Introduction to Data Science

with Python

Alexis Bogroff

May 26, 2022

Presenter

Alexis Bogroff
Lecturer and Mentor in Data Science
at Paris 1 Panthéon-Sorbonne, ESILV,
Openclassrooms, EM-Lyon

Presenter

Alexis Bogroff
Lecturer and Mentor in Data Science
at Paris 1 Panthéon-Sorbonne, ESILV,
Openclassrooms, EM-Lyon

- 4 years Teaching Assistant and lecturer in VBA, Python for finance, SQL, Data Analysis and Data Science
- 9 months Researcher Assistant at Paris 1 Panthéon-Sorbonne within H2020 European Project
- 1 year Data Scientist at Pléiade Asset Management

Why Python?

- Versatile
- Simple
- Open Source
- Most used for Data Science

Programming Environment

- Interactive console

```
(eda) alexisbogroff@Alexiss-MacBook-Pro src % python
Python 3.9.2 | packaged by conda-forge | (default, Feb 21 2021, 05:00:30)
[Clang 11.0.1 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>> x = 1
>>> x += 2
>>> x
3
>>> def say_hello():
...     print("hello")
...
>>> say_hello()
hello
>>>
```

Programming Environment

- Interactive console
- Scripts

Users > alexisbogroff > git_repositories > geo_challenge > part_3_4_main.py

```python
"""
Report alerts on sensible metrics

Alerts:
- variation of ships count (by type)
- static ships between two snaps, or on the given period
"""
import part_3_4_settings as stng
from part_3_4_fcts import Detector, load_data

# Load data
data_t0 = load_data(stng.PATH_DATA, date_max=stng.DATE_MIN)
data_t1 = load_data(stng.PATH_DATA, date_min=stng.DATE_MIN,
                                    date_max=stng.DATE_MAX)

# Init detector
detector = Detector(data_t0, data_t1)

# Run scans
detector.detect_large_variations(threshold=1.7)
detector.detect_static_objects(n_periods_min=20, smooth=5)

# Save alerts to file
detector.export(stng.PATH_ALERTS)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh

```
(base) alexisbogroff@Alexiss-MacBook-Pro geo_challenge % python part_3_4_main.py


Ouliters count:

                  date             ship_type count  breach_pct bound_type
0  2020-02-25 12:00:00  Imperial SpaceShuttle     9       -8.01  bound_min
1  2020-01-19 12:00:00  Imperial SpaceShuttle    12        1.47  bound_max
2  2020-05-09 12:00:00  Imperial SpaceShuttle    13        9.92  bound_max
3  2020-07-09 12:00:00  Imperial SpaceShuttle    12        1.47  bound_max
4  2020-01-11 12:00:00             TIE Fighter   23        3.07  bound_max
5  2020-01-26 12:00:00             TIE Fighter   23        3.07  bound_max
6  2020-02-06 12:00:00             TIE Fighter   27       20.99  bound_max
7  2020-02-22 12:00:00             TIE Fighter   37       65.80  bound_max
```

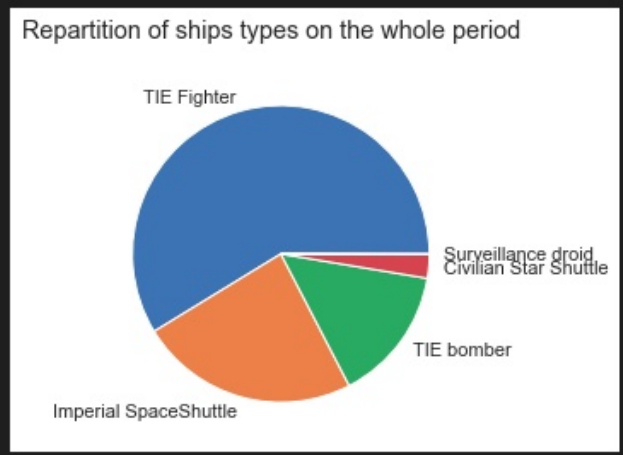
Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks

Ship_type repartition

```
# Total period
ship_type_total = data['ship_type'].value_counts()
ship_type_total.plot.pie(label="")
plt.suptitle('Repartition of ships types on the whole period')
print(ship_type_total)
```

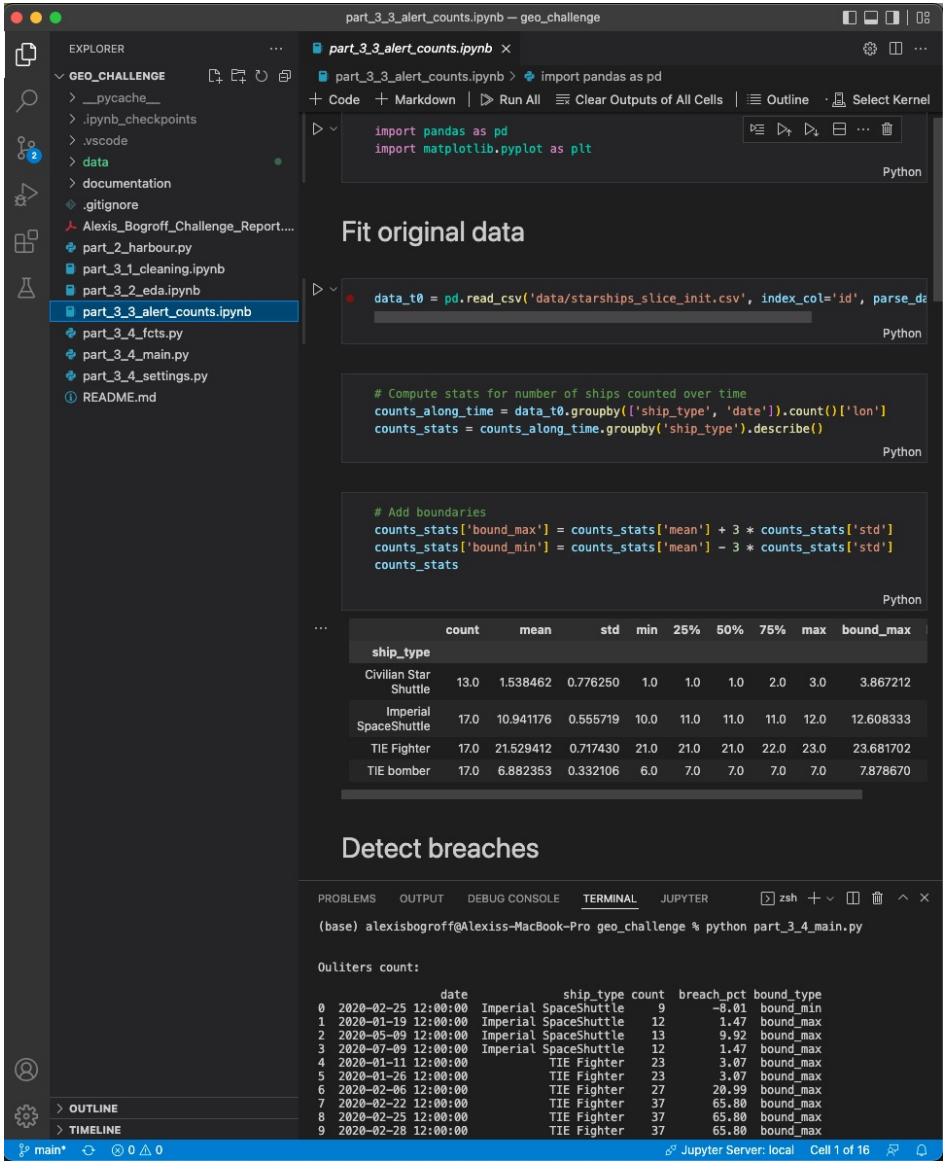
```
TIE Fighter             2034
Imperial SpaceShuttle    827
TIE bomber               511
Civilian Star Shuttle     89
Surveillance droid         2
Name: ship_type, dtype: int64
```




```
# Get extreme dates
```

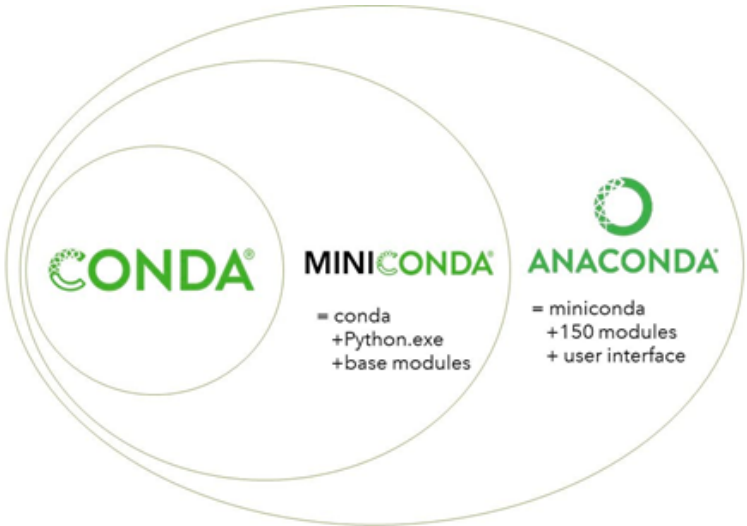
Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks
- Code editors (VS Code)



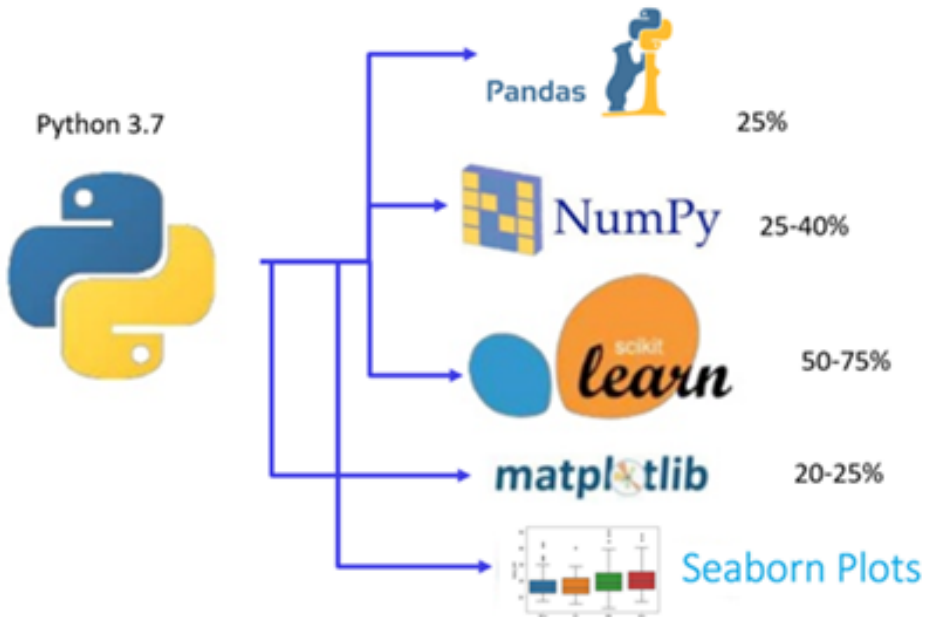
Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks
- Code editors (VS Code)
- Packages managers (Conda, Pip)



Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks
- Code editors (VS Code)
- Packages managers (Conda, Pip)
- Packages / Libraries (Pandas, Matplotlib)



Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks
- Code editors (VS Code)
- Packages managers (Conda, Pip)
- Packages / Libraries (Pandas, Matplotlib)
- Virtual Environments

- Virtualenv
- Pipenv
- Venv
- Poetry

Programming Environment

- Interactive console
- Scripts
- Jupyter Notebooks
- Code editors
  (VS Code)
- Packages managers
  (Conda, Pip)
- Packages / Libraries
  (Pandas, Matplotlib)
- Virtual Environments
- Version Control Systems
  (Git, Github, Gitlab)

Essentials in the Python language

- Data types and structures
  - Numbers
  - Text (strings)
  - Iterables
  - Other

```
# Text and numbers
12          # int (integer)
1.5         # float
'hola'      # str (string)
"hola"
"""hola"""

# Iterables
[42, 58, 209, 42]  # list
(42, 58, 209, 42)  # tuple
{42, 58, 209}      # set
{'name': ['akiko', 'julie'], 'age': [12, 43]}  # dict (dictionary)
```

- Operators
  - Greater than, lower than

Class methods:

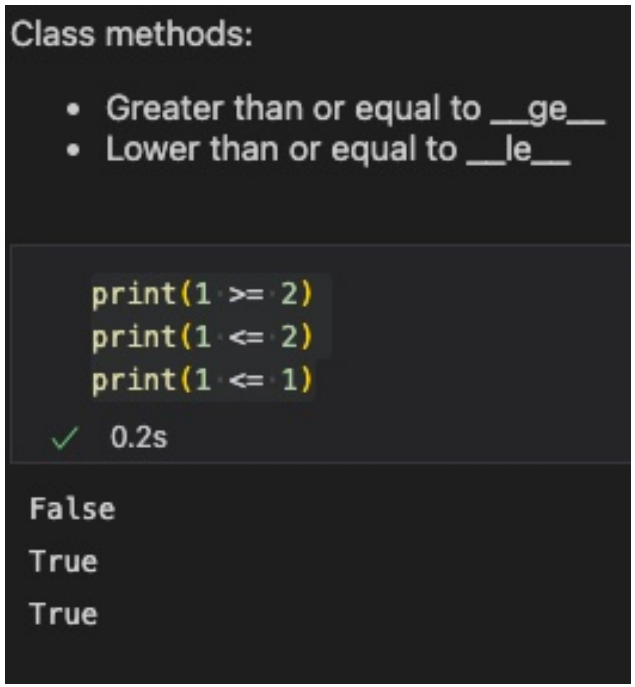
- Greater than __gt__
- Lower than __lt__

```
print(1 > 2)
print(1 < 2)
print(1 < 1)
print('a' < 'b')
print('a' > 'b')
print([1] < [2, 3])
print([1] > [2, 3])
```

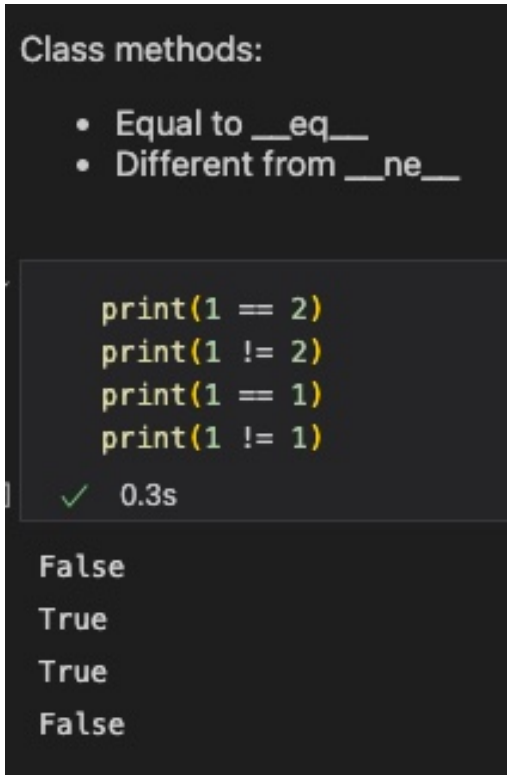
✓ 0.2s

```
False
True
False
True
False
False
True
```

- Operators
  - Greater than, lower than
  - Greater or equal than, lower or equal than

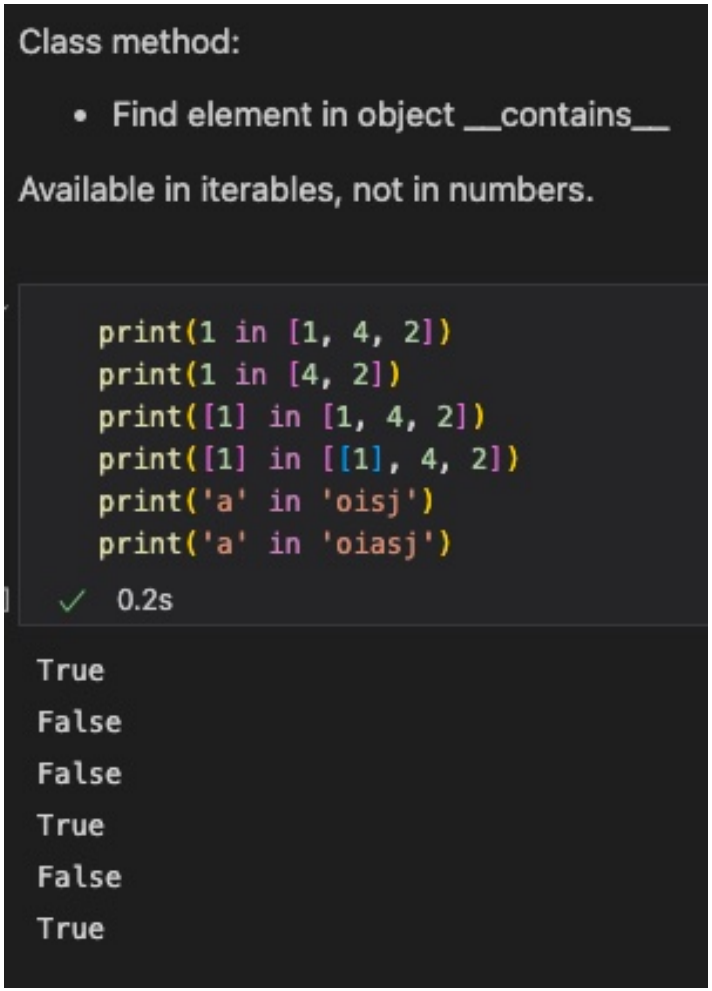


- Operators
  - Greater than, lower than
  - Greater or equal than, lower or equal than
  - Equals to, different from



Essentials in the Python language

- Operators
  - Greater than, lower than
  - Greater or equal than, lower or equal than
  - Equals to, different from
  - in



- Operators
  - Greater than, lower than
  - Greater or equal than, lower or equal than
  - Equals to, different from
  - in
  - not

Negation operator:

- not
- ~

```
print(1 == 1)
print(not 1 == 1)
print(~ 1 == 1)
print(1 != 1)
print(not 1 != 1)
print(~ 1 != 1)
```

✓ 0.2s

```
True
False
False
False
True
True
```

Essentials in the Python language

- Operators
  - Greater than, lower than
  - Greater or equal than, lower or equal than
  - Equals to, different from
  - in
  - not
  - is

Not a class method:

- Check if element is (True, False, None, np.nan)

```
print(True is True)
print(True is False)

print([1] is None)
x = 12
print(x is None)
x = None
print(x is None)

import numpy as np
print(x is np.nan)
```

✓ 0.2s

```
True
False
False
False
True
False
```

- Conditional structures
  - if else statement

Simple "if, else" condition

```
x = 3

if x == 3:
    print("Yes, x is equal to 3")
else:
    print("No, x is not equal to 3")
```

✓ 0.2s

```
Yes, x is equal to 3
```

Essentials in the Python language

- Control structures
  - for loop

```
for i in [2, 54, 39]:
    print(i)
```

✓ 0.1s

```
2
54
39
```

- Functions
  - A function can:
    - take no, to many arguments
    - arguments can be "Positional" or "Keyword" arguments
    - return nothing (None) or anything (to many things)
    - synonyms: arguments / parameters / inputs
  - One must:
    - Define the function
    - Call the function

Functions

Simplest form:

- No argument required
- No return

```
# Define the function
def say_something():
    print("Something")

# Call the function
say_something()
```

✓ 0.3s

```
Something
```

Function with:

- an argument
- no return

```
def say_my_name(name):
    print(name)

say_my_name("Alexis")
```

✓ 0.2s

```
Alexis
```

Function with:

- an argument
- a return

```
def square(x):
    return x**2

result = square(4)
result
```

✓ 0.3s

```
16
```

Function with:

- no argument
- multiple returns

```
def return_many_things():
    return 'alexis', 'bogroff', 'data'

return_many_things()
```

✓ 0.2s

```
('alexis', 'bogroff', 'data')
```

Function with:

- multiple arguments
- a return

```
def add(a, b, c):
    return a + b + c

result = add(4, 2, 9)
result
```

✓ 0.2s

```
15
```

Function with:

- no argument
- multiple returns

```
def return_many_things():
    return 'alexis', 'bogroff', 'data'

return_many_things()
```

✓ 0.2s

```
('alexis', 'bogroff', 'data')
```

Functions

Function with:

- a keyword argument
  - is thus optional
  - must be positioned after the positional arguments
- no return

```
def say_what_you_doing(name, course='data'):
    print(f"{name} doing {course}")

say_what_you_doing("Alexis")
say_what_you_doing("Alexis", "writing the course")
```

✓ 0.2s

```
Alexis doing data
Alexis doing writing the course
```

Function with:

- a (positional) argument awaiting a function
- no return

```
def complex_fct(func):
    print("This function will say")
    func()

complex_fct(say_something)
```

✓ 0.2s

```
This function will say
Something
```

Objects - init

```
class Truc:
    # Define instanciator (init)
    def __init__(self):
        self.age = 10
        self.name = 'truc'
```

✓ 0.3s

```
truc_1 = Truc()
```

✓ 0.2s

```
truc_1
```

✓ 0.7s

```
<__main__.Truc at 0x1077e4340>
```

```
print(truc_1.name)
print(truc_1.age)
```

✓ 0.3s

```
truc
10
```

- Define init
- Define properties / attributes (internal variables)
- Access through **self**
- Instanciate
- Object reference
- Access properties

Objects - method

```
class Truc:

    # Define instanciator (init)
    def __init__(self):
        self.age = 10
        self.name = 'truc'

    def present(self):
        print(f"My name is: {self.name}, "
              f"I'm {self.age} years old")
```

✓ 0.3s

```
truc_1 = Truc()
```

✓ 0.2s

```
truc_1
```

✓ 0.2s

```
<__main__.Truc at 0x33bffdb80>
```

```
truc_1.name
```

✓ 0.4s

```
'truc'
```

```
truc_1.present()
```

✓ 0.4s

```
My name is: truc, I'm 10 years old
```

- Create method
- Pass **self** argument
- Access attributes via **self.attribute**
- Re-instanciate object 
- New reference
- Access method via **self.method**

Objects - method with return

```
class Truc:

    # Define instanciator (init)
    def __init__(self):
        self.age = 10
        self.name = 'truc'

    def present(self):
        print(f"My name is: {self.name}, "
              f"I'm {self.age} years old")

    def dog_age(self):
        return self.age * 7
```

✓ 0.3s

```
truc_1 = Truc()
```

✓ 0.3s

```
dog_age = truc_1.dog_age()
```

✓ 0.3s

```
dog_age
```

✓ 0.3s

```
70
```

- Create method with return
- Set variable using return value

Objects - init with arguments

```
class Truc:

    # Define instanciator (init)
    def __init__(self, name, age):
        self.age = age
        self.name = name

    def present(self):
        print(f"My name is: {self.name}, "
              f"I'm {self.age} years old")

    def dog_age(self):
        return self.age * 7
```

✓ 0.3s

```
truc_1 = Truc('Yuko', 7)
truc_2 = Truc('Mila', 13)
```

✓ 0.5s

```
print(f"{truc_1.name} is {truc_1.dog_age()} old")
print(f"{truc_2.name} is {truc_2.dog_age()} old")
```

✓ 0.2s

```
Yuko is 49 old
Mila is 91 old
```

- Set specific init values
- Create different objects

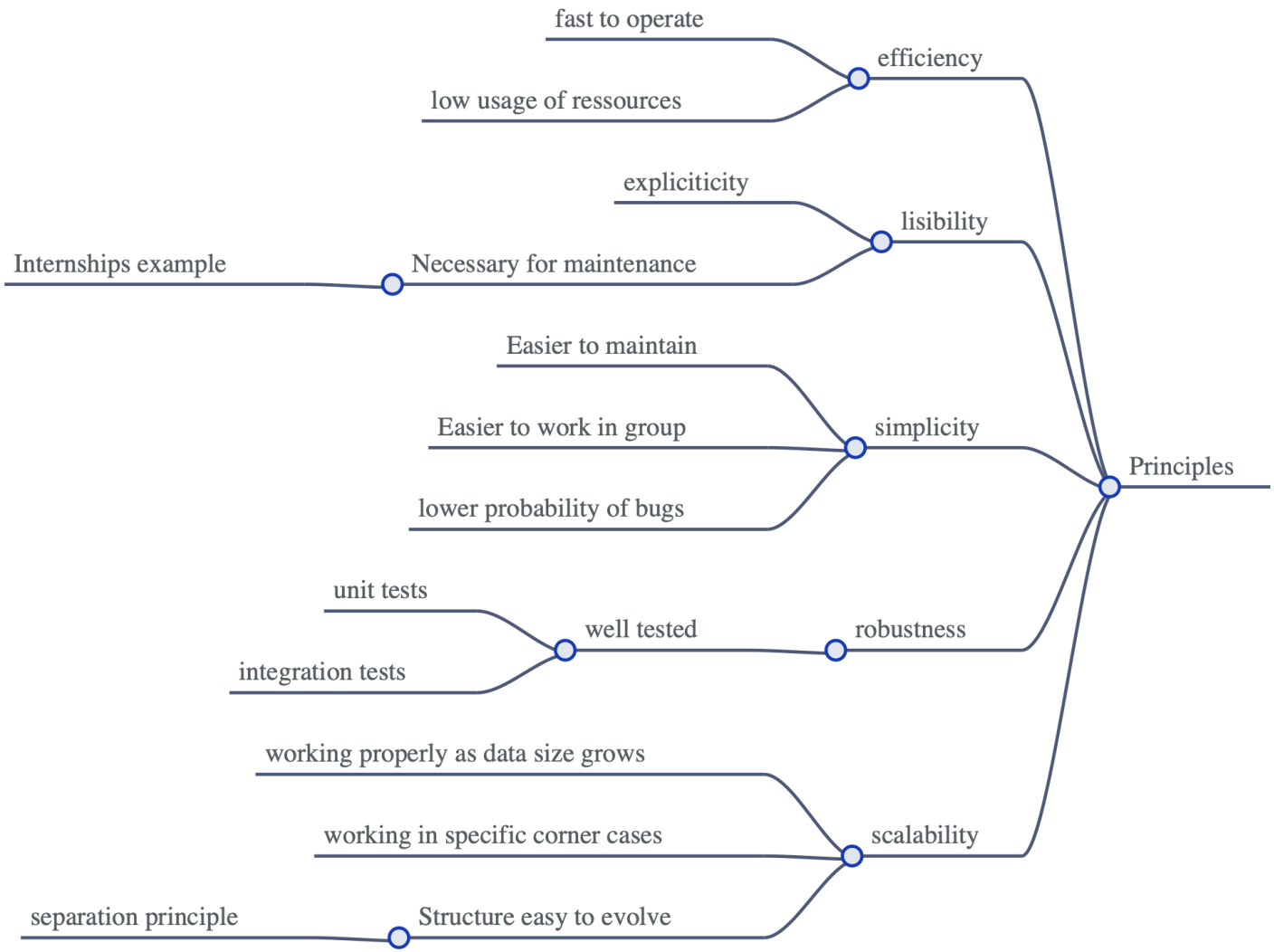
Coding conventions

- Names
  - variables: snake_style
  - constants: CAPITAL_SNAKE_STYLE
  - functions: snake_style
  - classes: First_letter_capital
- Spaces
- Max number characters by row: **79**
- Creation of iterables (lists, dicts)
- Comments
- File, functions, classes description

Coding conventions

- Code order in a file:
  - Description
  - Imports
  - Constants
  - Functions and Classes alphabetically
  - Body (functions calls, loops, variables)
- Code organisation between files (script):
  - Main file
  - Functions and Classes file
  - Settings file
- Code organisation files (Jupyter Notebook):
  - Load and prepare data file
  - Analysis file
  - Predictions file

Programming in general, good practices



Programming in general, good practices

- Vectorization
- Don't use loops when its possible to vectorize
- Same in Python, Matlab
- *This code is explained in the next course*

```
# /!\ Never do it this way (loops) /!\
for i, val in enumerate(df_temp['nums']):
    df_temp.loc[i, 'nums'] = val * 2
```

```
# Do this way (vectorized)
df_temp['nums'] = df_temp['nums'] * 2
```

Programming in general, other languages

- Difference between programming for:
  - Analysis, statistics
  - Software development
    - Front-end
    - Back-end

How to learn programming

- Trial and error
- Could be enough at first:
  - Python official documentation: *Here*
  - Exercises, i.e. on Coding game: *Here*
  - Google, Stackoverflow, Blogs: *Here*
- Progress:
  - Choose project (company, Kaggle, personal): *Here*
  - Peers, open-source project, e.g. Data For Good: *Here*
  - MOOC, e.g. machine learning on Coursera: *Here*