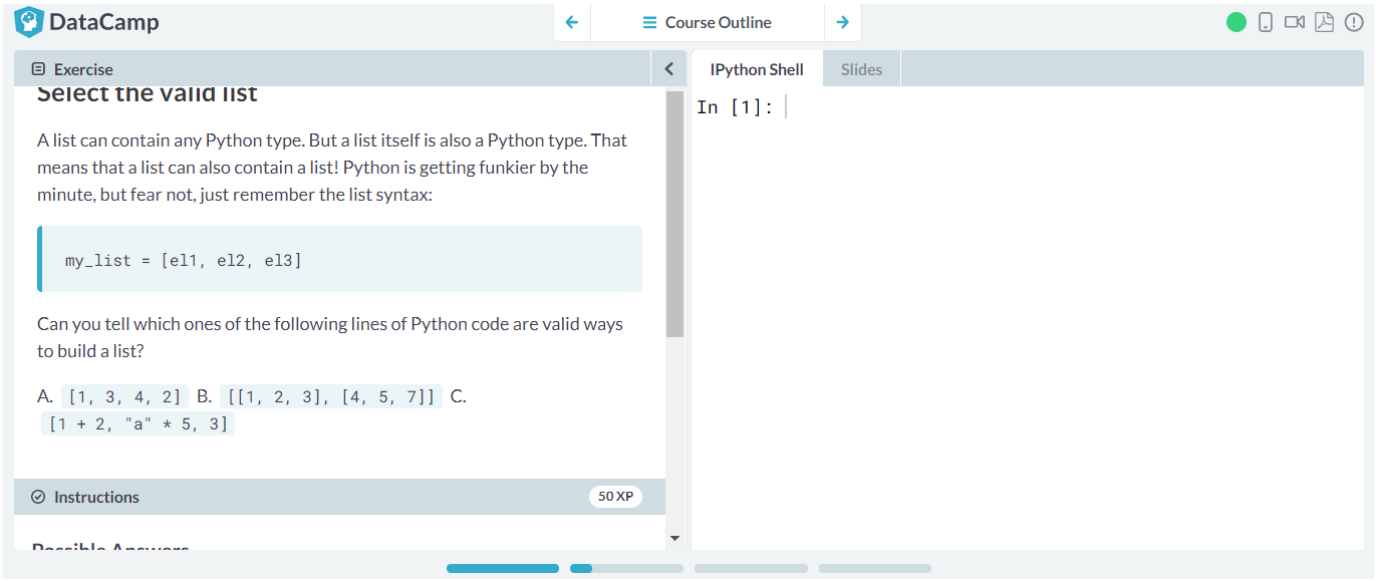
```
[['liz', 1.73], ['emma', 1.68], ['mom', 1.71], ['dad', 1.89]]
```

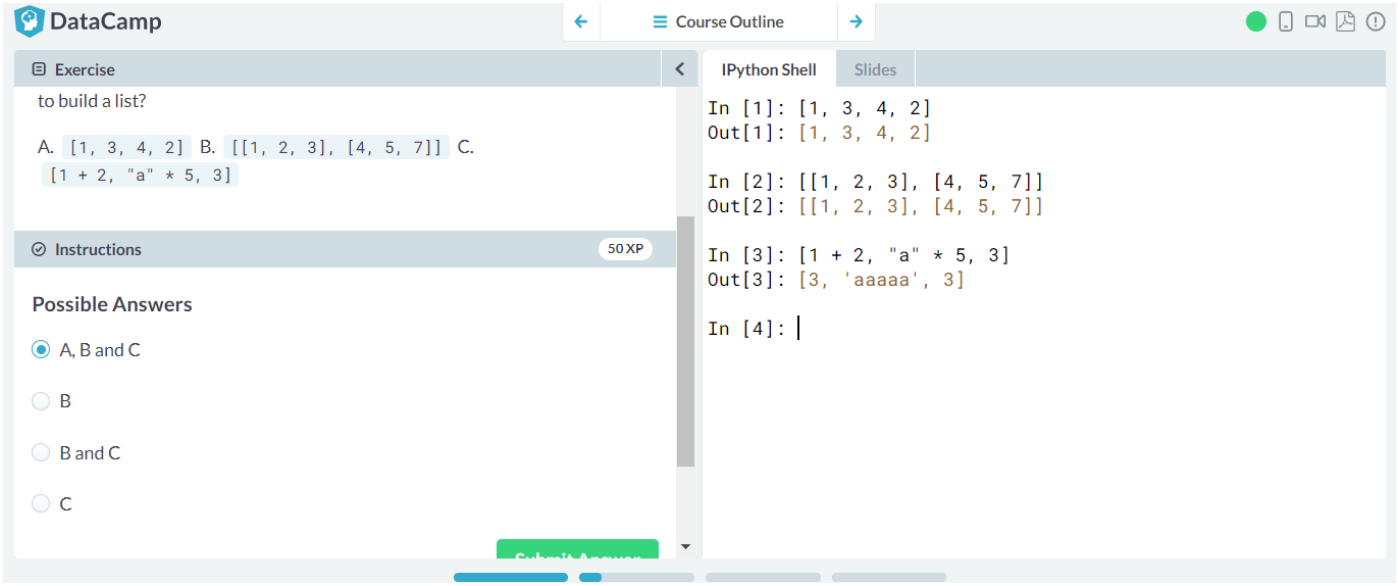
The main list contains 4 sub-lists.

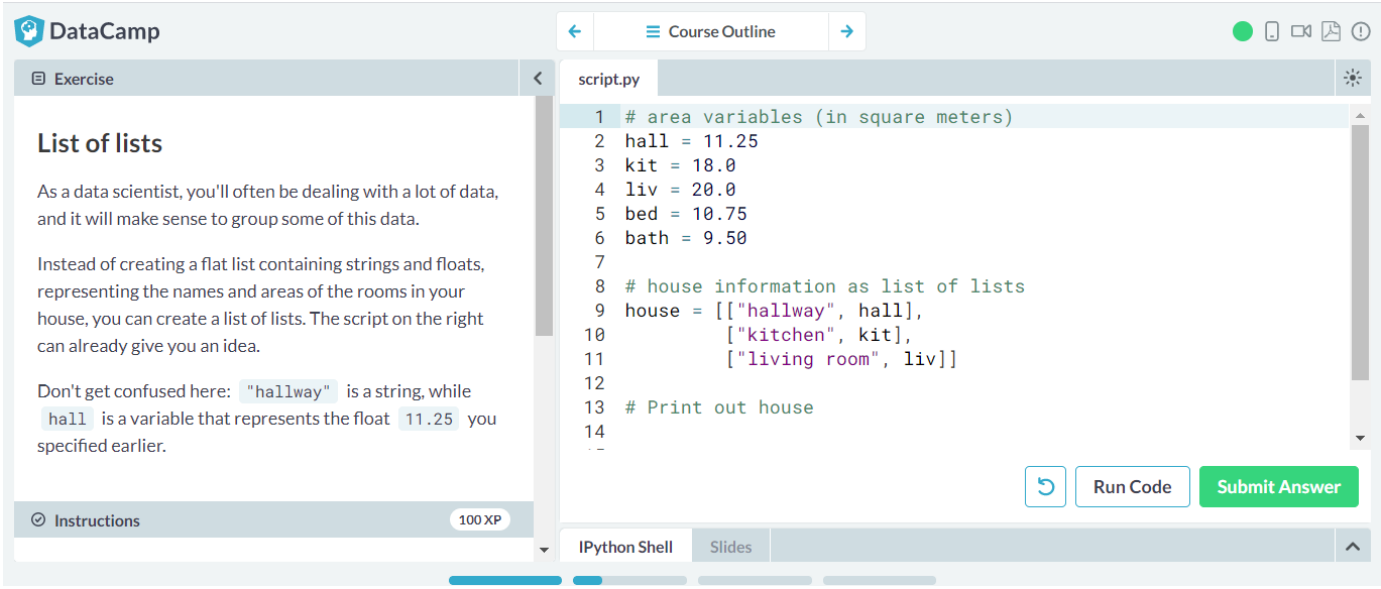

DataCamp
Course Outline
Exercise
my_list = ["my", "list", a, b]
After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are stored in separate variables for now, as shown in the script.
Instructions
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)
script.py
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
8 # Create list areas
9 areas=[hall,kit,liv,bed,bath]
10
11 # Print areas
12 print(areas)
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
Create list with different types
A list can contain any Python type. Although it's not really common, a list can also contain a mix of Python types including strings, floats, booleans, etc.
The printout of the previous exercise wasn't really satisfying. It's just a list of numbers representing the areas, but you can't tell which area corresponds to which part of your house.
The code on the right is the start of a solution. For some of the areas, the name of the corresponding room is already placed in front. Pay attention here! "bathroom" is a string, while bath is a variable that represents the float 9.50 you specified earlier.
Instructions
100 XP
script.py
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
9.50 you specified earlier.
Instructions
100 XP
Finish the line of code that creates the areas list. Build the list so that the list first contains the name of each room as a string and then its area. In other words, add the strings "hallway", "kitchen" and "bedroom" at the appropriate locations.
Print areas again; is the printout more informative this time?
Take Hint (-30 XP)
script.py
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7 # Adapt list areas
8 areas = ["hallway",hall, "kitchen",kit, "living room", liv,
"bedroom",bed, "bathroom", bath]
9 # Print areas
10 print(areas)
Run Code
Submit Answer
IPython Shell
Slides


DataCamp
Course Outline
Exercise
Select the valid list
A list can contain any Python type. But a list itself is also a Python type. That means that a list can also contain a list! Python is getting funkier by the minute, but fear not, just remember the list syntax:
my_list = [el1, el2, el3]
Can you tell which ones of the following lines of Python code are valid ways to build a list?
A. [1, 3, 4, 2] B. [[1, 2, 3], [4, 5, 7]] C. [1 + 2, "a" * 5, 3]
Instructions
50 XP
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
to build a list?
A. [1, 3, 4, 2] B. [[1, 2, 3], [4, 5, 7]] C. [1 + 2, "a" * 5, 3]
Instructions
50 XP
Possible Answers
A, B and C
B
B and C
C
IPython Shell
Slides
In [1]: [1, 3, 4, 2]
Out[1]: [1, 3, 4, 2]
In [2]: [[1, 2, 3], [4, 5, 7]]
Out[2]: [[1, 2, 3], [4, 5, 7]]
In [3]: [1 + 2, "a" * 5, 3]
Out[3]: [3, 'aaaaa', 3]
In [4]:


DataCamp
Course Outline
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 flat list containing strings and floats, representing the names and areas of the rooms in your house, you can create a list of lists. The script on the right can already give you an idea.
Don't get confused here: "hallway" is a string, while hall is a variable that represents the float 11.25 you specified earlier.
Instructions
100 XP
script.py
1 # area variables (in square meters)
2 hall = 11.25
3 kit = 18.0
4 liv = 20.0
5 bed = 10.75
6 bath = 9.50
7
8 # house information as list of lists
9 house = [["hallway", hall],
10 ["kitchen", kit],
11 ["living room", liv]]
12
13 # Print out house
14
Run Code
Submit Answer
IPython Shell
Slides

DataCamp

Course Outline

Exercise

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 flat list containing strings and floats, representing the names and areas of the rooms in your house, you can create a list of lists. The script on the right can already give you an idea.

Don't get confused here: `"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?
- Print out the type of `house` . Are you still dealing with a list?

script.py

```
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)

# Print out the type of house
print(type(house))
```

Run all/selected code Ctrl+Shift+Enter

Run Code Submit Answer

IPython Shell Slides

```
<class 'list'>

In [2]:
```

## Subsetting Lists

50 XP

# Subsetting lists

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1]  # <-
```

```
1.89
```

```
fam[7] # <-
```

```
1.89
```

Apart from indexing, there's also something called slicing,

DataCamp INTRODUCTION TO PYTHON

# List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

```
[1.68, 'mom']
```

[ start : end ]

starts, is included, while the index you specify after the colon, where the slice ends, is not.



# List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

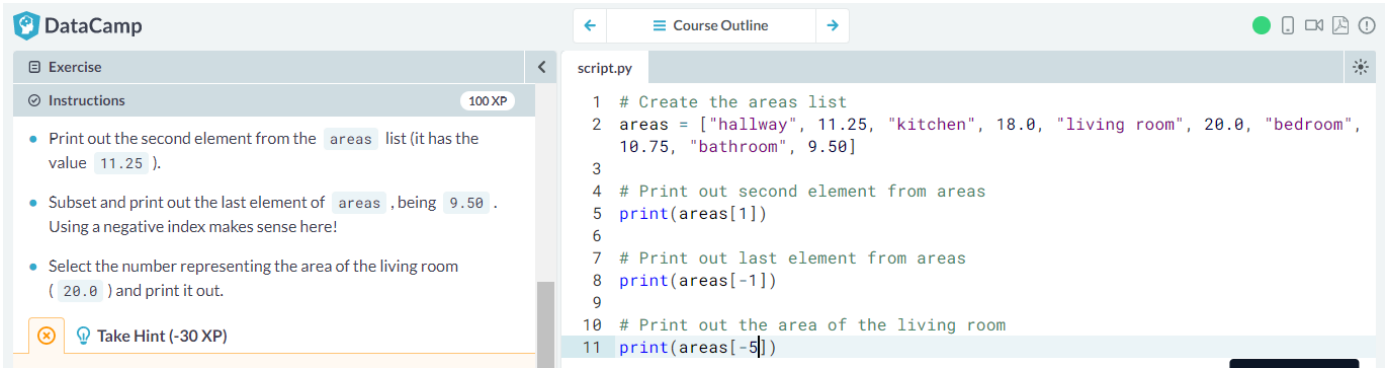
```
[1.68, 'mom']
```

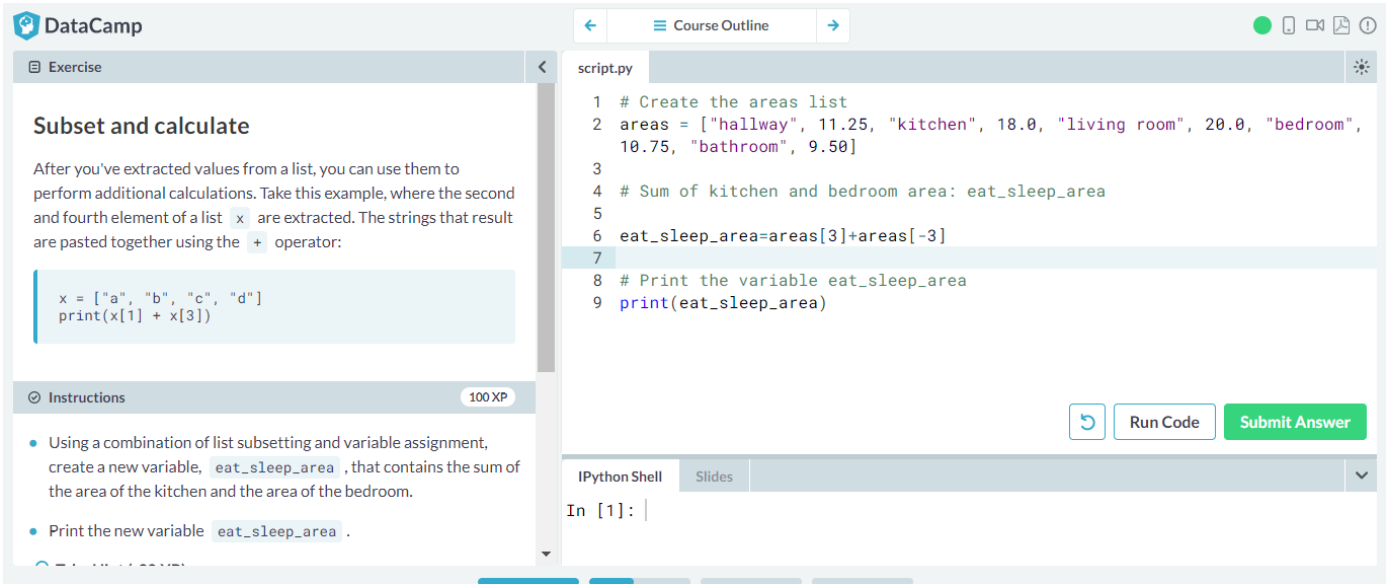
```
fam[1:4]
```

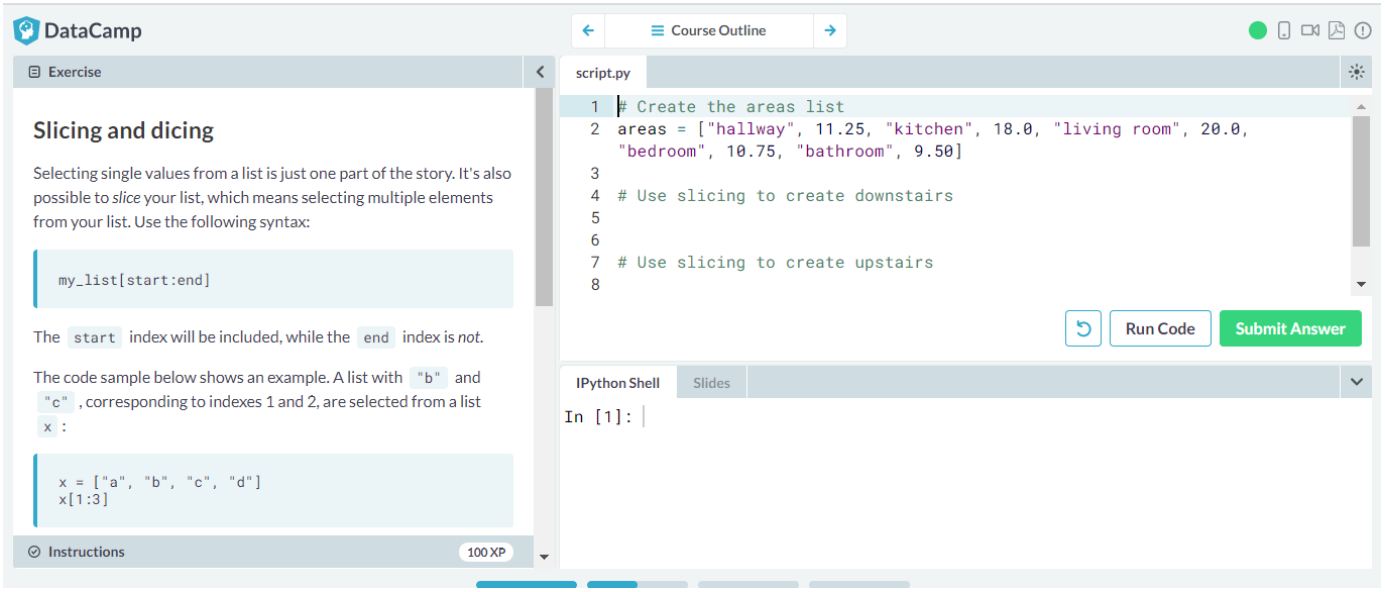
```
[1.73, 'emma', 1.68]
```

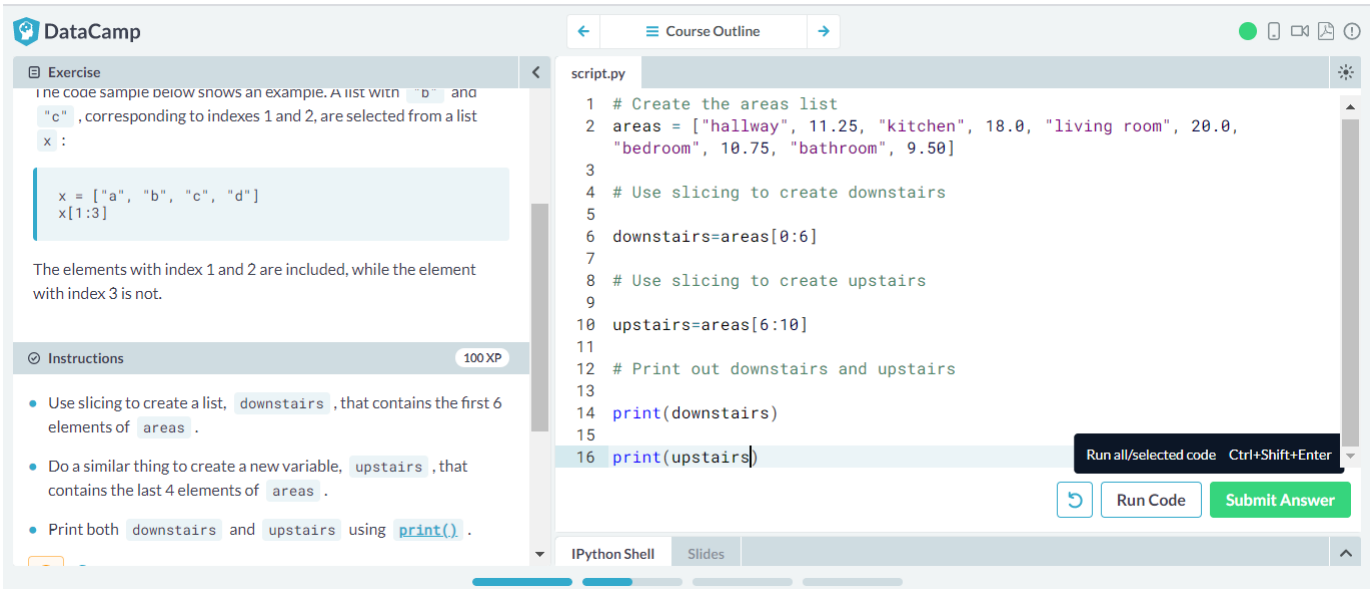
[ start : end ]

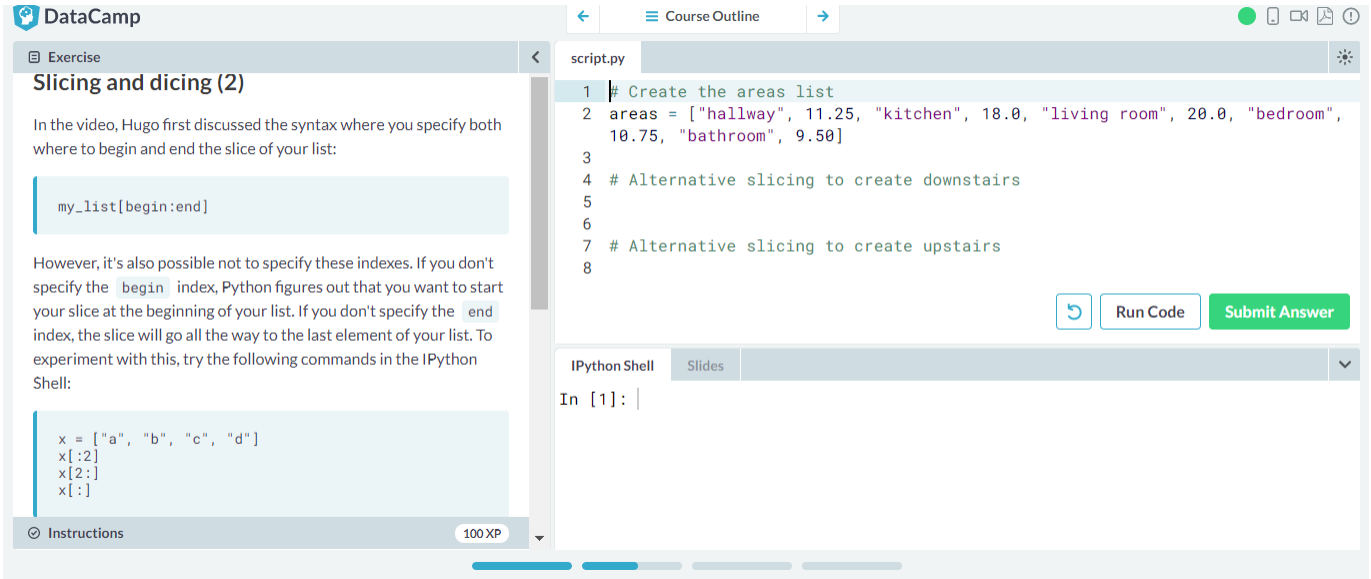
elements, corresponding to the elements with index 1, 2 and 3 of the fam list.

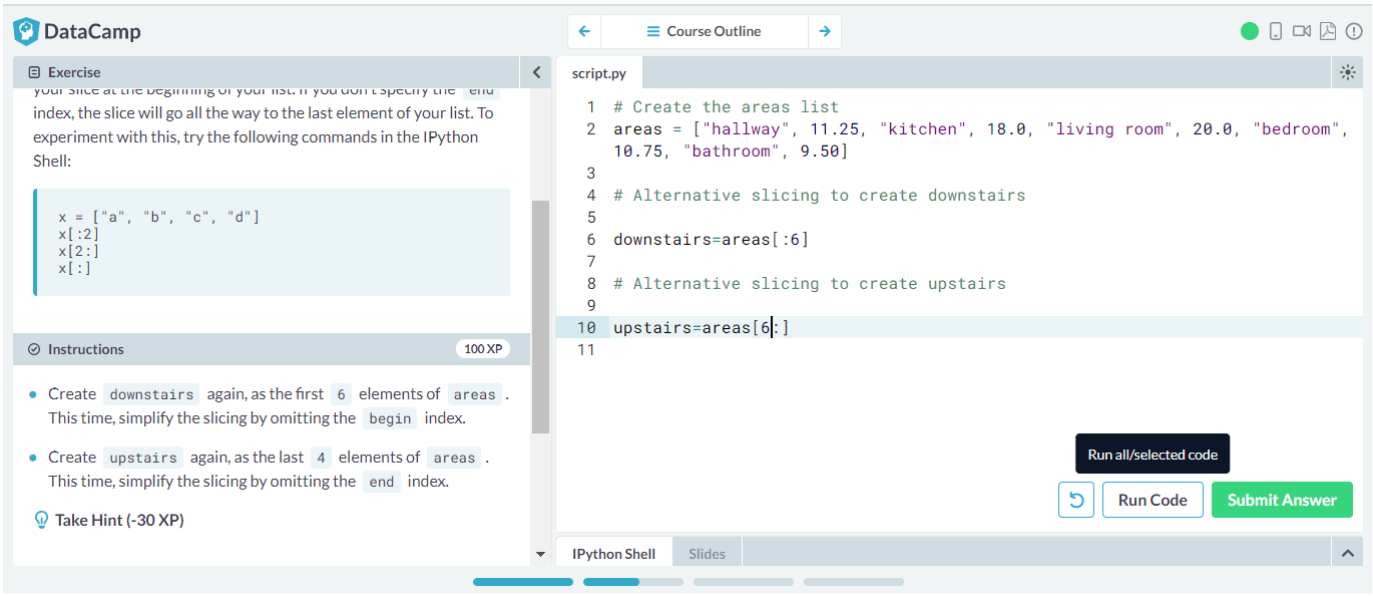

DataCamp
Course Outline
Exercise
Instructions
100 XP
Print out the second element from the areas list (it has the value 11.25 ).
Subset and print out the last element of areas , being 9.50 . Using a negative index makes sense here!
Select the number representing the area of the living room ( 20.0 ) and print it out.
Take Hint (-30 XP)
script.py
# 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[-5])

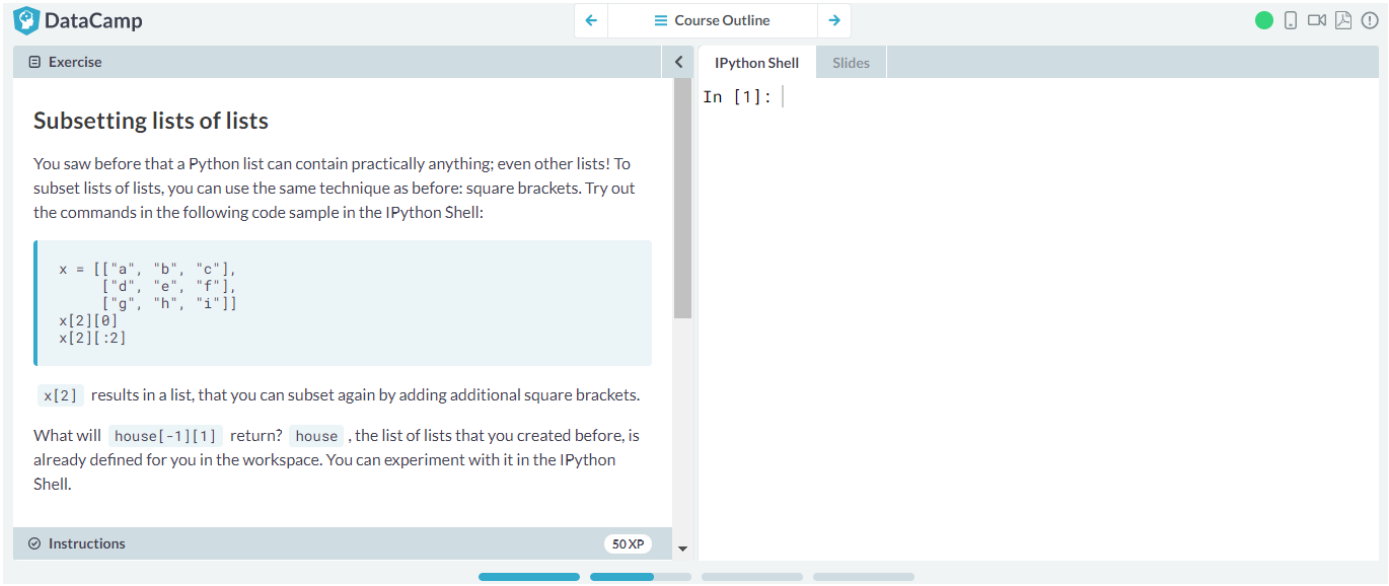

DataCamp
Course Outline
Exercise
Subset and calculate
After you've extracted values from a list, you can use them to perform additional calculations. Take this example, where the second and fourth element of a list x are extracted. The strings that result are pasted together using the + operator:
x = ["a", "b", "c", "d"]
print(x[1] + x[3])
Instructions
100 XP
Using a combination of list subsetting and variable assignment, create a new variable, eat_sleep_area , that contains the sum of the area of the kitchen and the area of the bedroom.
Print the new variable eat_sleep_area .
script.py
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]
# Sum of kitchen and bedroom area: eat_sleep_area
eat_sleep_area=areas[3]+areas[-3]
# Print the variable eat_sleep_area
print(eat_sleep_area)
Run Code
Submit Answer
IPython Shell
Slides
In [1]:

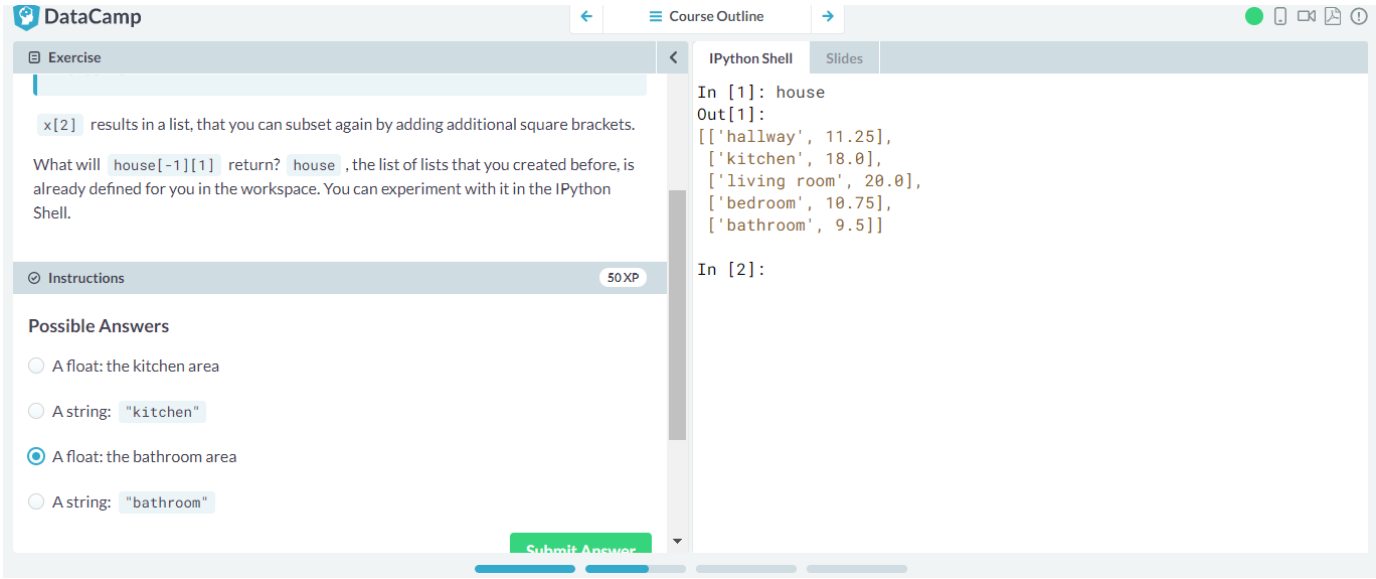

DataCamp
Course Outline
Exercise
Slicing and dicing
Selecting single values from a list is just one part of the story. It's also possible to slice your list, which means selecting multiple elements from your list. Use the following syntax:
my_list[start:end]
The start index will be included, while the end index is not.
The code sample below shows an example. A list with "b" and "c" , corresponding to indexes 1 and 2, are selected from a list x :
x = ["a", "b", "c", "d"]
x[1:3]
Instructions
100 XP
script.py
# 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
# Use slicing to create upstairs
Run Code
Submit Answer
IPython Shell
Slides
In [1]:

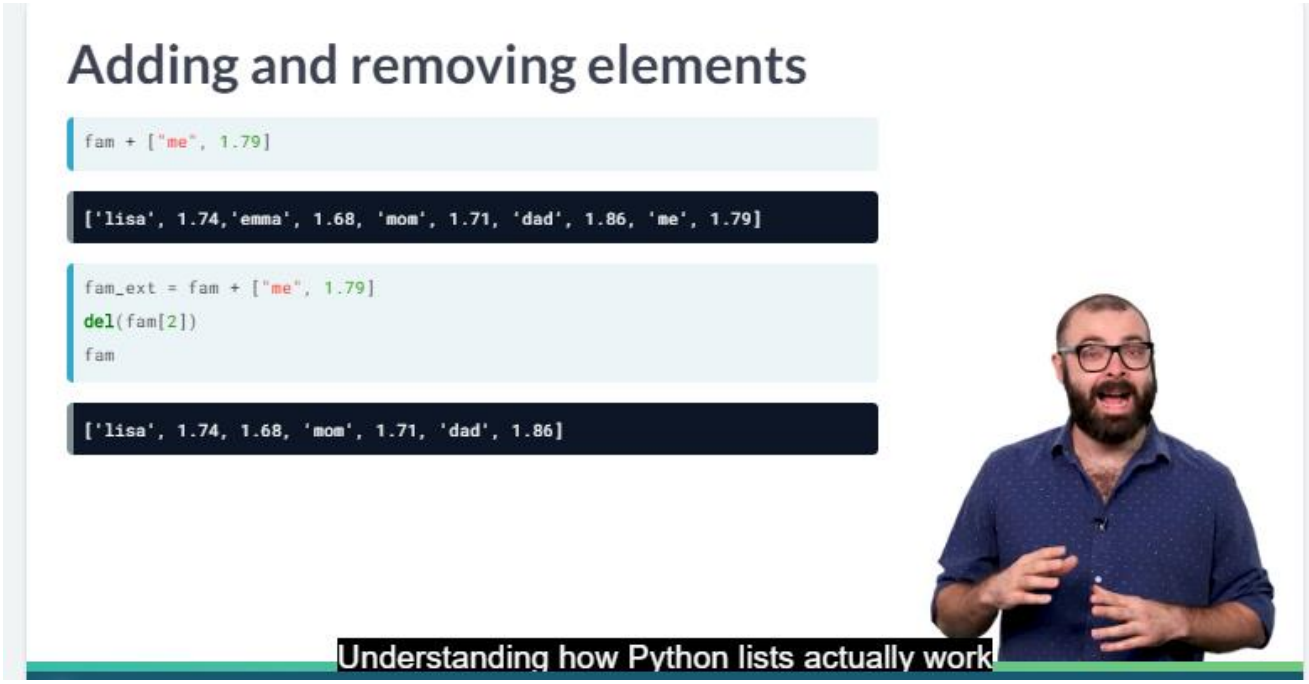

DataCamp
Course Outline
Exercise
The code sample below shows an example. A list with "b" and "c", corresponding to indexes 1 and 2, are selected from a list x:
x = ["a", "b", "c", "d"]
x[1:3]
The elements with index 1 and 2 are included, while the element with index 3 is not.
Instructions
100 XP
Use slicing to create a list, downstairs, that contains the first 6 elements of areas.
Do a similar thing to create a new variable, upstairs, that contains the last 4 elements of areas.
Print both downstairs and upstairs using print().
script.py
# 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[0:6]
# Use slicing to create upstairs
upstairs=areas[6:10]
# Print out downstairs and upstairs
print(downstairs)
print(upstairs)
Run all/selected code Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides


DataCamp
Course Outline
Exercise
Slicing and dicing (2)
In the video, Hugo first discussed the syntax where you specify both where to begin and end the slice of your list:
my_list[begin:end]
However, it's also possible not to specify these indexes. If you don't specify the begin index, Python figures out that you want to start your slice at the beginning of your list. If you don't specify the end index, the slice will go all the way to the last element of your list. To experiment with this, try the following commands in the IPython Shell:
x = ["a", "b", "c", "d"]
x[:2]
x[2:]
x[:]
Instructions
100 XP
script.py
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]
# Alternative slicing to create downstairs
# Alternative slicing to create upstairs
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
index, the slice will go all the way to the last element of your list. To experiment with this, try the following commands in the IPython Shell:
x = ["a", "b", "c", "d"]
x[:2]
x[2:]
x[:]
Instructions
100 XP
Create downstairs again, as the first 6 elements of areas. This time, simplify the slicing by omitting the begin index.
Create upstairs again, as the last 4 elements of areas. This time, simplify the slicing by omitting the end index.
Take Hint (-30 XP)
script.py
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]
# Alternative slicing to create downstairs
downstairs=areas[:6]
# Alternative slicing to create upstairs
upstairs=areas[6:]
Run all/selected code
Run Code
Submit Answer
IPython Shell
Slides

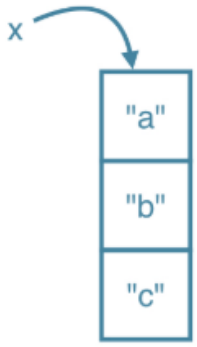

DataCamp
Course Outline
Exercise
Subsetting lists of lists
You saw before that a Python list can contain practically anything; even other lists! To subset lists of lists, you can use the same technique as before: square brackets. Try out the commands in the following code sample in the IPython Shell:
x = [["a", "b", "c"],
["d", "e", "f"],
["g", "h", "i"]]
x[2][0]
x[2][:2]
x[2] results in a list, that you can subset again by adding additional square brackets.
What will house[-1][1] return? house, the list of lists that you created before, is already defined for you in the workspace. You can experiment with it in the IPython Shell.
Instructions
50 XP
IPython Shell
Slides
In [1]:

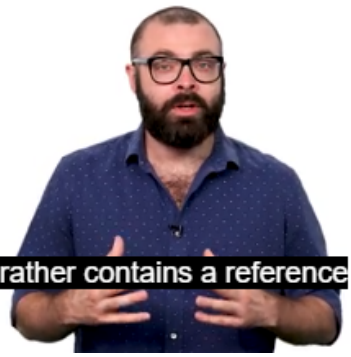

DataCamp
Course Outline
Exercise
IPython Shell
Slides
x[2] results in a list, that you can subset again by adding additional square brackets.
What will house[-1][1] return? house , the list of lists that you created before, is already defined for you in the workspace. You can experiment with it in the IPython Shell.
Instructions
50 XP
Possible Answers
A float: the kitchen area
A string: "kitchen"
A float: the bathroom area
A string: "bathroom"
Submit Answer
In [1]: house
Out[1]:
[['hallway', 11.25],
['kitchen', 18.0],
['living room', 20.0],
['bedroom', 10.75],
['bathroom', 9.5]]
In [2]:


Adding and removing elements
fam + ["me", 1.79]
['lisa', 1.74,'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
fam_ext = fam + ["me", 1.79]
del(fam[2])
fam
['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
Understanding how Python lists actually work

# Behind the scenes (1)

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

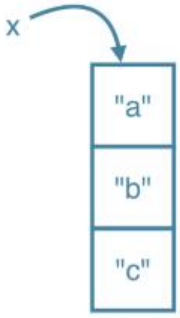


This means that x does not actually contain all the list elements, it rather contains a reference to the list.

# Behind the scenes (1)

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




For basic operations, the difference is not that important, but it becomes more so when you start copying lists.

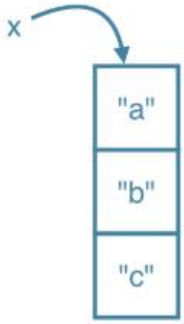
# Behind the scenes (1)

```
x = ["a", "b", "c"]
y = x
y[1] = "z"
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```




That's because when you copied x to y



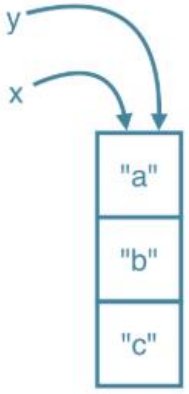
# Behind the scenes (1)

```
x = ["a", "b", "c"]
y = x
y[1] = "z"
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```

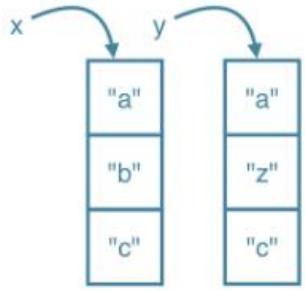



you copied the reference to the list, not the actual values themselves.

# Behind the scenes (2)

```
x = ["a", "b", "c"]
y = list(x)
y = x[:]
y[1] = "z"
x
```

```
['a', 'b', 'c']
```




make a change to the list y points to, x is not affected.

DataCamp -0:16 INTRODUCTION TO PYTHON

DataCamp

Course Outline

Exercise

## Replace list elements

Replacing list elements is pretty easy. Simply subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

Use the IPython Shell to experiment with the commands below. Can you tell what's happening and why?

```
x = ["a", "b", "c", "d"]
x[1] = "r"
x[2:] = ["s", "t"]
```

For this and the following exercises, you'll continue working on the `areas` list that contains the names and areas of different rooms in a house.

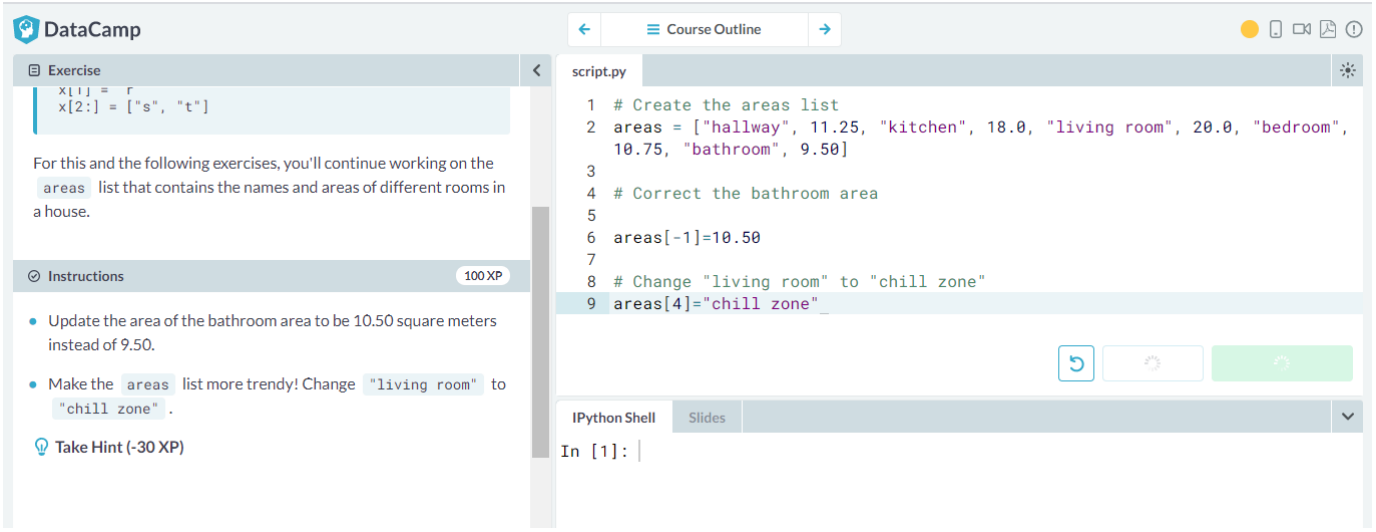
Instructions 100 XP

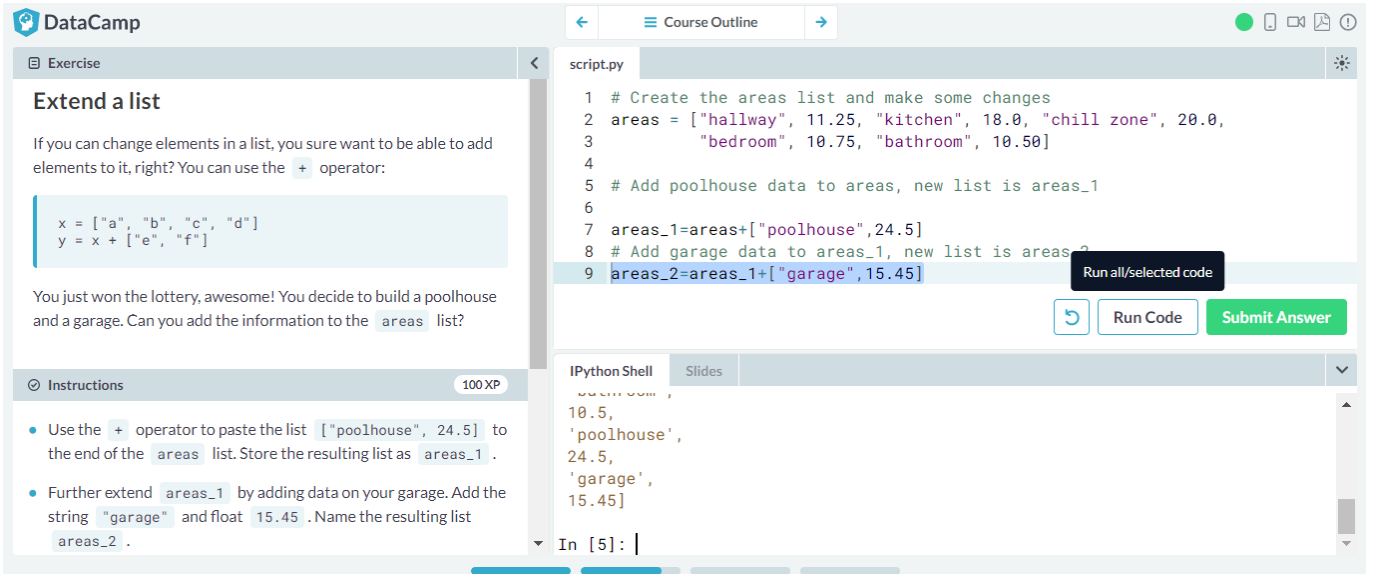
script.py

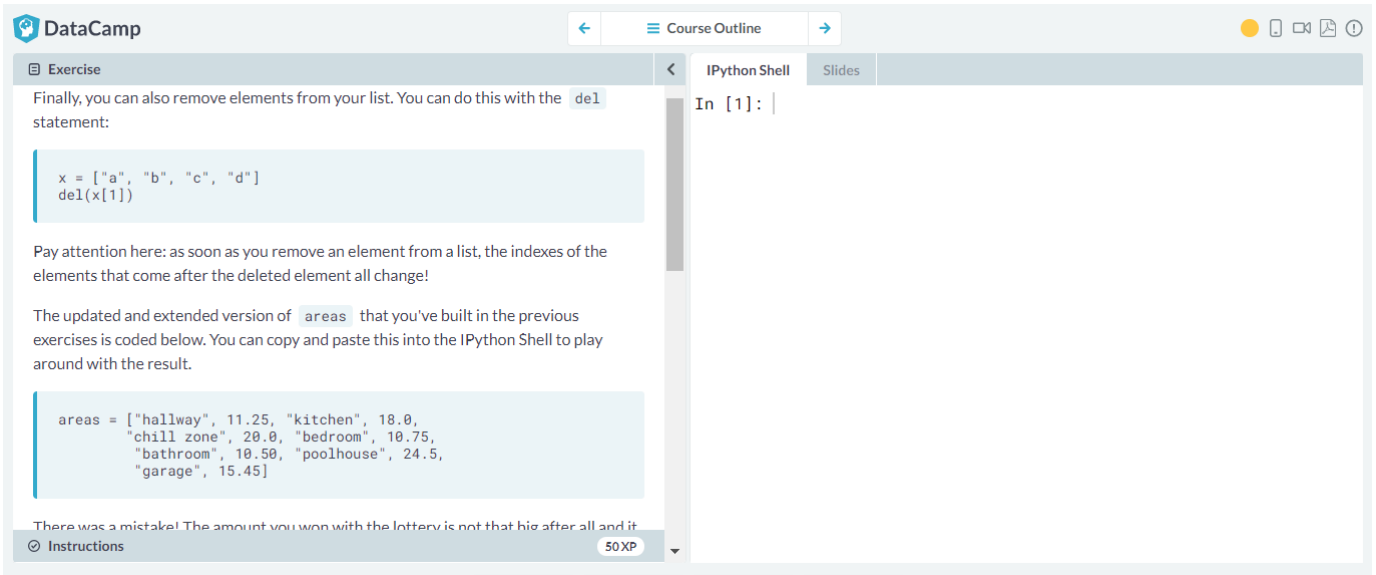
```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom",
  10.75, "bathroom", 9.50]
3
4 # Correct the bathroom area
5
6 _
7 # Change "living room" to "chill zone"
8
```

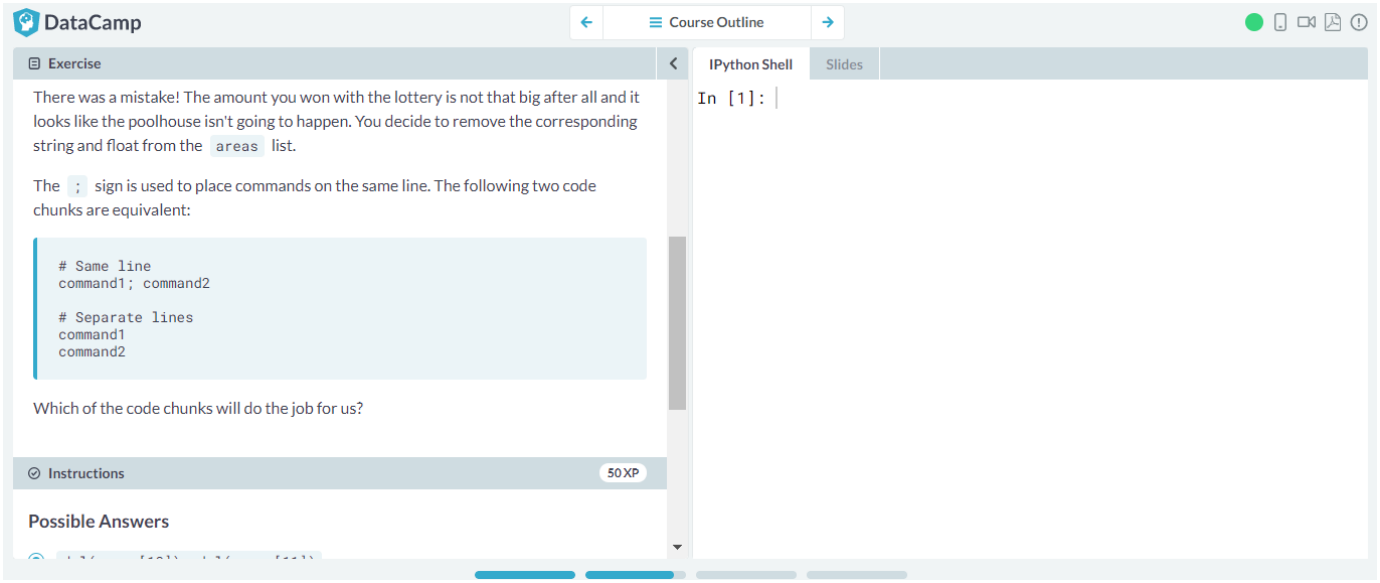
IPython Shell Slides

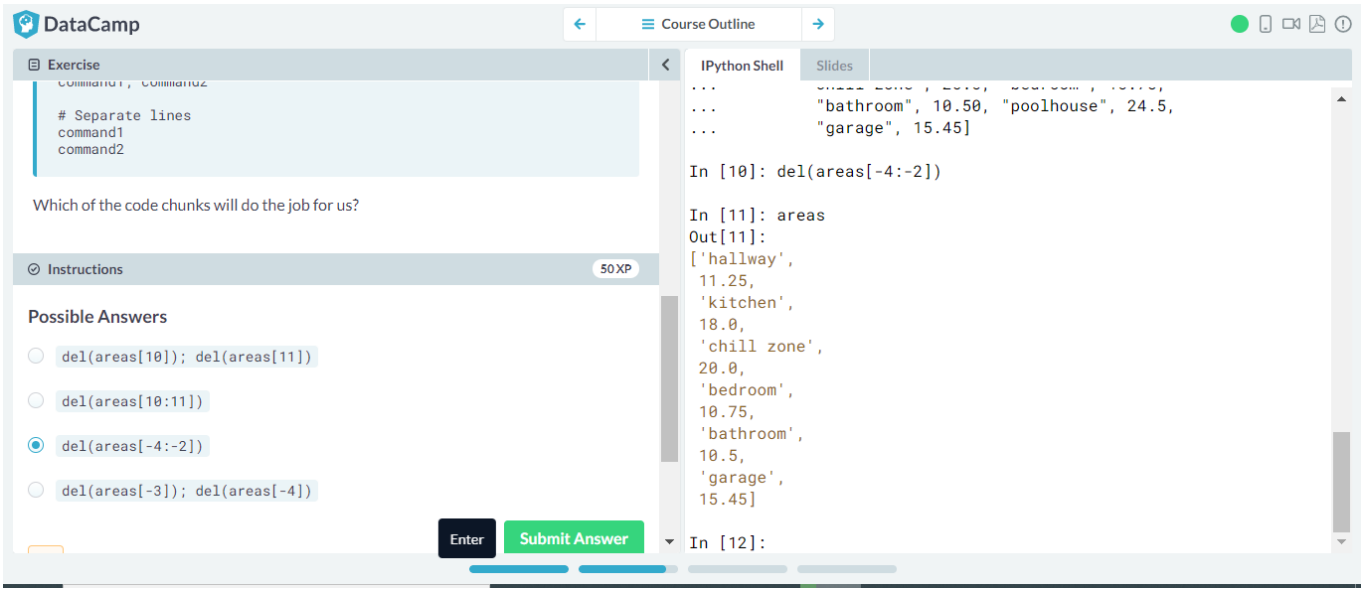
```
In [1]: areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0,
    "bedroom", 10.75, "bathroom", 9.50]

In [2]:
```

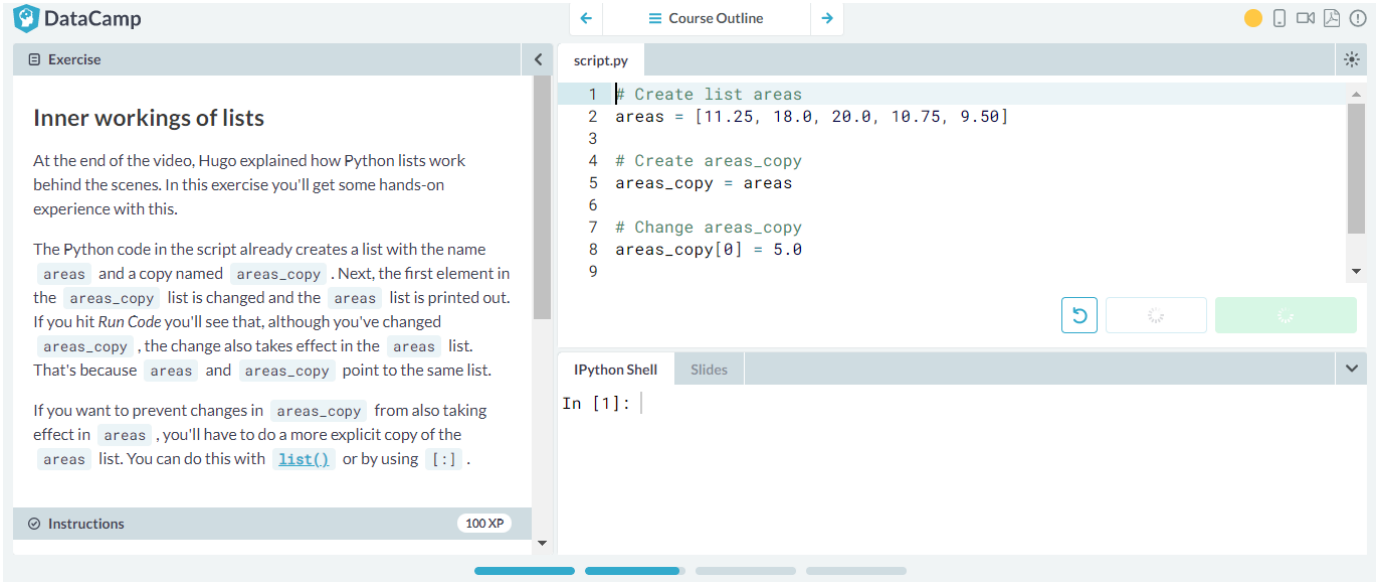

DataCamp
Course Outline
Exercise
x[2:] = ["s", "t"]
For this and the following exercises, you'll continue working on the areas list that contains the names and areas of different rooms in a house.
Instructions
100 XP
Update the area of the bathroom area to be 10.50 square meters instead of 9.50.
Make the areas list more trendy! Change "living room" to "chill zone".
Take Hint (-30 XP)
script.py
# Create the areas list
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]
# Correct the bathroom area
areas[-1]=10.50
# Change "living room" to "chill zone"
areas[4]="chill zone"
IPython Shell
Slides
In [1]:

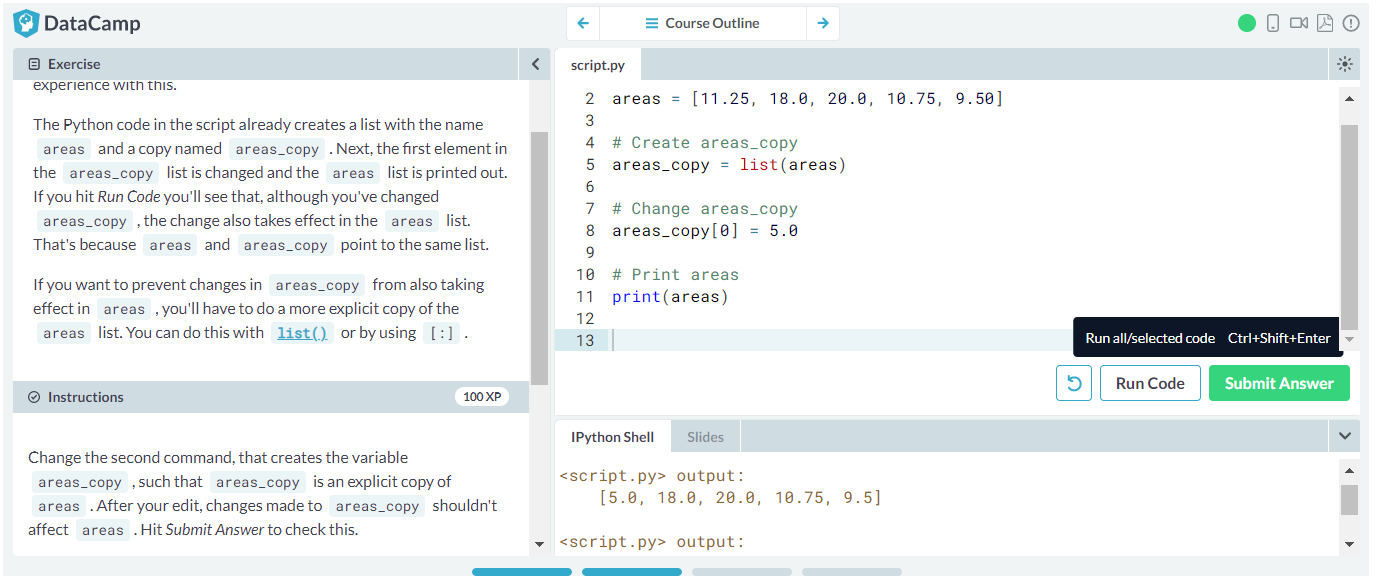

DataCamp
Course Outline
Exercise
Extend a list
If you can change elements in a list, you sure want to be able to add elements to it, right? You can use the + operator:
x = ["a", "b", "c", "d"]
y = x + ["e", "f"]
You just won the lottery, awesome! You decide to build a poolhouse and a garage. Can you add the information to the areas list?
Instructions
100 XP
Use the + operator to paste the list ["poolhouse", 24.5] to the end of the areas list. Store the resulting list as areas_1.
Further extend areas_1 by adding data on your garage. Add the string "garage" and float 15.45. Name the resulting list areas_2.
script.py
# Create the areas list and make some changes
areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0, "bedroom", 10.75, "bathroom", 10.50]
# Add poolhouse data to areas, new list is areas_1
areas_1=areas+["poolhouse",24.5]
# Add garage data to areas_1, new list is areas_2
areas_2=areas_1+["garage",15.45]
Run all/selected code
Run Code
Submit Answer
IPython Shell
Slides
10.5,
'poolhouse',
24.5,
'garage',
15.45]
In [5]:


DataCamp
Course Outline
Exercise
Finally, you can also remove elements from your list. You can do this with the del statement:
x = ["a", "b", "c", "d"]
del(x[1])
Pay attention here: as soon as you remove an element from a list, the indexes of the elements that come after the deleted element all change!
The updated and extended version of areas that you've built in the previous exercises is coded below. You can copy and paste this into the IPython Shell to play around with the result.
areas = ["hallway", 11.25, "kitchen", 18.0,
"chill zone", 20.0, "bedroom", 10.75,
"bathroom", 10.50, "poolhouse", 24.5,
"garage", 15.45]
Instructions
50 XP
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
There was a mistake! The amount you won with the lottery is not that big after all and it looks like the poolhouse isn't going to happen. You decide to remove the corresponding string and float from the areas list.
The ; sign is used to place commands on the same line. The following two code chunks are equivalent:
# Same line
command1; command2
# Separate lines
command1
command2
Which of the code chunks will do the job for us?
Instructions
50 XP
Possible Answers
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
# Separate lines
command1
command2
Which of the code chunks will do the job for us?
Instructions
50 XP
Possible Answers
del(areas[10]); del(areas[11])
del(areas[10:11])
del(areas[-4:-2])
del(areas[-3]); del(areas[-4])
Enter
Submit Answer
IPython Shell
Slides
... "bathroom", 10.50, "poolhouse", 24.5,
... "garage", 15.45]
In [10]: del(areas[-4:-2])
In [11]: areas
Out[11]:
['hallway',
11.25,
'kitchen',
18.0,
'chill zone',
20.0,
'bedroom',
10.75,
'bathroom',
10.5,
'garage',
15.45]
In [12]:


DataCamp
Course Outline
Exercise
Inner workings of lists
At the end of the video, Hugo explained how Python lists work behind the scenes. In this exercise you'll get some hands-on experience with this.
The Python code in the script already creates a list with the name areas and a copy named areas_copy . Next, the first element in the areas_copy list is changed and the areas list is printed out. If you hit Run Code you'll see that, although you've changed areas_copy , the change also takes effect in the areas list. That's because areas and areas_copy point to the same list.
If you want to prevent changes in areas_copy from also taking effect in areas , you'll have to do a more explicit copy of the areas list. You can do this with list() or by using [:] .
Instructions
100 XP
script.py
1 # Create list areas
2 areas = [11.25, 18.0, 20.0, 10.75, 9.50]
3
4 # Create areas_copy
5 areas_copy = areas
6
7 # Change areas_copy
8 areas_copy[0] = 5.0
9
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
experience with this.
The Python code in the script already creates a list with the name areas and a copy named areas_copy . Next, the first element in the areas_copy list is changed and the areas list is printed out. If you hit Run Code you'll see that, although you've changed areas_copy , the change also takes effect in the areas list. That's because areas and areas_copy point to the same list.
If you want to prevent changes in areas_copy from also taking effect in areas , you'll have to do a more explicit copy of the areas list. You can do this with list() or by using [:] .
Instructions
100 XP
Change the second command, that creates the variable areas_copy , such that areas_copy is an explicit copy of areas . After your edit, changes made to areas_copy shouldn't affect areas . Hit Submit Answer to check this.
script.py
2 areas = [11.25, 18.0, 20.0, 10.75, 9.50]
3
4 # Create areas_copy
5 areas_copy = list(areas)
6
7 # Change areas_copy
8 areas_copy[0] = 5.0
9
10 # Print areas
11 print(areas)
12
13
Run all/selected code Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
<script.py> output:
[5.0, 18.0, 20.0, 10.75, 9.5]
<script.py> output:

# Functions

- Nothing new!
- type()

type, for example, is a function that returns the type of a value.



# Functions

- Nothing new!
- type()
- Piece of reusable code

Simply put, a function is a piece of reusable code, aimed at solving a particular task.

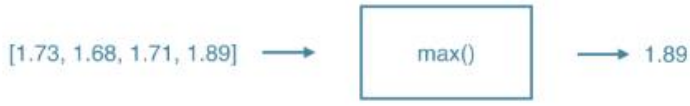
# Example

```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

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

```
max(fam)
```

```
1.89
```




and produced an output.



# round()

```
round(1.68, 1)
```

```
1.7
```

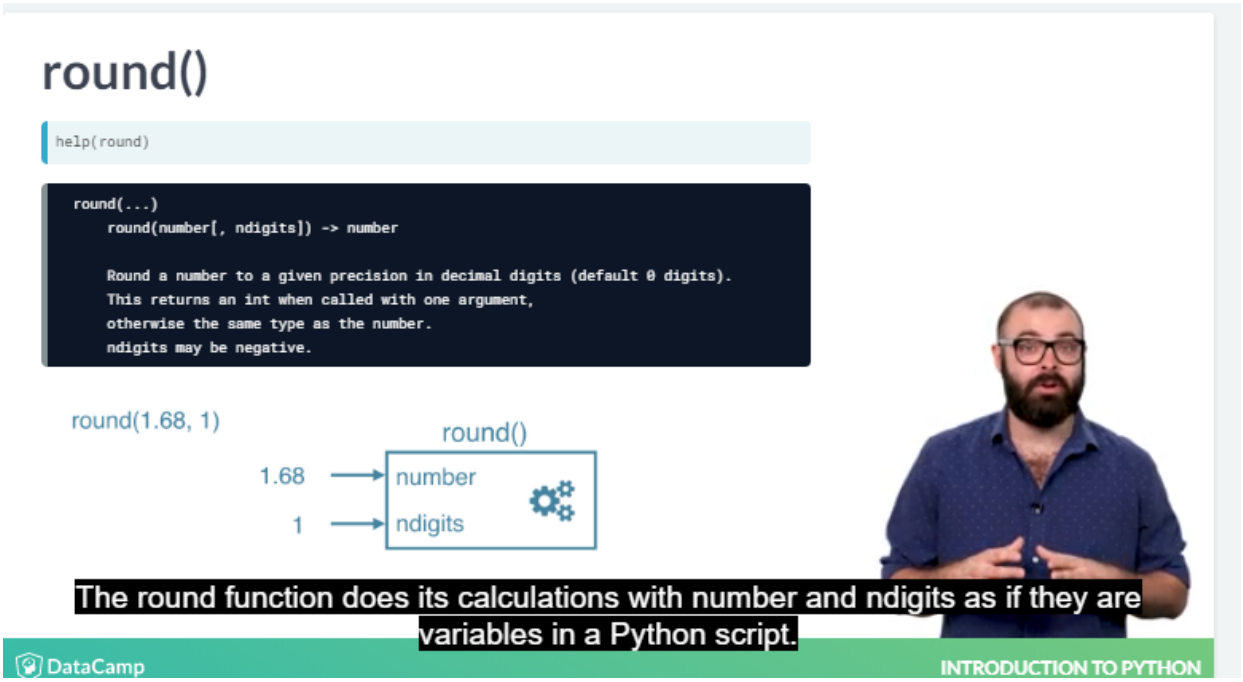
```
round(1.68)
```

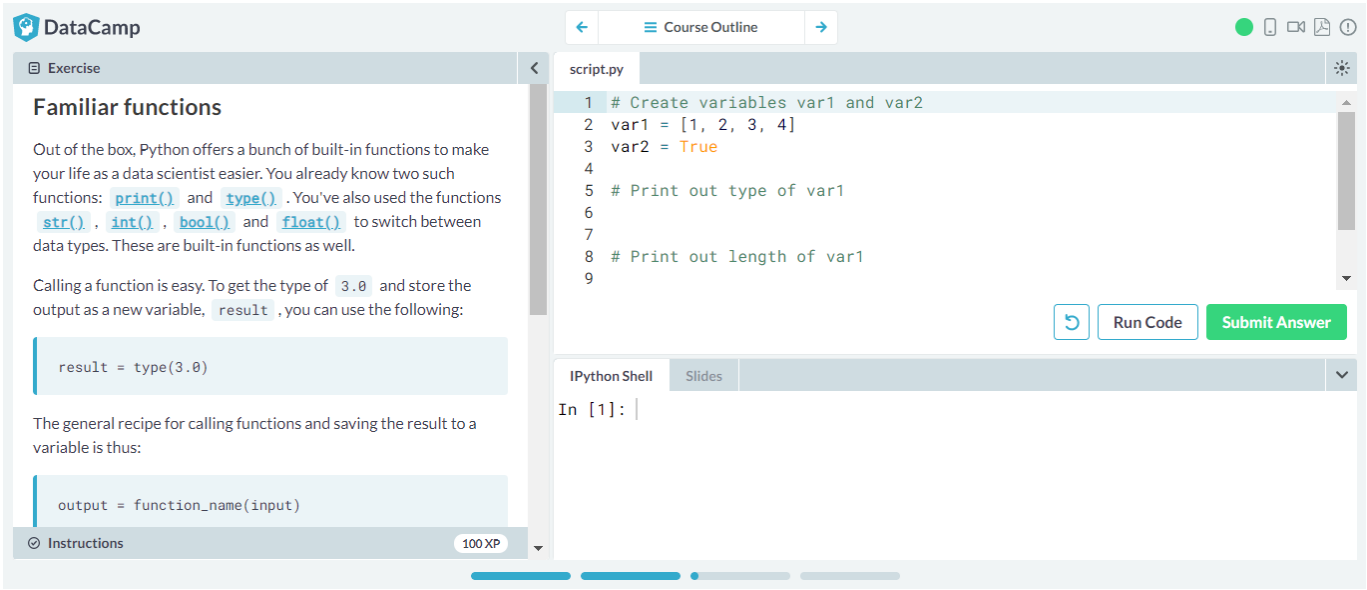
```
2
```

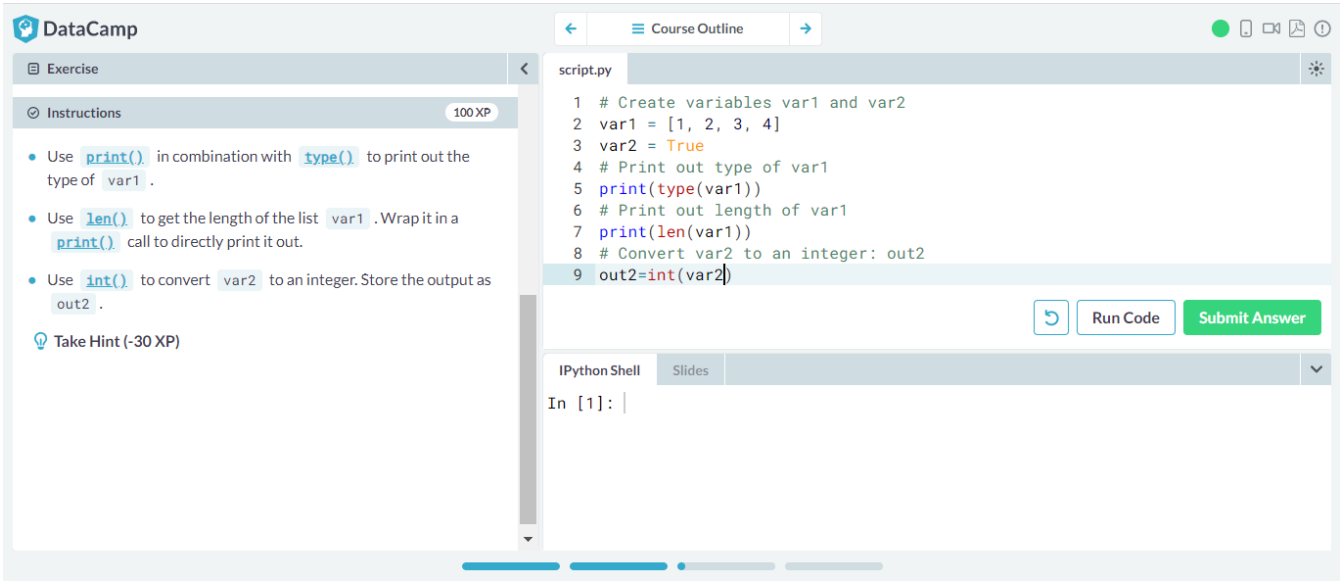
```
help(round) # Open up documentation
```

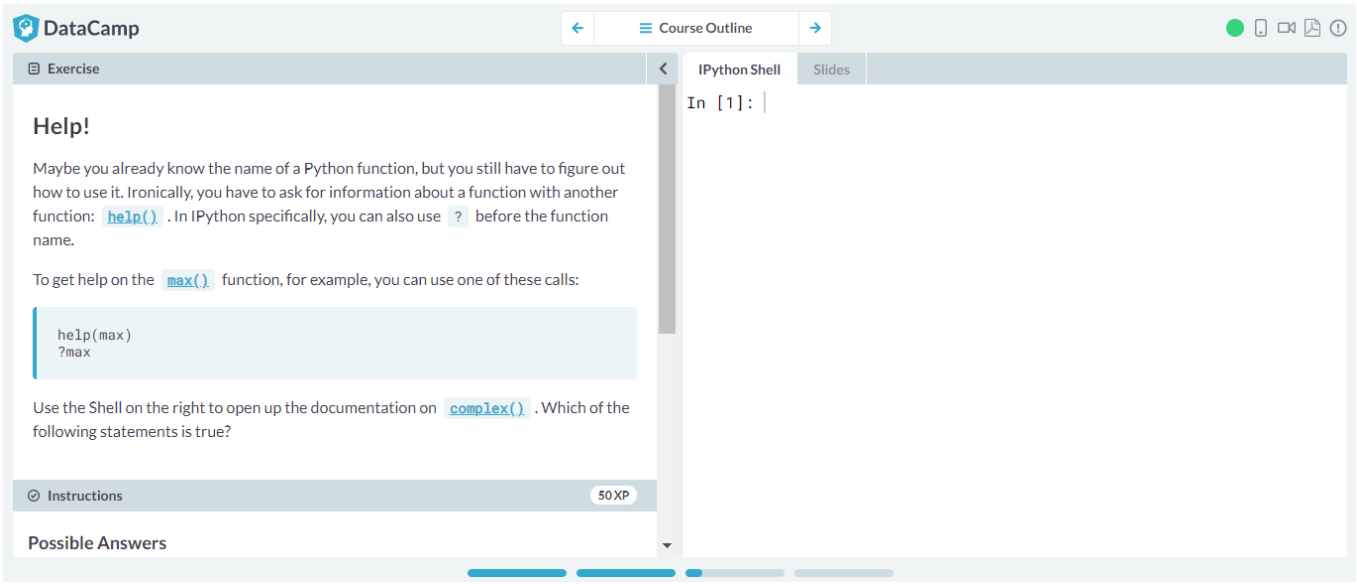
```
round(...)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument,
    otherwise the same type as the number.
    ndigits may be negative.
```

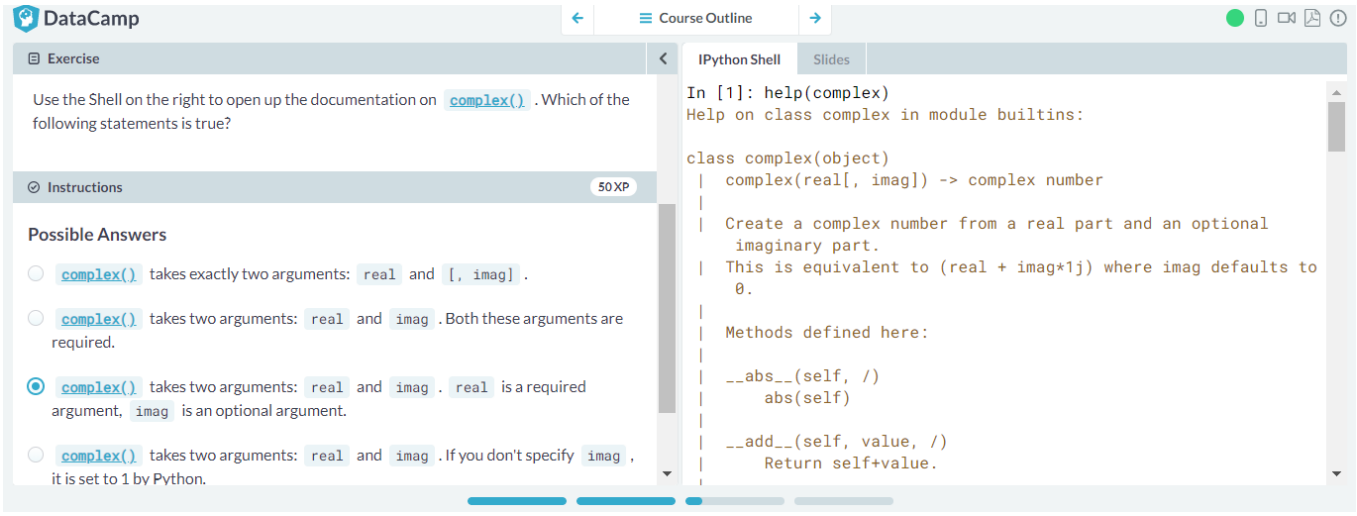
It appears that round takes two inputs.

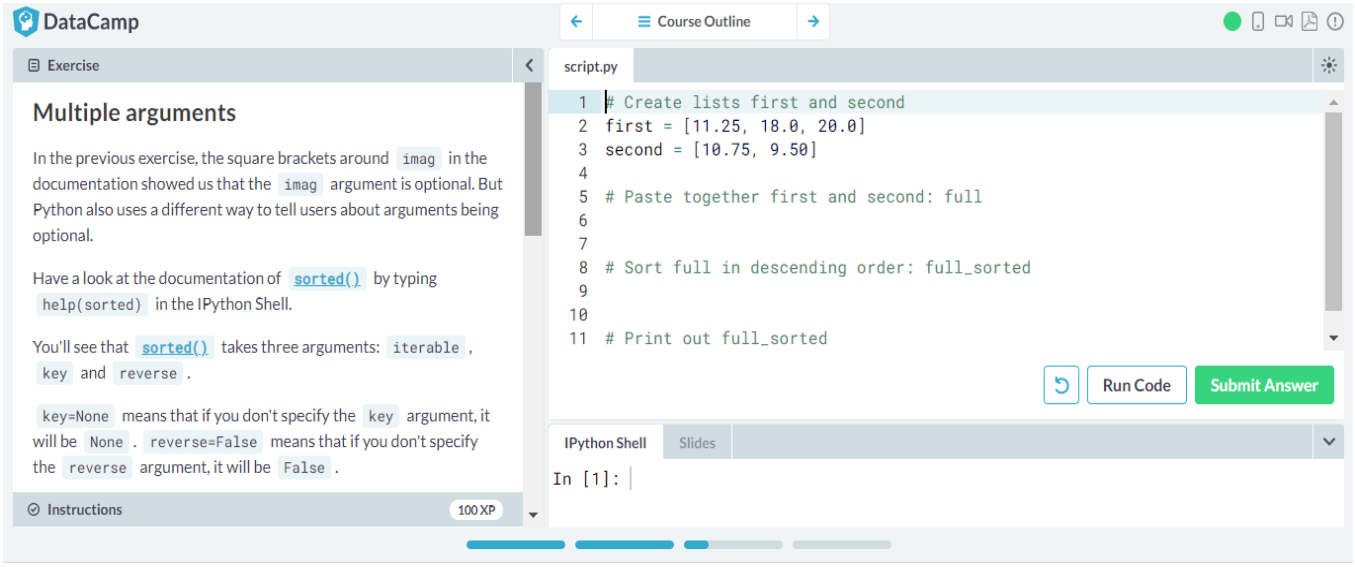

round()
help(round)
round(...)
round(number[, ndigits]) -> number
Round a number to a given precision in decimal digits (default 0 digits).
This returns an int when called with one argument,
otherwise the same type as the number.
ndigits may be negative.
round(1.68, 1)
round()
1.68
number
1
ndigits
The round function does its calculations with number and ndigits as if they are variables in a Python script.
DataCamp
INTRODUCTION TO PYTHON

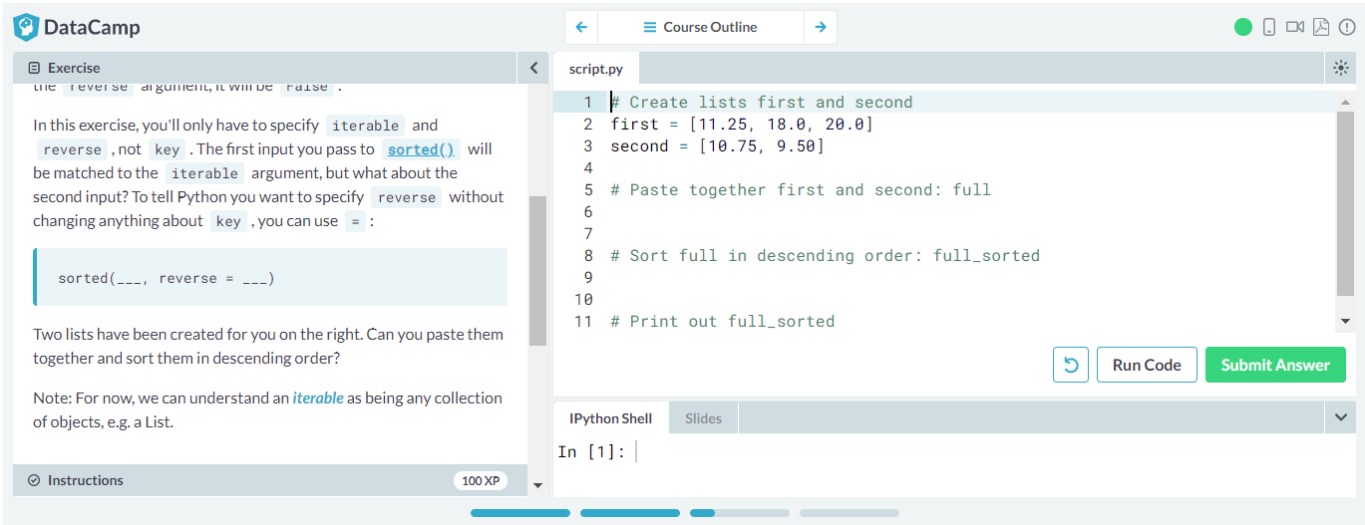

DataCamp
Course Outline
Exercise
Familiar functions
Out of the box, Python offers a bunch of built-in functions to make your life as a data scientist easier. You already know two such functions: print() and type() . You've also used the functions str() , int() , bool() and float() to switch between data types. These are built-in functions as well.
Calling a function is easy. To get the type of 3.0 and store the output as a new variable, result , you can use the following:
result = type(3.0)
The general recipe for calling functions and saving the result to a variable is thus:
output = function_name(input)
Instructions
100 XP
script.py
1 # Create variables var1 and var2
2 var1 = [1, 2, 3, 4]
3 var2 = True
4
5 # Print out type of var1
6
7
8 # Print out length of var1
9
Run Code
Submit Answer
IPython Shell
Slides
In [1]:

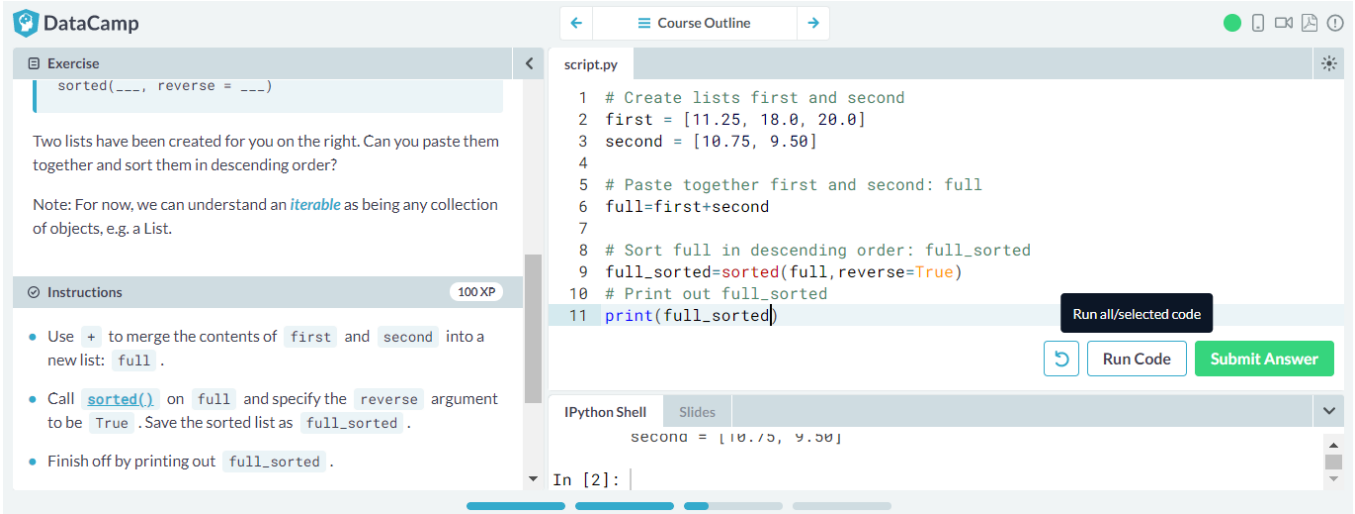

DataCamp
Course Outline
Exercise
Instructions
100 XP
Use print() in combination with type() to print out the type of var1 .
Use len() to get the length of the list var1 . Wrap it in a print() call to directly print it out.
Use int() to convert var2 to an integer. Store the output as out2 .
Take Hint (-30 XP)
script.py
1 # Create variables var1 and var2
2 var1 = [1, 2, 3, 4]
3 var2 = True
4 # Print out type of var1
5 print(type(var1))
6 # Print out length of var1
7 print(len(var1))
8 # Convert var2 to an integer: out2
9 out2=int(var2)
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
IPython Shell
Slides
In [1]:
Help!
Maybe you already know the name of a Python function, but you still have to figure out how to use it. Ironically, you have to ask for information about a function with another function: help() . In IPython specifically, you can also use ? before the function name.
To get help on the max() function, for example, you can use one of these calls:
help(max)
?max
Use the Shell on the right to open up the documentation on complex() . Which of the following statements is true?
Instructions
50 XP
Possible Answers


DataCamp
Course Outline
Exercise
IPython Shell
Slides
Use the Shell on the right to open up the documentation on complex() . Which of the following statements is true?
Instructions
50 XP
Possible Answers
complex() takes exactly two arguments: real and [, imag] .
complex() takes two arguments: real and imag . Both these arguments are required.
complex() takes two arguments: real and imag . real is a required argument, imag is an optional argument.
complex() takes two arguments: real and imag . If you don't specify imag , it is set to 1 by Python.
In [1]: help(complex)
Help on class complex in module builtins:
class complex(object)
| complex(real[, imag]) -> complex number
| Create a complex number from a real part and an optional imaginary part.
| This is equivalent to (real + imag*1j) where imag defaults to 0.
| Methods defined here:
| __abs__(self, /)
| abs(self)
| __add__(self, value, /)
| Return self+value.


DataCamp
Course Outline
Exercise
Multiple arguments
In the previous exercise, the square brackets around imag in the documentation showed us that the imag argument is optional. But Python also uses a different way to tell users about arguments being optional.
Have a look at the documentation of sorted() by typing help(sorted) in the IPython Shell.
You'll see that sorted() takes three arguments: iterable , key and reverse .
key=None means that if you don't specify the key argument, it will be None . reverse=False means that if you don't specify the reverse argument, it will be False .
Instructions
100 XP
script.py
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]
# Paste together first and second: full
# Sort full in descending order: full_sorted
# Print out full_sorted
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
In this exercise, you'll only have to specify iterable and reverse , not key . The first input you pass to sorted() will be matched to the iterable argument, but what about the second input? To tell Python you want to specify reverse without changing anything about key , you can use = :
sorted(___, reverse = ___)
Two lists have been created for you on the right. Can you paste them together and sort them in descending order?
Note: For now, we can understand an iterable as being any collection of objects, e.g. a List.
Instructions
100 XP
script.py
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]
# Paste together first and second: full
# Sort full in descending order: full_sorted
# Print out full_sorted
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
sorted(___, reverse = ___)
Two lists have been created for you on the right. Can you paste them together and sort them in descending order?
Note: For now, we can understand an iterable as being any collection of objects, e.g. a List.
Instructions
100 XP
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 .
script.py
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]
# 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)
Run all/selected code
Run Code
Submit Answer
IPython Shell
Slides
In [2]:

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

The only input is the string "mom", the element you want to get the index for.



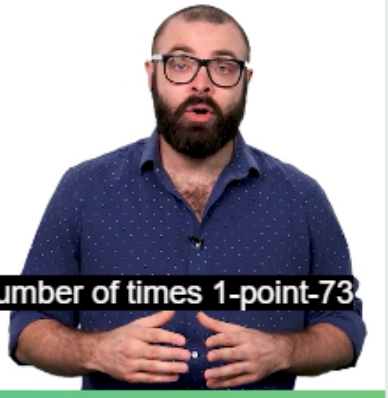
```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

```
fam.count(1.73)
```



Similarly, I can use the count method on the fam list to count the number of times 1-point-73 occurs in the list.

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

It returns a string where the first letter is capitalized now.



```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

In the output, "z" is replaced with "sa".

- Everything = object
- Object have methods associated, depending on type

```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

A string object like sister has a replace method, but a list like fam doesn't have this, as you can see from this error.



# Methods

```
sister.index("z")
```

```
2
```

```
fam.index("mom")
```

```
4
```

This means that, depending on the type of the object, the methods behave differently.

# Methods (2)

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.append("me")
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']
```

list again, we see that it has been extended with the string "me".

DataCamp

Course Outline

Exercise

## String Methods

Strings come with a bunch of methods. Follow the instructions closely to discover some of them. If you want to discover them in more detail, you can always type `help(str)` in the IPython Shell.

A string `place` has already been created for you to experiment with.

Instructions 100 XP

- Use the `upper()` method on `place` and store the result in `place_up`. Use the syntax for calling methods that you learned in the previous video.
- Print out `place` and `place_up`. Did both change?
- Print out the number of o's on the variable `place` by calling `count()` on `place` and passing the letter `'o'` as an input to the method. We're talking about the variable `place`, not the word `"place"`!

script.py

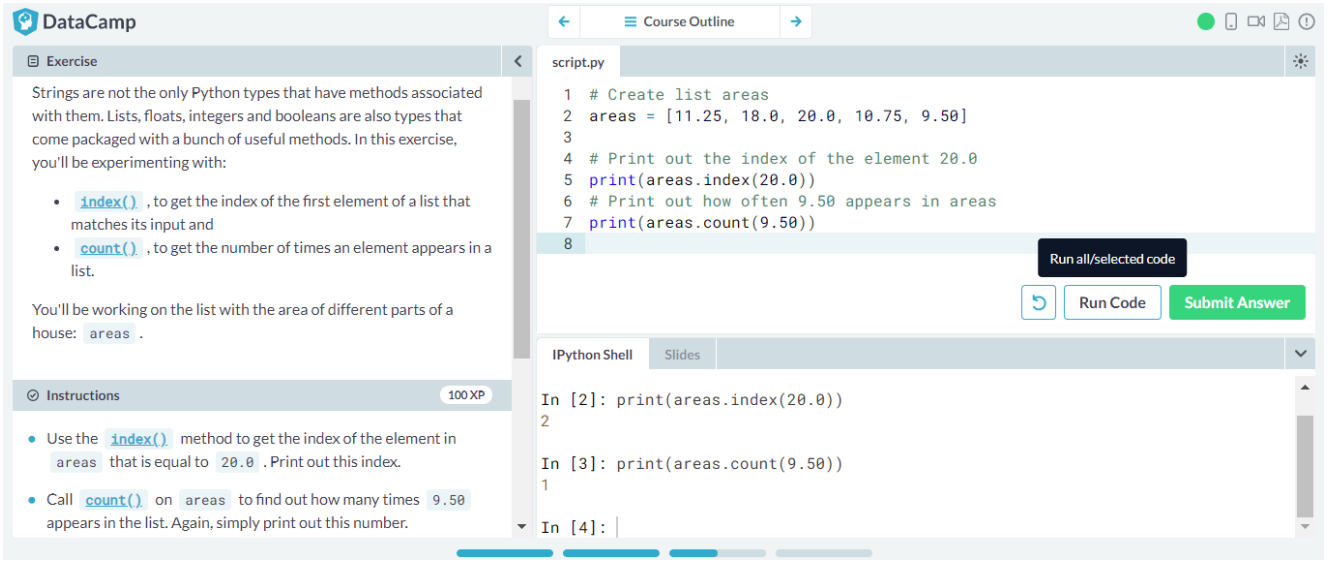
```
1 # string to experiment with: place
2 place = "poolhouse"
3 # Use upper() on place: place_up
4 place_up=place.upper()
5 # Print out place and place_up
6 print(place)
7 print(place_up)
8 # Print out the number of o's in place
9 print(place.count("o"))
```

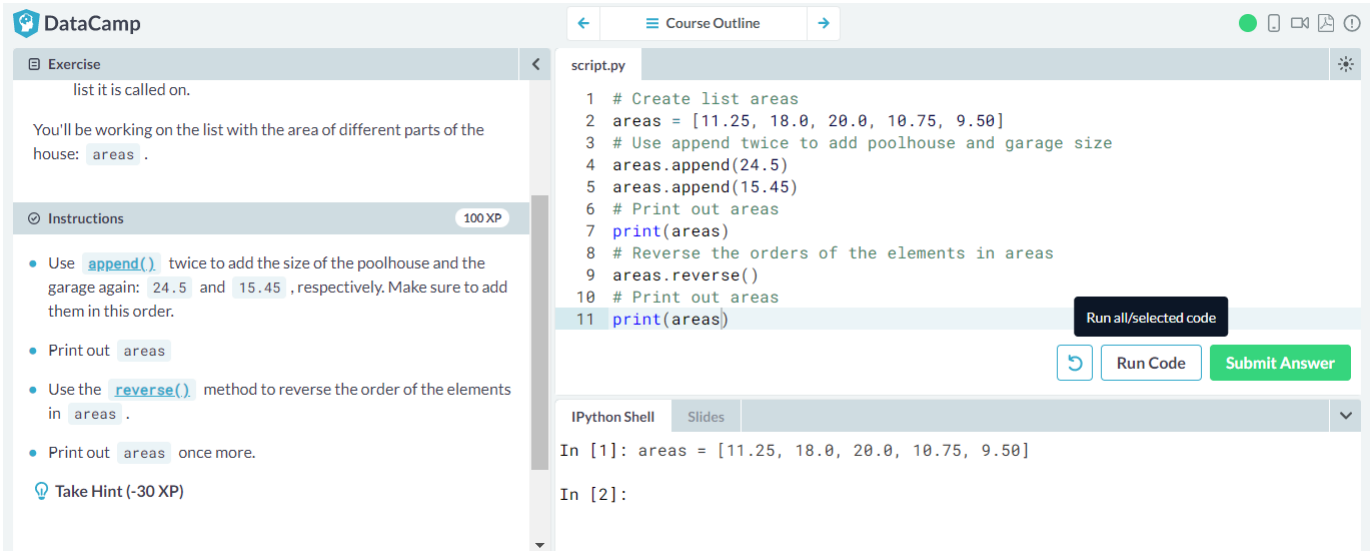
Run all/selected code

Run Code | Submit Answer

IPython Shell | Slides

```
poolhouse
POOLHOUSE

In [12]: print(place.count("o"))
3

In [13]:
```


DataCamp
Course Outline
Exercise
Strings are not 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
100 XP
Use the index() method to get the index of the element in areas that is equal to 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.
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))
Run all/selected code
Run Code
Submit Answer
IPython Shell
Slides
In [2]: print(areas.index(20.0))
2
In [3]: print(areas.count(9.50))
1
In [4]:


DataCamp
Course Outline
Exercise
list it is called on.
You'll be working on the list with the area of different parts of the house: areas .
Instructions
100 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 out areas
Use the reverse() method to reverse the order of the elements in areas .
Print out areas once more.
Take Hint (-30 XP)
script.py
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]
# Use 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)
Run all/selected code
Run Code
Submit Answer
IPython Shell
Slides
In [1]: areas = [11.25, 18.0, 20.0, 10.75, 9.50]
In [2]:

# Packages

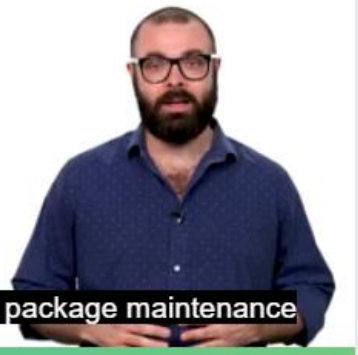
- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
  - Numpy
  - Matplotlib
  - Scikit-learn

```
pkg/
   mod1.py
   mod2.py
   ...
```

numpy to efficiently work with arrays, matplotlib for data visualization, and scikit-learn for machine learning.



# Install package



install them on your own system, you'll want to use pip, a package maintenance system for Python.

# Import package

```
import numpy
array([1, 2, 3])
```

```
NameError: name 'array' is not defined
```

```
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

The Numpy array is very useful to do data science, but more on that later.



# Import package

```
import numpy
array([1, 2, 3])
```

```
import numpy as np
np.array([1, 2, 3])
```

```
NameError: name 'array' is not defined
```

```
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

Now, instead of numpy-dot-array, you'll have to use np-dot-array to use Numpy's array function.

```
import numpy
array([1, 2, 3])
```

```
NameError: name 'array' is not defined
```

```
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
import numpy as np
np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
from numpy import array
array([1, 2, 3])
```

```
array([1, 2, 3])
```

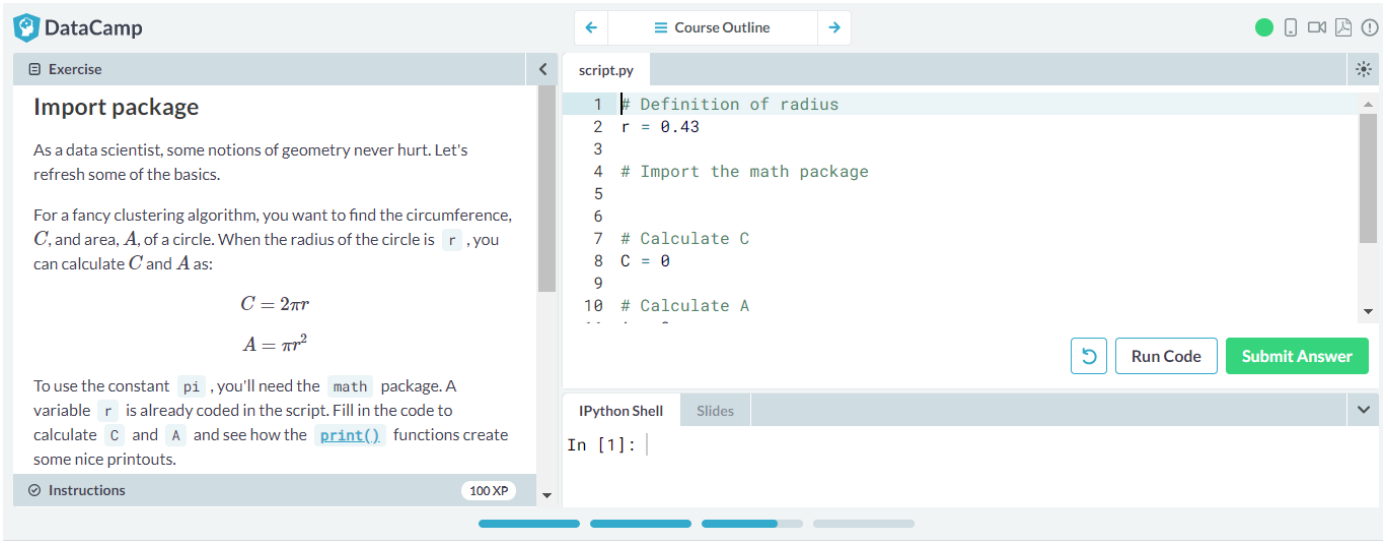
This time, you can simply call the array function like this, no need to use numpy dot here.

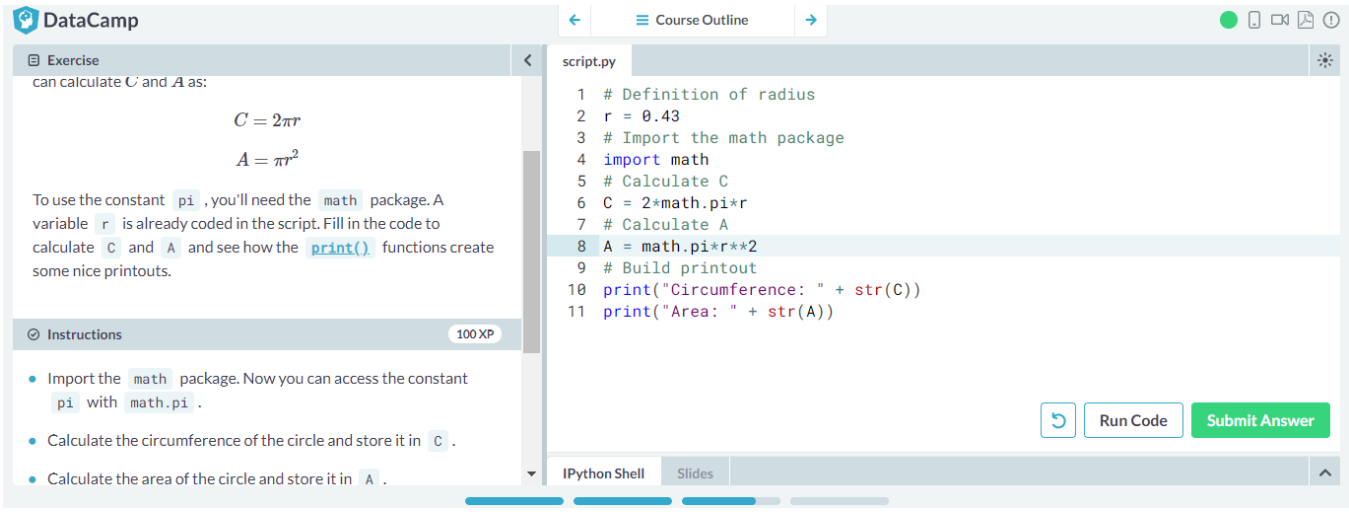


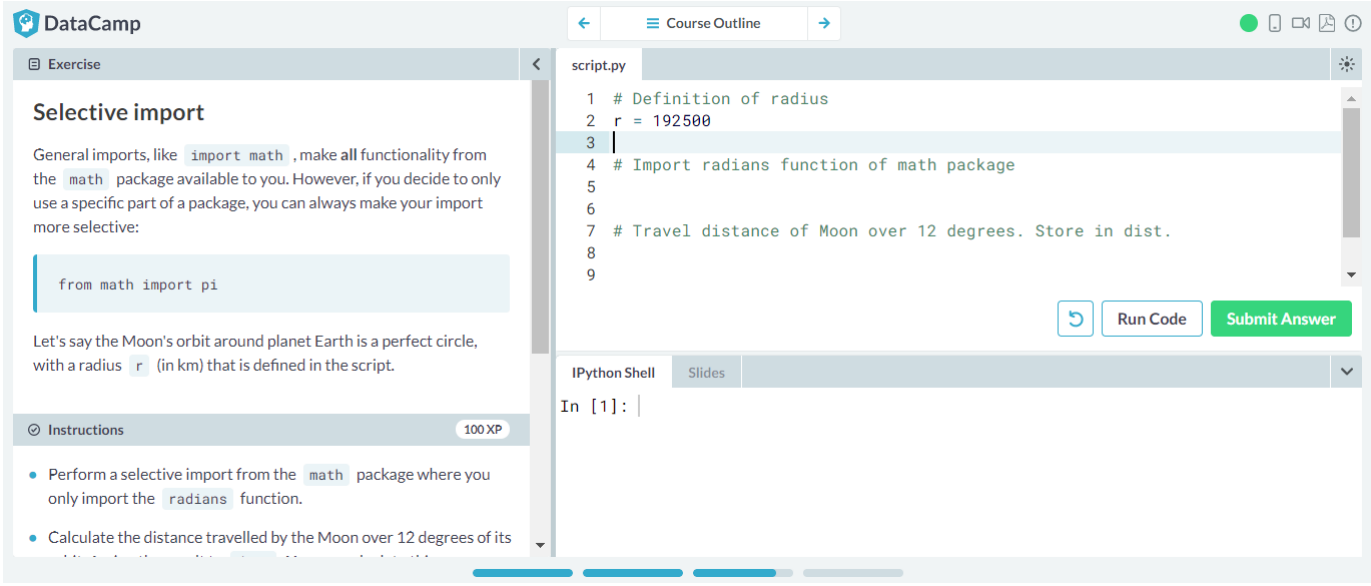
# import numpy

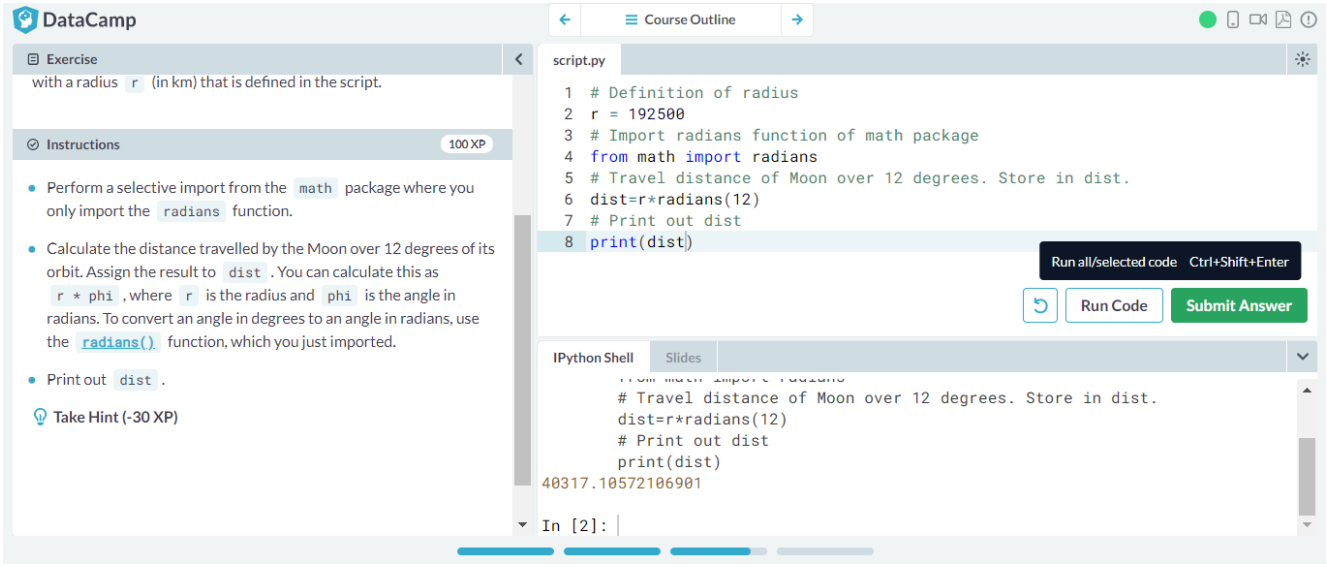
```
import numpy as np

fam = ["liz", 1.73, "emma", 1.68,
    "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = np.array(fam_ext) # Clearly using Numpy
```

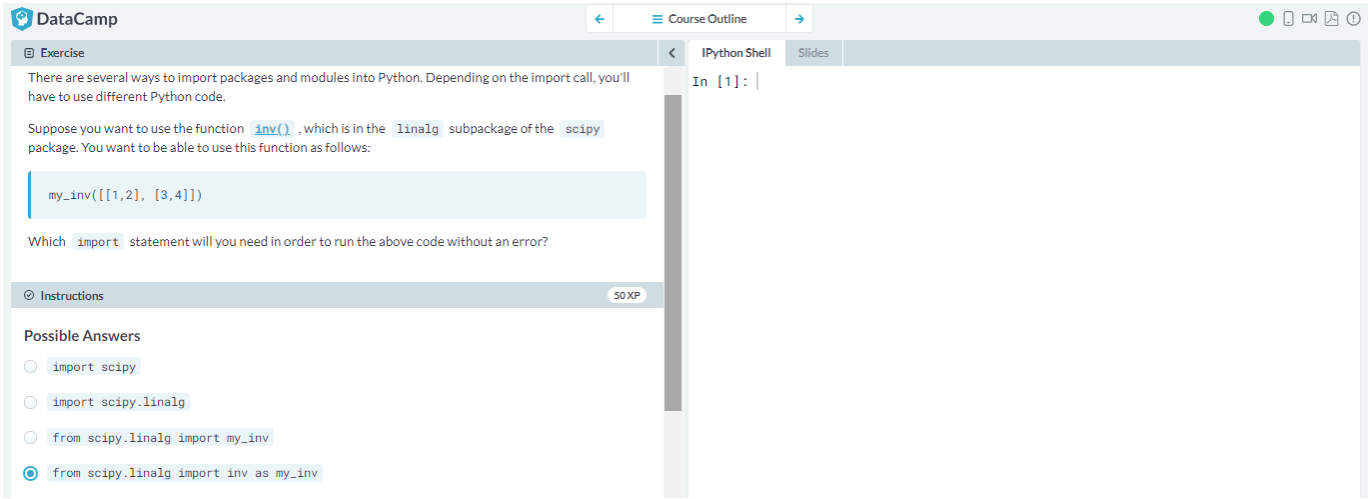
your function call is numpy-dot-array, making it very clear that you're working with Numpy.


DataCamp
Course Outline
Exercise
Import package
As a data scientist, some notions of geometry never hurt. Let's refresh some of the basics.
For a fancy clustering algorithm, you want to find the circumference, $C$, and area, $A$, of a circle. When the radius of the circle is r, you can calculate $C$ and $A$ as:
$$C = 2\pi r$$
$$A = \pi r^2$$
To use the constant pi, you'll need the math package. A variable r is already coded in the script. Fill in the code to calculate C and A and see how the print() functions create some nice printouts.
Instructions
100 XP
script.py
# Definition of radius
r = 0.43
# Import the math package
# Calculate C
C = 0
# Calculate A
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
can calculate $C$ and $A$ as:
$$C = 2\pi r$$
$$A = \pi r^2$$
To use the constant pi, you'll need the math package. A variable r is already coded in the script. Fill in the code to calculate C and A and see how the print() functions create some nice printouts.
Instructions
100 XP
Import the math package. Now you can access the constant pi with math.pi.
Calculate the circumference of the circle and store it in C.
Calculate the area of the circle and store it in A.
script.py
# Definition of radius
r = 0.43
# Import the math package
import math
# Calculate C
C = 2*math.pi*r
# Calculate A
A = math.pi*r**2
# Build printout
print("Circumference: " + str(C))
print("Area: " + str(A))
Run Code
Submit Answer
IPython Shell
Slides


DataCamp
Course Outline
Exercise
Selective import
General imports, like import math, make all functionality from the math package available to you. However, if you decide to only use a specific part of a package, you can always make your import more selective:
from math import pi
Let's say the Moon's orbit around planet Earth is a perfect circle, with a radius r (in km) that is defined in the script.
Instructions
100 XP
Perform a selective import from the math package where you only import the radians function.
Calculate the distance travelled by the Moon over 12 degrees of its
script.py
# Definition of radius
r = 192500
# Import radians function of math package
# Travel distance of Moon over 12 degrees. Store in dist.
Run Code
Submit Answer
IPython Shell
Slides
In [1]:


DataCamp
Course Outline
Exercise
with a radius r (in km) that is defined in the script.
Instructions
100 XP
Perform a selective import from the math package where you only import the radians function.
Calculate the distance travelled by the Moon over 12 degrees of its orbit. Assign the result to dist . You can calculate this as r * phi , where r is the radius and phi is the angle in radians. To convert an angle in degrees to an angle in radians, use the radians() function, which you just imported.
Print out dist .
Take Hint (-30 XP)
script.py
1 # Definition of radius
2 r = 192500
3 # Import radians function of math package
4 from math import radians
5 # Travel distance of Moon over 12 degrees. Store in dist.
6 dist=r*radians(12)
7 # Print out dist
8 print(dist)
Run all/selected code Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
# Travel distance of Moon over 12 degrees. Store in dist.
dist=r*radians(12)
# Print out dist
print(dist)
40317.10572106901
In [2]:


DataCamp
Course Outline
Exercise
There are several ways to import packages and modules into Python. Depending on the import call, you'll have to use different Python code.
Suppose you want to use the function inv() , which is in the linalg subpackage of the scipy package. You want to be able to use this function as follows:
my_inv([[1,2], [3,4]])
Which import statement will you need in order to run the above code without an error?
Instructions
50 XP
Possible Answers
import scipy
import scipy.linalg
from scipy.linalg import my_inv
from scipy.linalg import inv as my_inv
IPython Shell
Slides
In [1]:

Methods

50 XP

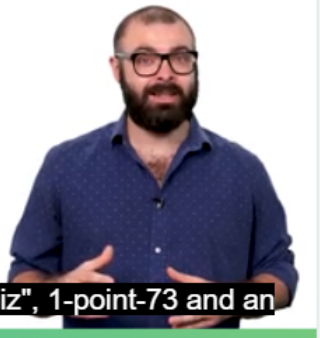
# Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

| | type |
|---|---|
| Object | str |
| Object | float |
| Object | list |



course they represent the values you gave them, such as "liz", 1-point-73 and an entire list.



Methods

50 XP

# Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

| | type | examples of methods |
|---|---|---|
| Object | str | capitalize()<br>replace() |
| Object | float | bit_length()<br>conjugate() |
| Object | list | index()<br>count() |

- Methods: Functions that belong to objects

but also objects of type float and list have specific methods depending on the type.

Methods

50 XP

# list methods

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

Python returns 4, which indeed is the index of the string "mom".



Methods

50 XP

# str methods

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

In the output, "z" is replaced with "sa".

Methods 50 XP

# Methods

- Everything = object

To be absolutely clear: in Python, everything is an object, and each object has specific methods associated.

DataCamp INTRODUCTION TO PYTHON

Methods 50 XP

# Methods

- Everything = object
- Object have methods associated, depending on type

```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

A string object like sister has a replace method, but a list like fam doesn't have this, as you can see from this error.

DataCamp INTRODUCTION TO PYTHON

-1:59 1x auto

Methods

50 XP

# Methods

```
sister.index("z")
```

```
2
```

```
fam.index("mom")
```

```
4
```

This means that, depending on the type of the object, the methods behave differently.

DataCamp INTRODUCTION TO PYTHON

-1:34 1x auto

Methods

50 XP

# Methods (2)

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.append("me")
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']
```

list again, we see that it has been extended with the string "me".

DataCamp INTRODUCTION TO PYTHON

Methods 50 XP

# Summary

## Functions

```
type(fam)
```

```
list
```

## Methods: call functions on objects

```
fam.index("dad")
```

```
6
```

There's much more to tell about Python objects, methods and how Python

DataCamp INTRODUCTION TO PYTHON

DataCamp Course Outline Daily XP 1450

Exercise

### String Methods

Strings come with a bunch of methods. Follow the instructions closely to discover some of them. If you want to discover them in more detail, you can always type `help(str)` in the IPython Shell.

A string `place` has already been created for you to experiment with.

Instructions 100 XP

- Use the `upper()` method on `place` and store the result in `place_up` . Use the syntax for calling methods that you learned in the previous video.
- Print out `place` and `place_up` . Did both change?
- Print out the number of o's on the variable `place` by calling `count()` on `place` and passing the letter `'o'` as an input to the method. We're talking about the variable `place` , not the word `"place"` !

Take Hint (-30 XP)

script.py

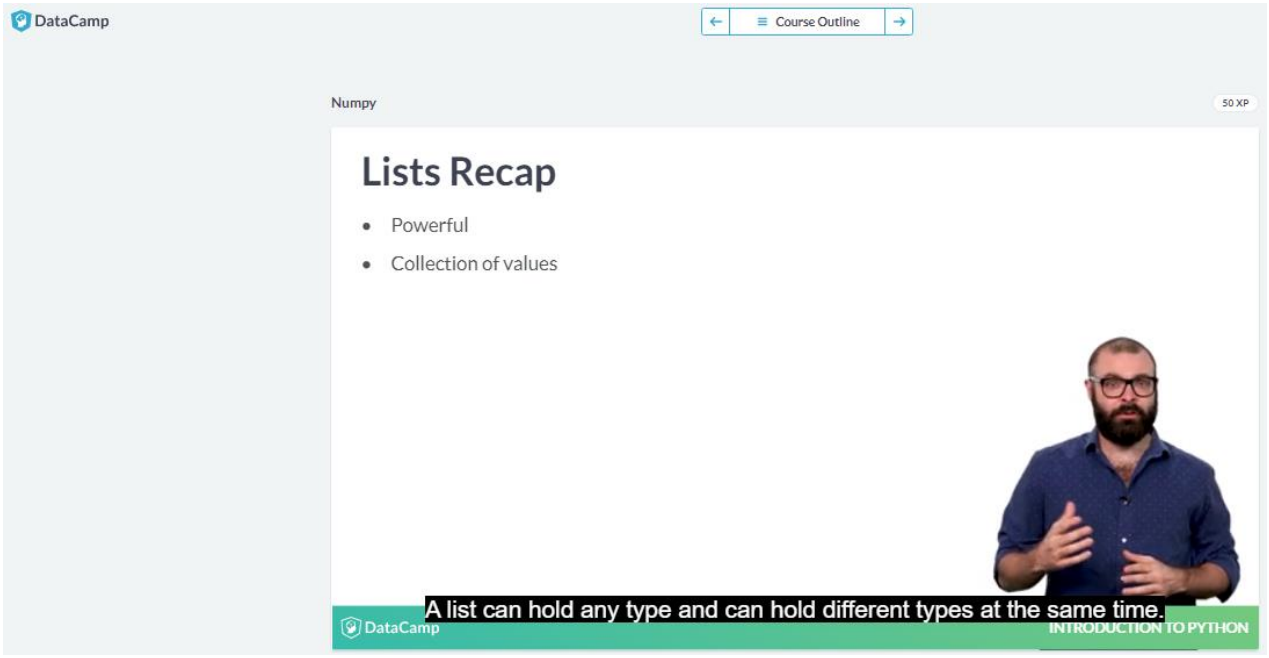
```
# string to experiment with: place
place = "poolhouse"
# Use upper() on place: place_up
place_up=place.upper()
# Print out place and place_up
print(place)
print(place_up)
# Print out the number of o's in place
print(place.count("o"))
```

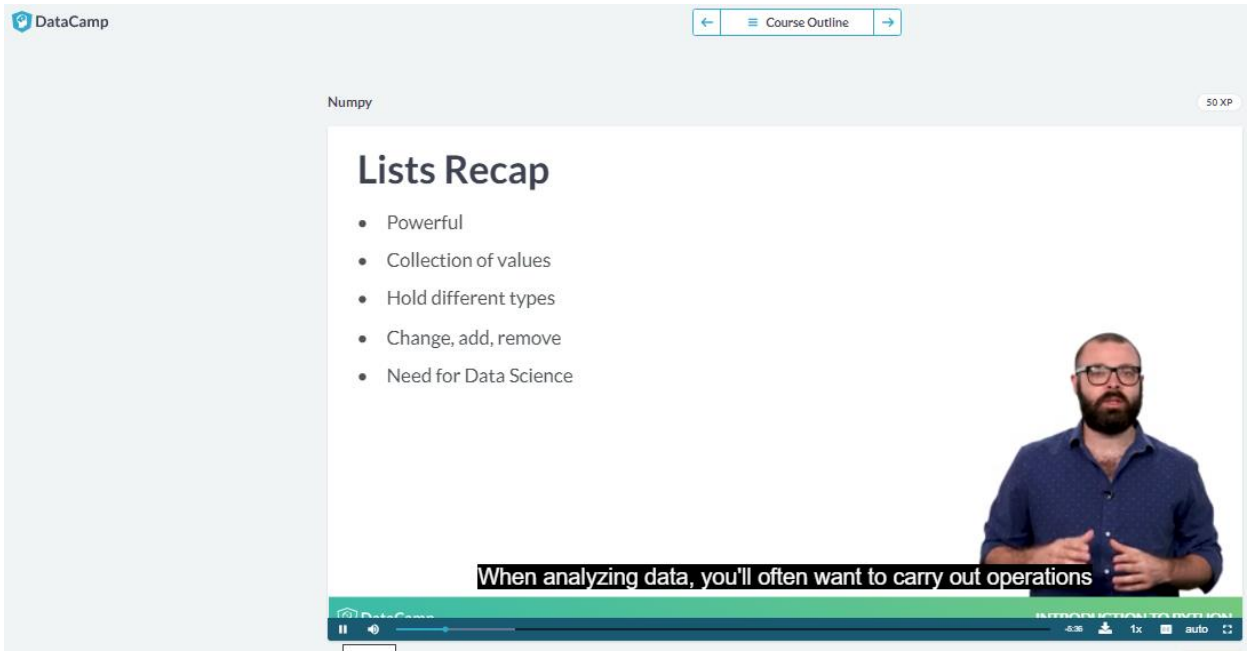
Ctrl+Shift+Enter

Run Code Submit Answer

IPython Shell Slides

```
In [1]: # string to experiment with: place
        place = "poolhouse"
        # Use upper() on place: place_up
        place_up=place.upper()
        # Print out place and place_up
        print(place)
        print(place_up)
        # Print out the number of o's in place
        print(place.count("o"))
poolhouse
POOLHOUSE
3

In [2]:
```


DataCamp
Course Outline
Numpy
50 XP
Lists Recap
Powerful
Collection of values
A list can hold any type and can hold different types at the same time.
DataCamp
INTRODUCTION TO PYTHON


DataCamp
Course Outline
Numpy
50 XP
Lists Recap
Powerful
Collection of values
Hold different types
Change, add, remove
Need for Data Science
When analyzing data, you'll often want to carry out operations
1x
auto

DataCamp

Course Outline

Numpy

50 XP

# Illustration

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

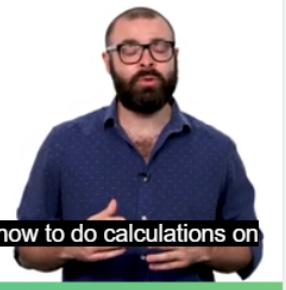
```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
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 **: 'list' and 'int'
```



Unfortunately, Python throws an error, because it has no idea how to do calculations on lists.

DataCamp INTRODUCTION TO PYTHON

Got It!

DataCamp

Course Outline

Numpy

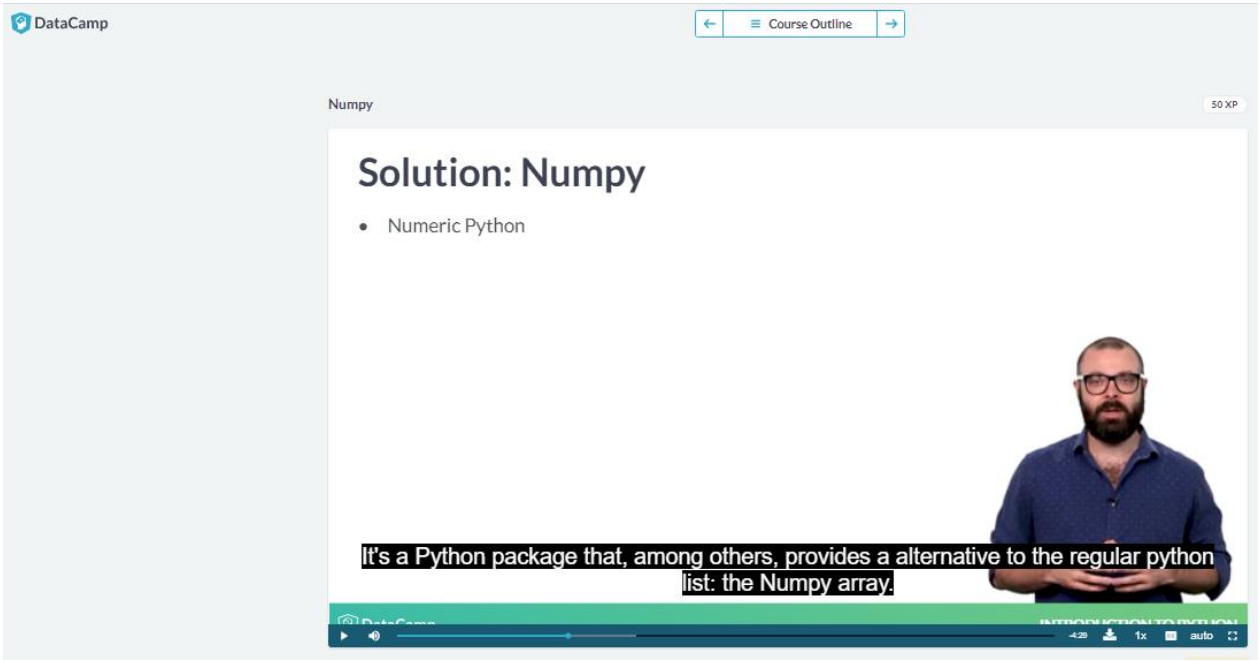
50 XP

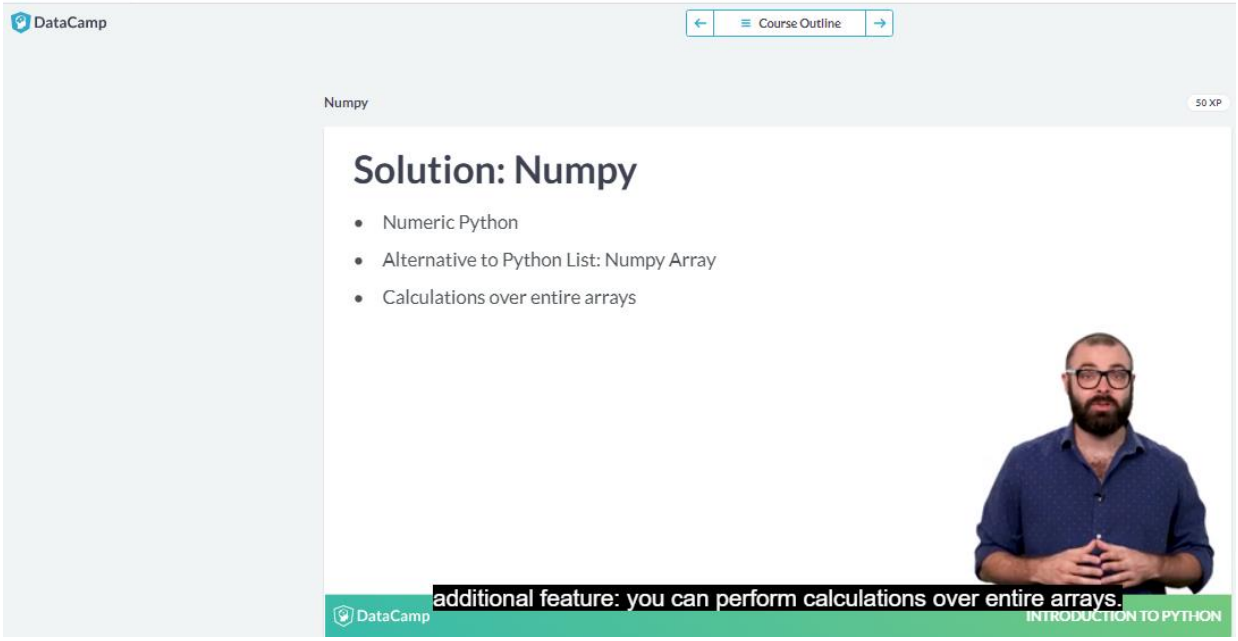
# Solution: Numpy

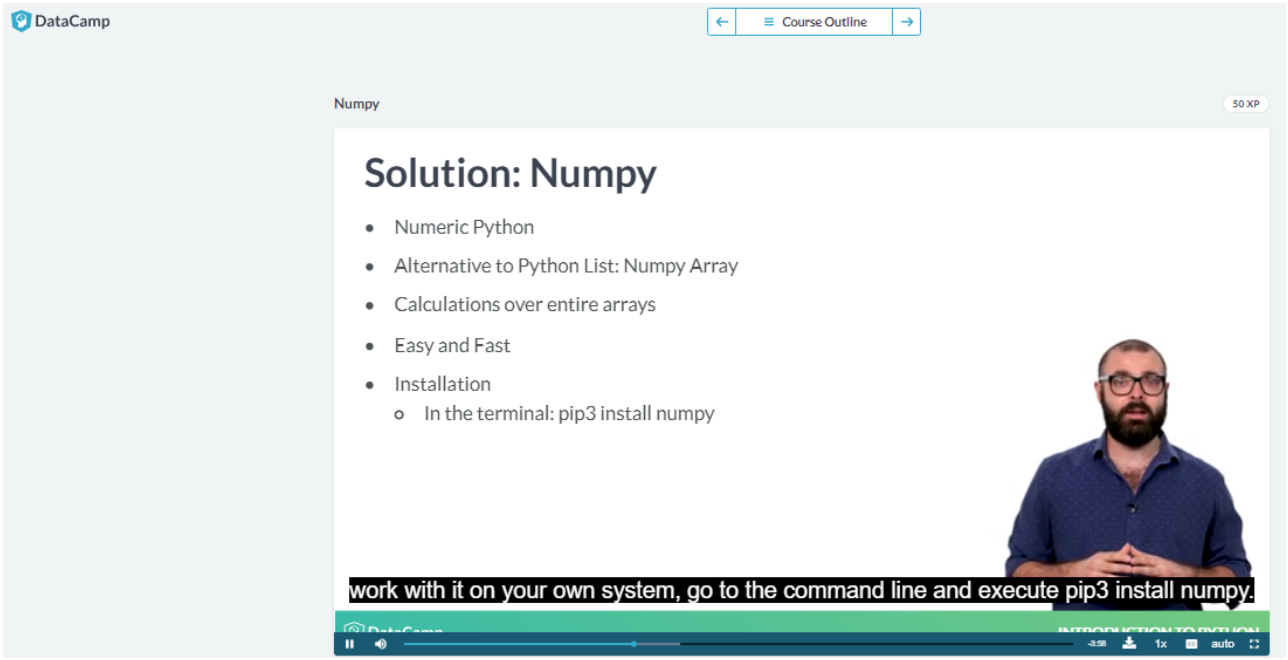
- Numeric Python

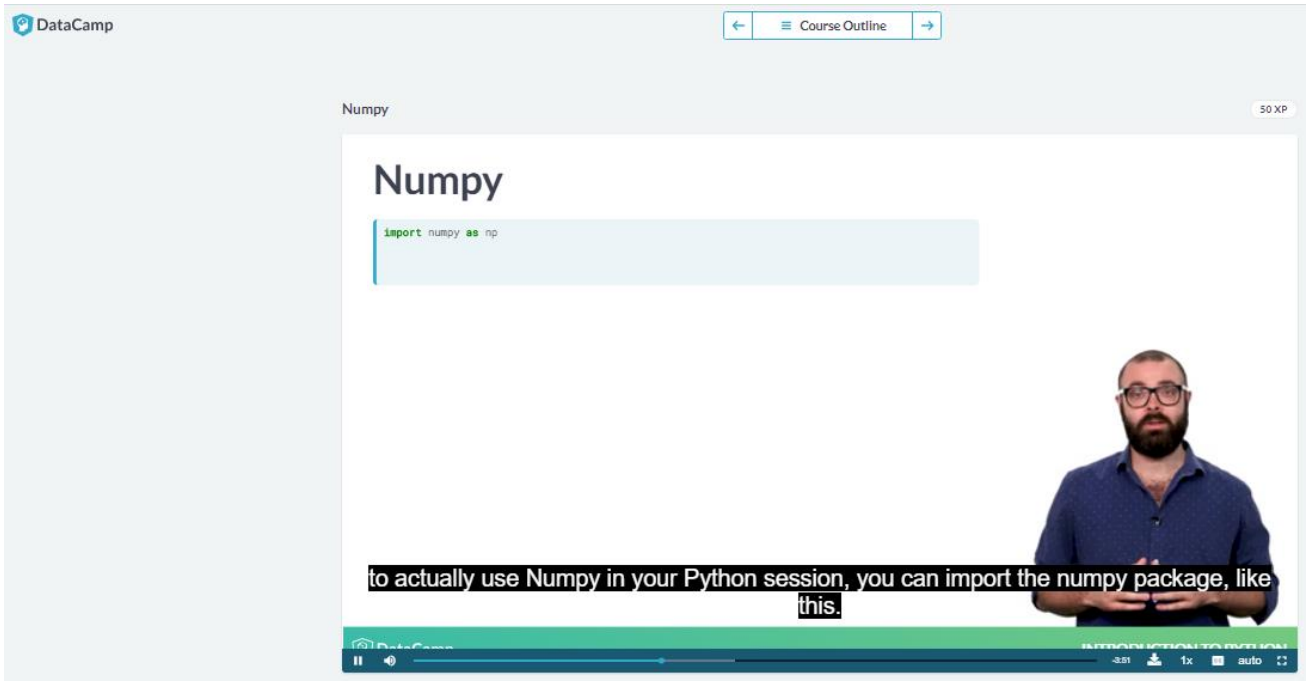
A way more elegant solution is to use NumPy, or Numeric Python.

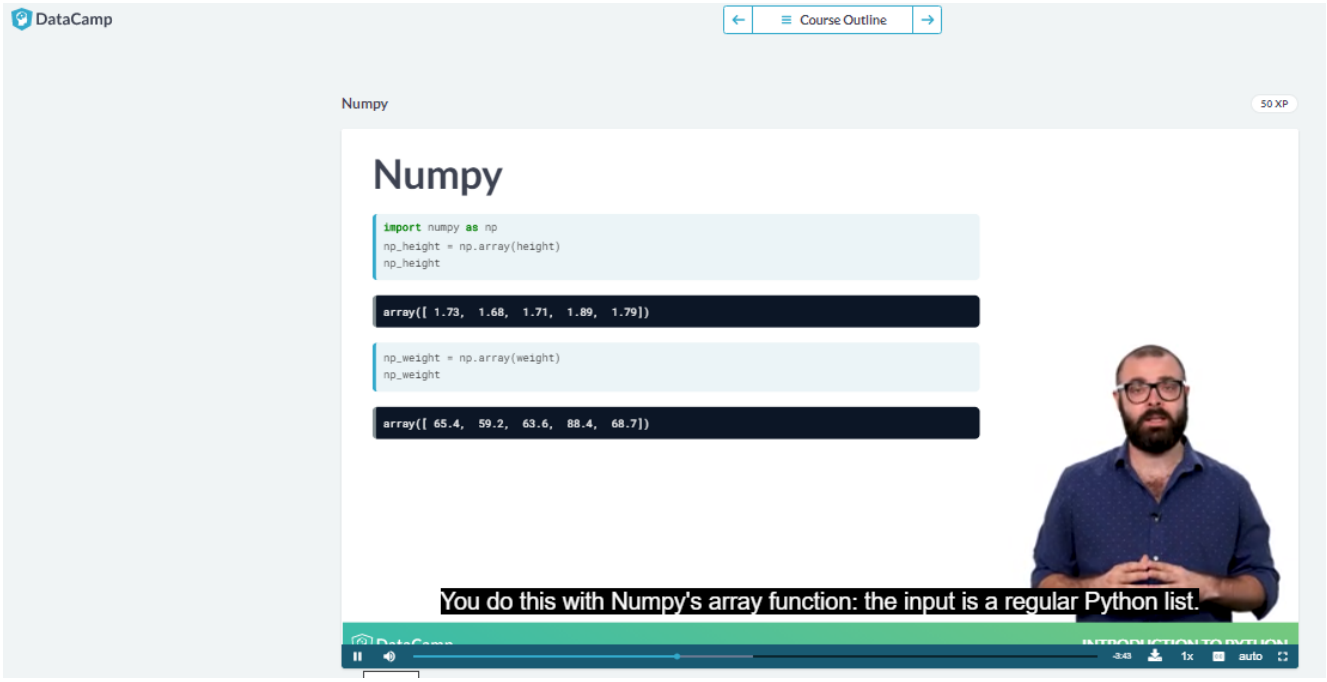
1:32 1:44 -4:31 1x auto

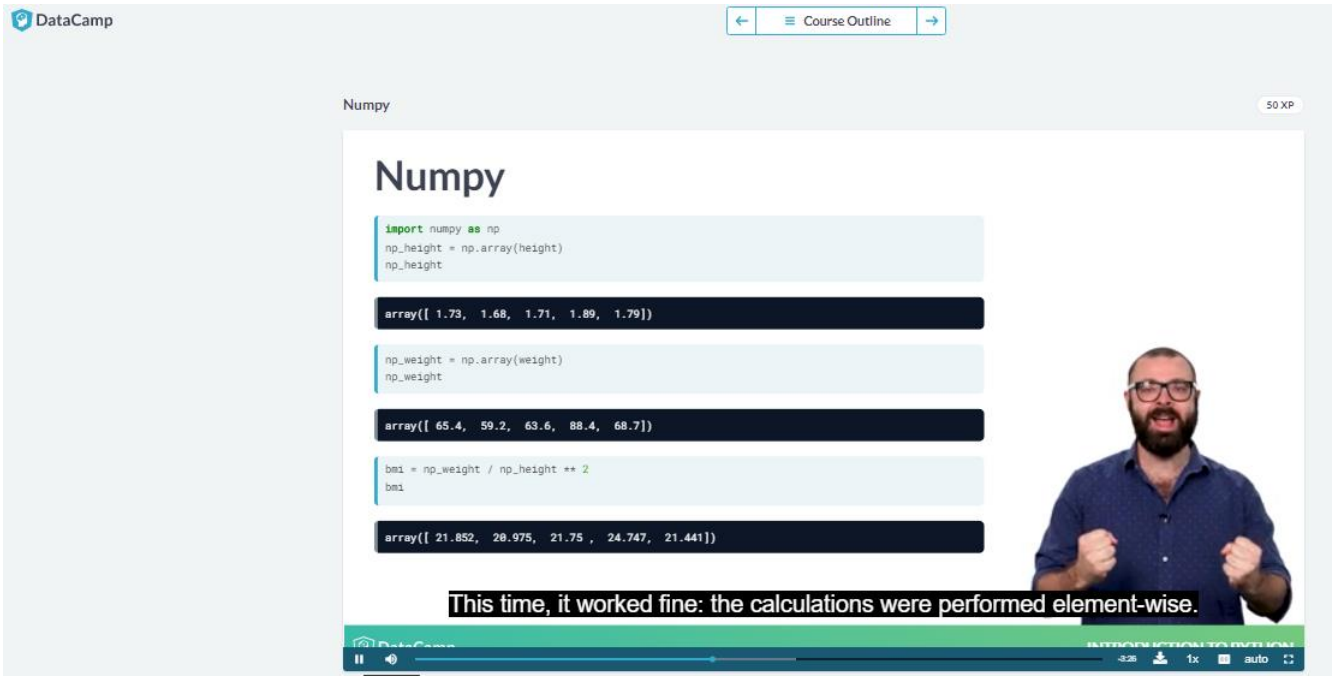

DataCamp
Course Outline
Numpy
50 XP
Solution: Numpy
• Numeric Python
It's a Python package that, among others, provides a alternative to the regular python list: the Numpy array.
1x
auto


DataCamp
Course Outline
Numpy
50 XP
Solution: Numpy
• Numeric Python
• Alternative to Python List: Numpy Array
• Calculations over entire arrays
additional feature: you can perform calculations over entire arrays.
DataCamp
INTRODUCTION TO PYTHON


DataCamp
Course Outline
Numpy
50 XP
Solution: Numpy
Numeric Python
Alternative to Python List: Numpy Array
Calculations over entire arrays
Easy and Fast
Installation
In the terminal: pip3 install numpy
work with it on your own system, go to the command line and execute pip3 install numpy.
INTRODUCTION TO PYTHON
1x
auto


DataCamp
Course Outline
Numpy
50 XP
Numpy
import numpy as np
to actually use Numpy in your Python session, you can import the numpy package, like this.
INTRODUCTION TO PYTHON
1x
auto


DataCamp
Course Outline
Numpy
50 XP
Numpy
import numpy as np
np_height = np.array(height)
np_height
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])
You do this with Numpy's array function: the input is a regular Python list.
INTRODUCTION TO PYTHON
-3:43
1x
auto


DataCamp
Course Outline
Numpy
50 XP
Numpy
import numpy as np
np_height = np.array(height)
np_height
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([ 21.852, 20.975, 21.75 , 24.747, 21.441])
This time, it worked fine: the calculations were performed element-wise.
INTRODUCTION TO PYTHON
-3:28
1x
auto

DataCamp

Course Outline

Numpy

50 XP

# Numpy: remarks

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

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

- Numpy arrays: contain only one type

assumes that your Numpy array can only contain values of a single type.

DataCamp

Course Outline

Numpy

50 XP

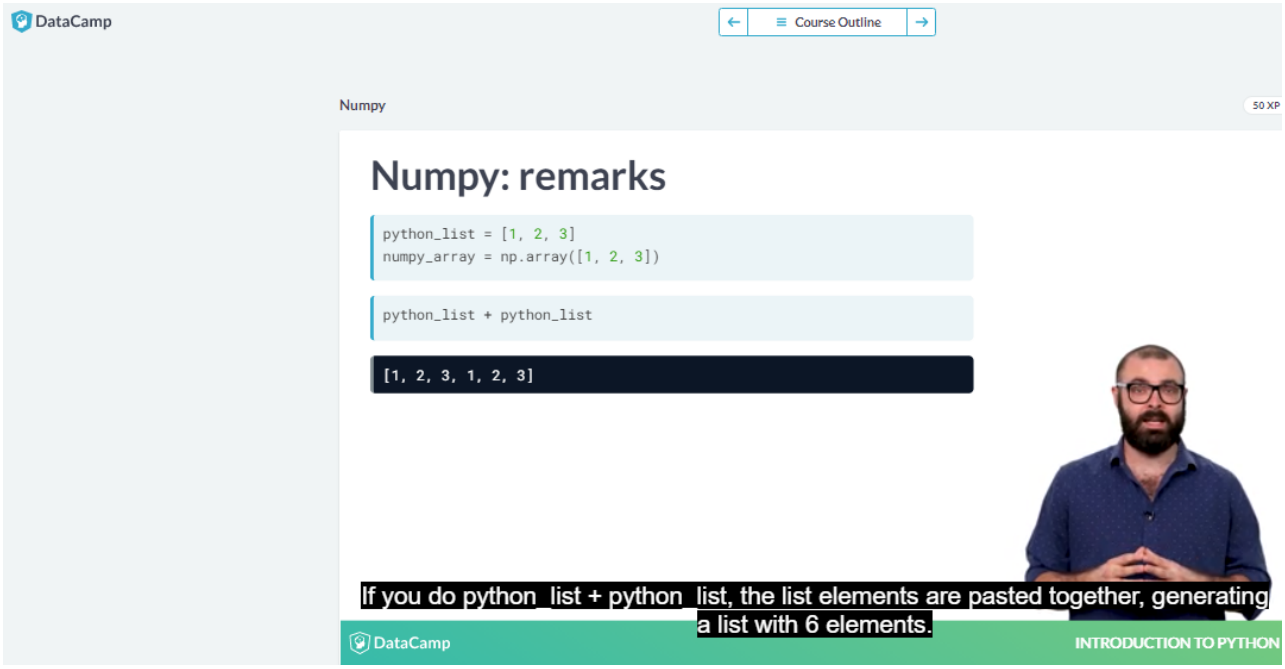
# Numpy: remarks

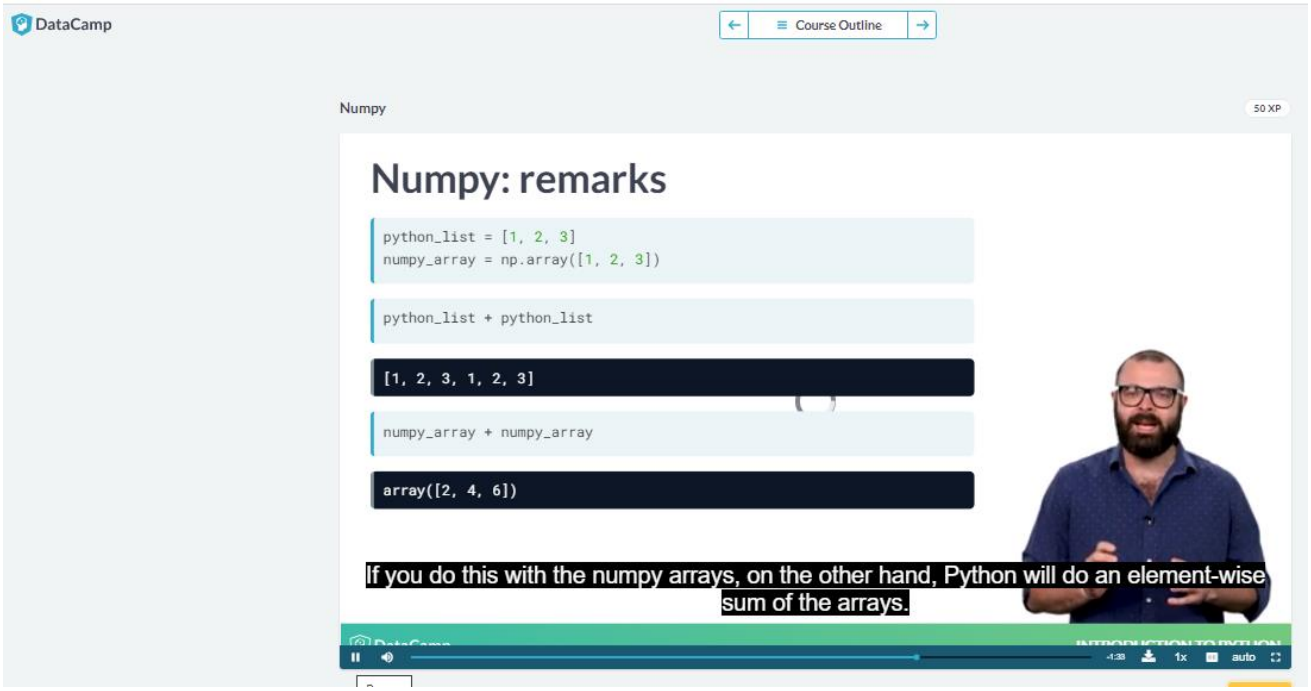
```
np.array([1.0, "is", True])
```

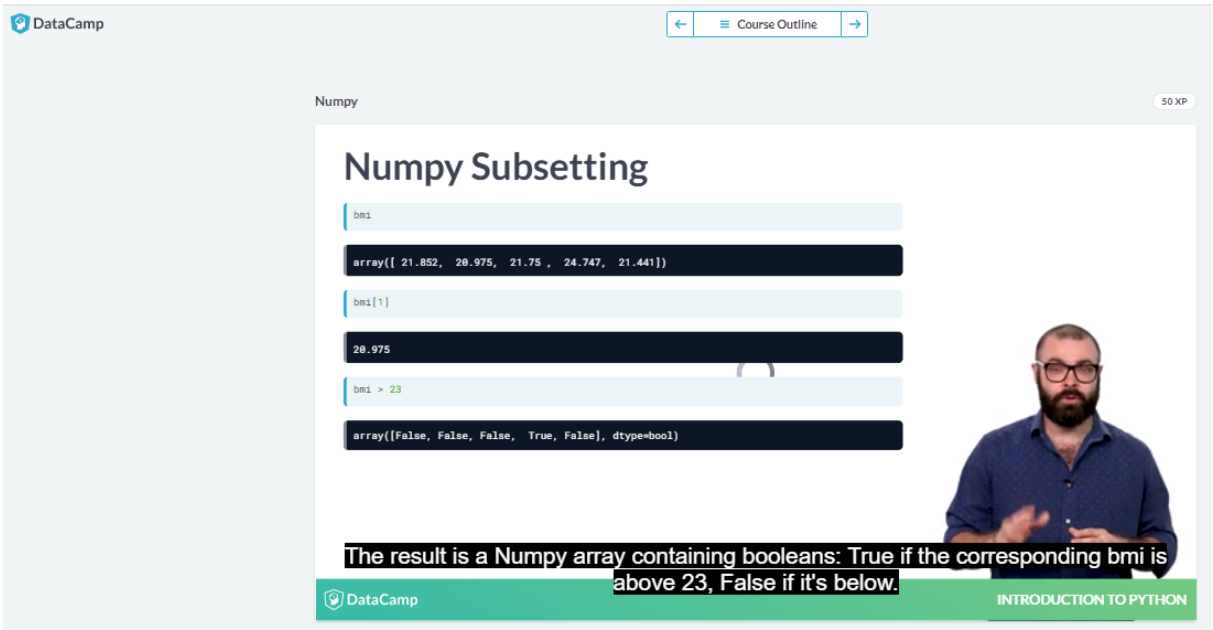
```
array(['1.0', 'is', 'True'],
      dtype='<U32')
```

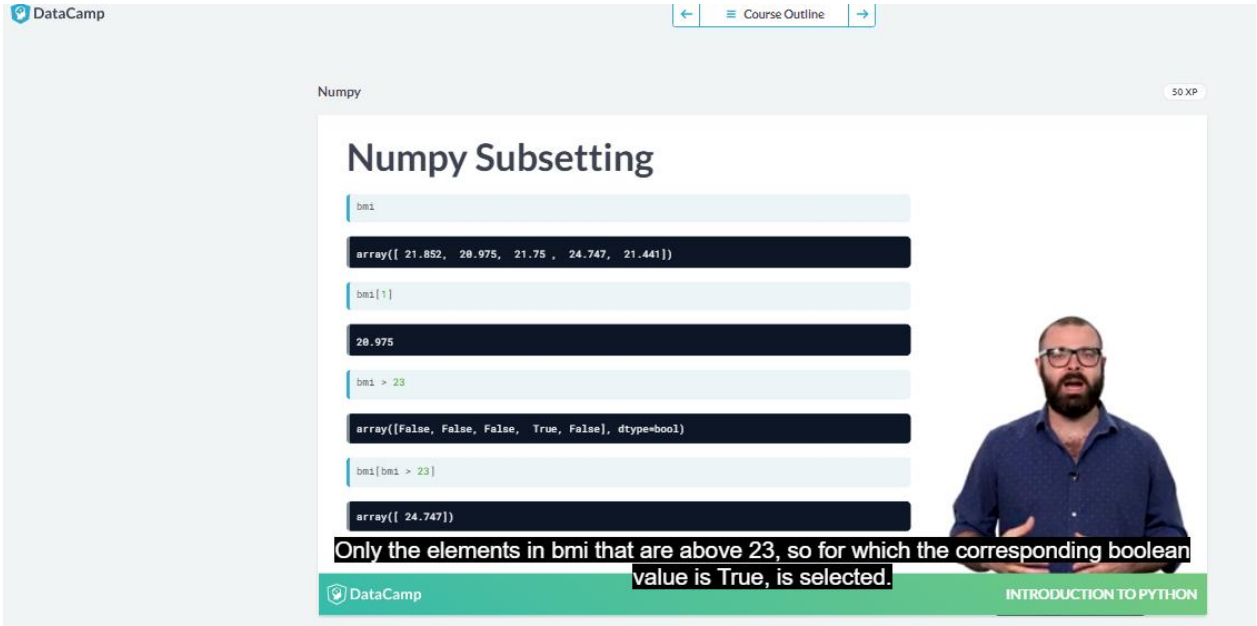
- Numpy arrays: contain only one type

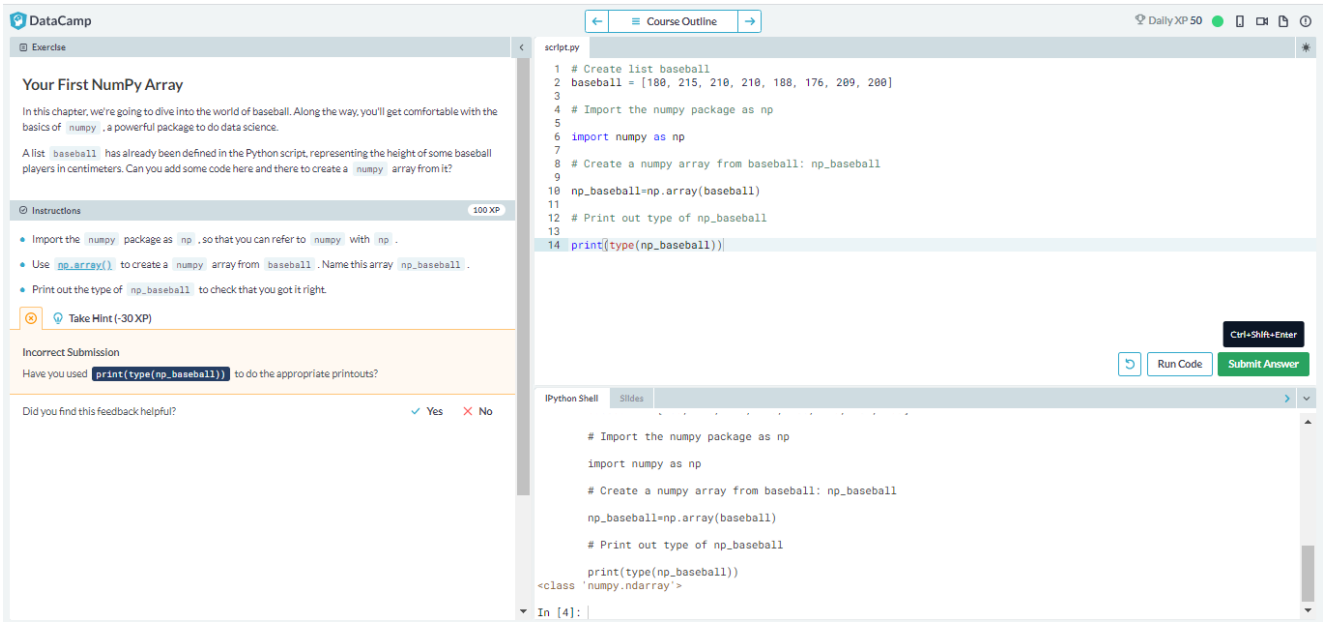
The boolean and the float were both converted to strings.

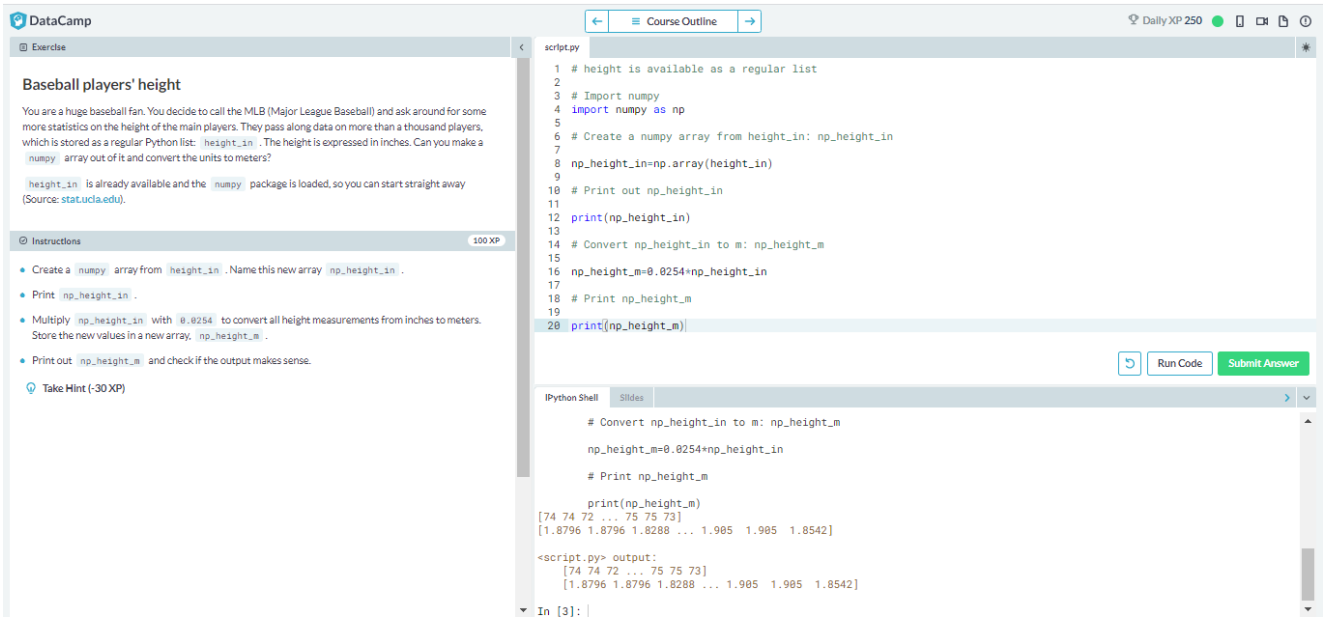

DataCamp
Course Outline
Numpy
50 XP
Numpy: remarks
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])
python_list + python_list
[1, 2, 3, 1, 2, 3]
If you do python_list + python_list, the list elements are pasted together, generating a list with 6 elements.
DataCamp
INTRODUCTION TO PYTHON

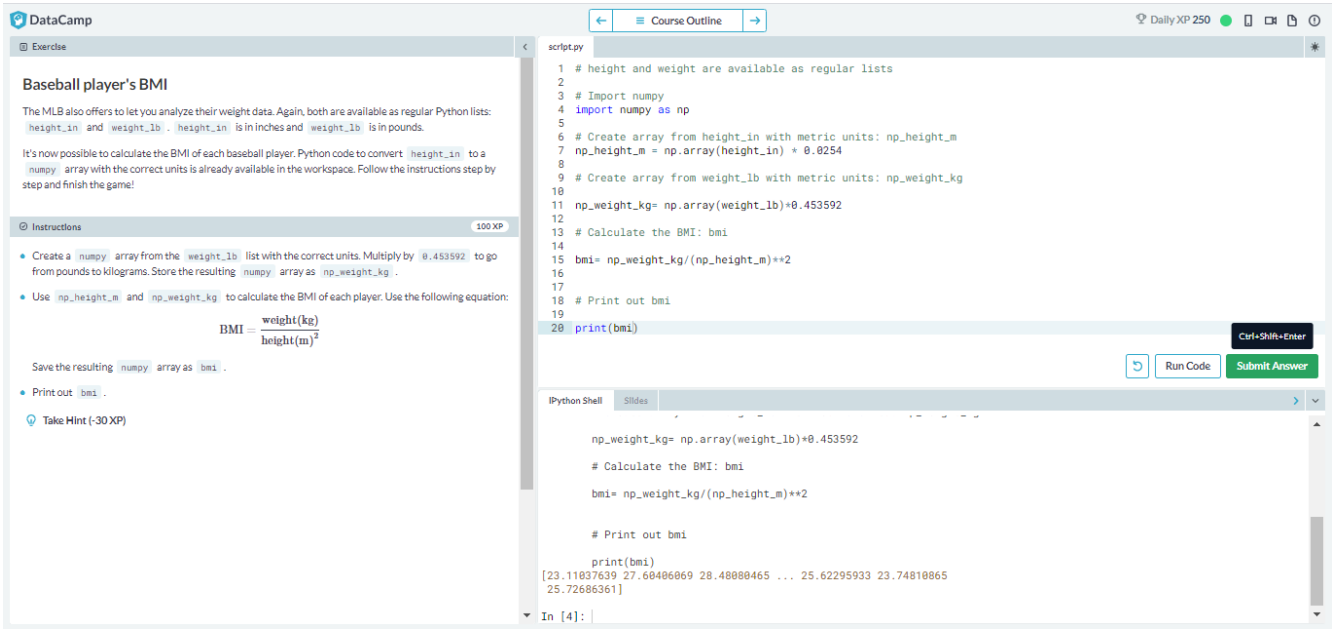

DataCamp
Course Outline
Numpy
50 XP
Numpy: remarks
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])
python_list + python_list
[1, 2, 3, 1, 2, 3]
numpy_array + numpy_array
array([2, 4, 6])
If you do this with the numpy arrays, on the other hand, Python will do an element-wise sum of the arrays.
1x
auto

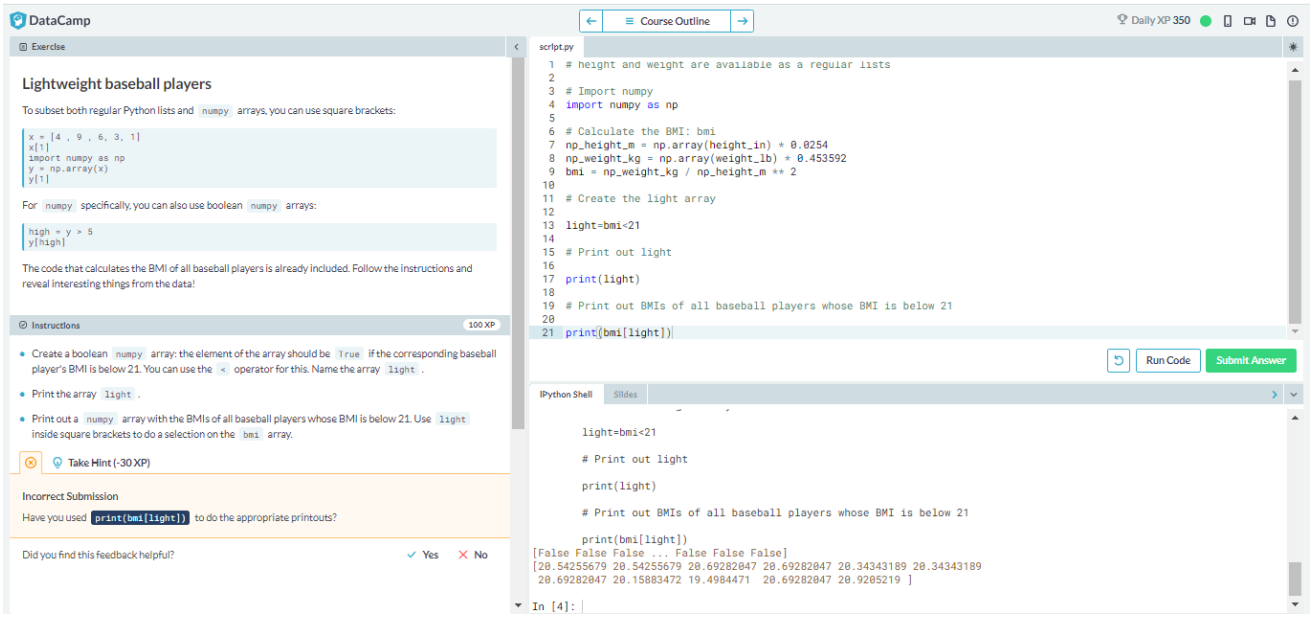

DataCamp
Course Outline
Numpy
50 XP
Numpy Subsetting
bmi
array([ 21.852, 20.975, 21.75 , 24.747, 21.441])
bmi[1]
20.975
bmi > 23
array([False, False, False, True, False], dtype=bool)
The result is a Numpy array containing booleans: True if the corresponding bmi is above 23, False if it's below.
DataCamp
INTRODUCTION TO PYTHON

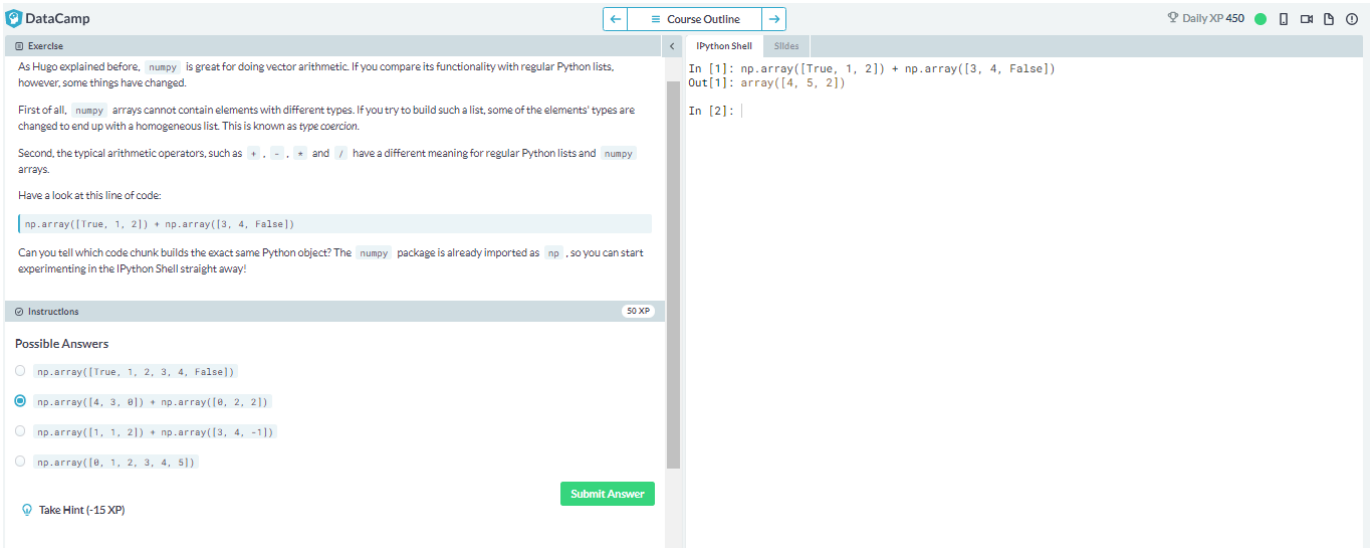

DataCamp
Course Outline
Numpy
50 XP
Numpy Subsetting
bmi
array([ 21.852, 20.975, 21.75 , 24.747, 21.441])
bmi[1]
20.975
bmi > 23
array([False, False, False, True, False], dtype=bool)
bmi[bmi > 23]
array([ 24.747])
Only the elements in bmi that are above 23, so for which the corresponding boolean value is True, is selected.
DataCamp
INTRODUCTION TO PYTHON

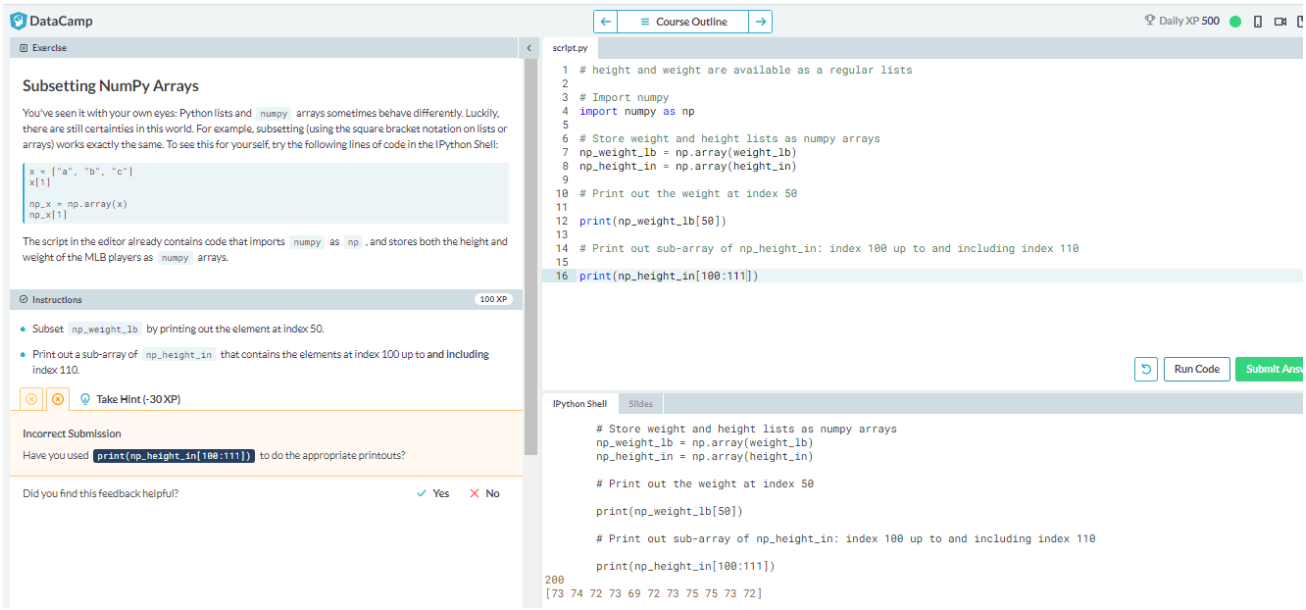

DataCamp
Course Outline
Daily XP 50
Exercise
Your First NumPy Array
In this chapter, we're going to dive into the world of baseball. Along the way, you'll get comfortable with the basics of numpy, a powerful package to do data science.
A list baseball has already been defined in the Python script, representing 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() 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)
Incorrect Submission
Have you used print(type(np_baseball)) to do the appropriate printouts?
Did you find this feedback helpful?
Yes
No
script.py
# 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))
Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
# 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 [4]:

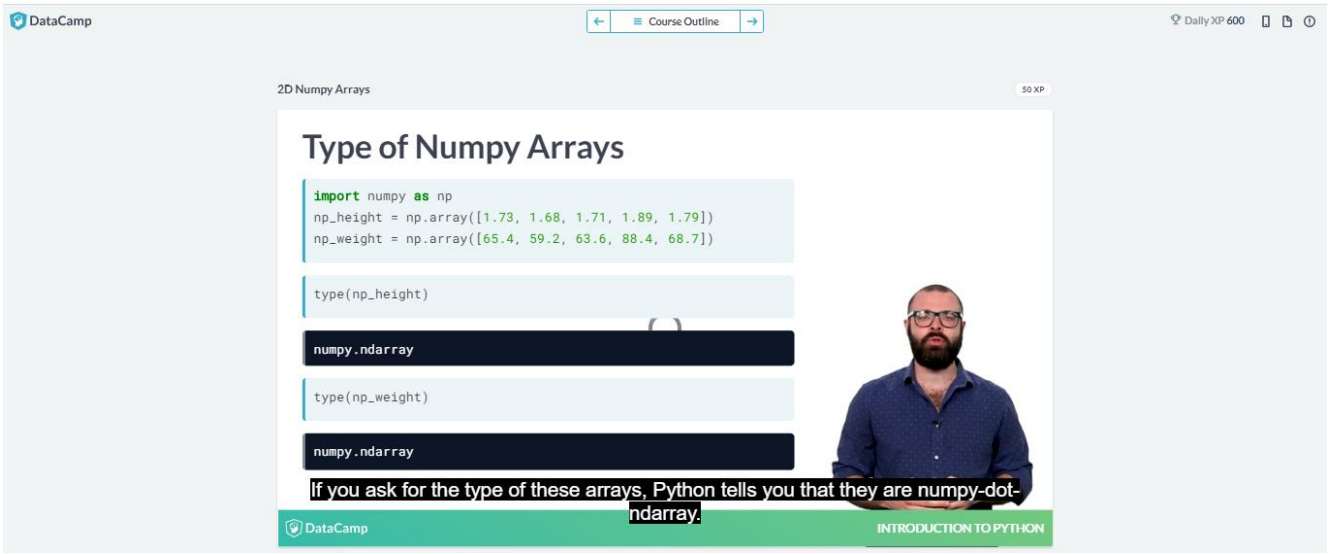

DataCamp
Course Outline
Daily XP 250
Exercise
Baseball players' height
You are a huge baseball fan. You decide to call the MLB (Major League Baseball) and ask around for some more statistics on the height of the main players. They pass along data on more than a thousand players, which is stored as a 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.
Take Hint (-30 XP)
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=0.0254*np_height_in
# Print np_height_m
print(np_height_m)
Run Code
Submit Answer
IPython Shell
Slides
# Convert np_height_in to m: np_height_m
np_height_m=0.0254*np_height_in
# 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]
<script.py> output:
[74 74 72 ... 75 75 73]
[1.8796 1.8796 1.8288 ... 1.905 1.905 1.8542]
In [3]:

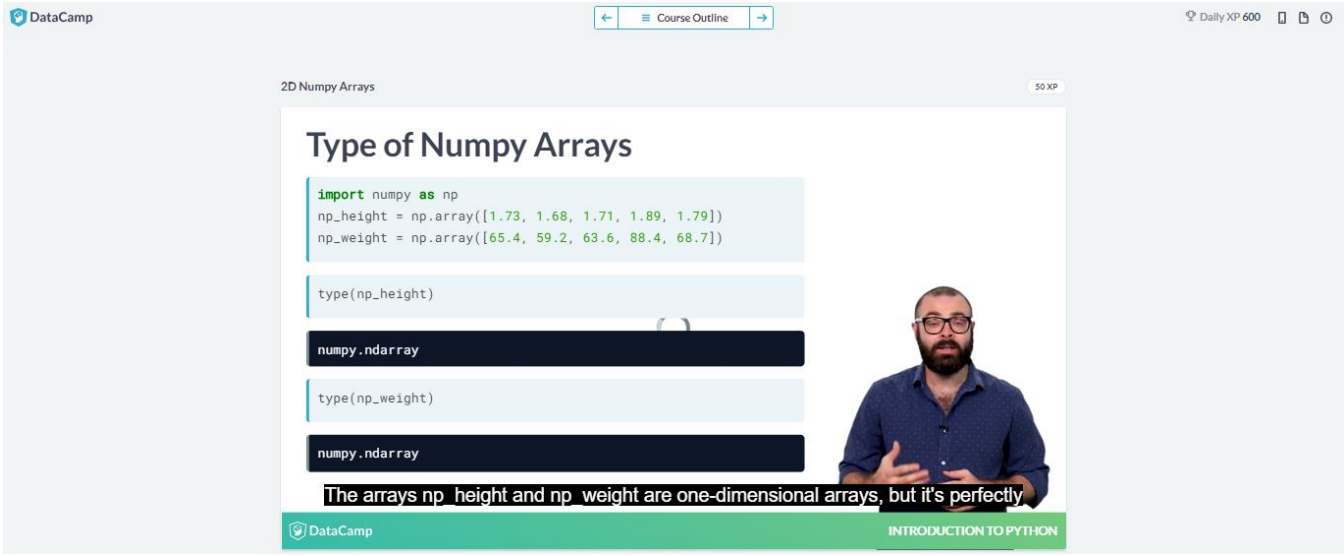

DataCamp
Course Outline
Daily XP 250
Exercise
Baseball player's BMI
The MLB also offers to let you analyze their weight data. Again, both are available as regular Python lists: height_in and weight_lb . height_in is in inches and weight_lb is in pounds.
It's now possible to calculate the BMI of each baseball player. Python code to convert height_in to a numpy array 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:
$$\text{BMI} = \frac{\text{weight(kg)}}{\text{height(m)}^2}$$
Save the resulting numpy array as bmi .
Print out bmi .
Take Hint (-30 XP)
script.py
# 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)
Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
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 [4]:

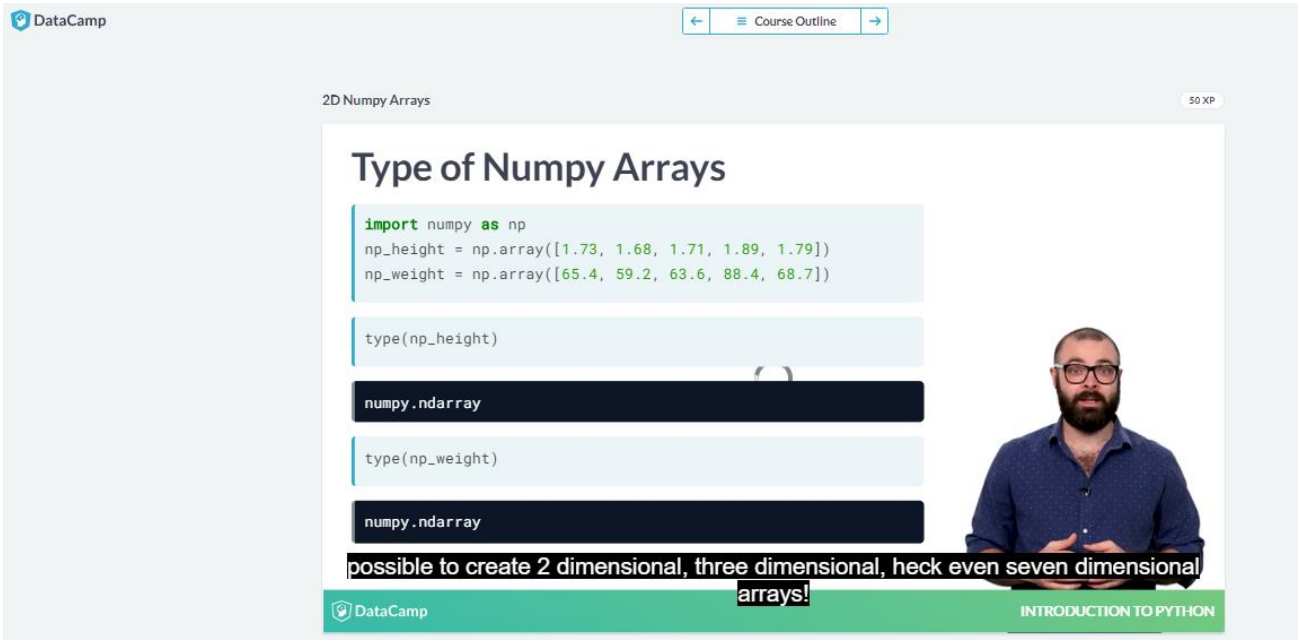

DataCamp
Course Outline
Daily XP 350
Exercise
Lightweight baseball players
To subset both regular Python lists and numpy arrays, you can use square brackets:
x = [4 , 9 , 6, 3, 1]
x[1]
import numpy as np
y = np.array(x)
y[1]
For numpy specifically, you can also use boolean numpy arrays:
high = y > 5
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.
Take Hint (-30 XP)
Incorrect Submission
Have you used print(bmi[light]) to do the appropriate printouts?
Did you find this feedback helpful?
Yes
No
script.py
# 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=bmi<21
# Print out light
print(light)
# Print out BMIs of all baseball players whose BMI is below 21
print(bmi[light])
Run Code
Submit Answer
IPython Shell
Slides
light=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 [4]:

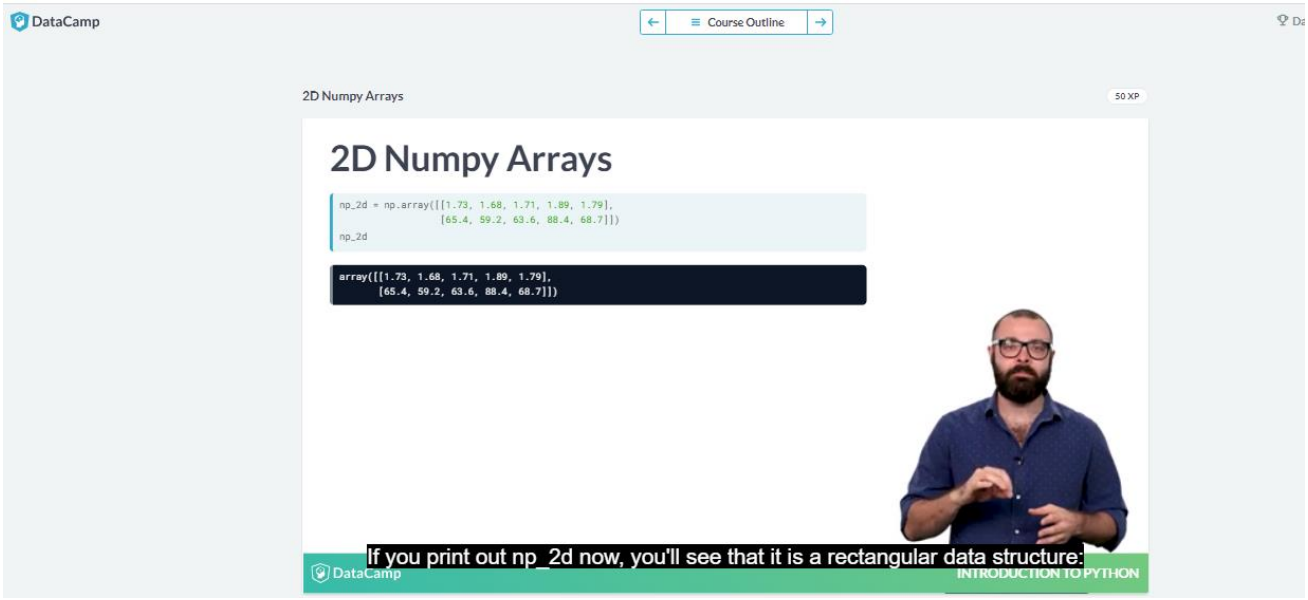

DataCamp
Course Outline
Daily XP 450
Exercise
As Hugo explained before, numpy is great for doing vector arithmetic. If you compare its functionality with regular Python lists, however, some things have changed.
First of all, numpy arrays cannot contain elements with different types. If you try to build 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])
Submit Answer
Take Hint (-15 XP)
IPython Shell
Slides
In [1]: np.array([True, 1, 2]) + np.array([3, 4, False])
Out[1]: array([4, 5, 2])
In [2]:


DataCamp
Course Outline
Daily XP 500
Exercise
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 in the editor 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.
Take Hint (-30 XP)
Incorrect Submission
Have you used print(np_height_in[100:111]) to do the appropriate printouts?
Did you find this feedback helpful?
Yes
No
script.py
# 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])
Run Code
Submit Answ
IPython Shell
Slides
# 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]


DataCamp
Course Outline
Daily XP 600
2D Numpy Arrays
50 XP
Type of Numpy Arrays
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])
type(np_height)
numpy.ndarray
type(np_weight)
numpy.ndarray
If you ask for the type of these arrays, Python tells you that they are numpy-dot-ndarray.
DataCamp
INTRODUCTION TO PYTHON


DataCamp
Course Outline
Daily XP 600
2D Numpy Arrays
50 XP
Type of Numpy Arrays
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])
type(np_height)
numpy.ndarray
type(np_weight)
numpy.ndarray
The arrays np_height and np_weight are one-dimensional arrays, but it's perfectly
DataCamp
INTRODUCTION TO PYTHON


DataCamp
Course Outline
2D Numpy Arrays
50 XP
Type of Numpy Arrays
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])
type(np_height)
numpy.ndarray
type(np_weight)
numpy.ndarray
possible to create 2 dimensional, three dimensional, heck even seven dimensional arrays!
DataCamp
INTRODUCTION TO PYTHON


DataCamp
Course Outline
2D Numpy Arrays
50 XP
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]])
If you print out np_2d now, you'll see that it is a rectangular data structure:
DataCamp
INTRODUCTION TO PYTHON

2D Numpy Arrays

50 XP

# 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]])
```

Each sublist in the list, corresponds to a row in the two dimensional numpy array.

DataCamp

INTRODUCTION TO PYTHON

2D Numpy Arrays

50 XP

# 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
```

shape is a so-called attribute of the np2d array, that can

DataCamp

INTRODUCTION TO PYTHON

-2:46 1x auto

# 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"]])
```

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

will be coerced to strings, to end up with a homogeneous array.

DataCamp INTRODUCTION TO PYTHON

# Subsetting

```
          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]
```

```
1.71
```

Basically you're selecting the row, and then from that row do another selection.

-1:37 1x auto

# Subsetting

```
          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]
```

```
1.71
```

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

```
1.71
```

The value before the comma specifies the row, the value after the comma specifies the column.





# Subsetting

```
          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 ]])
```

The intersection gives us a 2D array with 2 rows and 2 columns:

# Subsetting

```
          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 ]])
```

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

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

weight of all family members like this: you only want the second row, so put 1 before the comma.



# Subsetting

```
          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
```

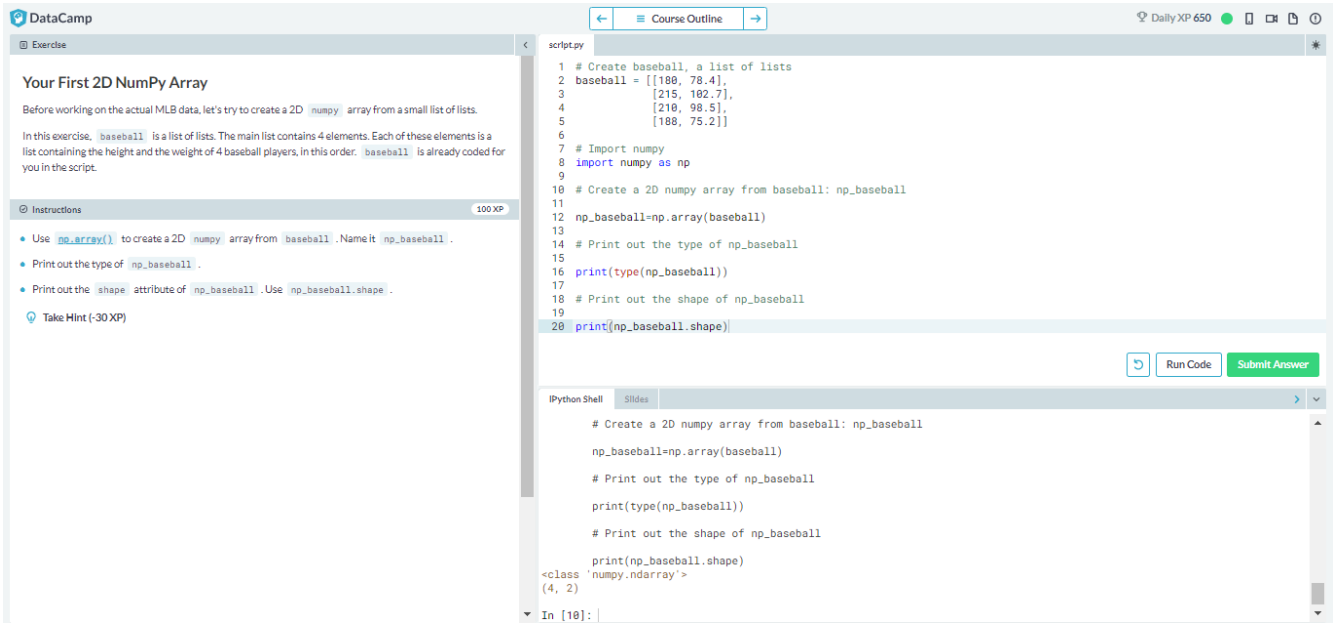
```
np_2d[:,1:3]
```

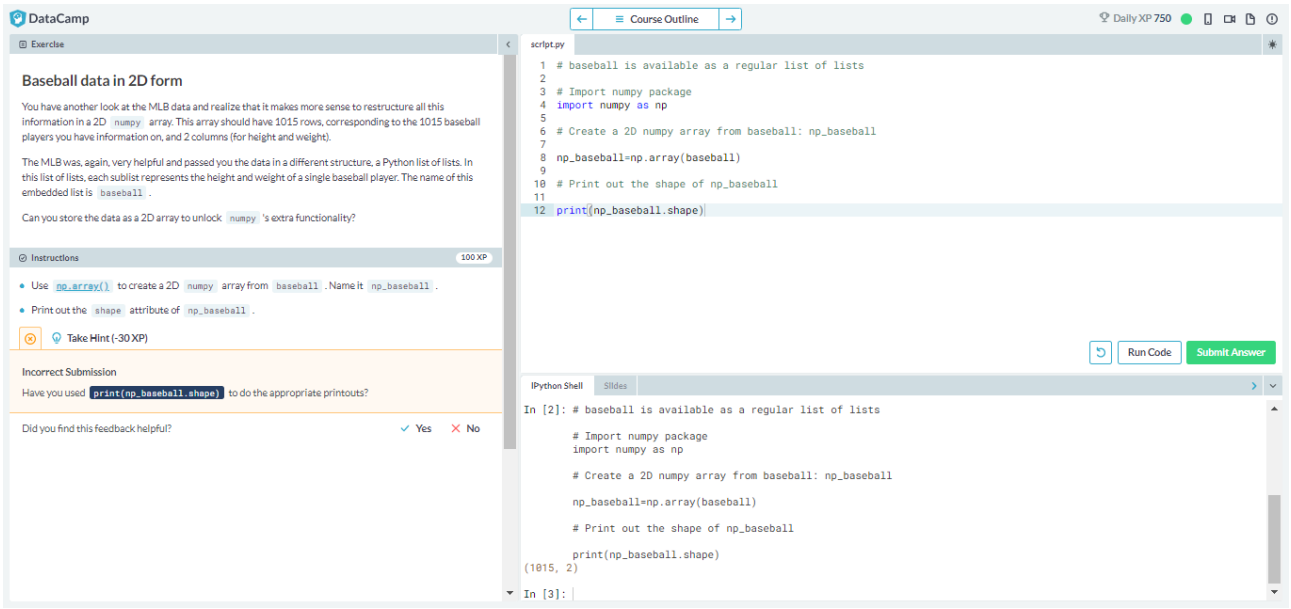
```
array([[  1.68,  1.71],
       [ 59.2 , 63.6 ]])
```

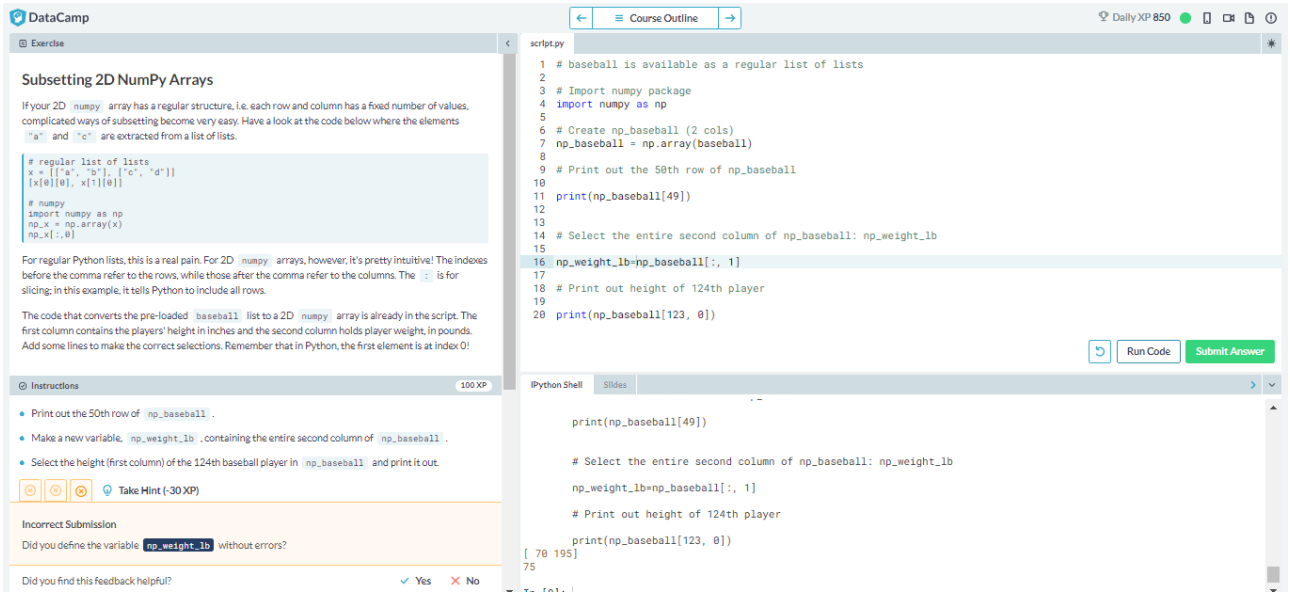
```
np_2d[1,:]
```

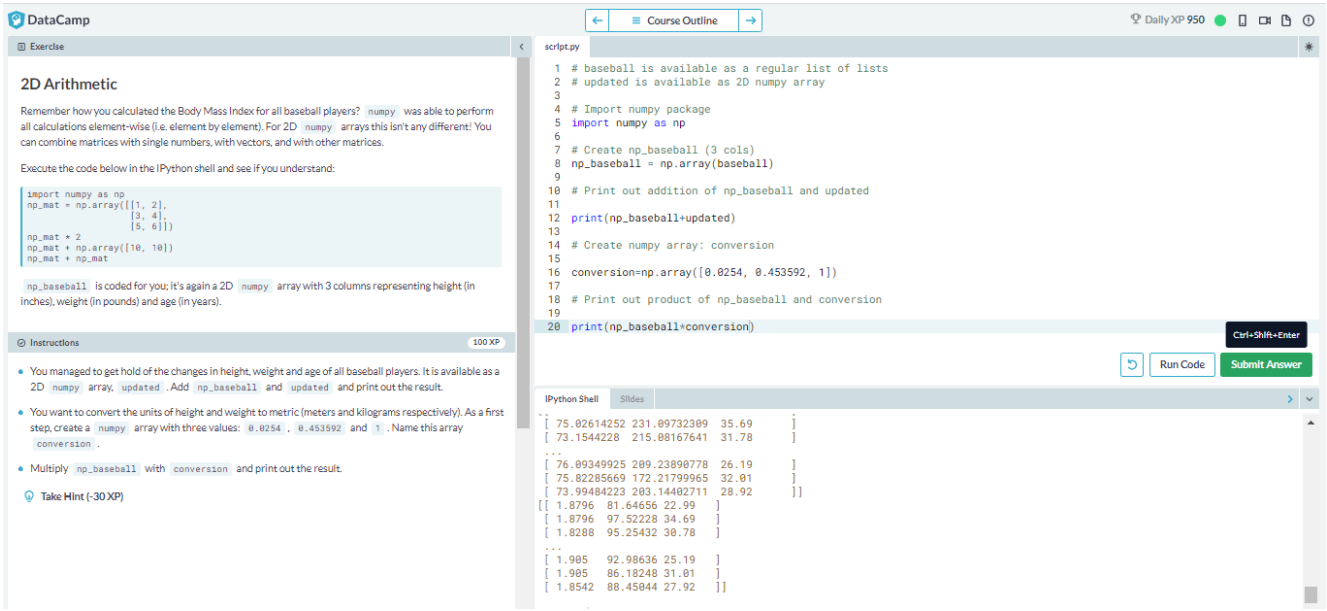
```
array([  65.4,  59.2,  63.6,  88.4,  68.7])
```

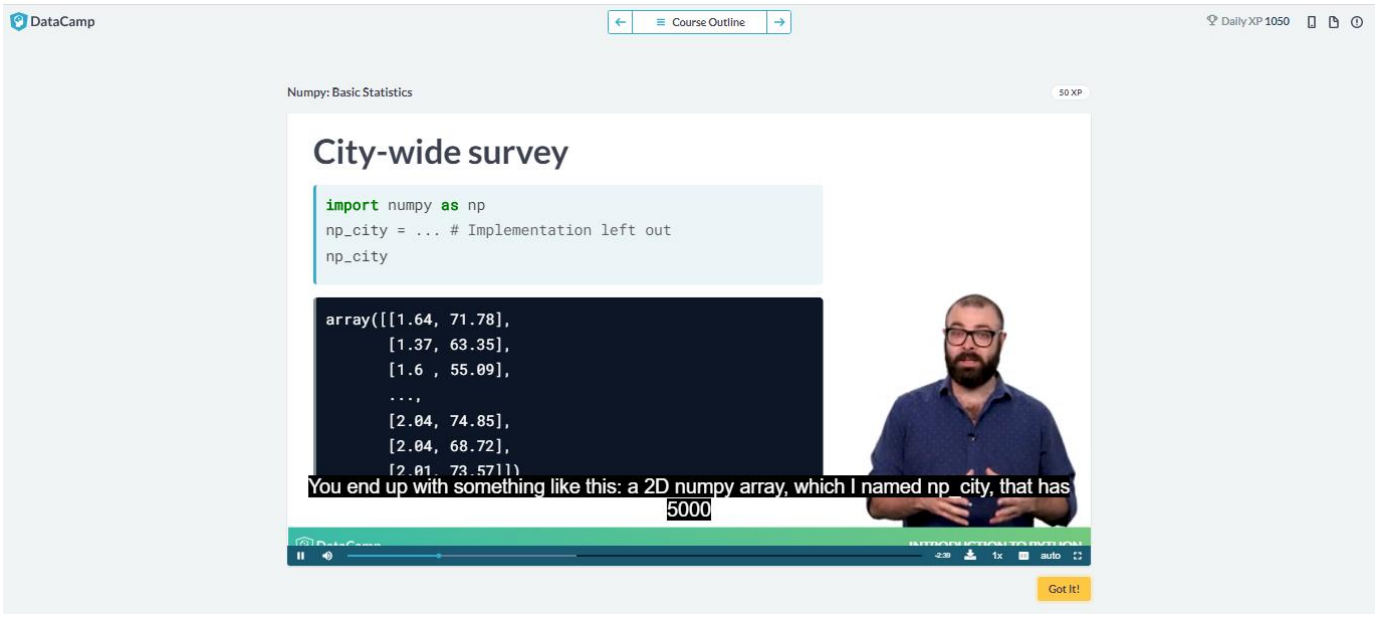
Finally, 2D numpy arrays enable you to do element-wise calculations, the same way you did it with 1D numpy arrays.

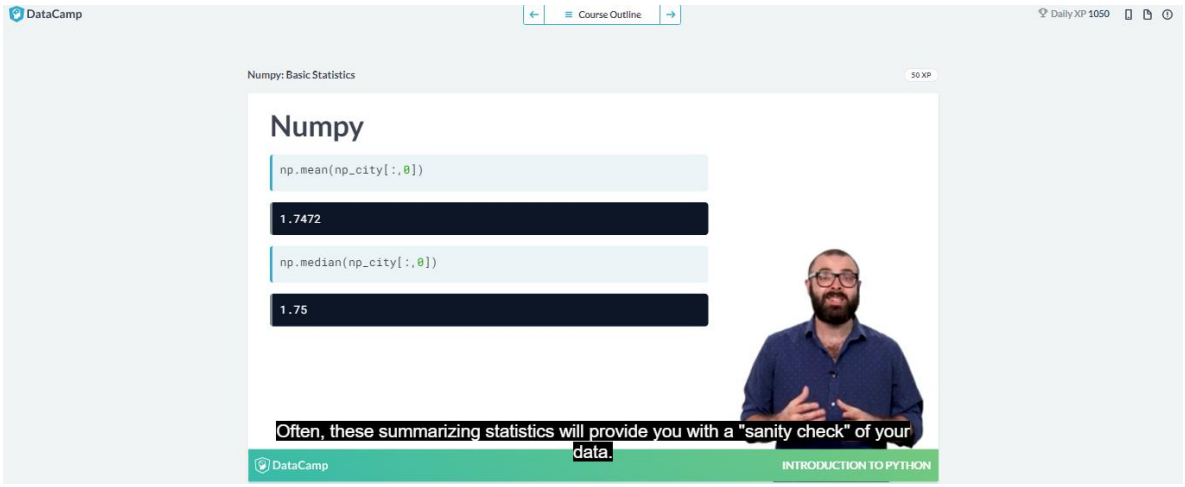

DataCamp
Course Outline
Daily XP 650
Exercise
Your First 2D NumPy Array
Before working on the actual MLB data, let's try to create a 2D numpy array from a small list of lists.
In this exercise, baseball is a list of lists. The main list contains 4 elements. Each of these elements is a list containing the height and the weight of 4 baseball players, in this order. baseball is already coded for you in the script.
Instructions
100 XP
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)
script.py
# 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)
Run Code
Submit Answer
IPython Shell
Slides
# 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 [10]:


DataCamp
Course Outline
Daily XP 750
Exercise
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() to create a 2D numpy array from baseball. Name it np_baseball.
Print out the shape attribute of np_baseball.
Take Hint (-30 XP)
Incorrect Submission
Have you used print(np_baseball.shape) to do the appropriate printouts?
Did you find this feedback helpful?
Yes
No
script.py
# 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)
Run Code
Submit Answer
IPython Shell
Slides
In [2]: # 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 [3]:


DataCamp
Course Outline
Daily XP 850
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.
# 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 numpy arrays, 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
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.
Take Hint (-30 XP)
Incorrect Submission
Did you define the variable np_weight_lb without errors?
Did you find this feedback helpful?
Yes
No
script.py
# 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])
Run Code
Submit Answer
IPython Shell
Slides
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


DataCamp
Course Outline
Daily XP 950
Exercise
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 numpy array with 3 columns representing height (in inches), weight (in pounds) and age (in years).
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.
Take Hint (-30 XP)
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=np.array([0.0254, 0.453592, 1])
# Print out product of np_baseball and conversion
print(np_baseball*conversion)
Ctrl+Shift+Enter
Run Code
Submit Answer
IPython Shell
Slides
[ 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 ]]


DataCamp
Course Outline
Daily XP 1050
Numpy: Basic Statistics
50 XP
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]])
You end up with something like this: a 2D numpy array, which I named np_city, that has 5000
-2:39
1x
auto
Got It!


DataCamp
Course Outline
Daily XP 1050
Numpy: Basic Statistics
50 XP
Numpy
np.mean(np_city[:,0])
1.7472
np.median(np_city[:,0])
1.75
Often, these summarizing statistics will provide you with a "sanity check" of your data.
DataCamp
INTRODUCTION TO PYTHON

Numpy: Basic Statistics

50 XP

# 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!

However, the big difference here is speed.



Numpy: Basic Statistics

50 XP

# Generate data

- Arguments for `np.random.normal()`
  - 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)
```

with the data in this video: We simulated it with Numpy functions!

Numpy: Basic Statistics

50 XP

# Generate data

- Arguments for `np.random.normal()`
  - 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))
```

Another awesome thing that Numpy can do!

DataCamp

Course Outline

Daily XP 1100

Exercise

## 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

- 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`.

Take Hint (-30 XP)

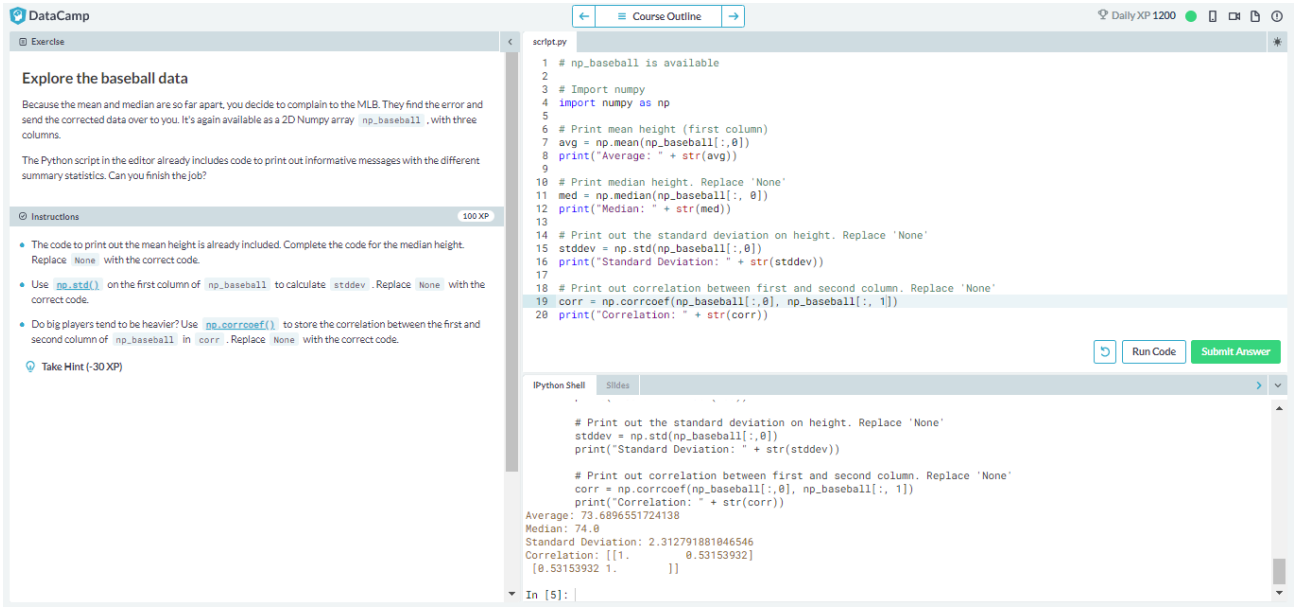
script.py

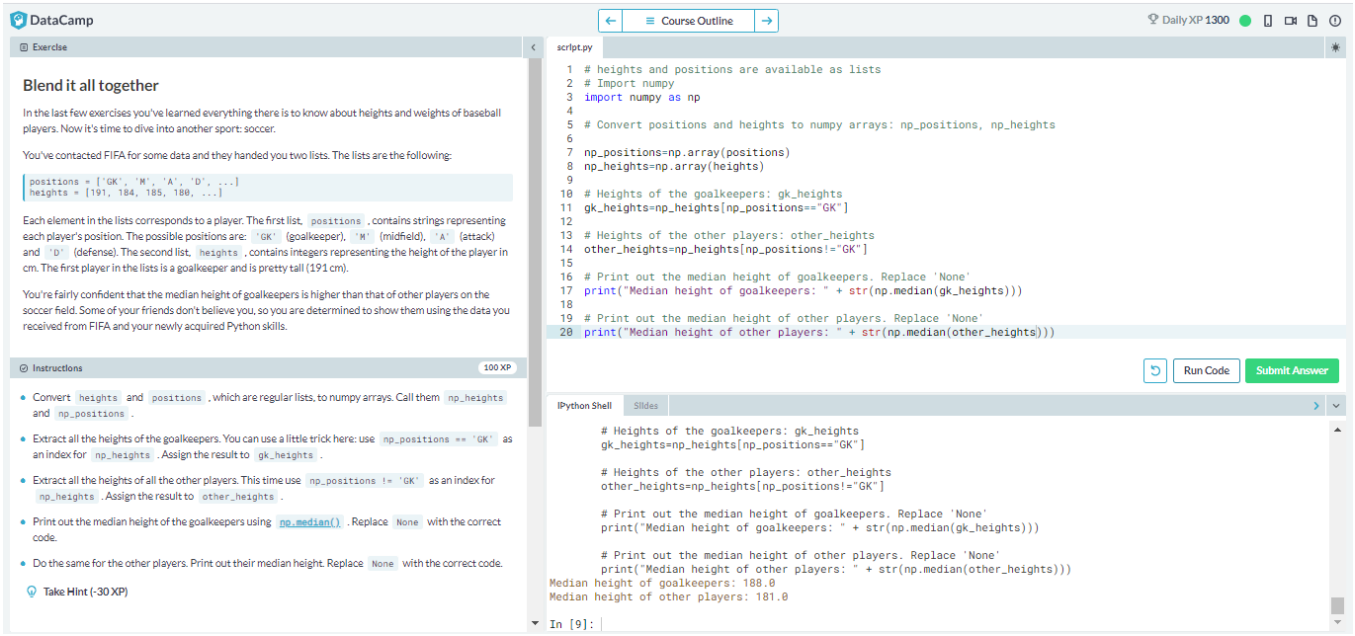
```
# np_baseball is available

# Import numpy
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))
```

Run Code | Submit Answer

IPython Shell | Slides

```
In [3]: # 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))
1586.4610837438424
74.0

In [4]:
```


DataCamp
Course Outline
Daily XP 1200
Exercise
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. Can you finish the job?
Instructions
100 XP
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.
Take Hint (-30 XP)
scrlpt.py
# 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))
Run Code
Submit Answer
IPython Shell
Slides
# 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: 73.6896551724138
Median: 74.0
Standard Deviation: 2.31279881046546
Correlation: [[1. 0.53153932]
[0.53153932 1. ]]
In [5]:


DataCamp
Course Outline
Daily XP 1300
Exercise
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
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(). Replace None with the correct code.
Do the same for the other players. Print out their median height. Replace None with the correct code.
Take Hint (-30 XP)
scrlpt.py
# 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)))
Run Code
Submit Answer
IPython Shell
Slides
# 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
In [9]: