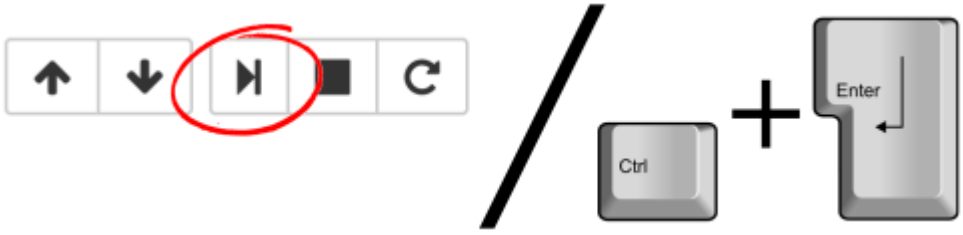
1. This is a Jupyter Notebook!

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 [2]:
```
# I'm a code cell, click me, then run me!
256 * 60 * 24 # Children × minutes × hours
```

Out[2]:
```
368640
```

In [3]:
```
%%nose
# No tests
```

Out[3]:
```
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 [4]:
```
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 Jane Bond.
greet('Daniel', 'Mortensen')
```

Out[4]:
```
'My name is Mortensen, Daniel Mortensen!'
```

In [5]:
```
%%nose
# No tests
```

Out[5]:
```
No tests found.
```

3. Jupyter Notebooks ♡ SQL (part i)

We've seen that notebooks can display basic objects such as numbers and strings. But notebooks also support and display the outputs of SQL commands! Using an open source Jupyter extension called ipython-sql (https://github.com/catherinedevlin/ipython-sql), we can connect to a database and issue SQL commands within our notebook. For example, we can connect to a PostgreSQL (https://www.postgresql.org/) database that has a table that contains country data, then inspect the first three rows of the table by putting `%%sql` ahead of the SQL commands (more on the meaning of `%%` later).

In [6]:
```
%%sql postgresql:///countries
SELECT * FROM countries LIMIT 3;
```

```
3 rows affected.
```

Out[6]:

code	name	continent	region	surface_area	indep_year	local_name	gov_fo
AFG	Afghanistan	Asia	Southern and Central Asia	652090.0	1919	Afganistan/Afqanestan	Islar Emir
NLD	Netherlands	Europe	Western Europe	41526.0	1581	Nederland	Constitutio Monarc
ALB	Albania	Europe	Southern Europe	28748.0	1912	Shqiperia	Repub

In [7]:
```
%%nose
# No tests
```

Out[7]:
```
No tests found.
```

4. Jupyter Notebooks ♡ SQL (part ii)

And after the first connection to the database, the connection code (`postgresql:///countries`) can be omitted. Let's do a different query this time and select the row in the `countries` table for Belgium. Note the single `%` this time. Again, more on that later.

In [8]:
```
# Query the database
%sql SELECT * FROM countries WHERE name = 'Belgium';
```

```
 * postgresql:///countries
1 rows affected.
```

Out[8]:

code	name	continent	region	surface_area	indep_year	local_name	gov_form	capit
BEL	Belgium	Europe	Western Europe	30518.0	1830	Belgie/Belgique	Constitutional Monarchy, Federation	Brusse

In [9]:
```
%%nose

last_value = _

def test_belgium():
    assert last_value[0]['name'] == 'Belgium', \
        "The name of the country queried should be Belgium."
```

Out[9]:
```
1/1 tests passed
```

5. Jupyter Notebooks ♡ SQL (part iii)

We can even convert our SQL results to a pandas DataFrame! Let's convert the entire `countries` table.

In [10]:
```
# SQL Query
result = %sql SELECT * FROM countries;

# To pandas DataFrame
df = result.DataFrame()
df.info()
```

```
 * postgresql:///countries
206 rows affected.
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 206 entries, 0 to 205
Data columns (total 11 columns):
code            206 non-null object
name            206 non-null object
continent       206 non-null object
region          206 non-null object
surface_area    206 non-null float64
indep_year      188 non-null float64
local_name      206 non-null object
gov_form        206 non-null object
capital         201 non-null object
cap_long        204 non-null float64
cap_lat         204 non-null float64
dtypes: float64(4), object(7)
memory usage: 17.8+ KB
```

In [11]:
```
%%nose
# This needs to be included at the beginning of every @tests cell.

correct_df = result.DataFrame()

def test_df():
    assert correct_df.equals(df), \
    "The variable df should contain the ResultSet from the provided SQL quer
y."
```

Out[11]:
```
1/1 tests passed
```

6. Jupyter Notebooks ♡ SQLAlchemy

If SQLAlchemy is your thing, you can do that in this notebook, too! Jupyter Notebooks love everything, apparently…

What's SQLAlchemy (https://www.sqlalchemy.org/), you ask? SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. Next, we'll run the last query we just ran except after connecting to and querying the database using SQLAlchemy.

In [12]:
```
# Connect to database
from sqlalchemy import create_engine
engine = create_engine("postgresql:///countries");

# Query database
result = engine.execute("SELECT * FROM countries;")

# Display column names
result.keys()
```

Out[12]:
```
['code',
 'name',
 'continent',
 'region',
 'surface_area',
 'indep_year',
 'local_name',
 'gov_form',
 'capital',
 'cap_long',
 'cap_lat']
```

In [13]:
```
%%nose
# No tests
```

Out[13]:
```
No tests found.
```

7. 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 some more 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 `%%` (just like we saw with `%sql` and `%%sql`) 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.

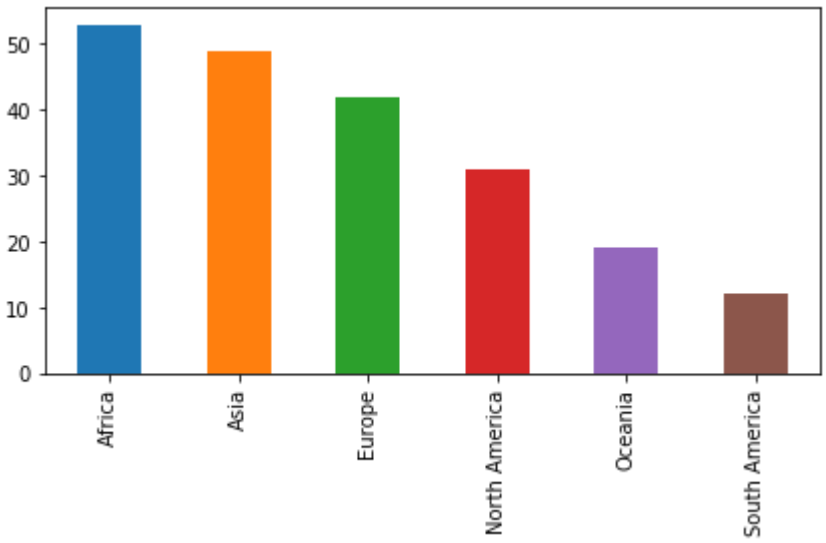
Using the previously created pandas DataFrame that we named `df`, let's plot the number of countries in each continent as a bar chart using the `plot()` method of pandas DataFrames.

Now, for the difference between `%%sql` and `%sql`: ordinary assignment works for single-line `%sql` queries while `%%sql` is for multi-line queries. See the [Assignment (https://github.com/catherinedevlin/ipython-sql#assignment)](https://github.com/catherinedevlin/ipython-sql#assignment) ipython-sql documentation section for more info.

In [14]:
```
# Setting up inline plotting using Jupyter Notebook "magic"
%matplotlib inline

# Plotting number of countries in each continent
df.continent.value_counts().plot(kind='bar')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb71ef31128>
```



In [15]:
```
%%nose
# No tests
```

Out[15]:
```
No tests found.
```

8. Goodbye for now!

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! Everything in this collection of Jupyter Widgets (http://jupyter.org/widgets) can be displayed in this notebook.

But that's enough for now! This was just a short introduction to Jupyter Notebooks, an open source technology that is increasingly used for data science and analysis. We hope you enjoyed it! :)

In [16]:
```
# Are you ready to get started with  DataCamp projects?
I_am_ready = True

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

In [17]:
```
%%nose

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

Out[17]:
```
1/1 tests passed
```