Python for datascience

track 1 - Python bases – 6th March 2021

Examen

Between 1h and 2h.

On last Friday (April 02nd).

Program

Saturday 06/03	4h	Python introduction + exercices
Saturday 13/03	4h	Pandas introduction + exercices
Friday 19/03	4h	Pandas exercices
Saturday 20/03	4h	Machine Learning introduction + exercices
Saturday 27/03	4h	Pandas + ML exercices
Friday 02/04	4h	Examen + exercices

Slack

https://bit.ly/30fHRnh

Outline

1 Quick history

2 Python, how to

3 Bases

4 Hand's on lab

0 Who am I?

Previously at kapten_.

Now a data engineering freelancer.

christophe.blefari@gmail.com

Data Engineer? Wut?

- Three data job exists as of today

  1. Data analysts

  2. Data scientists

  3. Data engineers

Data Engineer? Wut?

Data analysts work with data and answers to all business related questions like how many orders did we do last week? *etc.*

Data scientists are here to find correlation between data and develop models (machine learning based models for instance) that helps understand or predict the business.

Data engineers makes pipelines that get data from outside systems to the data systems.
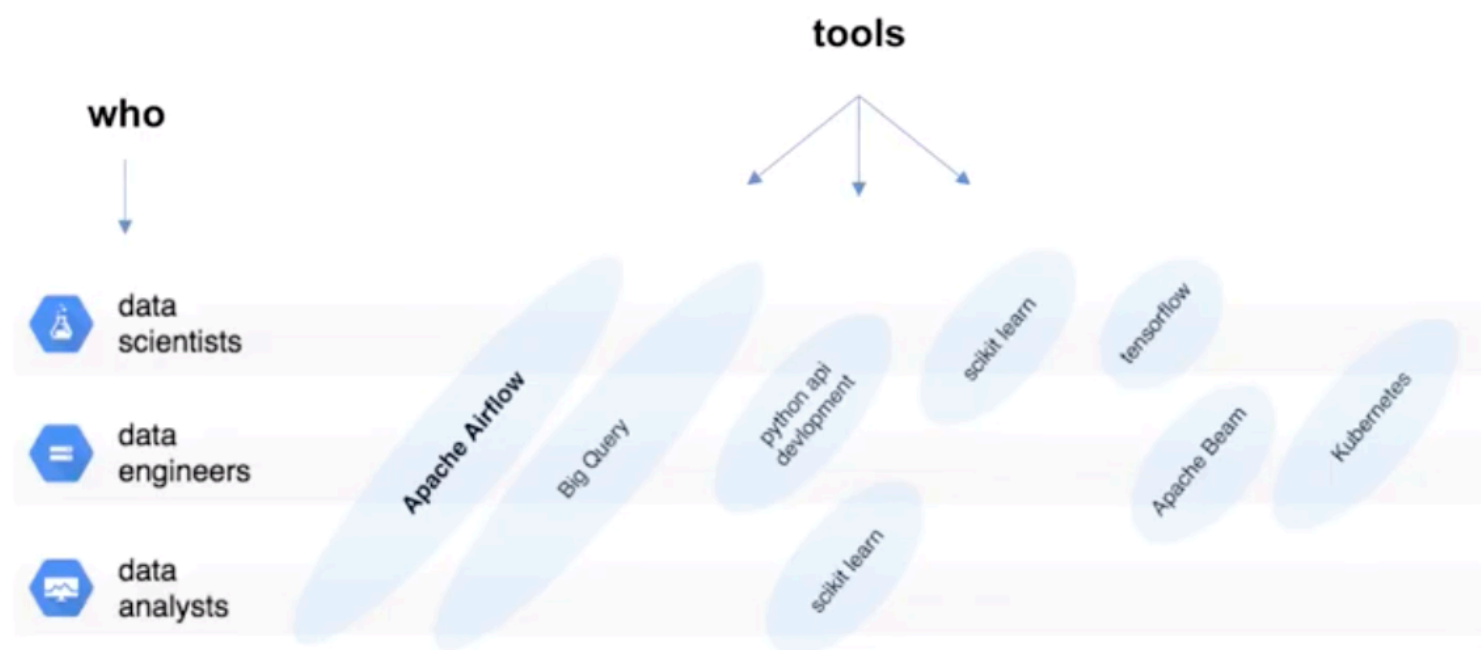
Data Engineer? Wut?

Quick history

Python v2.7
(major release)
End of life 2020

Python v3.8

Python v1.0

2020

2010

1994

Jan.
2020

2006

1991

pandas v1.0.0

NumPy start by Travis
Oliphant

Started by Guido Van
Rossum
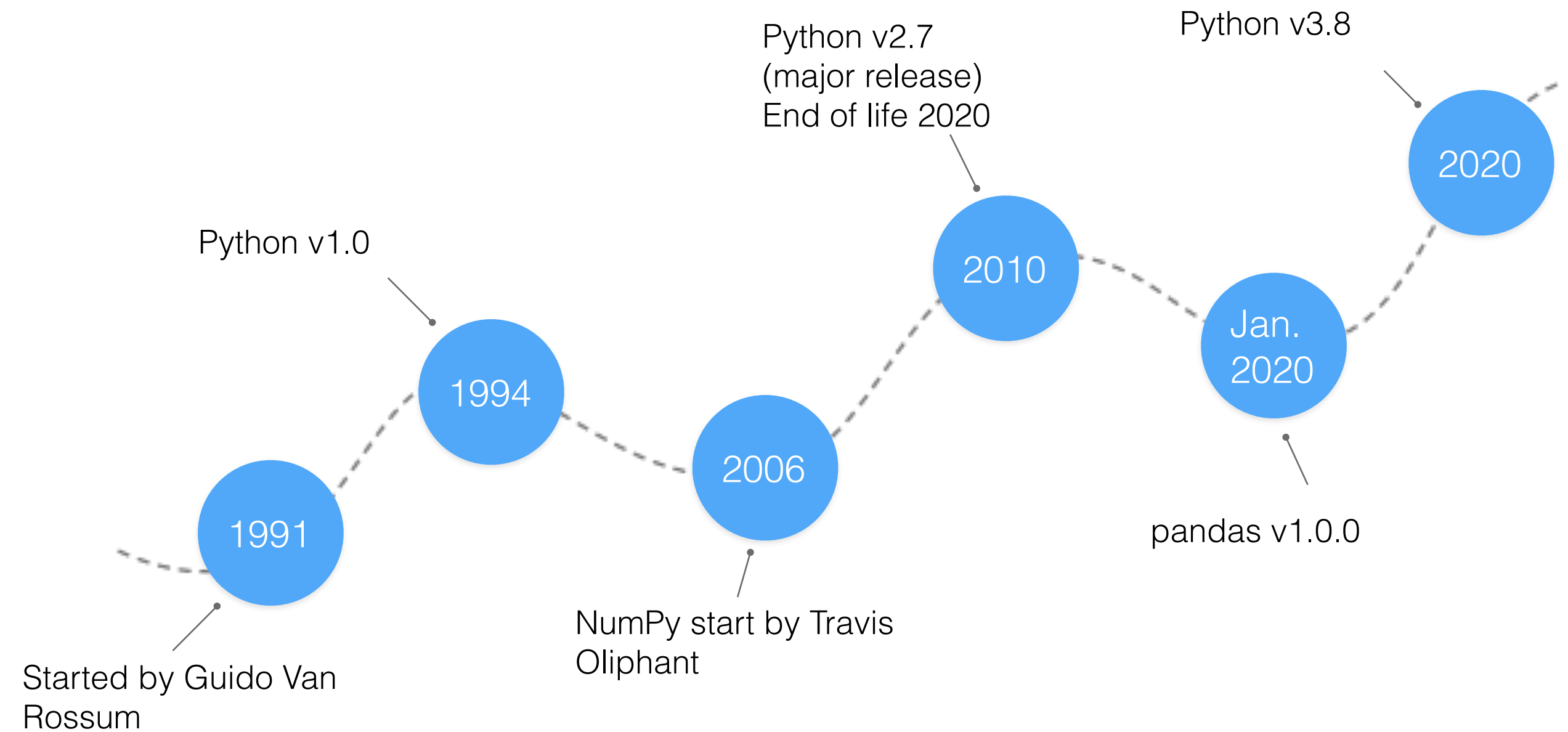
1 Quick history: remarks

- There are a lot of web resources about Python

- Use conventions

- Versioning your code

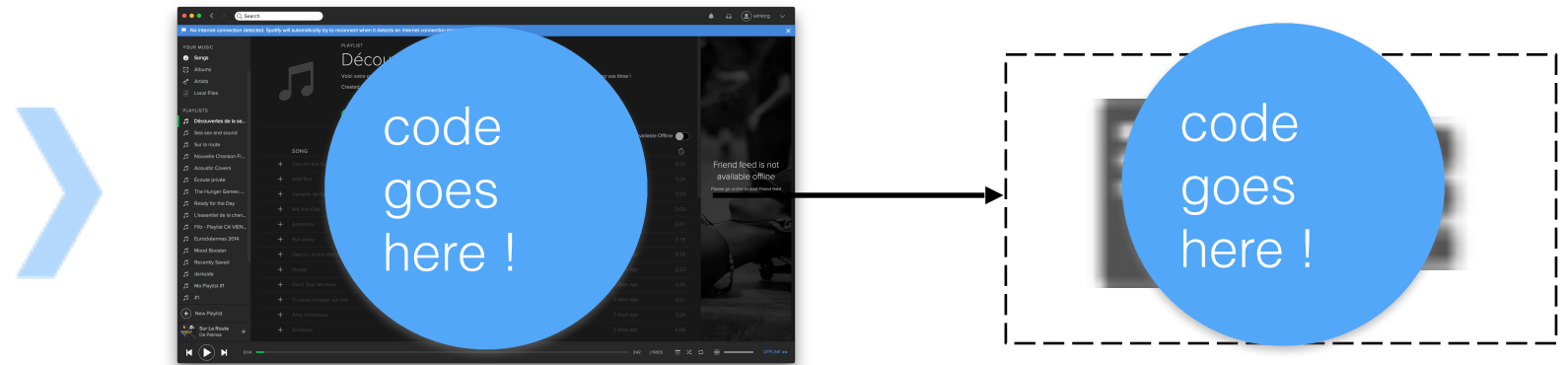- You can start learning Python with 3.8 or 3.9.

- Pros:
  - ‣ Big community
  - ‣ Simple and fast learning: productivity gain
  - ‣ Easy to script, to read
  - ‣ Lot of packages on PyPI for everything
  - ‣ Interpreted

Quick history: pros/cons

- Cons:
  ‣ No real typing (: speed issues)
  ‣ Performance (Python is high-level)
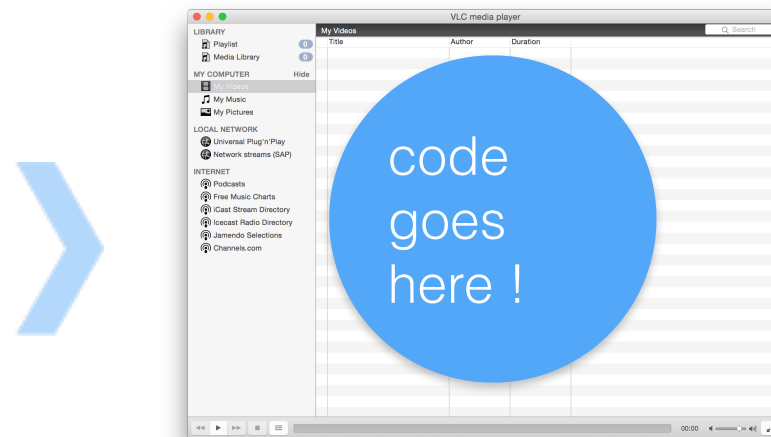
you often use **local** computer

code
goes
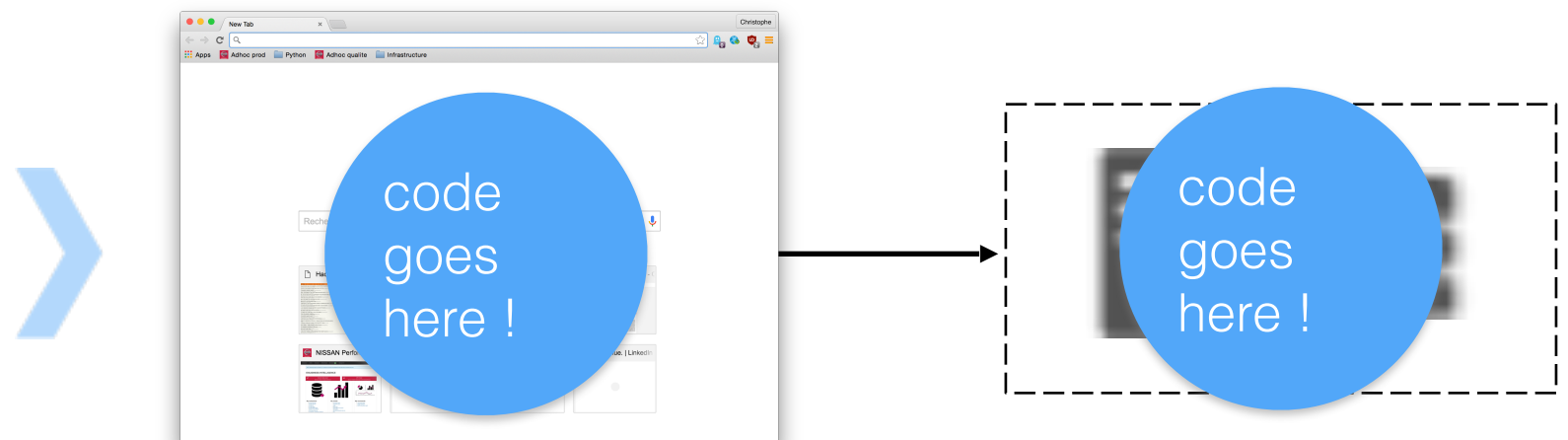here !

for your own activities

code
goes
here !

code
goes
here !

code
goes
here !

but it connects to **distant** servers

*to make things happens
(ex: get data)*

code
goes
here !

code
goes
here !

2 Global context

local machine

distant machines called servers

Every machine has an unique id that is the **MAC address**
And a network id that is the **IP address**

ssh, telnet, tcp

we use **terminals** to execute programs
more generally to do everything

we write code with program called **IDE** or text
editors

we **release** and **deploy** code on distant machines
with versionning tools like Git, SVN and
deployment tools like Ansible, Chief, etc. or
custom scripts

Today will see **Notebooks** that are web interfaces
where it's easy to develop, actually the code is
executed on a machine but from the web browser
with a socket mechanism between the browser
and the machine

Distant machine are kept in **datacenters**.

Several providers of distant machines like:
• aws, google cloud engine, microsoft azure
• ovh, 1&1, digital ocean, gandhi
• jolicloud, heroku

On a distant machine you can install everything
from scratch like on your computer:
• windows, linux in graphical or server mode

When an OS is installed you can start hacking
with your favourite language.

On the OS you can install software like
databases solutions (MySQL, PostgreSQL,
Elastic Search, etc.).

Remember: a server is just a computer that is not on your desk

15

2 Python, how to use?

- You can use Python from
  ‣ command line by typing `python`
  ‣ ipython notebook server
  ‣ Anaconda packages

- With virtualenv to isolate your packages versions

Python, how to *develop*

- You need to use **virtualenvs** to separate your different projects with different versions of packages

```
~ $ python3.4 -m venv /Users/Blef/dev/python/virtualenvs/test
~ $ source /Users/Blef/dev/python/virtualenvs/test/bin/activate
(test) ~ $ pip install requests
You are using pip version 6.0.8, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting requests
  Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
    100% |################################| 471kB 821kB/s
Installing collected packages: requests

Successfully installed requests-2.7.0
```

- We'll use iPython notebook because it's more easier to develop with for data analysis (and for one TD one IDE).

- But, for development we develop scripts inside files and execute them from command line:

- `$ python file.py`

- So, run your notebook server with anaconda

- Yo also can develop with programs likes

  ‣ sublime-text, notepad(++), IDE, vim/emacs…

2 Python, how to [debug](#)

- Do debug! DO!

- Go on Google

- Read exceptions

- Don't be afraid ; time and experience with programming will give you reflexes

```
1  # We assign a with the 4 value
2  a = 4
3  # We can reassign a with a new value
4  a = 2
5
6  # So now a has 2 value inside
7  # We can assign 40 to b
8  b = 40
9
10 # Make operations and assign to another variable
11 c = a + b
12
13 # Which value has c
14 # You can output the value with 'print'
15 print(c)
16
17 # Multi assignation
18 d, e, f = 1, 2, 4
```

- variable name can't start with a number

- By convention:

  ‣ constants are in CAPS

  ‣ names are > 3 letters

  ‣ spaces around '='

  ‣ use clear names

3   Bases: Types

```python
c = 42
# We can get the type of the variables with 'type'
type(c)
# Out: int

d = 2.4
type(d)
# Out: float

my_string = "Hello world!"
type(my_string)
# Out: str

my_list = [1, 1, 2, 3, 5, 8, 13, 21]
type(my_list)
# Out: list

my_dictionary = {"barack": 59, "françois": 61, "angela": 61}
type(my_dictionary)
# Out: dict

my_range = range(10)
type(my_range)
# Out: range

```

- Null element is **None** with a cap

- Python interpreter lets you make calculs

```python
1  a = 4
2  b = -4
3  # Multiply 'a' with 'b' and print
4  print(a * b)
5  # Out: -16
6  # You can also divide '//', add '+' and subtract '-'
7
8  # You can get the rest of the division with %
9  e = 23 % 2  # e will get 1
10
11 c = 1.5
12 d = 2.5
13 # Multiply 'c' with 'd' and print
14 print(c * d)
15 # Out: 3.75
16 # You can also divide '/', add '+' and subtract '-'
17
18 # You can combine 'float' and 'int' to get a new float
19 print(a * c)
20 # Out: 6.0
```

- Here we detailed base operations

- But you can combine variables to build a more complex expression

```python
1  # We can create an empty list, two methods
2  a = []
3  a = list()
4
5  # or not
6  b = [1, 2, 4, 6, 8]
7  c = ["hello", "how", "are", "you?"]
8
9  # use different types in a list
10 d = [1, 1.5, "hello"]
11
12 # The lists are indexed to get items
13 b[0]
14 b[-1]
15 b[::-1]
16 b[1:3]
17
18 # some methods are available to manipulate lists
19 a.append(10)  # append 10 to the list
20 d.pop()  # pop the last item of the list (here: 'hello')
21
22 # or to operate on the list without modifying the list
23 len(a)  # returns the length of the list
24
25 " ".join(c)
26 # returns list items joined with a space (: "hello how are you?")
```

- Lists have lot of built-in methods

- We can **iterate** over a list (see after)

- We can **sort** a list

- Some methods:

  ‣ Change (or not) the list

  ‣ Return (or not) a result

- All types can be addable in a list (like a list for instance)

- A list is **always ordered**

- 🚧   Indexing starts at 0

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

```python
1  # An empty string
2  a = ""
3
4  # Or not
5  b = "Christophe"
6
7  # Strings support indexing (careful indexing starts at 0)
8  b[0]  # returns ?
9  len(b)  # returns 10
10
11 # We can format a string, two syntax:
12 c = "Hello my name is %s" % b  # Very old way
13 d = "Hello my name is {0}".format(b)  # Old way
14 e = f"Hello my name is {b}" # New way
15
16 # Of split a string
17 columns = "Age;Name;FirstName".split(";")
18
19 # Make upper
20 b_up = b.upper()
```

- A str can be behaviour is like a list sometimes

  ‣ Indexed (can get item and has a length)

  ‣ Can be iterated

  ‣ Can be sorted

Bases: Types operations (str)

```python
1  # Declare a variable a
2  a = "Christophe"
3
4  # Count how many times `ophe` appears in a
5  a.count("ophe")
6
7  # Find the first occurrence of a given substring
8  a.find("h")
9
10 # Returns true if each character is a letter
11 a.isalpha()
12
13 # Returns true if each character is a number
14 a.isdigit()
15
16 # Replace first argument occurrence by the second argument
17 a.replace('ophe', 'ine')
18
19 # Join a list of str using a another string
20 ",".join(["Hello", "it's", "me", "Mario!"])
```

```
 1  import random
 2
 3  # I defined a variable
 4  a = "book"
 5
 6  # I defined a string using a f-string
 7  my_str = f"I read a {a}!"
 8
 9  # More complicated
10  print(f"A random number {random.randint()}")
11
```

- fstrings are a new type of string that appears recently in Python

- fstrings are evaluated like a string but variable inside are interpreted

```
1  # An empty dict
2  my_dict = {}
3  my_dict = dict()
4
5  # Or with values
6  my_dictionary = {"barack": 54, "françois": 61, "angela": 61}
7
8  # returns an iterator
9  my_dictionary.values()
10 my_dictionary.keys()
11
12 # To combines keys and values
13 my_dictionary.items()
14
15 # To get an element
16 my_dictionary.get("angela")
17 my_dictionary["angela"]
18
19 # To update the dict
20 my_dictionary.update({"barack": 28})
21 my_dictionary["barack"] = 28
22
23 # To get the length
24 len(my_dictionary)
```

- You can do almost everything with Python base structures (list + dict)

- A dict **is not always ordered (depends on Python version)**

```python
# You can iterate over a simple list
a = [1, 2, 3, 4, 5]
for element in a:
    print(element * 2)

# Over an iterator
for element in range(10):
    print(element)

# Over a string
for element in "Christophe":
    print(element.upper())

# Over a dict
for key, item in my_dictionary.items():
    print(f"{key} is {item} years old")

# And you can use while (but be careful!)
value = 10
while value > 0:
    print(value / 2)
    value = value - 1
```

- You can loop over lists or iterators

- element is a variable usable only in the 'for'

- The indentation of the code is very important in Python

Bases: If statements

```python
# Simple if with a simple condition
a = 10
if a > 20:
    print("Yes!")

# An if with an else, if 'if' is false then else is executed
b = 34
if a < 20 and a < b:
    pass
else:
    print('Bouh :(!')

# Same as previous but we test two conditions
if a == 1:
    a = 4
elif a is not None:
    print('Cool')
else:
    print('Sad!')
```

- If statements are based on **boolean** value

- It's only logic and you can combine everything to make **logical** expression:

  ‣ or, and, in

  ‣ not, is

  ‣ >,<, <=, >=, ==

- Python syntax is verbose and simple

- You can use parenthesis to factorise expressions

- **T**rue and **F**alse are capitalised

```python
ITEMS = [1, 2, 3, 4, 5]
ITEMS_BIS = [6, 7, 8, 9, 10]

# You can multiply a list if you need to repeat
print(ITEMS * 2)
NEW_ITEMS = ITEMS + ITEMS_BIS

# You can in one instruction iterate over the list
# It called list comprehension
NEW_ITEMS = [i * 2 for i in ITEMS]

# More complex list comprehension
NEW_ITEMS_EVEN = [i * 2 for i in ITEMS + ITEMS_BIS if i % 2 == 0]

# It's the same than (but more concise)
NEW_ITEMS = []
for i in ITEMS:
    NEW_ITEMS.append(i * 2)

# Some operations are useful when we use lists like map and
filter, sorted
NEW_ITEMS = map(lambda x: x * 2, ITEMS)
FILTERED_ITEMS = filter(lambda x: x % 2 == 0, ITEMS)
SORTED_ITEMS = sorted(ITEMS, reverse=True)
```

- Lists are the most useful structure in Python for data analyses

- You need to master the list operations

- Index starts at 0 (reminder!)

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

30

map reduce   x2

[1 , 4 , 10 , 5 , 3] } map
                           x2
[2 , 8 , 20 , 10 , 6]

                        } reduce
[10 , 20 , 10 , 6]

[30 , 10 , 6]

[40 , 6]

(46)

Scalaire

```python
1  # You first function
2  def greetings(name):
3      sentence = f"Hello {name}"
4      print(sentence)
5
6  greetings("Christophe")
7  greetings("Jacques")
8  # /!\ Variable greetings not accessible from this part of the code
9
10
11 # Functions can return a value
12 def add(a, b):
13     """ This method returns a sum between two variables a and b
14     :param a: first param to add
15     :param b: second param to add
16     :return: the sum between a and b
17     """
18     return a + b
19
20 c = add(3, 5)
21 print(c)  # print 8
22
23 # Sometimes we want to be precise with function parameters
24 def identity(name, age):
25     print(name, age)
26
27 identity(name="Roméo", age=31)
28 identity(age=34, name="Juliette")
29
```

- Functions can return a value (if not it returns **None**)

- We can say also a **method** or a **procedure**

- By convention we give explicit names to functions and we comment a lot the code

- A function has a **signature**

  ‣ Here add signature is to have *a* and *b* as **parameters**

32

```
1  # I create a function
2  def add_10(i):
3      return i + 10
4
5
6  # I use it
7  print(list(map(add_10, [1, 2, 3])))
8
9  # this is equivalent
10 print(list(map(lambda i: i + 10, [1, 2, 3])))
```

- A lambda is an anonymous function (meaning function without a name) and can be used sometimes in Python to simplify the code

```python
1  # You first function
2  def greetings(name):
3      sentence = "Hello %s" % name
4      print(sentence)
5
6  # Main block
7  if __name__ == "__main__":
8      greetings("Christophe")
9      greetings("Jacques")
10     greetings("Emmanuelle")
11
```

- We use the "main" block to separate the code execution and the import call

- i.e. if we import a module all code except the main will be executed

- If we call the file himself the main will be executed

34

Bases: Open and read files

```python
1  # Method 1
2  with open("file.csv", "r") as my_file:
3      data = my_file.read()
4      rows = data.split('\n')
5
6  print(rows)
7
8  # Method 2
9  f = open("file.csv", "r")
10 data = f.read()
11 rows = data.split('\n')
12 f.close()
13
14 print(rows)
```

- There are two methods to **open** files, I prefer the first one presented here

- In data analyses you often use files, so this snippet on code is very important

- Next week we'll see how to connect to other kinds of data (like databases or API)

- open() second parameter is the open mode: here we read the file so "r"

```python
data = [
    ["Name", "Gender", "Age"],
    ["A", "Male", 5],
    ["B", "Male", 10],
    ["C", "Female", 20],
    ["D", "Female", 30],
]

with open("file.txt", "w") as f:
    for item in data:
        f.write("%s\n" % ";".join(item))
```

- The file will be created in the path given

- Here we write the file so "w" for the open mode

Bases: Imports

```python
1  # You can import a simple package
2  import datetime
3
4  # Import with an alias
5  import pandas as pd
6
7  # Import a specific method or module in a package
8  from sys import path
9  from os.path import splitext
10
11 # Or all but it's not advised because everything will be in you
12 code
13 from sys import *
14
15 # So for usage
16 today = datetime.datetime.now()
17 dataframe = pd.DataFrame()
18 name, ext = splitext("path")
```

- We you want to develop you will always have to use external packages **imports** are the key

- Hint: order your imports at the top of the file alphabetically

Bases: datetime

```python
import datetime

# We can have the today datetime
today = datetime.datetime.now()
today_date = today.date()

# We can parse datetimes with a given format
date = datetime.datetime.strptime("2015-01-01", "%Y-%m-%d")

# And we can format datetime
date.strftime("%Y-%m")

# We can also substract or add days to a given datetime
tomorrow = today + datetime.timedelta(days=1)
yesterday = today - datetime.timedelta(days=1)
```

- Date times formats are described in the official docs

Bases: Exceptions

ZeroDivisionError

```
>>> 4 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- An exception is an issue **raise** by the code

- You can choose to **catch** exceptions and so write limit cases of your code

- If the exception is not caught somewhere the code will fail and you will get a **traceback**

TypeError

```
>>> 5 + "rrr"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

SyntaxError

```
>>> def hello(p1 p2):
  File "<stdin>", line 1
    def hello(p1 p2):
                   ^
SyntaxError: invalid syntax
```

And a lot more …

Bases: Exceptions

```python
 1  # A try except block is to catch exceptions
 2  # In other languages we call him try/catch block sometimes
 3  try:
 4      data = [1, 2, 3]
 5      last = data[4]
 6  except IndexError as er:
 7      print("Yes! We caugth you babe: %s" % er)
 8  except Exception as er:
 9      print("Unexpected error happened")
10  finally:
11      # This block code is always executed at the end of try except
12      pass
```

- An exception is an issue **raise** by the code

- You can choose to **catch** exceptions and so write limit cases of your code

- If the exception is not caught somewhere the code will fail and you will get a **traceback**

40

```python
# Everything is defined and a line can't contains more than 80
characters
# At the end of your file you must have a blank line
# You have to use all your imports
import os


# Here we have two spaces between import and function
def hello(name):
    a = name  # Spaces around '=' (note the two spaces before '#'
    print("Hello %s" % a)  # Here spaces around '%'

    if name == "Christophe":
        print("Oh yeah! Same name than me.")

hello("Christophe")
hello(name="Christophe")  # But here no space around '='
```
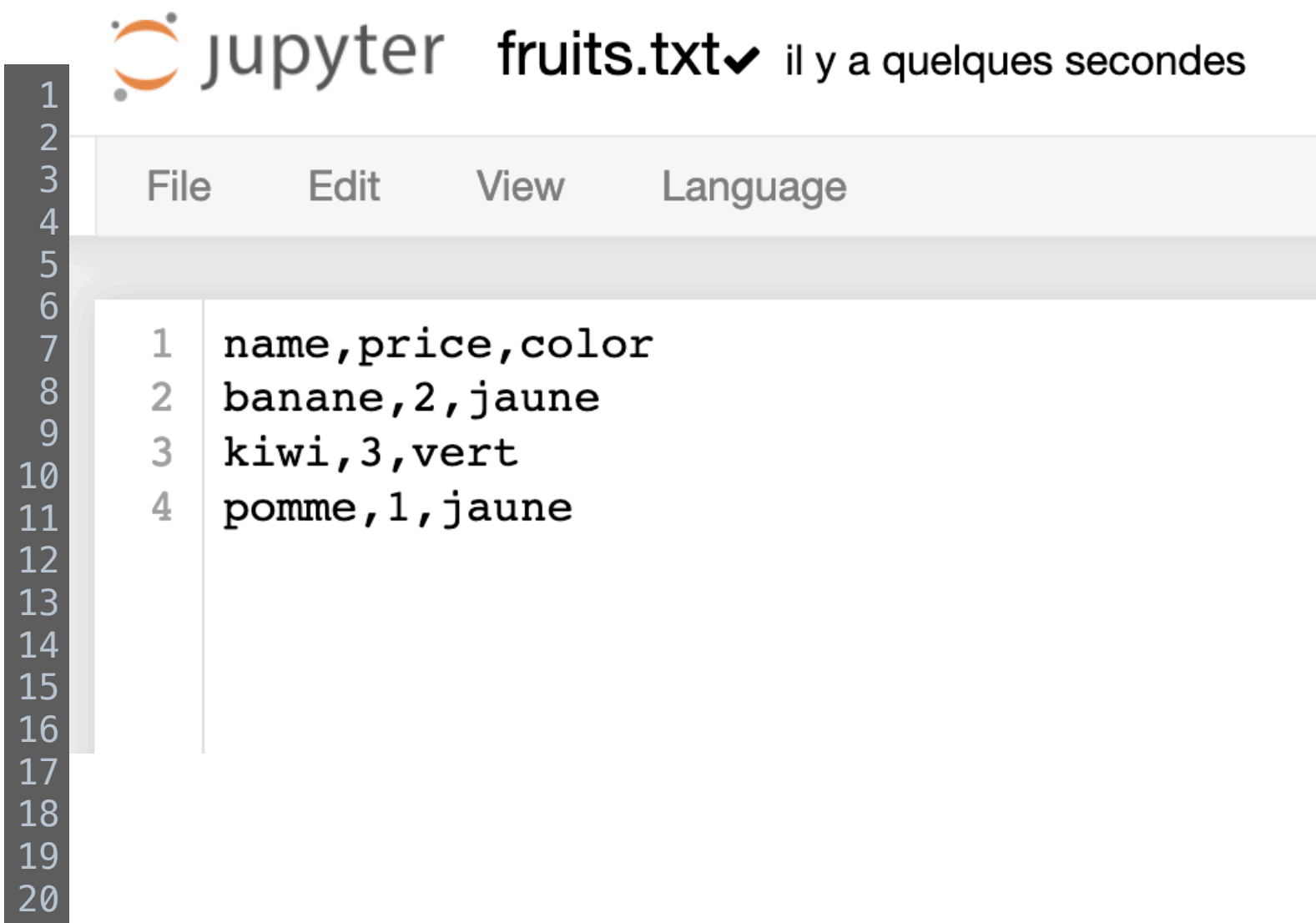
- PEP8 is a convention to write clean code and readable by anyone

41

```
1 ord('a')
2 # returns 97
3
4 chr(97)
5 # returns 'a'
6
7 # Decrypt this message "Qebobfpkljbpp^db"
8 # We subtract 3 to each letters of the origin message
```

- ord and chr are useful to get character number in the ASCII table

Jupyter fruits.txt✔ il y a quelques secondes

File   Edit   View   Language

```
1  name,price,color
2  banane,2,jaune
3  kiwi,3,vert
4  pomme,1,jaune
```

Create a CSV with 3 columns et 4 rows.

1. Open the CSV
2. Create a list of list of data
3. Count with a for #columns
4. Count with a for #rows

Bases: Practice #2

```
1  # Write a True condition with a greater than
2  a = 12 > 5
3
4  # Write a False condition with a less than
5  b = 45
6  c = 67 < b
7
8  # Write a True condition by combining 'or' + 'and'
9  d = 1 < 2 and (a or c)
10
11
12 # Write a function that returns if a number is odd 'is_odd'
13 def is_odd(number):
14     return number % 2 == 1
```

Booleans.

3 Bases: Misc

- Python driven by the indentation in your code

- Don't hesitate to comment your code

- Code slowly and test at every step your code

- Exceptions and errors in Python are very clear, so **read them please** 😥

4 Hand's on lab

- We'll go through:
  - ✓ simple snippets of Python code
  - ✓ algorithm problems
  - ✓ practical data cases

- Google* is your friend if you get stuck

- I'm your friend too

Hand's on lab: Simple programs

1. Write a program that prints a given name

2. Write a function that compares a given date (YYYY-MM-DD) with the present date and returns true if superior and false if not

3. Write a program that lists files in a given directory (use os.walk())

4. Write a *"Rock-Paper-Scissors"* game and play against the computer

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.

Hand's on lab: Algorithms problems

Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.

```
37107287533902102798797998220837590246510135740250
46376937677490009712648124896970078050417018260538
74324986199524741059474233309513058123726617309629
91942213363574161572522430563301811072406154908250
23067588207539346171171980310421047513778063246676
89261670696623633820136378418383684178734361726757
28112879812849979408065481931592621691275889832738
44274228917432520321923589422876796487670272189318
47451445736001306439091167216856844588711603153276
70386486105843025439939619828891759366568675793 4951
62176457141856560629502157223196586755079324193331
64906352462741904929101432445813822663347944758178
92575867718337217661963751590579239728245598838407
58203565325359399008402633568948830189458628227828
80181199384826282014278194139940567587151170094390
35398664372827112653829987240784473053190104293586
86515506006295864861532075273371959191420517255829
71693888707715466499115593487603532921714970056938
54370070576826668462461495650076471787294438377604
53282654108756828443191190634694037855217779295145
36123272525000296071075082563815656710885258350721
45876576172410976447339110607218265236877223636045
17423706905851860660448207621209813287860733969412
81114266041808683061932846080811191061556940512689692
51934325451728388641918047049293215058642563049483
62467221648435076201727918039944693004732956340691
15732444386908125794514089057706229429197107928209
55037687525678773091862540744969844508330393682126
18336384825330154686196124348767681297534375946515
80386287592878490201521685554828717201219257766954
78182833757993103614740356856449095527097864797581
16726320100436897842553539920931837441497806860984
48403098129077791799088218795327364475675590848030
87086987551392711854517078544161852424320693150332
59959406895756536782107074926966537676326235447210
69793950679652694742597709739166693763042633987085
41052684708299908521139942736573411618276031 5001271
(…)
```

Given two strings *s* and *t* of equal length, the **Hamming distance** between *s* and *t*, denoted *dH(s, t)*, is the number of corresponding symbols that differ in *s* and *t.*

Given: Two DNA strings s and t of equal length

Return: The Hamming distance *dH(s, t)*

GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT

This input should returns a distance of 7

Batman wants to escape

https://www.codingame.com/ide/puzzle/shadows-of-the-knight-episode-1

A Resources

- Python website: https://www.python.org/

- Codecademy: https://www.codecademy.com/tracks/python

- Python 3 docs: https://docs.python.org/3/contents.html

- Project Euler: https://projecteuler.net/

- CodeWars: https://www.codewars.com/

- Coding Game: https://www.codingame.com/

- http://dataquest.io

B Course index

- Variables

- Types

- Types operations (int + float)

- Types operations (list)

- Types operations (str)

- Types operations (dict)

- Loops

- If statements

- Types operations (list) - advanced

- Functions

- Main block

- Open and read files

- Open and write files

- Imports

- datetime

- Exceptions

- PEP8

C How to version your code with Git

- If you are on linux install git (apt-get install git)

- On windows go to: https://git-scm.com/downloads

D Use pip // virtualenvs

```
$ python -m venv ENV_DIR            create a virtualenv

$ source ENV_DIR/bin/activate       activate the venv

$ pip install package               install a package

$ pip freeze > requirements.txt     freeze the version of the venv

$ pip uninstall package             uninstall a package

$ pip install -U package            upgrade a package

$ deactivate                        unactivate the venv

$ pip install package==<version>    install a specific version
```

0   Thanks a lot!