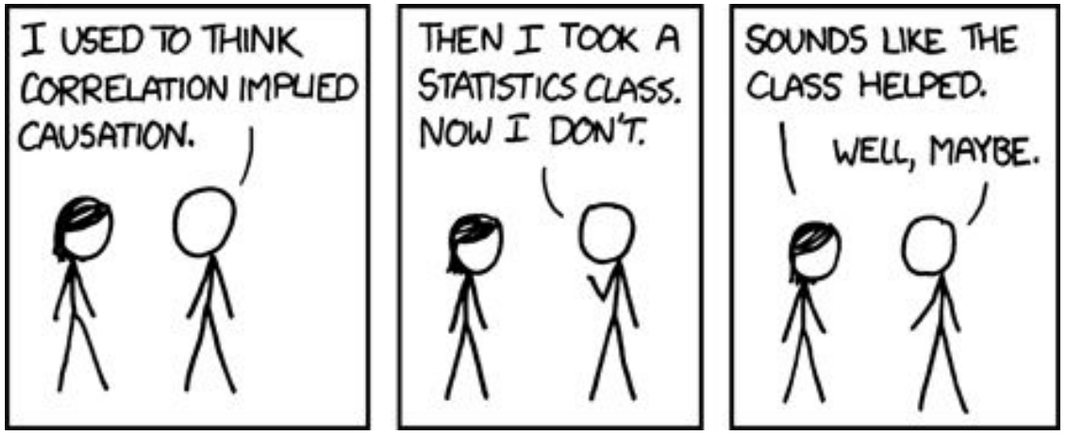
Python for Data Science

Python Core

ENSAE-ENSAI
Formation continue
(Cepe)

Hi !

- I am **Koby**
- **Data Scientist** @ Equancy (Marketing Consultancy)
  - Previously: Data Scientist at GDF SUEZ (Engie)
- Organizer of the **Kaggle** Meetup in Paris
- Studied Robotics in Tel Aviv University (2007)
  - Machine Learning in UPMC (2012)
  - Master of Business Administration (2013)



3-days plan

1. Pure Python - 1/2d
2. Pandas - 1/2d
3. More Pandas & Matplotlib - 1/2d
4. Scikit-Learn & Machine Learning - 1d
5. Spark - 1/2d

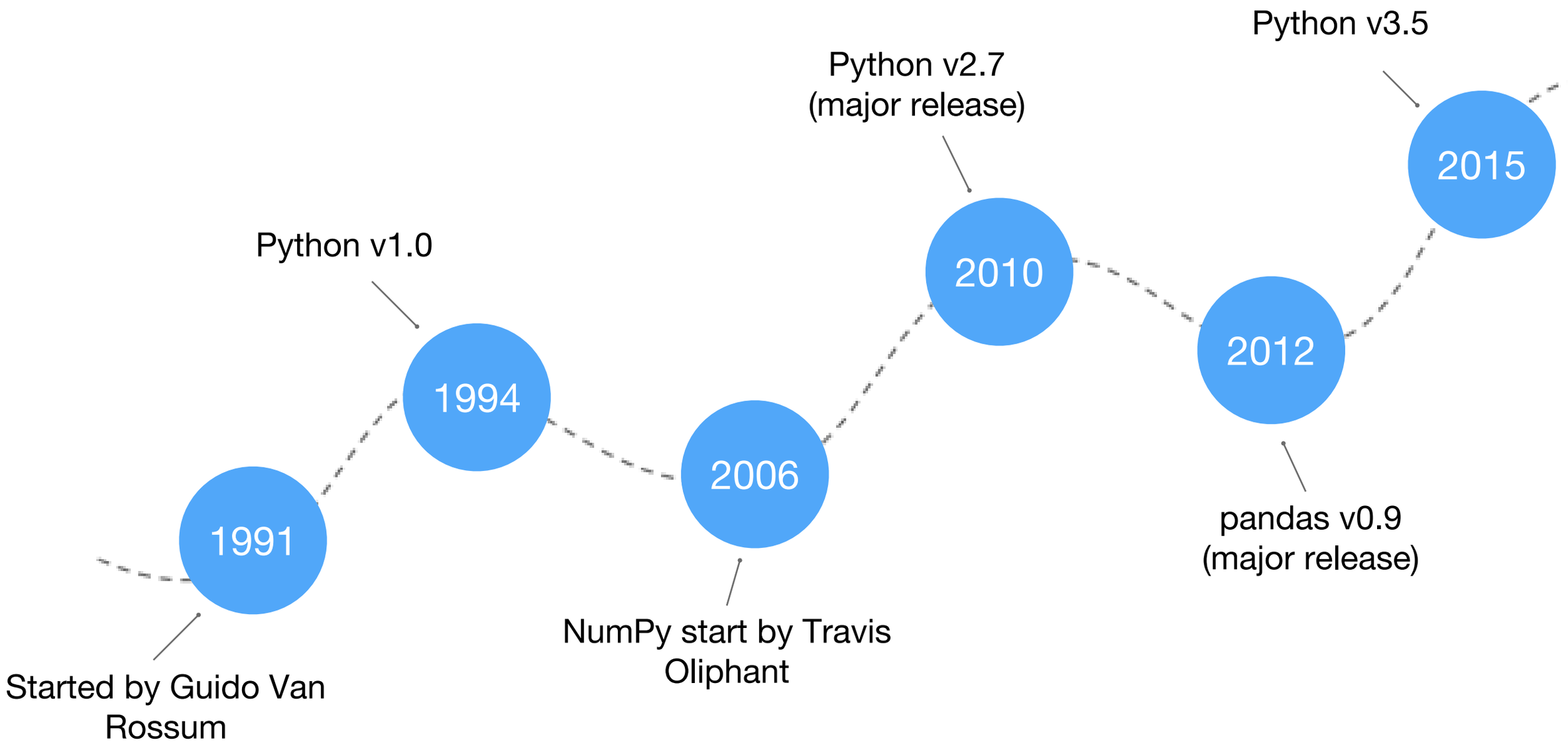
ENSAE-ENSAI
Formation continue
(Cepe)

Outline

1 Overview

2 Basics & Exercises

1 Quick history



1 What is Python

- Python is a Programming Language

- Three major version:
  - ➢ 1.X (1994) - Historical version
  - ➢ 2.X (2000) - Most common version
  - ➢ 3.X (2008) - The version we should all be using

- Python is an interpreted version and not compiled:
  - ➢ Runs on-the-fly
  - ➢ Easy debugging
  - ➢ No optimizations be compiler (slower…)

1 How to run Python?

1 - Python Console

- Enter one line of Python code
- Execute
- Display Output

This is called REPL - Read-Eval-Print Loop

```
karp@K:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello')
Hello
>>> exit()
```

1 How to run Python?

2 - Launch scripts

- Python files have the extension “py”
- The file script.py contains a single line:

script.py

```
print('this is the first line of script.py')
~
```

- To launch it we simply type “python <script name>”:

```
karp@K:~$ python script.py
this is the first line of script.py
```

- It’s good to use an IDE (integrated development environment) such as PyCharm to make coding more efficient, fast and fun

1 How to run Python?

IPython / Jupyter - interactive computing

Best alternative for analysis and debugging

- Command shell: *ipython*

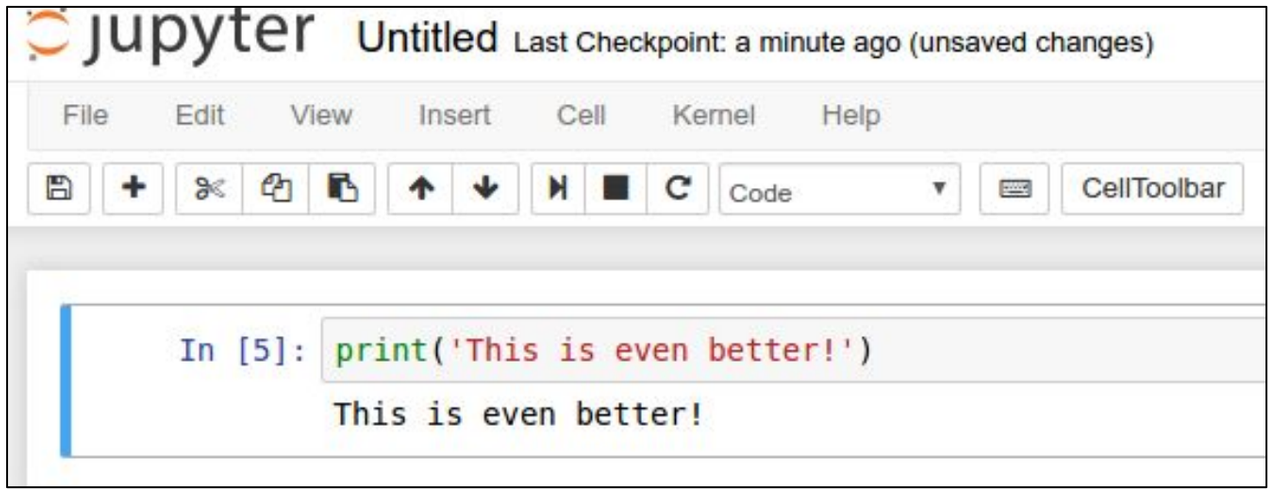
```
(kaggle)karp@K:~$ ipython
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
Type "copyright", "credits" or "license" for more information.

IPython 4.1.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: print('This is so much better than the python shell !')
This is so much better than the python shell !

In [2]:
```

- Browser-based: *ipython notebook*



1 Quick overcap: pros/cons

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

1 Quick history: pros/cons

- Cons:
  - Low performance compared to C or Java
    - This is (also) due to dynamical typing

1 Let’s code!

2 Basics: Variables

```
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
19
```

- variable name can’t start with a number
- By convention:
  - constants are in CAPS
  - names are > 3 letters
  - spaces around ‘=‘
  - use clear names

2 Basics: Types

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


d = 2.4
type(d)


my_string = "Hello world!"
type(my_string)


my_list = [1, 1, 2, 3, 5, 8, 13, 21]
type(my_list)


my_dictionary = {"barack": 54, "françois": 61, "angela": 61}
type(my_dictionary)


my_range = range(10)
type(my_range)

```

- In python types are dynamic: types are set during assignment, we don’t need to define them
- Null element is **None**

2 Basics: Types operations (int + float)

```
a = 4
b = -4
# Multiply 'a' with 'b' and print
print(a * b)



# You can get the rest of the division with %
23 % 2



c = 1.5
d = 2.5
# Multiply 'c' with 'd' and print
print(c * d)

# You can combine 'float' and 'int' to get a new float
print(a * c)

```

- Here we detailed base operations
- But you can combine variables to build a more complex expression

https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

2 Basics: List

```python
# We can create an empty list, two methods
a = []
a = list()

# or not
b = [1, 2, 4, 6, 8]
c = ["hello", "how", "are", "you?"]

# use different types in a list
d = [1, 1.5, "hello"]

# The lists are indexed to get items
B[0] # first element
B[-1] # last element
B[::-1] # reverse order
B[1:3] # subset starting from 1 (included) to 3 (not-included)

# some methods are available to manipulate lists
a.append(10)  # append 10 to the list
d.pop()  # pop the last item of the list

# or to operate on the list without modifying the list
len(a)  # returns the length of the list

# concatenate strings
" ".join(c)
```

- 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

2 Basics: Strings

```
# An empty string
a = ""

# Or not
b = "Koby"

# Strings support indexing (careful indexing starts at 0)
b[0]  # returns ?
len(b)  # returns ?

# We can format a string, two syntax:
c = "Hello my name is %s" % b  # Old way
d = "Hello my name is {0}".format(b)  # New way

# Of split a string
columns = "Age;Name;FirstName".split(";")

# Make upper
b_up = b.upper()
```

- A str has a behaviour similar to a list
  - Indexed (can get item and has a length)
  - Can be iterated
  - Can be sorted

https://docs.python.org/3/library/stdtypes.html#textseq

2 Basics: More Strings

```
# strings are created with " or '
"This is a string."
'This is also a string.'

# joining strings
mylist = ['spam', 'ham', 'eggs']
print(', '.join(mylist))

# replace
print('I like to eat popcorn'.replace('popcorn', 'chocolate'))

# contains
animals = 'cats and dogs'
if 'cats' in animals:
    print('we have cats')

# can put special characters in string
print('if we want to have an apostrophe we just need to do \' <- this')
print('this is a line\nand this is another one\t that\'s a tab')
```

https://docs.python.org/3/library/stdtypes.html#textseq

2 Basics: Dictionaries

```
# An empty dict
my_dict = {}
my_dict = dict()

# Or with values
my_dictionary = {"barack": 54, "françois": 61, "angela": 61}

# returns an iterator
my_dictionary.values()
my_dictionary.keys()

# To combines keys and values
my_dictionary.items()

# To get an element
my_dictionary.get(“angela") # will throw exception if key doesn’t exist
My_dictionary[“angela”] # will return None if key doesn’t exist

# To update the dict
my_dictionary.update({"barack": 28})
my_dictionary[“barack”] = 28

# To get the length
len(my_dictionary)
```

- You can do almost everything with Python base structures (list + dict)
- A dict **is not ordered**

https://docs.python.org/3/library/stdtypes.html#mapping-types-dict

2 Basics: For / While Loops

```
# 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 "%s is %s years old" % (key, item)

# 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’
- In python you must **indent** your code. The indentation lust be consistent (e.g. 4 spaces or tabs)

2 Basics: If statements

```
# 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 can be used for evaluation
b = ‘Great!’ if a == 1 else ‘Not Great!’
```

- 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

2 Exercise 1

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.

Answer: 233168

2 Basics: More Lists

```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 as:
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

2 Basics: Functions

```python
# You first function
def greetings(name):
    sentence = "Hello %s" % name
    print(sentence)

greetings("Koby")
greetings("Jacques")
# /!\ Variable greetings not accessible from this part of the code


# Functions can return a value
def add(a, b):
    """ This method returns a sum between two variables a and b
    :param a: first param to add
    :param b: second param to add
    :return: the sum between a and b
    """
    return a + b

c = add(3, 5)
print(c)

# We can name our function variables
def identity(name, age):
    print(name, age)

identity(name="Roméo", age=31)
identity(age=34, name="Juliette")

```

- 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 (e.g. with docstrings)

2 Exercise 2

Write a function:

1. It takes a first name as variable
2. It will print the length of the name
3. It will print the first letter of the name only if name's length is even (2, 4, 6, …)
4. It will return the name in reverse order

2 Exercise 3

Write a function

1. It receives 3 arguments: func(digit1, digit2, operation)
2. digit1 and digit2 are strings representing digits
3. Operation is a string:
    a. add
    b. substract
    c. multiply
    d. divide
4. The function will return the answer for the operation

2 Basics: Main block

```
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("Koby")
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

2 Basics: Open and read files

```
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() # don’t forget to close it
13
14  print(rows)
```

- There are two methods to **open** files
- In data analyses you often use files, so this snippet on code is very important
- open() second parameter is the open mode: here we read the file so “r”

2 Basics: Open and write files

```
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

2 Basics: Imports

```
# You can import a simple package
import datetime

# Import with an alias
import pandas as pd

# Import a specific method or module in a package
from sys import path
from os.path import splitext

# Or all but it's not advised because everything will be in you code
from sys import *

# So for usage
today = datetime.datetime.now()
dataframe = pd.DataFrame()
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

2 Basics: 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

2 Basics: Other basic types

```
# tuples
point1 = (1, 2)
point2 = (2, 5)
point1 + point2 # behaves as a list

point1[0] = 3


# sets
items = {1, 2, 'three'}

len(items)

set([1, 2, 3, 3])

```

2 Example

```
# My CSV:
# Name;Age;Gender
# Max;23;Male
# Lou;29;Female
# Paul;67;Male
# Marion;12;Female

# Open the file with ‘with’ syntax
with open('people.csv', 'r') as f:
    data = f.read()
    rows = data.split('\n')

full_data = []
count_row = 0
count_columns = 0
for row in rows:
    count_row += 1
    full_data.append(row.split(';'))

count_columns = len(full_data[0])
```

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

2 Exercise 4

Make a program that creates the file *today.txt* and writes inside today's date

2 Basics: Exceptions

ZeroDivisionError

TypeError

SyntaxError

IndexError

- 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**

3 Basics: Exceptions

```
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: %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
13
```

- 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**
- **Pass** is necessary if nothing appears in the close (applicable in loops and functions too)

2 Basics: PEP8

```python
# 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 == "Koby":
        print("Oh yeah! We have the same name.")

hello("Koby")
hello(name="Koby")  # But here no space around ‘=‘


```

- PEP8 is a convention to write clean code and readable by anyone

2 Basics: 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** 

2 Exercise 5

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

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

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

Thanks a lot!

kkarp@equancy.com

ENSAE-ENSAI
Formation continue
(Cepe)