

## Variables & Types

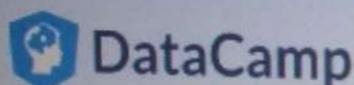


# Python Types (3)

```
In [16]: 2 + 3  
Out[16]: 5
```

```
In [17]: 'ab' + 'cd'  
Out[17]: 'abcd'
```





### Exercise

## Variable Assignment

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

```
x = 5
```

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

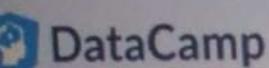
Remember, `=` in Python means *assignment*, it doesn't test equality!

### Instructions

100 XP

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

Take Hint (-30 XP)



## 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

- 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

Take Hint (-15 XP)

Submit Answer

Filip mentioned that different types behave differently in Python.

When you sum two strings, for example, you'll get different 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?

 Take Hint (-30 XP)

#### Incorrect Submission

Use `print(type(year1))` to print out the type of `year1`.

Did you find this feedback helpful?

Yes  No

#### script.py

```
2 growth_multiplier = 1.1
3 desc = "compound interest"
4
5 # Assign product of growth_multiplier and savings
6 year1=growth_multiplier*savings
7
8 # Print the type of year1
9 print(type(year1))
10
11 # Assign sum of desc and desc to doubledesc
12 doubledesc=desc+desc
13
14 # Print out doubledesc
15 print(doubledesc)
```

IPython Shell Slides

```
print(type(year1))

# Assign sum of desc and desc to doubledesc
doubledesc=desc+desc

# Print out doubledesc
print(doubledesc)
<class 'float'>
compound interestcompound interest
```

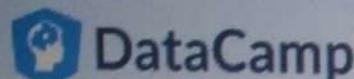
- In [3]:

## Type conversion | Python - Chromium

### Type conversion | Py x



https://campus.datacamp.com/courses/intro-to-python-for-data-science/type-conversion



#### Exercise

#### Type conversion

Using the `+` operator to paste together two strings can be very useful in building custom messages.

Suppose, for example, that you've calculated the return of your investment and want to summarize the results in a string. Assuming the floats `savings` and `result` are defined, you can try something like this:

```
print("I started with $" + savings + " and now have $" + re:
```



This will not work, though, as you cannot simply sum strings and floats.

To fix the error, you'll need to explicitly convert the types of your variables. More specifically, you'll need `str()`, to convert a value into a string. `str(savings)`, for example, will convert the float `savings` to a string.

Similar functions such as `int()`, `float()` and `bool()` will help you convert Python values into any type.

#### Instructions

- Hit Run Code to run the code on the right. Try to understand the error message.

100 XP

ence/chapter-1-python-basics?ex=12

## Course Outline

script.py

```
2 savings = 100
3 result = 100 * 1.10 ** 7
4
5 #to do str(savings)
6 #str(result)
7
8 # Fix the printout
9 print("I started with $" + str(savings) + " and now have $" + str(result)
Awesome!")
10
11 # Definition of pi_string
12 pi_string = "3.1415926"
13
14 # Convert pi_string into float: pi_float
```



Run Code

Submit

IPython Shell

Slides

```
print("I started with $" + str(savings) + " and now have $" + str(resu
Awesome!")
```

```
# Definition of pi_string
pi_string = "3.1415926"
```

```
# Convert pi_string into float: pi_float
pi_float=str(pi_string)
```

```
I started with $100 and now have $194.87171000000012. Awesome!
```

In [4]:

<https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-1-python-basics?ex=12>

taCamp

ercise

To fix the error, you'll need to explicitly convert the types of your variables. Specifically, you'll need `str()`, to convert a value into a string. `savings`, for example, will convert the float `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 on the right. Try to understand the error message.

Fix the code on the right 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`.

 Take Hint (-30 XP)

Incorrect Submission

In order to convert `pi_string` to a float, be sure to use the `float()` function.

Did you find this feedback helpful?

✓ Yes ✗ No

script.py

```
3 result = 100 * 1.10 ** 7
4
5 # to do str(savings)
6 #str(result)
7
8 # Fix the printout
9 print("I started with $" + str(savings) + " and now have"
10      "Awesome!")
11
12 # Definition of pi_string
13 pi_string = "3.1415926"
14
15 # Convert pi_string into float: pi_float
16 pi_float=float(pi_string)
```

IPython Shell Slides

```
print("I started with $" + str(savings) + " and now have"
      "Awesome!")

# Definition of pi_string
pi_string = "3.1415926"

# Convert pi_string into float: pi_float
pi_float=float(pi_string)
I started with $100 and now have $194.87171000000012. Awesome!
```

. In [5]:

## In Python handle everything?

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

[Enter](#)[Submit Answer](#)[Take Hint \(-15 XP\)](#)

### IPython Shell Slides

```
File "<stdin>", line 3, in <module>
    "The correct answer to this multiple choice exercise is answer number " + 2
TypeError: Can't convert 'int' object to str implicitly
In [2]: "I can add integers, like " + str(5) + " to strings."
...
Out[2]: 'I can add integers, like 5 to strings.'
In [3]: "I can add integers, like " + str(5) + " to strings."
... "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
File "<stdin>", line 1
    The correct answer to this multiple choice exercise is answer number " + 2
...
SyntaxError: invalid syntax
In [5]: True + False
Out[5]: 1
In [6]: "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
...
```

Lists, what are they?



Intra to Python

## Python List

[a, b, c]

```
In [7]: [1.73, 1.68, 1.71, 1.89]
Out[7]: [1.73, 1.68, 1.71, 1.89]

In [8]: fam = [1.73, 1.68, 1.71, 1.89]

In [9]: fam
Out[9]: [1.73, 1.68, 1.71, 1.89]
```



Lists, what are they?



Intro to Python

# Python List

[a, b, c]

```
In [10]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]

In [11]: fam
Out[11]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [11]: fam2 = [[{"name": "liz", "height": 1.73},
   ...:             {"name": "emma", "height": 1.68},
   ...:             {"name": "mom", "height": 1.71},
   ...:             {"name": "dad", "height": 1.89}]

In [12]: fam2
Out[12]: [[{'name': 'liz', 1.73}, {'name': 'emma', 1.68},
   ...:             {'name': 'mom', 1.71}, {'name': 'dad', 1.89}]]
```



Lists, what are they?



## List type

```
In [13]: type(fam)  
Out[13]: list
```

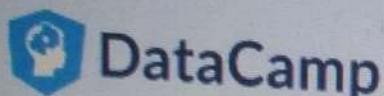
```
In [14]: type(fam2)  
Out[14]: list
```

- Specific functionality
- Specific behavior

## Create a list | Python - Chromium

Create a list | Python x

https://campus.datacamp.com/courses/intro-to-python-for-data-science/exercise-create-a-list#t-1000



### Exercise

## Create a list

As opposed to `int`, `bool` etc., a list is a **compound data type**; you can group values together:

```
a = "is"  
b = "nice"  
my_list = ["my", "list", a, b]
```

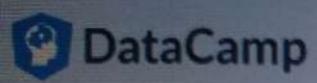
After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are stored in separate variables for now, as shown in the script.

#### 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)



## 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

#### Possible Answers

- A, B and C

## Exercise

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?



Take Hint (-30 XP)

## Incorrect Submission

Have you used `print(type(house))` to print out the type of the house variable?

Did you find this feedback helpful?

Yes  No

## script.py

```

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           ["bedroom", bed],
13           ["bathroom", bath]]
14 print(house)
15 print(type(house))

```

## IPython Shell

## Slides

```

In [1]: print(house)
       print(type(house))

# Print out house

```

```

# Print out the type of house
[[{"hallway": 11.25}, {"kitchen": 18.0}, {"living room": 20.0}, {"bedroom": 10.75}, {"bathroom": 9.5}]]
```

# Subsetting lists

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  
In [2]: fam  
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]  
index:  0      1      2      3      4      5      6      7  
       -8     -7     -6     -5     -4     -3     -2     -1  
  
In [3]: fam[3]  
Out[3]: 1.68  
  
In [4]: fam[6]           →  
Out[4]: 'dad'  
  
In [5]: fam[-1]
```

# List slicing :

```
In [7]: fam
```

```
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
          0   1   2   3   4   5   6   7
```

```
In [8]: fam[3:5]
```

```
Out[8]: [1.68, 'mom']
```

[ start : end ]

inclusive      exclusive

## Subsetting lists



Intro

# List slicing

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
         0   1   2   3   4   5   6   7

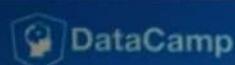
In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]

In [10]: fam[:4]
Out[10]: ['liz', 1.73, 'emma', 1.68] →

In [11]: fam[5:]
Out[11]: [1.71, 'dad', 1.89]
```

## NumPy



Intro to Python

# NumPy: remarks

```
In [19]: np.array([1.0, "is", True])  
Out[19]:  
array(['1.0', 'is', 'True'],  
      dtype='<U32')
```

NumPy arrays: contain only one type

```
In [20]: python_list = [1, 2, 3]
```

```
In [21]: numpy_array = np.array([1, 2, 3])
```

Different types: different behavior!

```
In [22]: python_list + python_list  
Out[22]: [1, 2, 3, 1, 2, 3]
```

```
In [23]: numpy_array + numpy_array  
Out[23]: array([2, 4, 6])
```



-1.07



# NumPy Subsetting

```
In [24]: bmi
```

```
Out[24]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
In [25]: bmi[1]
```

```
Out[25]: 20.975
```



```
In [26]: bmi > 23
```

```
Out[26]: array([False, False, False, True, False], dtype=bool)
```

```
In [27]: bmi[bmi > 23]
```

```
Out[27]: array([ 24.747])
```



Subset and calculate | Python - Chromium

Subset and calculate x https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-2-python-lists?ex=8

DataCamp

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`.

Take Hint (-30 XP)

script.py

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 18.75, "bathroom", 9.50]
3
4 # Sum of kitchen and bedroom area: eat_sleep_area
5 eat_sleep_area=areas[3]+areas[-3]
6
7 # Print the variable eat_sleep_area
8 print(eat_sleep_area)
```

Run all/selected code

Run Code

Submit Answer

iPython Shell Slides

```
areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 18.75, "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)
```

28.75

In [2]:

### 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()`.

 Take Hint (-30 XP)

### Incorrect Submission

`downstairs` is incorrect. Use `areas[0:6]` and slicing to select the elements you want, or something equivalent.

Did you find this feedback helpful?

Yes  No

### script.py

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0,
3           "bathroom", 9.50]
4
5 # Use slicing to create downstairs
6 downstairs=areas[1:7]
7 # Use slicing to create upstairs
8 upstairs=areas[-1:-5]
9
10 # Print out downstairs and upstairs
11 print(downstairs)
12 print(upstairs) 
```

### IPython Shell Sides

```
# Print out downstairs and upstairs
print(downstairs)
print(upstairs)
[11.25, 'kitchen', 18.0, 'living room', 28.0, 'bedroom']
```

<script.py> output:

```
[11.25, 'kitchen', 18.0, 'living room', 28.0, 'bedroom']
```

In [2]:

<https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-2-python-lists?ex=9>

DataCamp

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()`.

Take Hint (-30 XP)

Incorrect Submission

`upstairs` is incorrect. Use `areas[6:10]` and slicing to select the elements you want, or something equivalent.

Did you find this feedback helpful?

✓ Yes ✘ No

```
IPython Shell Slides
```

```
<script.py> output:
['hallway', 11.25, 'kitchen', 18.0, 'living room']
[]

<script.py> output:
['hallway', 11.25, 'kitchen', 18.0, 'living room',
 'bedroom', 10.75, 'bathroom', 9.5]
```

- In [3]:

Subsetting lists of lists

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-2-python-lists?ex=11

## DataCamp

### Exercise

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

#### Possible Answers

A float: the kitchen area

A string: "kitchen"

A float: the bathroom area

A string: "bathroom"

Submit Answer

# Changing list elements

```
In [1]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
In [2]: fam
Out[2]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
In [3]: fam[7] = 1.86
In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
In [5]: fam[0:2] = ["lisa", 1.74]
In [6]: fam
Out[6]: ['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```



2:30

# Adding and removing elements

```
In [7]: fam + ["me", 1.79]
Out[7]: ['lisa', 1.74, 'emma', 1.68,
         'mom', 1.71, 'dad', 1.86, 'me', 1.79]
In [8]: fam_ext = fam + ["me", 1.79]

In [9]: del(fam[2])

In [10]: fam
Out[10]: ['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
In [11]: del(fam[2])

In [12]: fam
Out[12]: ['lisa', 1.74, 'mom', 1.71, 'dad', 1.86]
```



4:39



ix



DataCamp

In

## Behind the scenes (1)

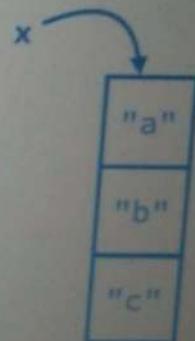
```
In [13]: x = ["a", "b", "c"]
```

```
In [14]: y = x
```

```
In [15]: y[1] = "z"
```

```
In [16]: y  
Out[16]: ['a', 'z', 'c']
```

```
In [17]: x  
Out[17]: ['a', 'z', 'c']
```



```
*Untitled Document 1 - gedit  
Open Save  
1  
2  
3|x is a reference to it
```

th: 8 ▾ Ln 3, Col 1 ▾ INS

# Behind the scenes (1)

```
In [13]: x = ["a", "b", "c"]
```

```
In [14]: y = x
```

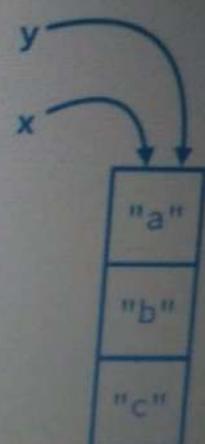
```
In [15]: y[1] = "z"
```

```
In [16]: y
```

```
Out[16]: ['a', 'z', 'c']
```

```
In [17]: x
```

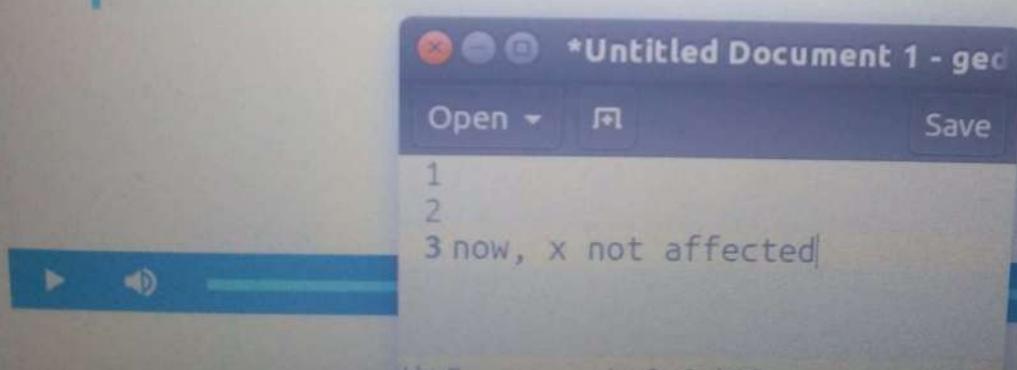
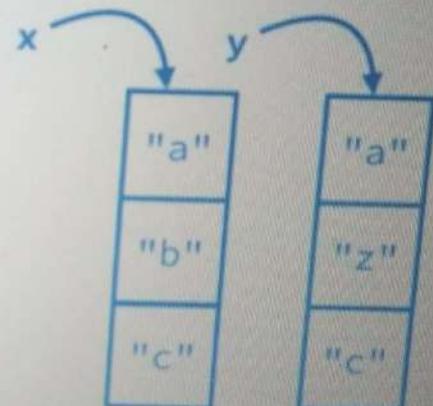
```
Out[17]: ['a', 'z', 'c']
```



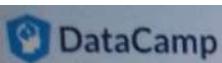
```
*Untitled Document 1 - gedit
Open Save
1
2
3|x is a reference to it
th: 8 ▾ Ln 3, Col 1 ▾ INS
```

## Behind the scenes (2)

```
In [18]: x = ["a", "b", "c"]  
In [19]: y = list(x)  
In [20]: y = x[:]  
In [21]: y[1] = "z"  
In [22]: x  
Out[22]: ['a', 'b', 'c']
```



```
*Untitled Document 1 - gec  
Open Save  
1  
2  
3 now, x not affected  
3:11 0:12
```



## Exercise

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

- 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

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen",
3           "bathroom", 9.50]
4 # Correct the bathroom area
5 areas[-1]=[10.50]
6
7 # Change "living room" to "chill zone"
8 areas[4]=["chill zone"]
9 areas
```

IPython Shell

```
11.25,
'kitchen',
18.0,
['chill zone'],
20.0,
'bedroom',
18.75,
'bathroom',
[10.5]]
```

In [3]:

In 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

- 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".

100 XP

Take Hint (-30 XP)

### Incorrect Submission

You can use `areas[-1] = 10.50` to update the bathroom area.

Did you find this feedback helpful?

Yes No

### script.py

```
1 # Create the areas list
2 areas = ["hallway", 11.25,
           "bathroom", 9.50]
3
4 # Correct the bathroom area
5 areas[-1]=10.50
6
7 # Change "living room" to "chill zone"
8 areas[4]=["chill zone"]
9 areas
```

### !Python Shell

### Slides

```
[11.25,
 'kitchen',
 18.0,
 ['chill zone'],
 20.0,
 'bedroom',
 19.75,
 'bathroom',
 10.5]
```

- In [5]:

# DataCamp

## Exercise

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

- 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)

### Incorrect Submission

You can use `areas[4] = "chill zone"` to update the bathroom area.

Did you find this feedback helpful?

Yes  No

## script.py

```
1 # Create the areas list
2 areas = ["hallway", 11.25, "kitchen", 18.0,
3           "bathroom", 9.50]
4
5 # Correct the bathroom area
6 areas[-1]=10.50
7
8 # Change "living room" to "chill zone"
9 areas[4]="chill zone"
9 areas
```

### IPython Shell Slides

```
11.25,
'kitchen',
18.0,
'chill zone',
20.0,
'bedroom',
10.75,
'bathroom',
10.5]
```

In [6]:

<https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-2-python-lists?ex=14>

dataCamp

Exercise

End of exercise

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"]
```

I 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`.

  Take Hint (-30 XP)

Incorrect Submission

Use `areas_1 + ["garage", 15.45]` to create `areas_2`. Watch out for typos!

Did you find this feedback helpful?  Yes  No

In [4]:

```
script.py
```

```
1 # Create the areas list and make some changes
2 areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,
3           "bedroom", 10.75, "bathroom", 10.50]
4
5 # Add poolhouse data to areas, new list is areas_1
6 areas_1=areas+["poolhouse",24.5]
7
8 # Add garage data to areas_1, new list is areas_2
9 areas_2=areas_1 + ["garage", 15.45]
10 #i wrote garge, therefore to check typos:spelling mistake
```

Run all/selected code

IPython Shell Slides

```
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]
#i wrote garge, therefore to check typos:spelling mistake
```

Delete list elements

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-2-python-lists?ex=15

DataCamp

Exercise

Course Outline

IPython Shell Slides

## Delete list elements

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", 10.5, "poolhouse", 24.5,
         "garage", 15.45]
```

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:

In [1]: `areas`

Out[1]:  
['hallway',  
 11.25,  
 'kitchen',  
 18.0,  
 'chill zone',  
 20.0,  
 'bedroom',  
 10.75,  
 'bathroom',  
 10.5,  
 'poolhouse',  
 24.5,  
 'garage',  
 15.45]

In [2]: `del(areas[-3]);del(areas[-4])`

In [3]: `#wrong as index changes`

## Exercise

```
command1; command2  
# 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:-5])`
- `del(areas[-3]); del(areas[-4])`

 Take Hint (-15 XP)

Submit Answer

## Incorrect Submission

`areas[10:11])` will only select the element at index 10

Did you find this feedback helpful?

Yes  No

In [5]:

```
18.0,  
'chill zone',  
20.0,  
'bedroom',  
10.75,  
'bathroom',  
10.5,  
'poolhouse',  
24.5,  
'garage',  
15.45]
```

In [2]: `del(areas[-3]);del(areas`

In [3]: `del(areas[-4:-2])`

```
In [4]: areas  
Out[4]:  
['hallway',  
11.25,  
'kitchen',  
18.0,  
'chill zone',  
20.0,  
'bedroom',  
10.75,  
'garage',  
15.45]
```

In [5]:

## Inner workings of lists

At the end of the video, Filip 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

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.

 Take Hint (-30 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
10 # Print areas
11 print(areas)
```

IPython Shell

Slides

```
# Create areas_copy
areas_copy = areas

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
```

[5.0, 18.0, 20.0, 10.75, 9.5]

In [2]:

## Inner workings of lists

At the end of the video, Filip 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

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.

Take Hint (-30 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=areas[ :]
9 #neeraj:another method
10 areas_copy=list(areas)
11
12 # Print areas
13 print(areas)
```

IPython Shell Slides

```
# Change areas_copy
areas_copy=areas[ :]
#another method
areas_copy=list(areas)

# Print areas
print(areas)
[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [4]:

## Exercise

### Inner workings of lists

At the end of the video, Filip 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

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.

 Take Hint (-30 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] = 10.0
9 print(areas)
10 #neeraj:another method
11 areas_copy1=list(areas)
12 print(areas_copy1)
13 # Print areas
14 print(areas)
```

#### iPython Shell Slides

```
print(areas)
#neeraj:another method
areas_copy1=list(areas)
print(areas_copy1)
# Print areas
print(areas)
[11.25, 18.0, 20.0, 10.75, 9.5]
[11.25, 18.0, 20.0, 10.75, 9.5]
[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [6]:

### Exercise

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.

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

### Incorrect Submission

It seems that `areas_copy` has not been updated correctly.

Did you find this feedback helpful?

Yes  No

### script.py

```
1 # Create list areas
2 areas = [11.25, 18.0, 20.0, 10.75, 9.5]
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)
```

### IPython Shell Slides

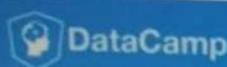
```
# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
[11.25, 18.0, 20.0, 10.75, 9.5]

<script.py> output:
[11.25, 18.0, 20.0, 10.75, 9.5]
```

- In [8]:

## Functions

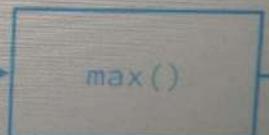


# Example

```
In [1]: fam = [1.73, 1.68, 1.71, 1.89]
```

```
In [2]: fam  
Out[2]: [1.73, 1.68, 1.71, 1.89]
```

```
In [3]: max(fam)  
Out[3]: 1.89
```

[1.73, 1.68, 1.71, 1.89] →  max() → 1.89



## Functions



# round()

```
In [6]: round(1.68, 1)  
Out[6]: 1.7
```

```
In [7]: round(1.68)  
Out[7]: 2
```

```
In [8]: help(round)      Open up documentation
```

Help on built-in function round in module builtins:

```
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.

\*Untitled Document 1 - gedit

Open ▾

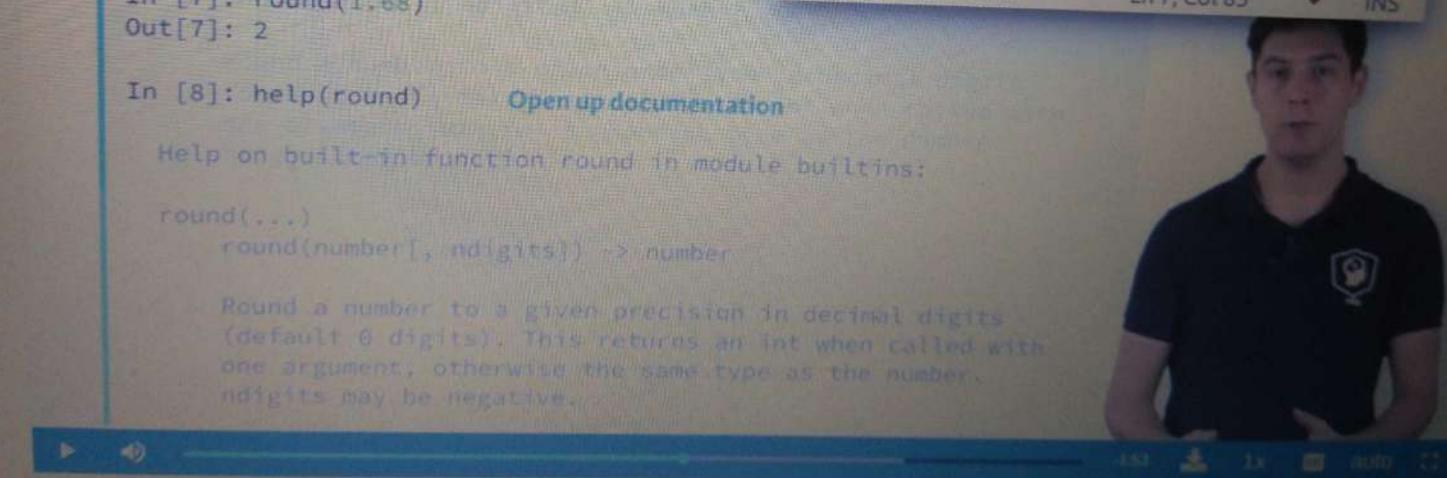
1

2

3 round takes 2 inputs:  
4 a)the no. u want to round  
5 b)precision:the no. of digits u want to  
keep after decimal

6

7 If didn't passed second argument,python  
automatically rounds off to closest  
integer|



# round()

```
In [8]: 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)
```



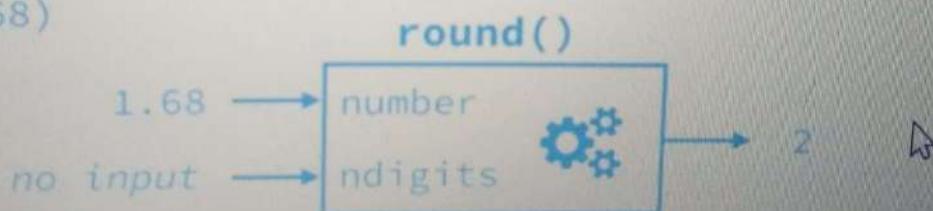
# round()

```
In [8]: help(round)
```

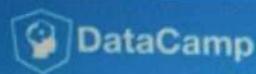
```
round(...)  
round(number[, ndigits]) -> number
```

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)
```



## Functions



Intro to i

# round()

In [8]: `help(round)`

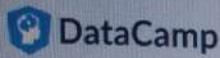
```
round(...)  
    round(number[, ndigits]) -> number
```

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(number)

round(number, ndigits)





## 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

- 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`. - In [2]:

#### 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 print(type(var1))
7
8 # Print out length of var1
9 print(len(var1))
10
11 # Convert var2 to an integer: out2
12 out2=int(var2)
13 print(out2)
```

#### IPython Shell Slides

```
<script.py> output:
<class 'list'>
4
```

```
<script.py> output:
<class 'list'>
4
1
```

## DataCamp

← Course Outline →

### Exercise

## 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

- `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 bui

```
class complex(object)  
| complex(real[, imag]) -> complex  
|  
| Create a complex number from a real  
| part.  
| This is equivalent to (real + imag*0j).  
|  
| Methods defined here:  
|  
|     __abs__(self, /)  
|         abs(self)  
|  
|     __add__(self, value, /)  
|         Return self+value.  
|  
|     __bool__(self, /)  
|         self != 0  
|  
|     __divmod__(self, value, /)  
|         Return divmod(self, value).  
|  
|     __eq__(self, value, /)  
|         Return self==value.  
|  
|     __float__(self, /)  
|         float(self)
```

```
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

- `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.

Submit Answer

 Take Hint (-15 XP)

#### Incorrect Submission

Incorrect. `[, imag]` shows that `imag` is an optional argument.

```
In [1]: help(complex)
Help on class complex in module builtins:

class complex(object)
|   complex(real=0.0, imag=0.0)
|   Create a complex number from real and imaginary components.
|   This is equivalent to (real+imagj).
|
|   Methods defined here:
|
|   __abs__(self, /)
|       abs(self)
|
|   __add__(self, value, /)
|       Return self+value.
|
|   __bool__(self, /)
|       self != 0
|
|   __divmod__(self, value, /)
|       Return divmod(self, value)
|
|   __eq__(self, value, /)
|       Return self==value
|
|   __float__(self, /)
|       float(self)
```

## 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`.

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?

Instructions

100 XP

## script.py

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

IPython Shell Slides

In [1]: `help(sorted)`

Help on built-in function sorted in module builtins:

```
sorted(iterable, key=None, reverse=False)
Return a new list containing all items from the iterable
```

A custom key function can be supplied to customise the sort order. The reverse flag can be set to request the result in descending order.

In [2]:

## Exercise

None . reverse=False means that if you don't specify the reverse argument, it will be False .

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

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

Take Hint (-30 XP)

## script.py

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

### IPython Shell Slides

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

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

# Print out full_sorted
print(full_sorted)
[20.0, 18.0, 11.25, 10.75, 9.5]
```

. In [3]:



# Back 2 Basics

```
In [1]: sister = "liz"           type  
                                Object  str  
  
In [2]: height = 1.73           type  
                                Object  float  
  
In [3]: fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  type  
                                Object  list
```

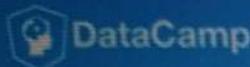
Methods: Functions that belong to objects



00:00 / 00:00

33

## Methods



Intro to Pyth

# Back 2 Basics

		type	examples of methods
In [1]:	sister = "liz"	Object str	capitalize() replace()
In [2]:	height = 1.73	Object float	bit_length() conjugate()
In [3]:	fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]	Object list	index() count()

Methods: Functions that belong to objects



# list methods

```
In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
In [5]: fam.index("mom")  
Out[5]: 4  
"Call method index() on fam"
In [6]: fam.count(1.73)
Out[6]: 1
```



## Methods



DataCamp

Intro

# str methods

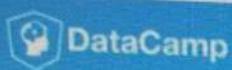
```
In [7]: sister  
Out[7]: 'liz'
```

```
In [8]: sister.capitalize()  
Out[8]: 'Liz'
```

```
In [9]: sister.replace("z", "s")  
Out[9]: 'lisa'
```



## Methods



Intro to Py

# Methods

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

```
In [10]: sister.replace("z", "sa")
Out[10]: 'lisa'

In [11]: fam.replace("mom", "mommy")
AttributeError: 'list' object has no attribute 'replace'

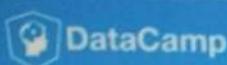
In [12]: sister.index("z")
Out[12]: 2

In [13]: fam.index("mom")
Out[13]: 4
```



1/27 1/1

## Methods



Intro to Python

# Methods (2)

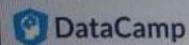
```
In [14]: fam
Out[14]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [15]: fam.append("me")

In [16]: fam
Out[16]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']

In [17]: fam.append(1.79)
          ^
In [18]: fam
Out[18]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me', 1.79]
```





## 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)

## Incorrect Submission

You have calculated the number of o's in `place` fine; now make sure to wrap `place.count('o')` call in a `print()` function to print out the result.

## script.py

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

## IPython Shell

```
print(place.count("o"))
```

```
poolhouse
```

```
POOLHOUSE
```

```
3
```

```
<script.py> output:
```

```
poolhouse
```

```
POOLHOUSE
```

```
3
```

```
In [5]:
```

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-3-functions-and-packages?ex=7

## DataCamp

### Exercise

## List Methods

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.

Take Hint (-30 XP)

```
script.py
```

```
1 # Create list areas
2 areas = [11.25, 18.0, 20.0, 10.75, 9.50]
3
4 # Print out the index of the element 20.0
5 print(areas.index(20.0))
6
7 # Print out how often 9.50 appears in areas
8 print(areas.count(9.50))
9
```

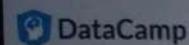
IPython Shell Slides

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

```
2
1
```

```
<script.py> output:
2
1
```

In [2]:



## Exercise

## List Methods (2)

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

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

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

```
areas .
```

## Instructions

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

```
1 # Create list areas
2 areas = [11.25, 18.0, 20.0, 10.75, 9.50]
3
4 # Use append twice to add poolhouse and garage size
5 areas.append(24.5)
6 areas.append(15.45)
7
8
9 # Print out areas
10 print(areas)
11
12 # Reverse the orders of the elements in areas
13 areas.reverse()
14 print(areas)
```

IPython Shell Slides

```
print(areas)
```

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

```
# Print out areas
```

```
[11.25, 18.0, 20.0, 10.75, 9.5, 24.5, 15.45]
[15.45, 24.5, 9.5, 10.75, 20.0, 18.0, 11.25]
```

In [2]:

## Packages



Intro to Python

# 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  
...



## Install package

- <http://pip.readthedocs.org/en/stable/installing/>
- Download get-pip.py
- Terminal:
  - python3 get-pip.py
  - pip3 install numpy



## Packages



Intro to

# Import package

```
In [1]: import numpy  
  
In [2]: array([1, 2, 3])  
NameError: name 'array' is not defined  
  
In [3]: numpy.array([1, 2, 3])  
Out[3]: array([1, 2, 3])  
  
In [4]: import numpy as np  
  
In [5]: np.array([1, 2, 3])  
Out[5]: array([1, 2, 3])  
  
In [6]: from numpy import array  
  
In [7]: array([1, 2, 3])  
Out[7]: array([1, 2, 3])
```



## Packages



# from numpy import array

```
my_script.py
from numpy import array

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 = array(fam_ext)      Using Numpy, but not very clear
```



# import numpy

my\_script.py

```
import numpy

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 = numpy.array(fam_ext)
```

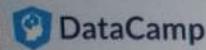
Clearly using Numpy



## Import package | Python - Chromium

Import package | Py x

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-3-functions-and-packages?ex=10



### Exercise

$$C = 2\pi r$$

$$A = \pi r^2$$

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`.

Take Hint (-30 XP)

script.py

```
2 r = 0.43
3
4 # Import the math package
5 import math
6
7 # Calculate C
8 C = 2*math.pi*r
9
10 # Calculate A
11 A = 0
12
13 # Build printout
14 print("Circumference: " + str(C))
15 print("Area: " + str(A))
```

IPython Shell Slides

In [1]:

## Exercise

$$C = 2\pi r$$

$$A = \pi r^2$$

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`.

 Take Hint (-30 XP)

### script.py

```
2 r = 0.43
3
4 # Import the math package
5 import math
6
7 # Calculate C
8 C = 2*(math.pi)*r
9
10 # Calculate A
11 A = (math.pi)*r**2
12
13 # Build printout
14 print("Circumference: " + str(C))
15 print("Area: " + str(A))
```

### IPython Shell

```
# Build printout
print("Circumference: " + str(C))
print("Area: " + str(A))
Circumference: 2.701769682087222
Area: 0.5808804816487527
```

<script.py> output:

```
Circumference: 2.701769682087222
Area: 0.5808804816487527
```

- In [2]:

## 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 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.
- Printout `dist`.

 Take Hint (-30 XP)

#### script.py

```
1 # Definition of radius
2 r = 192500
3
4 # Import radians function of math module
5 from math import radians
6
7
8 # Travel distance of Moon over 12 degrees
9 dist=r*radians(12)
10
11 # Print out dist
12 print(dist)
```

#### IPython Shell

#### Slides

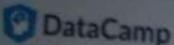
```
# Travel distance of Moon over 12 degrees
dist=r*radians(12)
```

```
# Print out dist
print(dist)
```

40317.10572106901

```
<script.py> output:
40317.10572106901
```

In [7]:



Exercise

Course Outline

IPython Shell

Slides

## Different ways of importing

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`

Take Hint (-15 XP)

Submit Answer

```
In [1]: from scipy import linalg
```

```
In [2]: my_inv([[1,2], [3,4]])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    my_inv([[1,2], [3,4]])  
NameError: name 'my_inv' is not defined
```

```
In [3]: from scipy import linalg as my_inv
```

```
In [4]: my_inv([[1,2], [3,4]])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    my_inv([[1,2], [3,4]])  
TypeError: 'module' object is not callable
```

In [5]:

## Different ways of importing

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  
  
④ from scipy.linalg import inv as my_inv
```

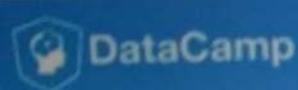
Take Hint (-15 XP)

Submit Answer

```
File "<stdin>", line 1, in <module>  
    my_inv([[1,2], [3,4]])  
TypeError: 'module' object is not callable  
  
In [9]: from scipy.linalg import my_inv  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    from scipy.linalg import my_inv  
ImportError: cannot import name 'my_inv'  
  
In [10]: from scipy.linalg import inv as my_inv()  
  File "<stdin>", line 1  
    from scipy.linalg import inv as my_inv()  
SyntaxError: invalid syntax  
  
In [11]: my_inv([[1,2], [3,4]])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    my_inv([[1,2], [3,4]])  
TypeError: 'module' object is not callable  
  
In [12]: from scipy.linalg import inv as my_inv  
  
In [13]: my_inv([[1,2], [3,4]])  
Out[13]:  
array([[-2.,  1.,  1.,  
       1.,  5., -0.5]])  
  
In [14]:
```



umPy



# Illustration

```
In [1]: height = [1.73, 1.68, 1.71, 1.89, 1.79]
```

```
In [2]: height  
Out[2]: [1.73, 1.68, 1.71, 1.89, 1.79]
```

```
In [3]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]
```

```
In [4]: weight  
Out[4]: [65.4, 59.2, 63.6, 88.4, 68.7]
```



```
In [5]: weight / height ** 2  
TypeError: unsupported operand type(s) for **: 'list' and 'int'
```





Course Outline



## NumPy



Intro

# Solution: NumPy

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





## NumPy



# Solution: NumPy

- Numeric Python
- Alternative to Python List: NumPy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`



# NumPy

```
In [6]: import numpy as np
```

Element-wise calculation

```
In [7]: np_height = np.array(height)
```

```
In [8]: np_height
```

```
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [9]: np_weight = np.array(weight)
```



```
In [10]: np_weight
```

```
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
In [11]: bmi = np_weight / np_height ** 2
```

```
In [12]: bmi
```

```
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```



# NumPy

```
In [6]: import numpy as np
```

Element-wise calculation

```
In [7]: np_height = np.array(height)
```

```
In [8]: np_height
```

```
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [9]: np_weight = np.array(weight)
```

```
In [10]: np_weight
```

```
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
In [11]: bmi = np_weight / np_height ** 2
```

```
In [12]: bmi
```

```
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
= 65.5/1.73 ** 2
```

## NumPy



# NumPy: remarks

```
In [19]: np.array([1.0, "is", True])
Out[19]:
array(['1.0', 'is', 'True'],
      dtype='|<U32')
```

NumPy arrays: contain only one type

\*Untitled Document 1 - gedit

Open Save

\*Untitled Document 1 4.c

```
1
2
3 NumPy arrays: contain only one type
4 Therefore, float and boolean converted to strings
```

n Text Tab Width: 8 Ln 4, Col 10 INS

```
In [20]: python_list = [1, 2, 3]
In [21]: numpy_array = np.array([1, 2, 3])
In [22]: python_list + python_list
Out[22]: [1, 2, 3, 1, 2, 3]
In [23]: numpy_array + numpy_array
Out[23]: array([2, 4, 6])
```

```
In [20]: python_list = [1, 2, 3]
```

```
In [21]: numpy_array = np.array([1, 2, 3])
```

```
In [22]: python_list + python_list  
Out[22]: [1, 2, 3, 1, 2, 3]
```

Different types: different behavior!

```
In [23]: numpy_array + numpy_array  
Out[23]: array([2, 4, 6])
```

# NumPy Subsetting

```
In [24]: bmi
```

```
Out[24]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
In [25]: bmi[1]
```

```
Out[25]: 20.975
```

```
In [26]: bmi > 23
```

```
Out[26]: array([False, False, False, True, False], dtype=bool)
```

```
In [27]: bmi[bmi > 23]
```

```
Out[27]: array([ 24.747])
```



Your First NumPy

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=2

DataCamp

Exercise

Course Outline

## 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)

script.py

```
1 # Create list baseball
2 baseball = [180, 215, 210, 210, 188, 176, 209, 200]
3
4 # Import the numpy package as np
5 import numpy as np
6
7 # Create a numpy array from baseball: np_baseball
8 np_baseball=np.array(baseball)
9
10 # Print out type of np_baseball
11 print(type(np_baseball))
```

IPython Shell Slides

```
# 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'>

<script.py> output:
<class 'numpy.ndarray'>
```

. In [2]:

## Baseball players' height

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

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

### Instructions

- Create a `numpy` array from `height_in`. Name this new array `np_height_in`.
- Print `np_height_in`.
- Multiply `np_height_in` with `0.0254` to convert all height measurements from inches to meters. Store the new values in a new array, `np_height_m`.
- Print out `np_height_m` and check if the output makes sense.

 Take Hint (-30 XP)

100 XP

script.py

```
1 # height is available as a regular list
2
3 # Import numpy
4 import numpy as np
5
6 # Create a numpy array from height_in:
7
8
9 # Print out np_height_in
10
11
12 # Convert np_height_in to m: np_height_m
13
14
```

IPython Shell Slides

```
In [1]: height_in
Out[1]:
[74,
 74,
 72,
 72,
 73,
 69,
 69,
 71,
 76]
```

## Baseball players' height

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

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

### Instructions

Create a `numpy` array from `height_in`. Name this new array `np_height_in`.

Print `np_height_in`.

Multiply `np_height_in` with `0.0254` to convert all height measurements from inches to meters. Store the new values in a new array, `np_height_m`.

Print out `np_height_m` and check if the output makes sense.

Take Hint (-30 xp)

```
script.py
```

```
3 # Import numpy
4 import numpy as np
5
6 # Create a numpy array from height_in: np_height_in
7 np_height_in=np.array(height_in)
8
9 # Print out np_height_in
10 print(np_height_in)
11
12 # Convert np_height_in to m: np_height_m
13 np_height_m=np_height_in*0.0254
14
15 # Print np_height_m
16 print(np_height_m)
```

### iPython Shell

```
[72, 77, 79, 78, ...]
```

<script.py> output:

```
[74 74 72 ..., 75 75 73]
[ 1.8796  1.8796  1.8288 ...,  1.905   1.905   1.8542]
```

- In [2]:

## Baseball player's BMI

The MLB also offers to let you analyze their weight data. Again, both are available as regular Python lists: `height_in` and `weight`. `height_in` is in inches and `weight_lb` is in pounds.

It's now possible to calculate the BMI of each baseball player. Python code to convert `height_in` to a 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

```
3 # Import numpy
4 import numpy as np
5
6 # Create array from height_in with metric units:
7 np_height_m = np.array(height_in) * 0.0254
8
9 # Create array from weight_lb with metric units:
10 np_weight_kg=np.array(weight_lb)*0.453592
11
12 # Calculate the BMI: bmi
13 bmi=np_weight_kg/np_height_m**2
14
15 # Print out bmi
16 print(bmi)
```

IPython Shell Slides

```
# Print out bmi
print(bmi)
[ 23.11037639  27.60406069  28.48080465 ...,  25.62295933  23.
 25.72686361]

<script.py> output:
[ 23.11037639  27.60406069  28.48080465 ...,  25.62295933
 25.72686361]

- In [2]:
```

## Lightweight baseball players

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

```
x = [4, 9, 6, 3, 1]
x[1]
import numpy as np
y = np.array(x)
y[1]
```

For `numpy` specifically, you can also use boolean `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`.

script.py

```
6 # Calculate the BMI: bmi
7 np_height_m = np.array(height_in)
8 np_weight_kg = np.array(weight_lb)
9 bmi = np_weight_kg / np_height_m
10
11 # Create the light array
12 light=np.array(bmi<21)
13
14 # Print out light
15 print(light)
16
17
18 # Print out BMIs of all baseball players
19 print(light[light<21])
```

IPython Shell Slides

20.34343189 20.69282047 20.15883472 19.9205219 ]

<script.py> output:

```
[ 23.11037639 27.60405069 28.48880465
 25.72686361]
[ 20.54255679 20.54255679 20.69282047
 20.34343189 20.69282047 20.15883472
 20.9205219 ]
```

In [5]:

## DataCamp

Exercise

specifically, you can also use boolean numpy arrays:

```
high = y > 5  
[high]
```

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

rections

100 XP

ate a boolean numpy array: the element of the array should be  
true if the corresponding baseball player's BMI is below 21. You can  
the < operator for this. Name the array light .

nt the array light .

nt out a numpy array with the BMIs of all baseball players whose BMI  
below 21. Use light inside square brackets to do a selection on the  
array.



Take Hint (-30 XP)

rect Submission

bmi < 21 to define light

ou find this feedback helpful?

✓ Yes ✗ No

script.py

```
6 # Calculate the BMI: bmi  
7 np_height_m = np.array(height_in) * 0.0254  
8 np_weight_kg = np.array(weight_lb) * 0.453592  
9 bmi = np_weight_kg / np_height_m ** 2  
10  
11 # Create the light array  
12 light=np.array(bmi<21)  
13  
14 # Print out light  
15 print(light)  
16  
17  
18 # Print out BMIs of all baseball players whose  
19 print(bmi[light])
```

IPython Shell Slides

```
[ 20.54255679  20.54255679  20.69282047  20.69282047  
 20.34343189  20.69282047  20.15883472  19.49844721  
 20.9205219 ]
```

<script.py> output:

```
[False False False ..., False False False]  
[ 20.54255679  20.54255679  20.69282047  20.69282047  
 20.34343189  20.69282047  20.15883472  19.49844721  
 20.9205219 ]
```

. In [5]:

NumPy Side Effects

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=6

DataCamp

Exercise

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)

NumPy Side Effects | Python - Chromium

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=6

DataCamp Course Outline IPython Shell Slides

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

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

In [2]:

+50 XP

Great job! `True` is converted to 1, `False` is converted to 0.

PRESS ENTER TO

Continue

Become a power user!

SUBMIT ANSWER: `CTRL + SHIFT + ENTER`

See all keyboard shortcuts

This screenshot shows a DataCamp Python course exercise titled "NumPy Side Effects". The exercise involves executing the following code in an IPython shell:

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

The output of this code is:

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

After executing the code, a message appears indicating that the user has completed the exercise successfully:

Great job! `True` is converted to 1, `False` is converted to 0.

A green button labeled "Continue" is present, along with a "PRESS ENTER TO" instruction above it. A "Become a power user!" section at the bottom provides keyboard shortcuts for submitting answers:

SUBMIT ANSWER: `CTRL + SHIFT + ENTER`

[See all keyboard shortcuts](#)

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=7

## DataCamp

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

Instructions 100 XP

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

Take Hint (-30 XP)

```
script.py
```

```
1 # height and weight are available as a regular lists
2
3 # Import numpy
4 import numpy as np
5
6 # Store weight and height lists as numpy arrays
7 np_weight_lb = np.array(weight_lb)
8 np_height_in = np.array(height_in)
9
10 # Print out the weight at index 50
11 print(np_weight_lb[50])
12
13 # Print out sub-array of np_height_in: index 100 up to and
14 print(np_height_in[100:111])
```

Run all/ Run

IPython Shell Slides

```
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
print(np_height_in[100:111])
```

200 [73 74 72 73 69 72 73 75 75 73 72]

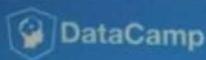
In [4]:

# Type of NumPy Arrays

```
In [1]: import numpy as np  
  
In [2]: np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])  
  
In [3]: np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])  
  
In [4]: type(np_height)  
Out[4]: numpy.ndarray  
  
In [5]: type(np_weight)  
Out[5]: numpy.ndarray
```

ndarray = N-dimensional array

## 2D NumPy Arrays



Intro to Pyth

# 2D NumPy Arrays

```
In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
[65.4, 59.2, 63.6, 88.4, 68.7]])
```

```
In [7]: np_2d  
Out[7]:  
array([[ 1.73, 1.68, 1.71, 1.89, 1.79],  
[ 65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

```
In [8]: np_2d.shape  
Out[8]: (2, 5) 2 rows, 5 columns
```

```
In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
[65.4, 59.2, 63.6, 88.4, "68.7"]])  
Out[9]:  
array([[1.73, '1.68', '1.71', '1.89', '1.79'],  
['65.4', '59.2', '63.6', '88.4', '68.7']],  
dtype='<U32') Single type!
```

-136

# 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
```

```
In [10]: np_2d[0]  
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])  
  
In [11]: np_2d[0][2]  
Out[11]: 1.71  
  
In [12]: np_2d[0,2]  
Out[12]: 1.71
```



Got it!

## 2D NumPy Arrays

The screenshot shows a DataCamp course slide titled "Subsetting". The slide includes a code editor with Jupyter Notebook-style syntax highlighting and a preview of a 2D NumPy array.

**Code Examples:**

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [11]: np_2d[0][2]
Out[11]: 1.71

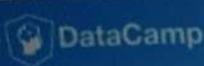
In [12]: np_2d[0,2]
Out[12]: 1.71

In [13]: np_2d[:,1:3]
Out[13]:
array([[ 1.68,  1.71],
       [ 59.2 ,  63.6 ]])
```

**Preview of 2D NumPy Array:**

	0	1	2	3	4
0	1.73	1.68	1.71	1.89	1.79
1	65.4	59.2	63.6	88.4	68.7

A male instructor is visible on the right side of the slide, and a "Got it!" button is at the bottom right.



# Subsetting

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

```
In [10]: np_2d[0]  
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])
```

```
In [11]: np_2d[0][2]  
Out[11]: 1.71
```

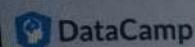
```
In [12]: np_2d[0,2]  
Out[12]: 1.71
```

```
In [13]: np_2d[:,1:3]  
Out[13]:  
array([[ 1.68,  1.71],  
       [ 59.2 ,  63.6 ]])
```

```
In [14]: np_2d[1,:]  
Out[14]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
*Untitled Document 1 - gedit  
Open ▾  
*Untitled Document 1  
1  
2  
3 np_2d[1,:]  
4 as we want 1 st row  
5 and all column elements[:,:]  
n Text ▾ Tab Width: 8 ▾ Ln 5, Col 27
```

Got it!



## 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

```
1 # Create baseball, a list of lists
2 baseball = [[180, 78.4],
3               [215, 102.7],
4               [210, 98.5],
5               [188, 75.2]]
6
7 # Import numpy
8 import numpy as np
9
10 # Create a 2D numpy array from baseball: np_baseball
11 np_baseball=np.array(baseball)
12
13 # Print out the type of np_baseball
14 print(type(np_baseball))
```

#### IPython Shell

```
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.shape(np_baseball))
<class 'list'>
(4, 2)

. In [2]:
```

Your First 2D NumPy Array | Python - Chromium

Your First 2D NumPy Array

<https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=9>

DataCamp

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`.

(x) 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

```
1 # Create baseball, a list of lists
2 baseball = [[180, 78.4],
3               [215, 102.7],
4               [210, 98.5],
5               [188, 75.2]]
6
7 # Import numpy
8 import numpy as np
9
10 # Create a 2D numpy array from baseball: np_baseball
11 np_baseball=np.array(baseball)
12
13 # Print out the type of np_baseball
14 print(type(np_baseball))
```

IPython Shell Slides

```
# Print out the shape of np_baseball
print(np.shape(baseball))
<class 'numpy.ndarray'>
(4, 2)

<script.py> output:
<class 'numpy.ndarray'>
(4, 2)
```

In [3]:

## 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)

### script.py

```
1 # baseball is available as a regular list o
2
3 # Import numpy package
4 import numpy as np
5
6 # Create a 2D numpy array from baseball: np_
7 np_baseball=np.array(baseball)
8
9 # Print out the shape of np_baseball
10 print(np.shape(np_baseball))
```

### IPython Shell Slides

```
In [3]: np_baseball
Out[3]:
array([[ 74, 180],
       [ 74, 215],
       [ 72, 210],
       [ 75, 205],
       [ 75, 190],
       [ 73, 195]])
```

In [4]:

## Camp

ise

### baseball data in 2D form

We another look at the MLB data and realize that it makes more sense to structure 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).

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`.

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

structions

100 XP

use `np.array()` to create a 2D numpy array from `baseball`. Name it `np_baseball`.

Print out the `shape` attribute of `np_baseball`.

Take Hint (-30 XP)

script.py

```
1 # baseball is available as a regular list of lists
2
3 # Import numpy package
4 import numpy as np
5
6 # Create a 2D numpy array from baseball: np_baseball
7 np_baseball=np.array(baseball)
8
9 # Print out the shape of np_baseball
10 print(np.shape(np_baseball))
```

IPython Shell Slides

`np_baseball=np.array(baseball)`

# Print out the shape of np\_baseball  
`print(np.shape(np_baseball))`

(1015, 2)

In [2]: `baseball`

Out[2]:

`[[74, 180],  
 [74, 215],  
 [72, 210],  
 [72, 210],  
 [73, 188],  
 [69, 176],`

## 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

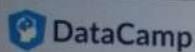
```
3 # Import numpy package
4 import numpy as np
5
6 # Create np_baseball (2 cols)
7 np_baseball = np.array(baseball)
8
9 # Print out the 50th row of np_baseball
10 print(np_baseball[50,:])
11
12 # Select the entire second column of np_baseball
13 np_weight_lb=np_baseball[:,2]
14
15 # Print out height of 124th player
16 print(np_baseball[[124][1]])
```

IPython Shell Slides

```
np_weight_lb=np_baseball[:,2]
IndexError: index 2 is out of bounds for axis 1 with size 2

<script.py> output:
[ 73 200]
Traceback (most recent call last):
  File "script.py", line 13, in <module>
    np_weight_lb=np_baseball[:,2]
IndexError: index 2 is out of bounds for axis 1 with size 2
```

, In [2]:



### Exercise

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

Have you used `print(np_baseball[123, 0])` to do the appropriate printouts?

Did you find this feedback helpful?

Yes  No

script.py

```
3 # Import numpy package
4 import numpy as np
5
6 # Create np_baseball (2 cols)
7 np_baseball = np.array(baseball)
8
9 # Print out the 50th row of np_baseball
10 print(np_baseball[49,:])
11
12 # Select the entire second column of np_baseball: np_weight_lb
13 np_weight_lb=np_baseball[:,1]
14
15 # Print out height of 124th player
16 print(np_baseball[[123][0]])
```

IPython Shell Slides

```
[ 70 195]
Traceback (most recent call last):
File "script.py", line 16, in <module>
    print(np_baseball[[124][1]])
IndexError: list index out of range

<script.py> output:
[ 70 195]
[ 75 200]
```

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=12

DataCamp

Exercise

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

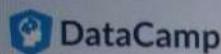
```
import numpy as np
6
7 # Create np_baseball (3 cols)
8 np_baseball = np.array(baseball)
9
10 # Print out addition of np_baseball and updated
11 print(np_baseball+updated)
12
13 # Create numpy array: conversion
14 conversion=np.array([0.0254,0.453592,1])
15
16 # Print out product of np_baseball and conversion
17 print(np_baseball*conversion)
```

Run Code

IPython Shell Slides

```
# Print out product of np_baseball and conversion
print(np_baseball*conversion)
[[ 75.2303559  168.83775102  23.99   ]
 [ 75.02614252 231.09732309  35.69   ]
 [ 73.1544228  215.08167641  31.78   ]
 ...
 [ 76.09349925 209.23890778  26.19   ]
 [ 75.82285669 172.21799965  32.81   ]
 [ 73.99484223 203.14482711  28.92   ]
 [ 1.8796    81.64656   22.99   ]
 [ 1.8796    97.52228   34.69   ]
 [ 1.8288    95.25432   38.78   ]]
```

<https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=12>



### Exercise

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`.

script.py

IPython Shell Slides

```
In [1]: import numpy as np
... np_mat = np.array([[1, 2],
...                   [3, 4],
...                   [5, 6]])
```

```
In [2]: np_mat * 2
Out[2]:
array([[ 2,  4],
       [ 6,  8],
       [10, 12]])
```

```
In [3]: np_mat + np.array([10, 10])
Out[3]:
array([[11, 12],
       [13, 14],
       [15, 16]])
```

```
In [4]: # baseball is available as a regular
# updated is available as 2D numpy a
# Import numpy package
import numpy as np
```

## NumPy: Basic Statistics



Intro to Python for Data Science

# NumPy

```
In [4]: np.mean(np_city[:,0])  
Out[4]: 1.7472
```

```
In [5]: np.median(np_city[:,0])  
Out[5]: 1.75
```

```
In [6]: np.corrcoef(np_city[:,0], np_city[:,1])  
Out[6]:  
array([[ 1.        , -0.01802],  
       [-0.01803,  1.        ]])
```

```
In [7]: np.std(np_city[:,0])  
Out[7]: 0.1992
```

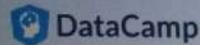
The screenshot shows a Jupyter Notebook interface with a blue header bar. On the right, a man in a black polo shirt with a DataCamp logo is visible. The main area displays four code cells (In [4] to In [7]) and their corresponding outputs. Below the cells is a text box containing explanatory notes about the results. The interface includes standard Jupyter controls like back/forward arrows, a search bar, and a file menu.

```
File Edit View Search Tools Documents Help  
Open Save  
1 mean: it appears on average people are 1.745 metres tall  
2 median: height of middle person if start from small to tall  
3 corrcoef: to check if height and weight are co-related  
4 std : standard deviation|
```

# Generate data

distribution  
mean      distribution  
standard dev.      number of  
samples

```
In [8]: height = np.round(np.random.normal(1.75, 0.20, 5000), 2)
In [9]: weight = np.round(np.random.normal(60.32, 15, 5000), 2)
In [10]: np_city = np.column_stack((height, weight))
```



## 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

```
1 # np_baseball is available
2
3 # Import numpy
4 import numpy as np
5
6 # Create np_height_in from np_baseball
7 np_height_in=np.array(np_baseball[:,0])
8
9 # Print out the mean of np_height_in
10 print(np.mean(np_height_in))
11
12 # Print out the median of np_height_in
13 print(np.median(np_height_in))
```

## IPython Shell

## Slides

&lt;script.py&gt; output:

1586.46108374  
74.0

In [1]:

Average versus median | Python - Chromium

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=14

DataCamp

+100 XP

An average height of 1586 inches, that doesn't sound right, does it? However, the median does not seem affected by the outliers: 74 inches makes perfect sense. It's always a good idea to check both the median and the mean, to get an idea about the overall distribution of the entire dataset.

Please rate this exercise:

PRESS ENTER TO

Continue

Become a power user!

SUBMIT ANSWER: CTRL + SHIFT + ENTER

script.py

```
1 # np_baseball is available
2
3 # Import numpy
4 import numpy as np
5
6 # Create np_height_in from np_baseball
7 np_height_in=np.array(np_baseball[:,0])
8
9 # Print out the mean of np_height_in
10 print(np.mean(np_height_in))
11
12 # Print out the median of np_height_in
13 print(np.median(np_height_in))
```

IPython Shell Slides

```
<script.py> output:  
1586.46108374  
74.0
```

```
<script.py> output:  
1586.46108374  
74.0
```

## Exercise

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

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

## Instructions

100 XP

- 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)

## Incorrect Submission

Check your call of `np.corrcoef()`. To calculate `corr`, the second argument to `np.corrcoef()` should be the second column of `np_baseball`. Instead of `[:, 1]`, use `[:, 2]` this time.

## script.py

```
7 avg = np.mean(np_baseball[:,0])
8 print("Average: " + str(avg))
9
10 # Print median height. Replace 'None'
11 med = np.median(np_baseball[:,0])
12 print("Median: " + str(med))
13
14 # Print out the standard deviation on height. Replace 'None'
15 stddev = np.std(np_baseball[:,0])
16 print("Standard Deviation: " + str(stddev))
17
18 # Print out correlation between first and second column. Replace 'None'
19 corr = np.corrcoef(np_baseball[:,0])
20 print("Correlation: " + str(corr))
```

Run Code

## IPython Shell Slides

```
<script.py> output:
Average: 73.6896551724
Median: 74.0
Standard Deviation: 2.31279188105
Correlation: 1.0
```

In [1]:

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)

#### Incorrect Submission

Check your call of `np.corrcoef()`. To calculate `corr`, the second argument to `np.corrcoef()` should be the second column of `np_baseball`. Instead of `[:,1]`, use `[:,2]` this time.

Did you find this feedback helpful?

Yes  No

#### script.py

```
7 avg = np.mean(np_baseball[:,0])
8 print("Average: " + str(avg))
9
10 # Print median height. Replace 'None'
11 med = np.median(np_baseball[:,0])
12 print("Median: " + str(med))
13
14 # Print out the standard deviation on height. Replace 'None'
15 stddev = np.std(np_baseball[:,0])
16 print("Standard Deviation: " + str(stddev))
17
18 # Print out correlation between first and second column. Replace 'None'
19 corr = np.corrcoef(np_baseball[:,0],np_baseball[:,1])
20 print("Correlation: " + str(corr))
```

#### iPython Shell Slides

```
# Print out correlation between first and second column. Replace 'None'
corr = np.corrcoef(np_baseball[:,0],np_baseball[:,1])
print("Correlation: " + str(corr))
Average: 73.6896551724
Median: 74.0
Standard Deviation: 2.31279188105
Correlation: [[ 1.          0.53153932]
               [ 0.53153932  1.        ]]
```

In [5]:

data | Python - Chromium

//campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=15

Course Outline

script.py

```
7 avg = np.mean(np_baseball[:,0])
8 print("Average: " + str(avg))
9
10 # Print median height. Replace 'None'
11 med = np.median(np_baseball[:,0])
12 print("Median: " + str(med))
13
14 # Print out the standard deviation on height. Replace 'None'
15 stddev = np.std(np_baseball[:,0])
16 print("Standard Deviation: " + str(stddev))
17
18 # Print out correlation between first and second column. Replace 'None'
19 corr = np.corrcoef(np_baseball[:,0],np_baseball[:,1])
20 print("Correlation: " + str(corr))
```

+100 XP

Time to use all of your new data science skills in the last exercise!

PRESS ENTER TO

Continue

Become a power user!

SUBMIT ANSWER: CTRL + SHIFT + ENTER

See all keyboard shortcuts

In [7]:

```
IPython Shell Slides
Correlation: [[ 1.          0.53153932]
 [ 0.53153932  1.        ]]
<script.py>: output:
    Average: 73.6896551724
    Median: 74.8
    Standard Deviation: 2.31279108105
    Correlation: [[ 1.          0.53153932]
 [ 0.53153932  1.        ]]
```

## 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

## script.py

```
1 # heights and positions are available as lists
2
3 # Import numpy
4 import numpy as np
5
6 # Convert positions and heights to numpy arrays: np_positions
7 np_heights=np.array(heights)
8 np_positions=np.array(positions)
9
10 # Heights of the goalkeepers: gk_heights
11 gk_heights=np_heights[(np_positions=='GK')]
12
13 # Heights of the other players: other_heights
14 other_heights=np_heights[(np_positions!='GK')]
```

## IPython Shell Slides

In [4]: # 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.array(heights)
np_positions=np.array(positions)

# Heights of the goalkeepers: gk_heights
gk_heights=np_heights[np_positions=='GK']
```

100 XP

are regular lists, to numpy  
positions.

You can use a little trick here:  
or `np_heights`. Assign the

s. This time use  
`np_heights`. Assign the

players using `np.median()`.

their median height. Replace

```
' " heights and positions are available as lists
2
3 # Import numpy
4 import numpy as np
5
6 # Convert positions and heights to numpy arrays: np_positions, np_heights
7 np_heights=np.array(heights)
8 np_positions=np.array(positions)
9
10 # Heights of the goalkeepers: gk_heights
11 gk_heights=np_heights[(np_positions=='GK')]
12
13 # Heights of the other players: other_heights
14 other_heights=np_heights[(np_positions!='GK')]
```



Run Code

Submit Answer

IPython Shell Slides

```
# 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)))

Traceback (most recent call last):
File "<stdin>", line 11, in <module>
    gk_heights=np_heights[np_positions=='GK']
NameError: name 'np_positions' is not defined
```

In [5]: #subsetting is done....put in braces to remove error

## Course Outline

script.py

```
1 # heights and positions are available as lists
2
3 # Import numpy
4 import numpy as np
5
6 # Convert positions and heights to numpy arrays: np_positions,
7 np_heights=np.array(heights)
8 np_positions=np.array(positions)
9
10 # Heights of the goalkeepers: gk_heights
11 gk_heights=np_heights[(np_positions=='GK')]
12
13 # Heights of the other players: other_heights
14 other_heights=np_heights[(np_positions!='GK')]
```



Run Code

IPython Shell

Slides

```
other_heights=np_heights[(np_positions!='GK')]

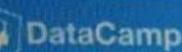
# Print out the median height of goalkeepers. Replace 'None'
print("Median height of goalkeepers: " + str(np.median(gk_heights)))

# 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 [7]:

## c plots with matplotlib

50 XP

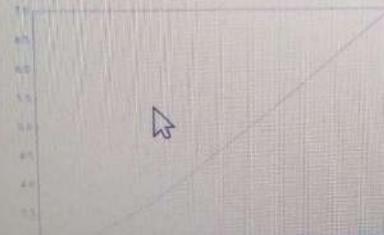


Intermediate Python for Data Science



# Matplotlib

```
In [1]: import matplotlib.pyplot as plt  
In [2]: year = [1950, 1970, 1990, 2010]  
In [3]: pop = [2.519, 3.692, 5.263, 6.972]  
In [4]: plt.plot(year, pop)  
      x     y  
In [5]: plt.show()
```



```
*Untitled Document 1 - gedit  
Open ▾  
1 to plot these data as a line chart:  
2 we call: plt.plot(year,pop)  
3 and use 2 lists as arguments  
4  
5 The 1st one corresponds to horizontal axis  
6 and the other to vertical axis  
7  
8 plt.show() - function to display the plot  
9  
10 pop is population.
```

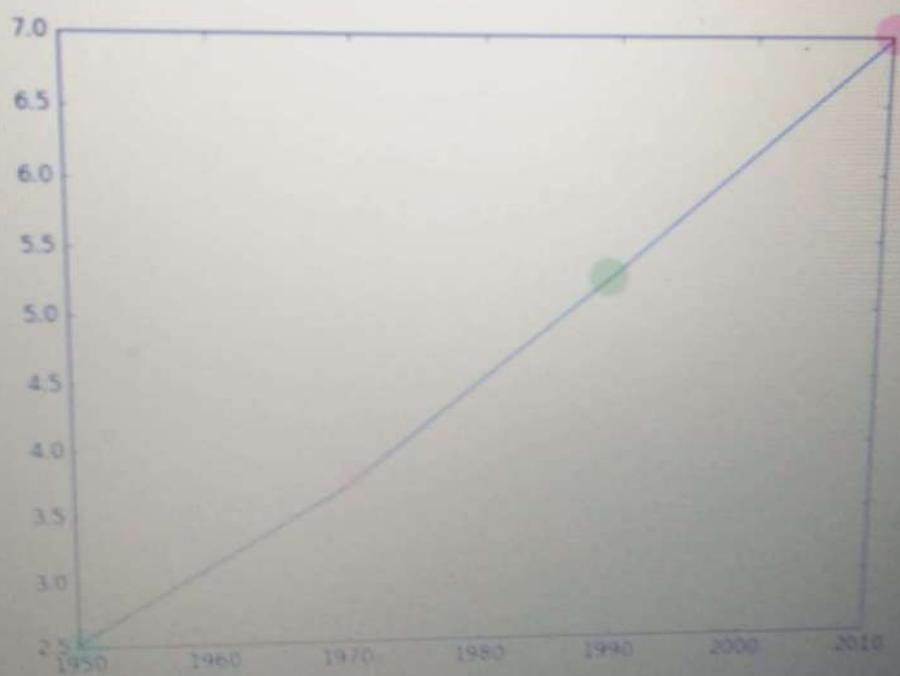
Plain Text ▾ Tab Width: 8 ▾

Ln 8, Col 42

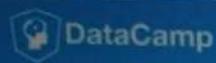
Got it!

# Matplotlib

```
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]
```



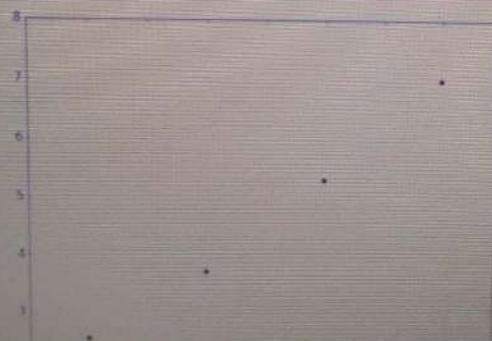
## Basic plots with matplotlib



Intermediate Python for Data Science

# Scatter plot

```
In [1]: import matplotlib.pyplot as plt  
In [2]: year = [1950, 1970, 1990, 2010]  
In [3]: pop = [2.519, 3.692, 5.263, 6.972]  
In [4]: plt.scatter(year, pop)  
In [5]: plt.show()
```



\*Untitled Document 1 - gedit  
Open Save  
1 plt.scatter() - function python doesn't connect lines with plots  
2 sometimes, its better than plt.plot

Line plot (1) | Python - Chromium  
Line plot (1) | Python x https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=2

DataCamp

Exercise

With matplotlib, you can create a bunch of different plots in Python. The most basic plot is the line plot. A general recipe is given here.

```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```

In the video, you already saw how much the world population has grown over the past years. Will it continue to do so? The world bank has estimates of the world population for the years 1950 up to 2100. The years are loaded in your workspace as a list called `year`, and the corresponding populations as a list called `pop`.

Instructions 100 XP

- `print()` the last item from both the `year` and the `pop` list to see what the predicted population for the year 2100 is. Use two `print()` functions.
- Before you can start, you should import `matplotlib.pyplot as plt`. `pyplot` is a sub-package of `matplotlib`, hence the dot.
- Use `plt.plot()` to build a line plot. `year` should be mapped on the horizontal axis, `pop` on the vertical axis. Don't forget to finish off with the `show()` function to actually display the plot.

IPython Shell

```
# Display the plot with plt.show()
2100
10.85
Out[3]: [<matplotlib.lines.Line2D at 0x7fbe2fc0ae48>]
In [4]:
```

Line plot (1) | Python

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=2

DataCamp

Exercise

With matplotlib, you can create a bunch of different plots in Python. The most basic plot is the line plot. A general recipe is given here.

```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```

In the video, you already saw how much the world population has grown over the past years. Will it continue to do so? The world bank has estimates of the world population for the years 1950 up to 2100. The years are loaded in your workspace as a list called `year`, and the corresponding populations as a list called `pop`.

Instructions 100 XP

- `print()` the last item from both the `year` and the `pop` list to see what the predicted population for the year 2100 is. Use two `print()` functions.
- Before you can start, you should import `matplotlib.pyplot as plt`. `pyplot` is a sub-package of `matplotlib`, hence the dot.
- Use `plt.plot()` to build a line plot. `year` should be mapped on the horizontal axis, `pop` on the vertical axis. Don't forget to finish off with the `show()` function to actually display the plot.

script.py

```
1 # Print the last item from year and
2 print(year[-1])
3 print(pop[-1])
4
5
6 # Import matplotlib.pyplot as plt
7 import matplotlib.pyplot as plt
8
9 # Make a line plot: year on the x
# -axis, pop on the y-axis
10 plt.plot(year,pop)
11
12 # Display the plot with plt.show()
13 plt.show()
```

Plots

Run Code Submit Answer

IPython Shell

```
# Display the plot with plt.show()
plt.show()
```

2100  
16.85

In [9]:

Show all downloads

Scatter Plot (1) | Pyt x https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=5

DataCamp

Exercise

## Scatter Plot (1)

When you have a time scale along the horizontal axis, the line plot is your friend. But in many other cases, when you're trying to assess if there's a correlation between two variables, for example, the scatter plot is the better choice. Below is an example of how to build a scatter plot.

```
import matplotlib.pyplot as plt  
plt.scatter(x,y)  
plt.show()
```

Let's continue with the `gdp_cap` versus `life_exp` plot, the GDP and life expectancy data for different countries in 2007. Maybe a scatter plot will be a better alternative?

Again, the `matplotlib.pyplot` package is available as `plt`.

Instructions 100 XP

- Change the line plot that's coded in the script to a scatter plot.
- A correlation will become clear when you display the GDP per capita on a logarithmic scale. Add the line `plt.xscale('log')`.
- Finish off your script with `plt.show()` to display the plot.

script.py

```
1 # Change the line plot below to a  
# scatter plot  
2 plt.scatter(gdp_cap, life_exp)  
3  
4 # Put the x-axis on a logarithmic  
# scale  
5  
6  
7 # Show plot  
8 plt.show()
```

Run Code Submit Answer Previous Plot 1/1

In [1]: # Change the line plot below to a scatter plot  
plt.scatter(gdp\_cap, life\_exp)  
# Put the x-axis on a logarithmic scale  
# Show plot  
plt.show()

In [2]:

Intro to python...pdf Show all

## Scatter Plot (1)

When you have a time scale along the horizontal axis, the line plot is your friend. But in many other cases, when you're trying to assess if there's a correlation between two variables, for example, the scatter plot is the better choice. Below is an example of how to build a scatter plot.

```
import matplotlib.pyplot as plt  
plt.scatter(x,y)  
plt.show()
```

Let's continue with the `gdp_cap` versus `life_exp` plot, the GDP and life expectancy data for different countries in 2007. Maybe a scatter plot will be a better alternative?

Again, the `matplotlib.pyplot` package is available as `plt`.

### Instructions

100 XP

- Change the line plot that's coded in the script to a scatter plot.
- A correlation will become clear when you display the GDP per capita on a logarithmic scale. Add the line `plt.xscale('log')`.
- Finish off your script with `plt.show()` to display the plot.

Intro to python...pdf

### script.py

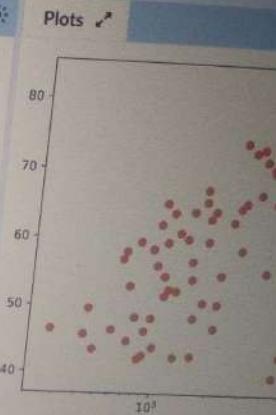
```
1 # Change the line plot below to a  
# scatter plot  
2 plt.scatter(gdp_cap, life_exp)  
3  
4 # Put the x-axis on a logarithmic  
# scale  
5 plt.xscale('log')  
6  
7 # Show plot  
8 plt.show()
```

IPython Shell Slides

In [2]: # Change the line plot below to a scatter plot  
`plt.scatter(gdp_cap, life_exp)`

```
# Put the x-axis on a logarithmic scale  
plt.xscale('log')  
  
# Show plot  
plt.show()
```

• In [3]:



← Previous Plot

2/2

rams

50 XP

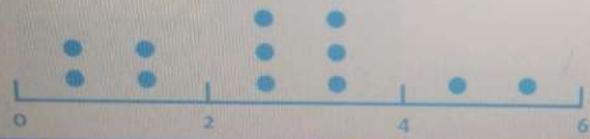
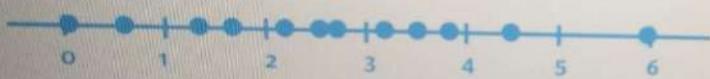
DataCamp

Intermediate Python for Data Science



# Histogram

- Explore dataset
- Get idea about distribution



\*Untitled Document 1 - gedit

Open Save

1 ) Dividing each line into equal parts called bins  
2 ) Accordingly, distribute no. of datapoints in each bin  
3 ) Draw a box for it. The height of each bin corresponds to no.of datapoints that fall in each bin.  
4 ) Result is a Histogram.  
5 result: there is more values below 2 than above 4.|

Text Tab Width: 8 Ln 5, Col 51 INS

1:50 1x auto

Got it!

Show all day

grams

DataCamp

Intermediate Python for Data Science



Save

# Matplotlib

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: help(plt.hist)
```

Help on function hist in module matplotlib.pyplot:

```
hist(x, bins=10, range=None, normed=False, weights=None,
cumulative=False, bottom=None, histtype='bar', align='mid',
orientation='vertical', rwidth=None, log=False, color=None,
label=None, stacked=False, hold=None, data=None, **kwargs)
Plot a histogram.
```

Compute and draw the histogram of `x`. The return value is a tuple `(xnx, xbins, xpatches)` or `([en0x, en1x, ...], [bins, [patches0, xpatches1, ...]])` if the input contains multiple data.

\*Untitled Document 1 - gedit

Open Save

1 Bunch of arguments can be specified, but  
the first 2 are very important ones.]

2

Text Tab Width: 8 Ln 1, Col 77 INS



Got it!

ograms

DataCamp

50 XP

Intermediate Python for Data Science



## Matplotlib example

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: help(plt.hist)
```

Help on function hist in module matplotlib.pyplot:

```
hist(x, bins=10, range=None, normed=False, weights=None,
cumulative=False, bottom=None, histtype='bar', align='mid',
orientation='vertical', rwidth=None, log=False, colors=None,
label=None, stacked=False, hold=None, data=None, **kwargs)
Plot a histogram.
```

Compute and draw the histogram of `xxx`. The return value is a tuple `(*nx*, *bins*, *patches*)` or `([*n0*, *n1*, ...], *bins*, [*patches0*, *patches1*,...])` if the input contains multiple data.

\*Untitled Document 1 - gedit

Open Save

1 Bunch of arguments can be specified, but the first 2 are very important ones.  
2 x- should be a list of values to be built histogram for.  
3 bins- to tell python in how many bins data shoul be divided.  
4 Bins argument 10 by default

Text Tab Width: 8 Ln 4, Col 28 INS

-1:12 1x auto

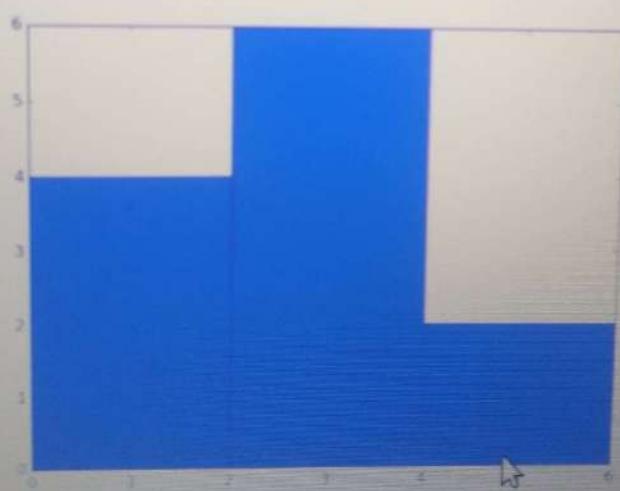
Got it!

## Histograms

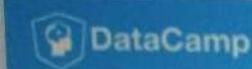


# Matplotlib example

```
In [3]: values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
In [4]: plt.hist(values, bins = 3)
In [5]: plt.show()
```

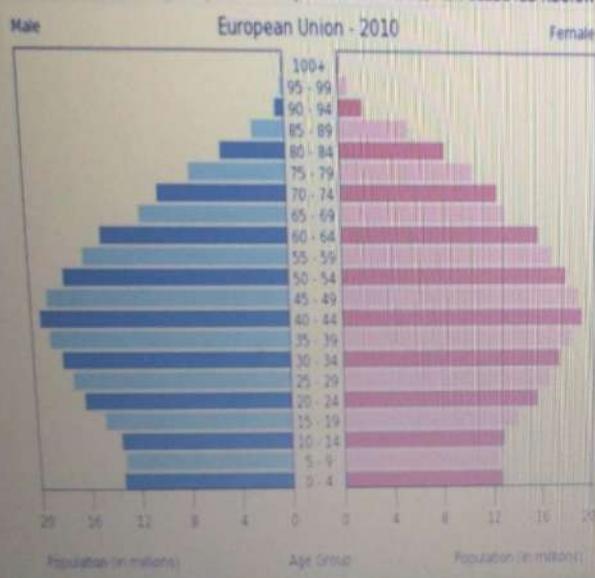


## Histograms



# Population Pyramid

Population Pyramid Graph - Special - European Union - TOTAL FOR SELECTED REGION



\*Untitled Document 1 - gedit

1 The population is highest for age group 40-44, i.e. 20 million population for both male and female. These are called baby-boomers

2 The histogram is flipped by 90(degree)

Build a histogram (2): bins | Python - Chromium  
Build a histogram (2) (77%)

<https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=9>

DataCamp

Exercise

## Build a histogram (2): bins

In the previous exercise, you didn't specify the number of bins. By default, Python sets the number of bins to 10 in that case. The number of bins is pretty important. Too few bins will oversimplify reality and won't show you the details. Too many bins will overcomplicate reality and won't show the bigger picture.

To control the number of bins to divide your data in, you can set the `bins` argument.

That's exactly what you'll do in this exercise. You'll be making two plots here. The code in the script already includes `plt.show()` and `plt.clf()` calls; `plt.show()` displays a plot; `plt.clf()` cleans it up again so you can start afresh.

As before, `life_exp` is available and `matplotlib.pyplot` is imported as `plt`.

Instructions 100 XP

- Build a histogram of `life_exp`, with 5 bins. Can you tell which bin contains the most observations?
- Build another histogram of `life_exp`, this time with 20 bins. Is this better?

script.py

```
1 # Build histogram with 5 bins
2 plt.hist(life_exp, bins=5)
3
4 # Show and clean up plot
5 plt.show()
6 plt.clf()
7
8 # Build histogram with 20 bins
9 plt.hist(life_exp, bins=20)
10
11 # Show and clean up again
12 plt.show()
13 plt.clf()
```

Plots

Run Code Submit Answer

IPython Shell

```
plt.clf()

# Build histogram with 20 bins
plt.hist(life_exp, bins=20)

# Show and clean up again
plt.show()
plt.clf()
```

1/2 Next Plot →

a histogram (2): bins | Python - Chromium  
Build a histogram (2) ×

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=9

DataCamp

Exercise

## Build a histogram (2): bins

In the previous exercise, you didn't specify the number of bins. By default, Python sets the number of bins to 10 in that case. The number of bins is pretty important. Too few bins will oversimplify reality and won't show you the details. Too many bins will overcomplicate reality and won't show the bigger picture.

To control the number of bins to divide your data in, you can set the `bins` argument.

That's exactly what you'll do in this exercise. You'll be making two plots here. The code in the script already includes `plt.show()` and `plt.clf()` calls: `plt.show()` displays a plot; `plt.clf()` cleans it up again so you can start afresh.

As before, `life_exp` is available and `matplotlib.pyplot` is imported as `plt`.

Instructions 100 XP

- Build a histogram of `life_exp`, with 5 bins. Can you tell which bin contains the most observations?
- Build another histogram of `life_exp`, this time with 20 bins. Is this better?

script.py

```
1 # Build histogram with 5 bins
2 plt.hist(life_exp,bins=5)
3
4 # Show and clean up plot
5 plt.show()
6 plt.clf()
7
8 # Build histogram with 20 bins
9 plt.hist(life_exp,bins=20)
10
11 # Show and clean up again
12 plt.show()
13 plt.clf()
```

Plots

Run Code Submit Answer ← Previous Plot 2/2

IPython Shell Slides

```
plt.clf()

# Build histogram with 20 bins
plt.hist(life_exp,bins=20)

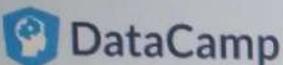
# Show and clean up again
plt.show()
plt.clf()
```

Show all discussions

## Choose the right plot (1) | Python - Chromium

Choose the right plot x

<https://campus.datacamp.com/courses/intermediate-python-for-data-science/>



Exercise



### Choose the right plot (1)

You're a professor teaching Data Science with Python, and you want to visually assess if the grades on your exam follow a particular distribution. Which plot do you use?

Instructions

50 XP

#### Possible Answers

Line plot

Scatter plot

Histogram

Take Hint (-15 XP)

**Submit Answer**

Intro to python... pdf

## Choose the right plot (2)

You're a professor in Data Analytics with Python, and you want to visually assess if longer answers on exam questions lead to higher grades. Which plot do you use?

### Instructions

50 XP

### Possible Answers

- Line plot
- Scatter plot
- Histogram



[Take Hint \(-15 XP\)](#)

[Submit Answer](#)

### Incorrect Submission

There's two variables involved: time to take the exam, and the corresponding grades. A histogram is not a suitable option in this case.

Did you find this feedback helpful?



Yes    No

## Exercise

Course C

### Choose the right plot (2)

You're a professor in Data Analytics with Python, and you want to visually assess if longer answers on exam questions lead to higher grades. Which plot do you use?

#### Instructions

50 XP

#### Possible Answers

- Line plot
- Scatter plot
- Histogram



Take Hint (-15 XP)

Submit Answer

#### Incorrect Submission

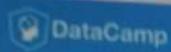
Making a line plot of this data will cause the lines to be all over the place. Do you still remember how the `gdp_cap` versus `life_exp` plot wasn't a good fit for the line plot?



Did you find this feedback helpful?

Yes  No

## Customization



Intermediate Python for Data Science

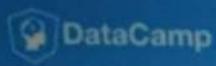
# Data Visualization

- Many options
  - Different plot types
  - Many customizations
- Choice depends on
  - Data
  - Story you want to tell



Code

## Customization



Intermediate Python for Data Science

# Axis labels

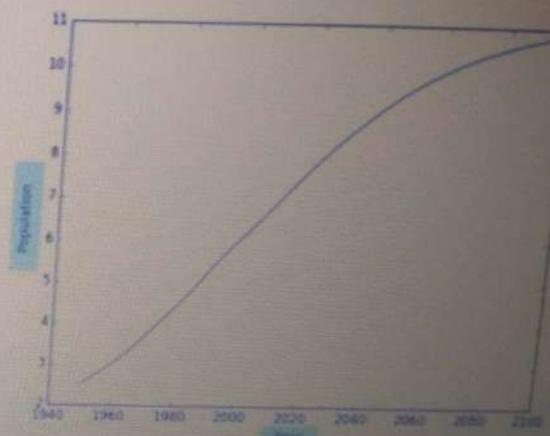
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')

plt.show()
```



Got it!

## Customization

Course Outline

50 XP

DataCamp

Intermediate Python for Data Science

# Title

population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.show()
```

World Population Projections

Population

Year

Got it!

## Customization

50 XP



Intermediate Python for Data Science

# Ticks

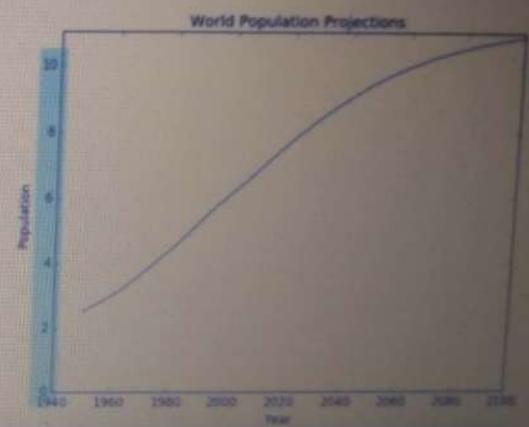
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10]) # ←

plt.show()
```



\*Untitled Document 1 - gedit

Open Save

1 to have y-axis start from 0  
2 use: yticks() function  
3 The plot changes accordingly with intervals of 2 upto 10.  
4 the curve shifts up

Text Tab Width: 8 Ln 4, Col 20

# Ticks (2)

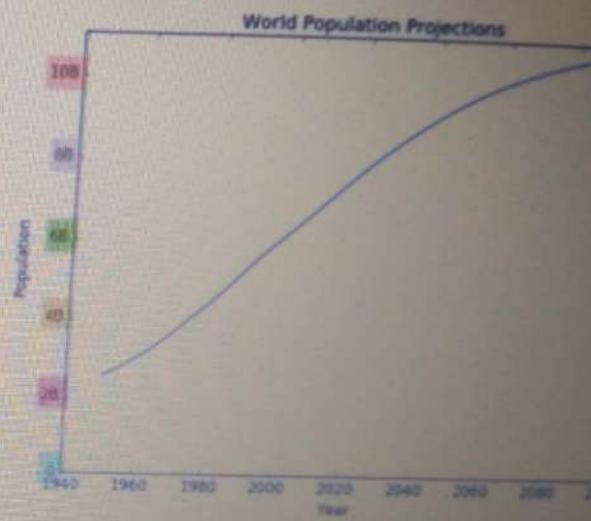
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10,
           'B', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



下  
\*Untitled Document 1 - gedit

Open ▾

```
1 now, to make clear population in
  billions.
2 we specify another list with B.
  3 0 as 0B , 1 as 1B..
4 The labels change accordingly.
5
```

Text ▾ Tab Width: 8 ▾ Ln 4, Col 37 ▾

imization

DataCamp

Intermediate Python for Data Science

# Add historical data

population.py

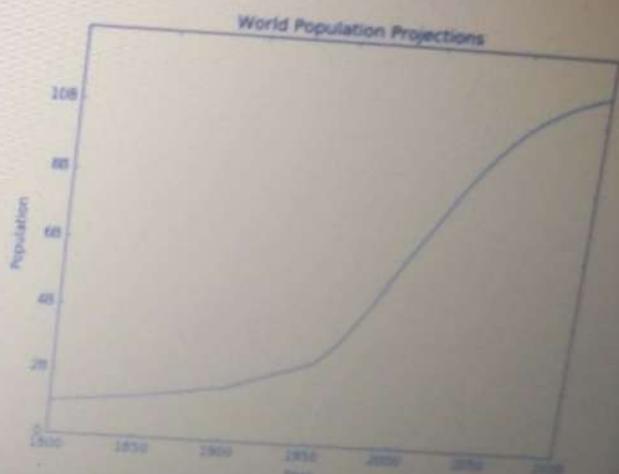
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

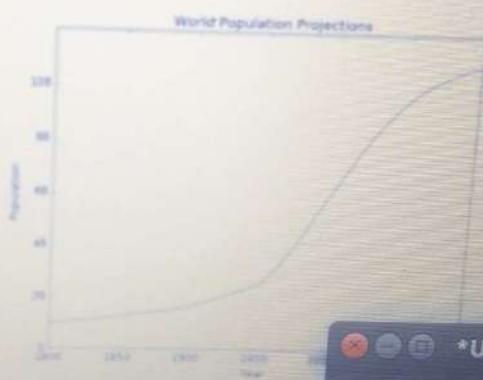
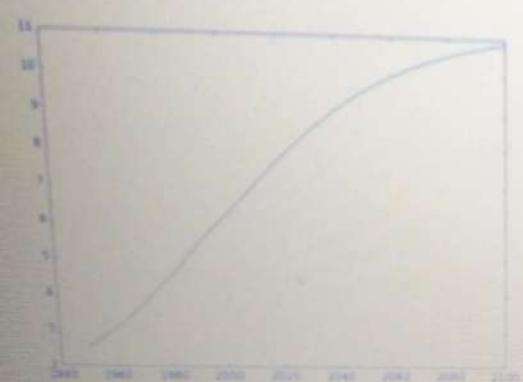
plt.show()
```



-0:14 ix 260p 21

Got it!

# Before vs After



\*Untitled Document 1 - gedit

Open ▾

1 now, gives more clear picture after  
adding more data.

2 Data Visualization.

3

Text ▾ Tab Width: 8 ▾

Ln 2, Col 19

JNS

Got It!



DataCamp

Exercise 14

It's time to customize your own plot. This is the fun part, you will see your plot come to life!

You're going to work on the scatter plot with world development data: GDP per capita on the x-axis (logarithmic scale), life expectancy on the y-axis. The code for this plot is available in the script.

As a first step, let's add axis labels and a title to the plot. You can do this with the `xlabel()`, `ylabel()` and `title()` functions, available in `matplotlib.pyplot`. This sub-package is already imported as `plt`.

Instructions 100 XP

- The strings `xlab` and `ylab` are already set for you. Use these variables to set the label of the x- and y-axis.
- The string `title` is also coded for you. Use it to add a title to the plot.
- After these customizations, finish the script with `plt.show()` to actually display the plot.

Take Hint (-30 XP)

Incorrect Submission  
Did you call `plt.title()`?

script.py

```
1 # Basic scatter plot, log scale
2 plt.scatter(gdp_cap, life_exp)
3 plt.xscale('log')
4
5 # Strings
6 xlab = 'GDP per Capita [in USD]'
7 ylab = 'Life Expectancy [in years]'
8 title = 'World Development in 2007'
9
10 # Add axis labels
11 plt.xlabel(xlab)
12 plt.ylabel(ylab)
13
```

Run Code Submit Answer

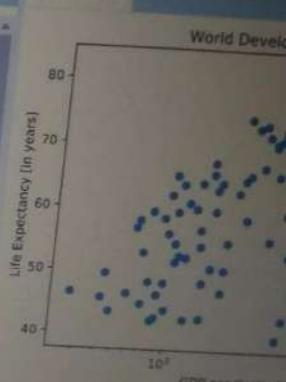
IPython Shell

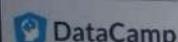
```
plt.xlabel=xlab
plt.ylabel=ylab

# Add title
plt.title=title

# After customizing, display the plot
plt.show()
```

In [3]:





## Exercise

### Ticks

The customizations you've coded up to now are available in the script, in a more concise form.

In the video, Filip has demonstrated how you could control the y-ticks by specifying two arguments:

```
plt.yticks([0,1,2], ["one", "two", "three"])
```

In this example, the ticks corresponding to the numbers 0, 1 and 2 will be replaced by one, two and three, respectively.

Let's do a similar thing for the x-axis of your world development chart, with the `xticks()` function. The tick values 1000, 10000 and 100000 should be replaced by 1k, 10k and 100k. To this end, two lists have already been created for you: `tick_val` and `tick_lab`.

#### Instructions

100 XP

- Use `tick_val` and `tick_lab` as inputs to the `xticks()` function to make the plot more readable.
- As usual, display the plot with `plt.show()` after you've added the customizations.

[Intro to python...pdf](#)

```
script.py
```

```
4 plt.xscale('log')
5 plt.xlabel('GDP per Capita [in USD]')
6 plt.ylabel('Life Expectancy [in years]')
7 plt.title('World Development in 2007')
8
9 # Definition of tick_val and tick_lab
10 tick_val = [1000, 10000, 100000]
11 tick_lab = ['1k', '10k', '100k']
12 # Adapt the ticks on the x-axis
13 plt.xticks(tick_val,tick_lab)
14 plt.show()
15
```



Run Code

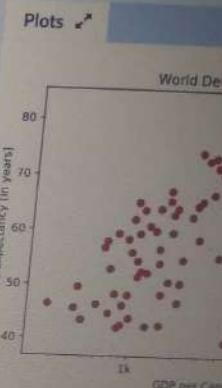
Submit Answer

IPython Shell

Slides

```
plt.title('World Development in 2007')
# Definition of tick_val and tick_lab
tick_val = [1000, 10000, 100000]
tick_lab = ['1k', '10k', '100k']
# Adapt the ticks on the x-axis
plt.xticks(tick_val,tick_lab)
plt.show()
```

In [5]:



## Exercise

## Sizes

Right now, the scatter plot is just a cloud of blue dots, indistinguishable from each other. Let's change this. Wouldn't it be nice if the size of the dots corresponds to the population?

To accomplish this, there is a list `pop` loaded in your workspace. It contains population numbers for each country expressed in millions. You can see that this list is added to the scatter method, as the argument `s`, for size.

## Instructions

100 XP

- Run the script to see how the plot changes.
- Looks good, but increasing the size of the bubbles will make things stand out more.
  - Import the `numpy` package as `np`.
  - Use `np.array()` to create a numpy array from the list `pop`. Call this Numpy array `np_pop`.
- Double the values in `np_pop` by assigning `np_pop * 2` to `np_pop` again. Because `np_pop` is a Numpy array, each array element will be doubled.

Intro to python...pdf

## script.py

```
1 # Import numpy as np
2
3
4 # Store pop as a numpy array: np_pop
5
6
7 # Double np_pop
8
9
10 # Update: set s argument to np_pop
11 plt.scatter(gdp_cap, life_exp, s = np_pop)
12
13 # Previous customizations
```

## IPython Shell Slides

In [1]: `help(plt.scatter)`

Help on function scatter in module matplotlib.pyplot:

```
scatter(x, y, s=None, c=None, marker=None, cmap=None,
alpha=None, linewidths=None, verts=None, edgecolor=None,
**kwargs)
```

A scatter plot of `*y*` vs `*x*` with varying marker sizes.

## Parameters



## DataCamp

## Exercise

from each other. Let's change this. Wouldn't it be nice if the size of the dots corresponds to the population?

To accomplish this, there is a list `pop` loaded in your workspace. It contains population numbers for each country expressed in millions. You can see that this list is added to the scatter method, as the argument `s`, for size.

## Instructions

100 XP

- Run the script to see how the plot changes.
- Looks good, but increasing the size of the bubbles will make things stand out more.
  - Import the `numpy` package as `np`.
  - Use `np.array()` to create a numpy array from the list `pop`. Call this Numpy array `np_pop`.
  - Double the values in `np_pop` by assigning `np_pop * 2` to `np_pop` again. Because `np_pop` is a Numpy array, each array element will be doubled.
  - Change the `s` argument inside `plt.scatter()` to be `np_pop` instead of `pop`.

Take Hint (-30 XP)

## script.py

```

1 # Import numpy as np
2 import numpy as np
3
4 # Store pop as a numpy array: np_pop
5 np_pop=np.array(pop)
6
7 # Double np_pop
8 #np_pop=np_pop*2
9
10 # Update: set s argument to np_pop
11 plt.scatter(gdp_cap, life_exp, s = np_pop
12 )

```



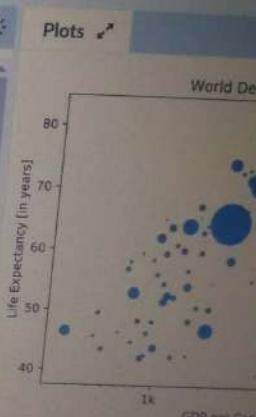
Run Code

Submit Answer

IPython Shell Slides

\* All arguments with the following names: 'c', 'color', 'edgecolors', 'facecolor', 'facecolors', 'linewidths', 's', 'x', 'y'.

In [2]: `pop`    Out[2]:  
 [31.889923,  
 3.600523,  
 33.333216,  
 12.428476]



Let's change this. Wouldn't it be nice if the size of the dots  
the population?

his, there is a list `pop` loaded in your workspace. It  
ation numbers for each country expressed in millions. You  
s list is added to the scatter method, as the argument `s`.

ot to see how the plot changes.

but increasing the size of the bubbles will make things stand

the numpy package as `np`.

`array()` to create a numpy array from the list `pop`. Call  
`array` `np_pop`.

the values in `np_pop` by assigning `np_pop * 2` to  
`np_pop` again. Because `np_pop` is a Numpy array, each array  
will be doubled.

the `s` argument inside `plt.scatter()` to be `np_pop`  
of `pop`.

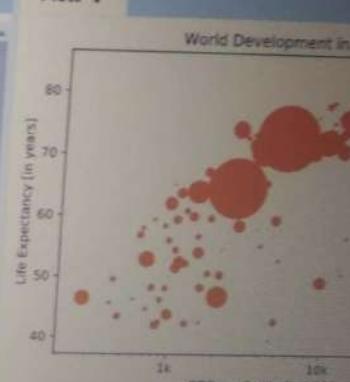
In [5]:

```
script.py
```

```
1 # Import numpy as np
2 import numpy as np
3
4 # Store pop as a numpy array: np_pop
5 np_pop=np.array(pop)
6
7 # Double np_pop
8 np_pop=np_pop*2
9
10 # Update: set s argument to np_pop
11 plt.scatter(gdp_cap, life_exp, s = np_pop)
12
```

Plots

World Development in 2007



Run Code Submit Answer Previous Plot 2/2

```
IPython Shell Slides
```

```
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])

# Display the plot
plt.show()
```

Colors | Python - Chromium  
Colors | Python

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=17

DataCamp

Exercise

## Colors

The code you've written up to now is available in the script on the right.

The next step is making the plot more colorful! To do this, a list `col` has been created for you. It's a list with a color for each corresponding country, depending on the continent the country is part of.

How did we make the list `col` you ask? The Gapminder data contains a list `continent` with the continent each country belongs to. A dictionary is constructed that maps continents onto colors:

```
dict = {  
    'Asia': 'red',  
    'Europe': 'green',  
    'Africa': 'blue',  
    'Americas': 'yellow',  
    'Oceania': 'black'  
}
```

Nothing to worry about now: you will learn about dictionaries in the next chapter.

Instructions

Add `c = col` to the arguments of the `plt.scatter()` function.

100 XP

script.py

```
1 # Specify c and alpha inside plt.scatter()  
2 plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop)  
3  
4 # Previous customizations  
5 plt.xscale('log')  
6 plt.xlabel('GDP per Capita [in USD]')  
7 plt.ylabel('Life Expectancy [in years]')  
8 plt.title('World Development in 2007')  
9 plt.xticks([1000, 10000, 100000], ['1k', '10k', '100k'])  
10  
11 # Show the plot  
12 plt.show()
```

IPython Shell Slides

```
In [1]: col  
Out[1]:  
['red',  
 'green',  
 'blue',  
 'blue',  
 'yellow',  
 'black',  
 'green',  
 'red']
```

thon - Chromium

| Python

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=17

Camp

se

ctions

100 XP

c = col to the arguments of the plt.scatter() function.

ange the opacity of the bubbles by setting the alpha argument to 1 inside plt.scatter(). Alpha can be set from zero to one, where 0 is totally transparent, and one is not at all transparent.

Hint (-30 XP)

script.py

```
1 # Specify c and alpha inside plt.scatter()
2 plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c=col)
3
4 # Previous customizations
5 plt.xscale('log')
6 plt.xlabel('GDP per Capita [in USD]')
7 plt.ylabel('Life Expectancy [in years]')
8 plt.title('World Development in 2007')
9 plt.xticks([1000,10000,100000], ['1k','10k','100k'])
10
11 # Show the plot
12 plt.show()
```

Run Code

Submit A

In [3]: ?plt.scatter

Signature: plt.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, v

=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, ha

=None, data=None, \*\*kwargs)

Docstring:

A scatter plot of \*y\* vs \*x\* with varying marker size and/or color.

Parameters

x, y : array-like, shape (n, )

Show all downloads

100 XP

col to the arguments of the `plt.scatter()` function.

the opacity of the bubbles by setting the `alpha` argument to

ide `plt.scatter()`. Alpha can be set from zero to one, where

ally transparent, and one is not at all transparent.

(-30 XP)

```
script.py
```

```
1 # Specify c and alpha inside plt
  .scatter()
2 plt.scatter(x = gdp_cap, y = life_exp
  , s = np.array(pop) * 2,c=col)
3
4 # Previous customizations
5 plt.xscale('log')
6 plt.xlabel('GDP per Capita [in USD]')
7 plt.ylabel('Life Expectancy [in
  years]')
8 plt.title('World Development in 2007'
  )
9 plt.xticks([1000,10000,100000], ['1k'
```

Plots

World Development in 2007

Run Code

Submit Answer

IPython Shell Slides

File: /usr/local/lib/python3.5/dist-packages/matplotlib/lib/pypiot.py

Type: function

```
In [4]: # Specify c and alpha inside plt.scatter()
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2,c=col)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
```

Show all downloads

ors | Python - Chromium  
Colors | Python

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=17

DataCamp

Exercise

Instructions

Add `c = col` to the arguments of the `plt.scatter()` function.

Change the opacity of the bubbles by setting the `alpha` argument to `0.8` inside `plt.scatter()`. Alpha can be set from zero to one, where zero is totally transparent, and one is not at all transparent.

Take Hint (-30 XP)

script.py

```
1 # Specify c and alpha inside plt.scatter()
2 plt.scatter(x = gdp_cap, y = life_exp, s =
3 np.array(pop) * 2, c=col, alpha=0.8)
4 # Previous customizations
5 plt.xscale('log')
6 plt.xlabel('GDP per Capita [in USD]')
7 plt.ylabel('Life Expectancy [in years]')
8 plt.title('World Development in 2007')
9 plt.xticks([1000,10000,100000], ['1k','10k',
10 , '100k'])
11 # Show the plot
```

Run Code Submit Answer

Plots

World Development in 2007

IPython Shell Slides

array is used. `*min*` and `*max*` are ignored if you pass a `marker` instance.

`alpha`: scalar, optional, default: None  
The alpha blending value, between 0 (transparent) and 1 (opaque).

`linewidths`: scalar or array-like, optional, default: None  
The linewidth of the marker edges. Note: The default `edgecolor` is "face". You may want to change this as well.  
If `None`, defaults to `rcParams["lines.linewidth"]`.

additional Customizations | Python - Chromium

Additional Customiz x

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=18

DataCamp

Exercise

## Additional Customizations

If you have another look at the script, under `# Additional Customizations`, you'll see that there are two `plt.text()` functions now. They add the words "India" and "China" in the plot.

Instructions 100 XP

- Add `plt.grid(True)` after the `plt.text()` calls so that gridlines are drawn on the plot.

Take Hint (-30 XP)

script.py

```
    )
7 plt.ylabel('Life Expectancy [in
years]')
8 plt.title('World Development in
2007')
9 plt.xticks([1000,10000,100000],
['1k', '10k', '100k'])
10
11 # Additional customizations
12 plt.text(1550, 71, 'India')
13 plt.text(5700, 80, 'China')
14
15 # Add grid() call
```

Plots

World Development in 2007

In [2]: `?plt.txt()`  
Object 'plt.txt' not found.

In [3]: `plt.grid`  
Out[3]: <function matplotlib.pyplot.grid>

In [4]: `?plt.grid`  
Signature: plt.grid(b=None, which='major', axis='both', \*\*kwargs)  
Docstring:

Intro to python... pdf

Show all download

re are two  
India" and

100 XP

ills so that gridlines

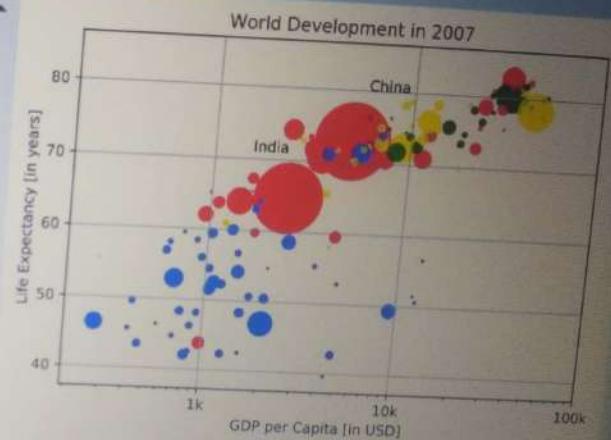
data science/matplotlib?ex=18

← Course Outline →

script.py

```
2007')
9 plt.xticks([1000, 10000, 100000],
[‘1k’, ‘10k’, ‘100k’])
10
11 # Additional customizations
12 plt.text(1550, 71, ‘India’)
13 plt.text(5700, 80, ‘China’)
14
15 # Add grid() call
16 plt.grid(True)
17
18 # Show the plot
19 plt.show()
```

Plots ↗



Run Code

Submit Answer

← Previous Plot

2/2

Next Plot ↗

IPython Shell

Slides

In [4]: ?plt.grid

Signature: plt.grid(b=None, which='major', axis='both', \*\*kwargs)

Docstring:

Turn the axes grids on or off.

Set the axes grids on or off; \*b\* is a boolean.

If \*b\* is \*None\* and `len(kwargs)==0` , toggle the grid state. If \*kwargs\* are supplied, it is assumed that you want a grid and \*b\* is thus set to \*True\*.

Show all downloads...

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=19

DataCamp

Exercise

Course Outline

Plots

## Interpretation

If you have a look at your colorful plot, it's clear that people live longer in countries with a higher GDP per capita. No high income countries have really short life expectancy, and no low income countries have very long life expectancy. Still, there is a huge difference in life expectancy between countries on the same income level. Most people live in middle income countries where difference in lifespan is huge between countries; depending on how income is distributed and how it is used.

What can you say about the plot?

Instructions 50 XP

Possible Answers

The countries in blue, corresponding to Africa, have both low life expectancy and a low GDP per capita.

There is a negative correlation between GDP per capita and life expectancy.

China has both a lower GDP per capita and low expectancy compared to India.

Take Hint (-15 XP)

Submit Answer

World Development in 2007

Life Expectancy (in years)

GDP per Capita (in USD)

Previous Plot 1/1

IPython Shell Slides

In [1]:

Intro to python...pdf

https://campus.datacamp.com/courses/intermediate-python-for-data-science/matplotlib?ex=19

+50 XP

Correct! Up to the next chapter, on dictionaries!

PRESS ENTER TO

Continue

Become a power user!

SUBMIT ANSWER: **CTRL + SHIFT + ENTER**

See all keyboard shortcuts

Plots

World Development

GDP per Capita (1990)

Life Expectancy (in years)

In [1]:



# pandas DataFrames

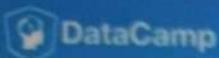
- Example: DataFrame of Apple Stock data

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216300	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
—	—	—	—	—	—	—



Get it!

## Review of pandas DataFrames



# Indexes and columns

```
In [1]: import pandas as pd
```

```
In [2]: type(AAPL)
```

```
Out[2]: pandas.core.frame.DataFrame
```

```
In [3]: AAPL.shape
```

```
Out[3]: (8514, 6)
```

```
In [4]: AAPL.columns
```

```
Out[4]:
```

```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'],  
      dtype='object')
```

```
In [5]: type(AAPL.columns)
```

```
Out[5]: pandas.indexes.base.Index
```

```
*Untitled Document 1 - gedit  
Save  
Open ▾  1 pandas-a library for data analysis  
2 the powertool of pandas -dataframe  
3 Dataframe-a tablured data structure with  
rows and column  
4  
5  
6 8514-rows  
7 6-columns  
8  
9 AAPL.columns give column names!
```

Text ▾ Tab Width: 8 ▾

Ln 9, Col 31

INS



1:12

Get it!

## Review of pandas DataFrames



# Indexes and columns

```
In [6]: AAPL.index
Out[6]:
DatetimeIndex(['2014-09-16', '2014-09-15', '2014-09-12',
                '2014-09-11', '2014-09-10', '2014-09-09',
                '2014-09-08', '2014-09-05', '2014-09-04',
                '2014-09-03',
                ...
                '1980-12-26', '1980-12-24', '1980-12-23',
                '1980-12-22', '1980-12-19', '1980-12-18',
                '1980-12-17', '1980-12-16', '1980-12-15',
                '1980-12-12'],
               dtype='datetime64[ns]', name='Date', length=8514,
               freq=None)

In [7]: type(AAPL.index)
Out[7]: pandas.tseries.index.DatetimeIndex
```



-253



# Slicing

```
In [8]: AAPL.iloc[:5,:]  
Out[8]:
```

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
2014-09-11	100.41	101.44	99.62	101.43	62353100	101.43
2014-09-10	98.01	101.11	97.76	101.00	100741900	101.00

```
In [9]: AAPL.iloc[-5:,:]  
Out[9]:
```

Date	Open	High	Low	Close	Volume	Adj Close
1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45

