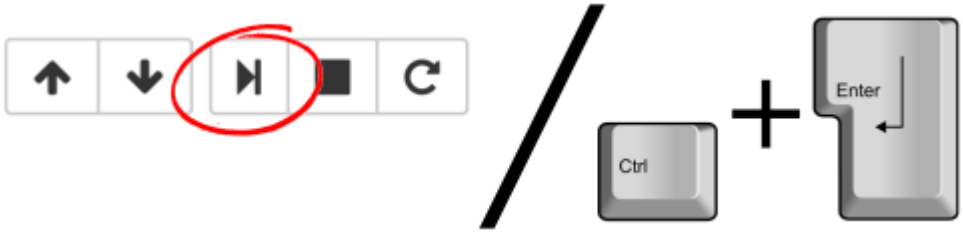
1. This is a Jupyter notebook!

A *Jupyter notebook* is a document that contains text cells (what you're reading right now) and code cells. What is special with a notebook is that it's *interactive*: You can change or add code cells, and then *run* a cell by first selecting it and then clicking the *run cell* button above (▶| Run) or hitting `ctrl + enter` .



The result will be displayed directly in the notebook. You *could* use a notebook as a simple calculator. For example, it's estimated that on average 256 children were born every minute in 2016. The code cell below calculates how many children were born on average on a day.

In [34]:
```
# I'm a code cell, click me, then run me!
256 * 60 * 24 # Children × minutes × hours
```

Out[34]:
```
368640
```

In [35]:
```
%%nose
# No tests
```

Out[35]:
```
No tests found.
```

2. Put any code in code cells

But a code cell can contain much more than a simple one-liner! This is a notebook running python and you can put *any* python code in a code cell (but notebooks can run other languages too, like R). Below is a code cell where we define a whole new function (`greet`). To show the output of `greet` we run it last in the code cell as the last value is always printed out.

In [36]:
```
def greet(first_name, last_name):
    greeting = 'My name is ' + last_name + ', ' + first_name + ' ' + last_name
+ '!'
    return greeting

# Replace with your first and last name.
# That is, unless your name is already James Bond.
greet('Daniel', 'Mortensen')
```

Out[36]:
```
'My name is Mortensen, Daniel Mortensen!'
```

In [37]:
```
%%nose
# No tests
```

Out[37]:
```
No tests found.
```

3. Jupyter notebooks ♡ data

We've seen that notebooks can display basic objects such as numbers and strings. But notebooks also support the objects used in data science, which makes them great for interactive data analysis!

For example, below we create a `pandas` DataFrame by reading in a `csv` -file with the average global temperature for the years 1850 to 2016. If we look at the `head` of this DataFrame the notebook will render it as a nice-looking table.

In [38]:
```
# Importing the pandas module
import pandas as pd

# Reading in the global temperature data
global_temp = pd.read_csv('datasets/global_temperature.csv')

# Take a look at the first datapoints
print(global_temp.head())
```

```
   year  degrees_celsius
0  1850             7.74
1  1851             8.09
2  1852             7.97
3  1853             7.93
4  1854             8.19
```

In [39]:
```
%%nose
# No tests
```

Out[39]:
```
No tests found.
```

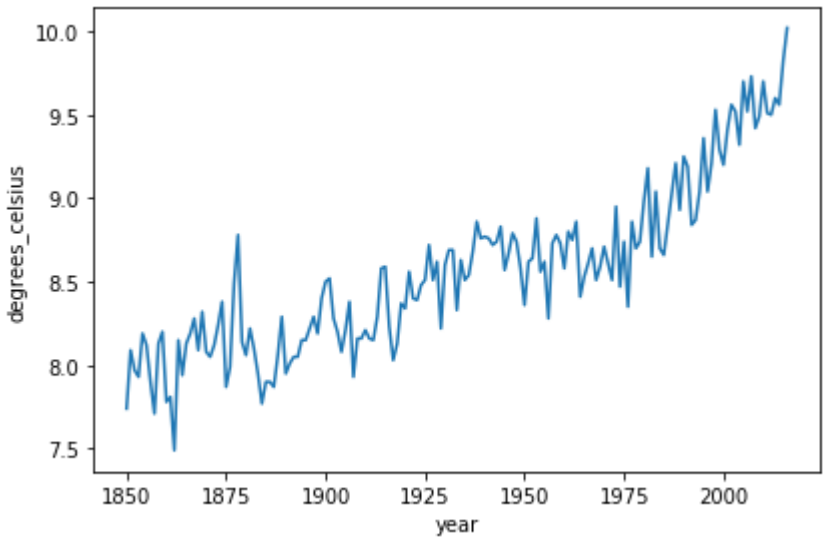
4. Jupyter notebooks ♡ plots

Tables are nice but — as the saying goes — *"a plot can show a thousand data points"*. Notebooks handle plots as well, but it requires a bit of magic. Here *magic* does not refer to any arcane rituals but to so-called "magic commands" that affect how the Jupyter notebook works. Magic commands start with either `%` or `%%` and the command we need to nicely display plots inline is `%matplotlib inline` . With this *magic* in place, all plots created in code cells will automatically be displayed inline.

Let's take a look at the global temperature for the last 150 years.

In [40]:
```
# Setting up inline plotting using jupyter notebook "magic"
%matplotlib inline

import matplotlib.pyplot as plt

# Plotting global temperature in degrees celsius by year
plt.plot(global_temp['year'], global_temp['degrees_celsius'])

# Adding some nice labels
plt.xlabel('year')
plt.ylabel('degrees_celsius')
```

Out[40]:
```
<matplotlib.text.Text at 0x7fd88c8d3be0>
```



In [41]:
```
%%nose
# No tests
```

Out[41]:
```
No tests found.
```

5. Jupyter notebooks ♡ a lot more

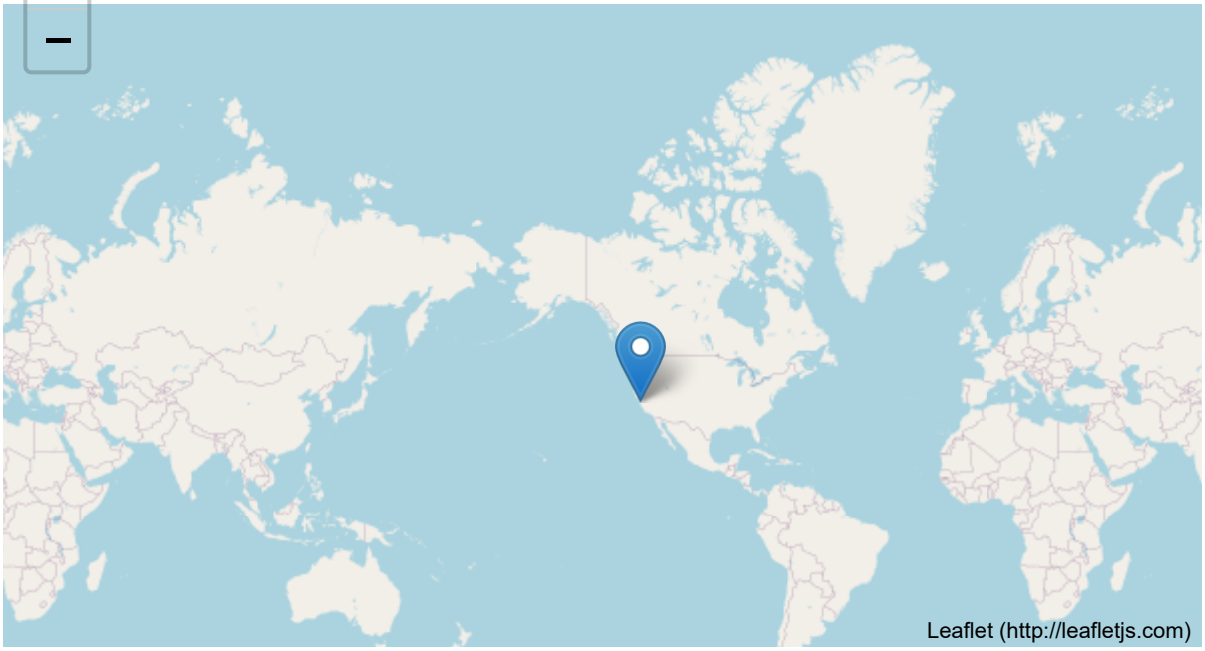
Tables and plots are the most common outputs when doing data analysis, but Jupyter notebooks can render many more types of outputs such as sound, animation, video, etc. Yes, almost anything that can be shown in a modern web browser. This also makes it possible to include *interactive widgets* directly in the notebook!

For example, this (slightly complicated) code will create an interactive map showing the locations of the three largest smartphone companies in 2016. You can move and zoom the map, and you can click the markers for more info!

In [42]:
```
# Making a map using the folium module
import folium
phone_map = folium.Map()

# Top three smart phone companies by market share in 2016
companies = [
    {'loc': [37.4970,  127.0266], 'label': 'Samsung: 20.5%'},
    {'loc': [37.3318, -122.0311], 'label': 'Apple: 14.4%'},
    {'loc': [22.5431,  114.0579], 'label': 'Huawei: 8.9%'}]

# Adding markers to the map
for company in companies:
    marker = folium.Marker(location=company['loc'], popup=company['label'])
    marker.add_to(phone_map)

# The last object in the cell always gets shown in the notebook
phone_map
```

Out[42]:

Make this Notebook Trusted to load map: File -> Trust Notebook



In [43]:
```
%%nose

def test_market_share_of_samsung():
    assert '20.5' in companies[0]['label'], \
        "The market share of Samsung should be 20.5%"

def test_market_share_of_apple():
    assert '14.4' in companies[1]['label'], \
        "The market share of Apple should be 14.4%"

def test_market_share_of_huawei():
    assert '8.9' in companies[2]['label'], \
        "The market share of Huawei should be 8.9%"
```

Out[43]:
```
3/3 tests passed
```

6. Goodbye for now!

This was just a short introduction to Jupyter notebooks, an open source technology that is increasingly used for data science and analysis. I hope you enjoyed it! :)

In [44]:
```
# Are you ready to get started with  DataCamp projects?
I_am_ready = True

# Ps.
# Feel free to try out any other stuff in this notebook.
# It's all yours!
```

In [45]:
```
%%nose

def test_if_ready():
    assert I_am_ready, \
        "I_am_ready should be set to True, if you are ready to get started wit
h DataCamp projects, that is."
```

Out[45]:
```
1/1 tests passed
```