FORMATION AU LANGAGE PYTHON

INTRODUCTION ET BASES

Jeremy Laforet jeremy.laforet@utc.fr

Équipe NSE - D2.16 - 4372

PLAN:

- Présentation et installations
- Bases du langage
  - Structures de données
  - Structures de contrôle
  - Pythonismes
  - Entrées - Sorties
  - Fonctions
  - Classes et objets
  - Interactions graphiques

MON PROFIL

- Master Automatique - Université Montpellier 2 (2006)
- Thèse en Robotique sur la modélisation des muscles lisses - LIRMM, Montpellier (2009)
- PostDoc: Modélisation EMG utérin et strié - BMBI (2010)
- Ingénieur de Recherche CNRS (2013 -)
  - Modélisation multiphysique des muscles
  - Implémentation logicielle des modèles
  - Analyses de sensibilité

Développements en Python depuis ~2007

POURQUOI PYTHON?

- Langage de haut niveau
- Ipython (jupyter) Notebook
- Transition directe vers un code stable
- Script exécutables / multiplateforme
- De nombreux formats de ficihers lisibles
- Possible interface graphique
- Ratio temps de programmeur/résultat !
- Écosysteme scientifique riche
- Outils de débug/profiling
- Utilisable du micro-contrôleur au supercalculateur!

Prototype - Code Glue - Développement complet

EXEMPLE DE DÉVELOPPEMENT

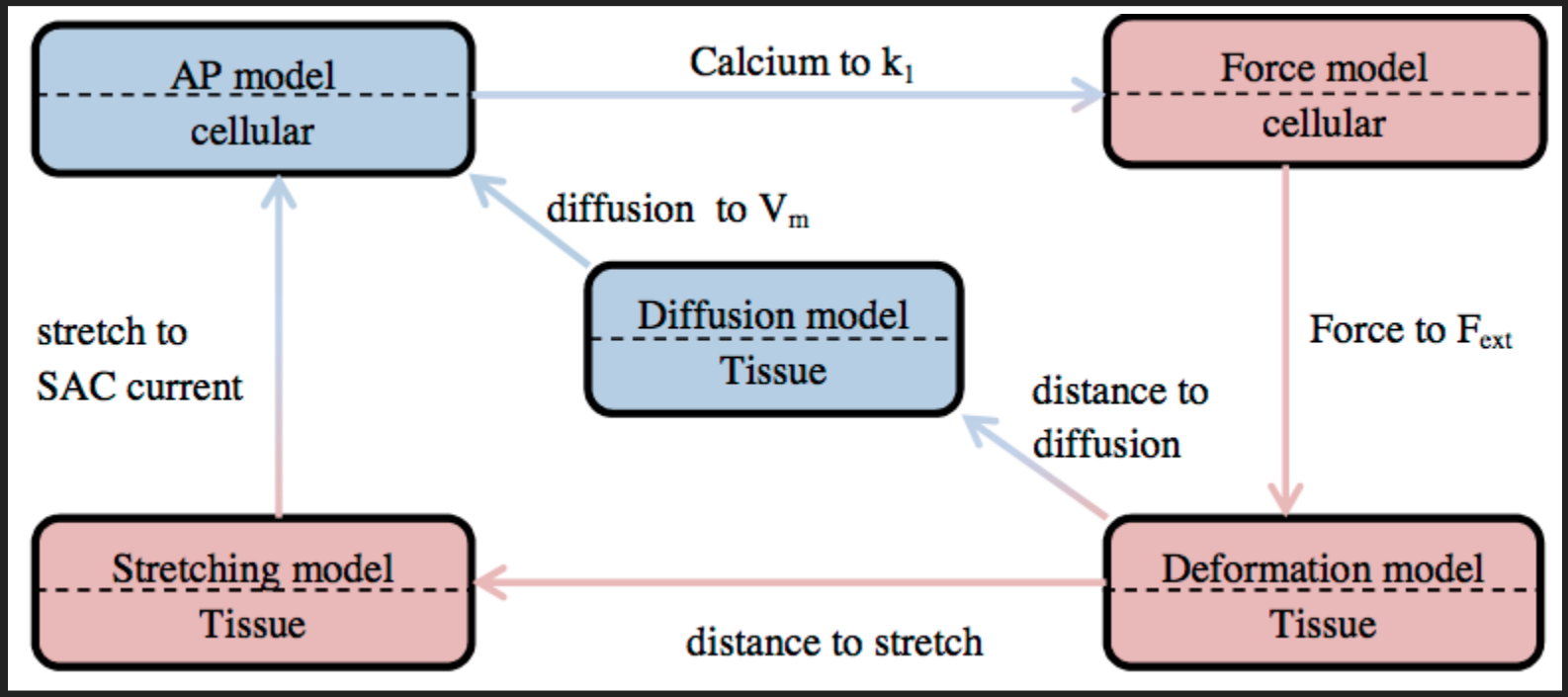
ELECTROMYOGRAMME UTÉRIN

Travaux de PostDoctorat de Maxime Yochum

(Mars 2015 - Présent)

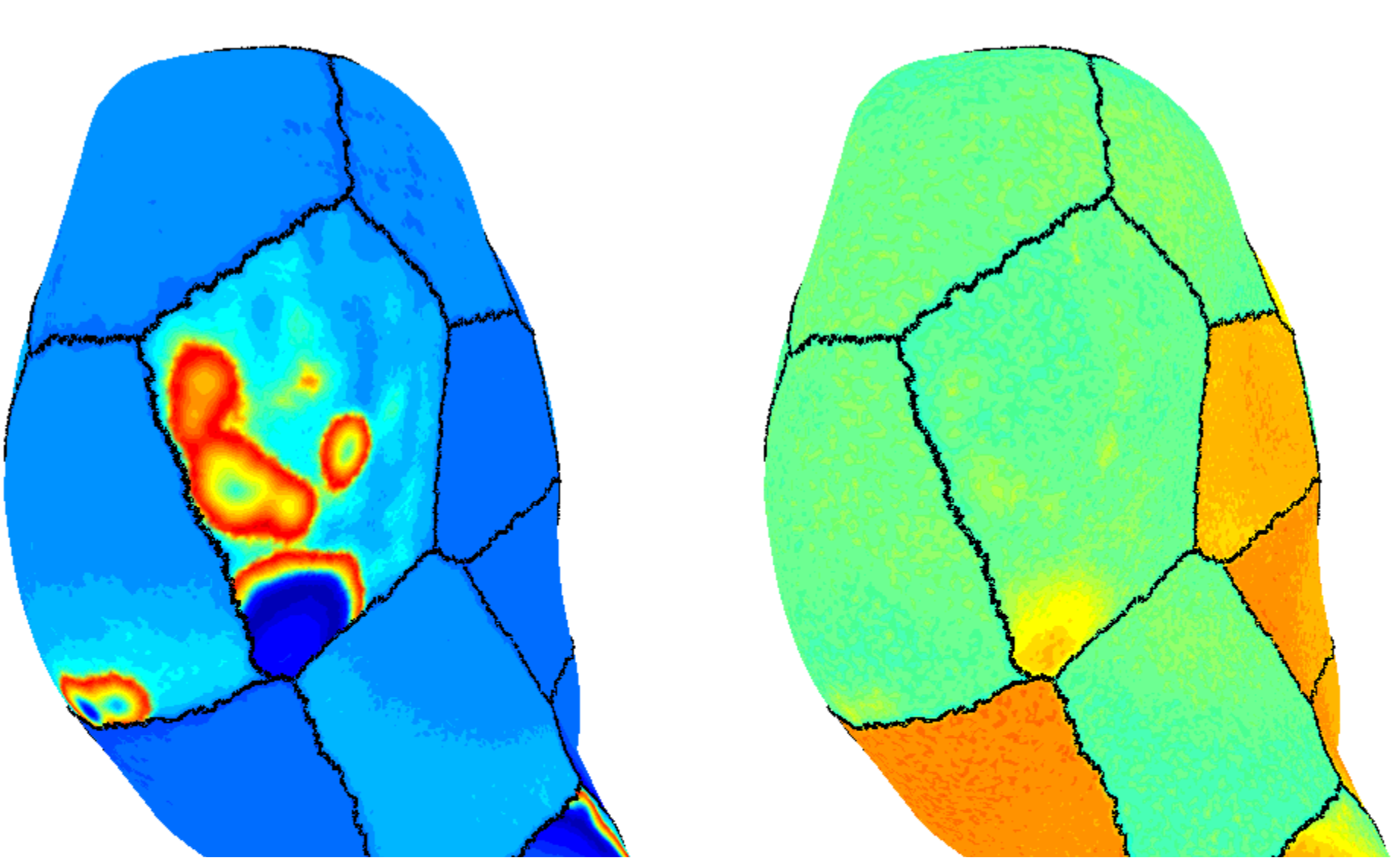
Débutant Python en février 2015

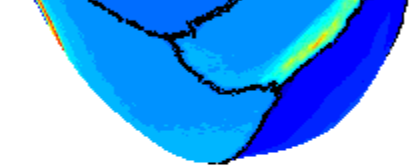
DESCRIPTION DU MODÈLE:

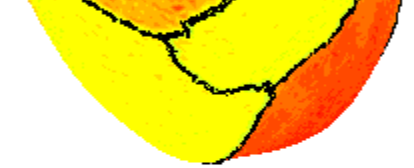


Application sur un mesh surfacique 3D (~100k noeuds)

SIMULATION COMPLÈTE:



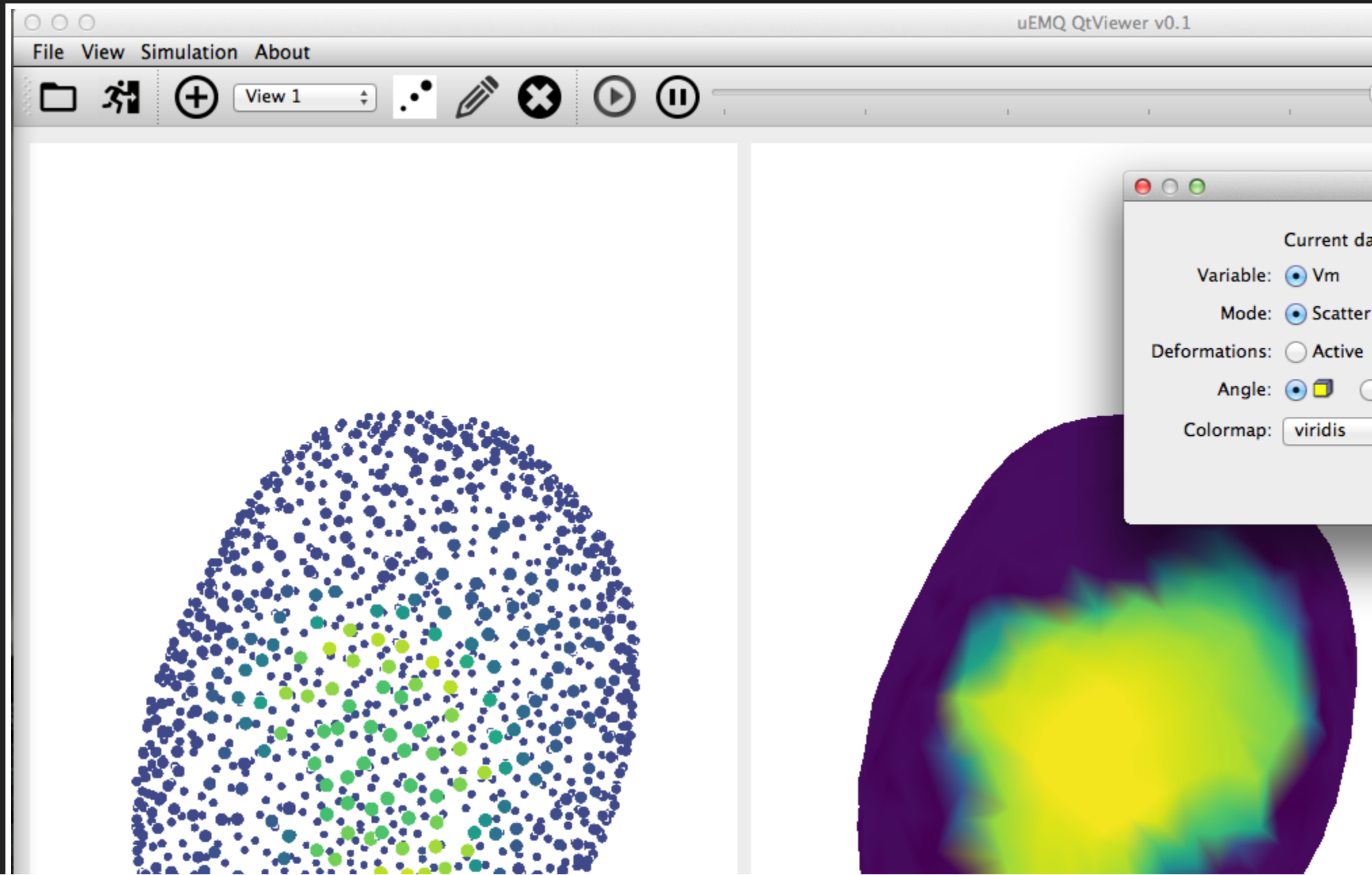


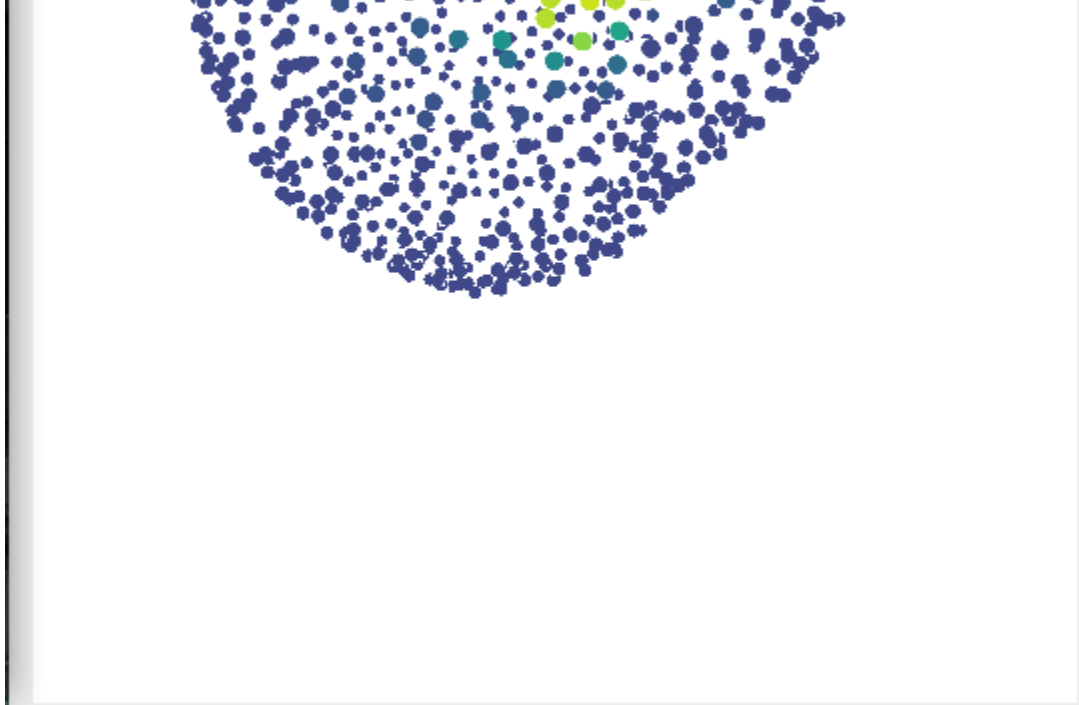


electrical activity, Time t = 14.70 s

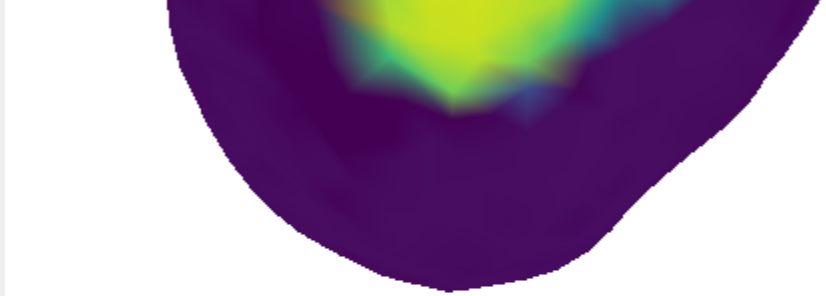
mechanical activity, Time t = 14.70 s

INTERFACE DE VISUALISATION





Electrical



Mechanical

INSTALLATION ET PRÉSENTATION

ANACONDA OU MINI-CONDA

- Distribution de Python
- ~300 Paquets
- Orienté sciences
- Mini-conda: installeur seul

PYCHARM(-EDU)

- IDE pour Python
- Coloration, complétion, vérification de code
- Execution, debuggage, profilage
- PyCharm-edu: version *simplifée* pour l'enseignement

ÉXECUTION

Interactive:

- python shell
- Ipython
- Jupyter Notebook

Batch:

- script
- executables

STRUCTURE

- script
- fonction
- module
- package

IMPORTS

Utilisation de modules (ou de fonctions des modules)

```
import sys
```

```
import numpy as np
```

```
from os import rename
```

- Un module entier
- Un module entier avec un nom personalisé
- une fonction d'un module

PYTHON 2.7.X VS PYTHON 3.Y

- Meilleur support unicode
- Changement de la syntaxe de certains *import*
- La division entière devient //
- *print* devient une fonction
- Tous les paquets ne sont pas encore en 3.y

le module __*future*__ permet d'écrire du code python 2.7 compatible avec python 3

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

BASES DU LANGAGE

PEP8

- Use 4-space indentation, and no tabs.
- Wrap lines so that they don’t exceed 79 characters.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- When possible, put comments on a line of their own.
- Use docstrings.

PEP8

- Use spaces around operators and after commas, but not directly inside bracketing constructs:
  `a = f(1, 2) + g(3, 4).`
- Name your classes and functions consistently; the convention is to use `CamelCase` for classes and `lower_case_with_underscores` for functions
- Don’t use fancy encodings if your code is meant to be used in international environments. Plain ASCII works best in any case.

STRUCTURES DE DONNÉES

- variable
- liste
- tuple
- set
- dictionnaire

VARIABLE

```
a = 2
```

```
2
```

```
b = 'test'
```

```
'test'
```

```
c = 3.14
```

```
3.14
```

- typage dynamique
- *type(var)* pour connaitre le type actuel de *var*

LIST

```
l = ['a', 2, 1, 2]
```

```
['a', 2, 1, 2]
```

```
l[0]
```

```
'a'
```

- types quelconques (mélanges possibles)
- conversation de l'ordre
- indexable
- mutable
- les chaines de charactères sont des listes

TUPLE

```
t = ('a', 2, 1, 2)
```

```
('a', 2, 1, 2)
```

```
t[0]
```

```
'a'
```

- types quelconques (mélanges possibles)
- conversation de l'ordre
- indexable
- immutable

SET

```
s = {'a', 2, 1, 2}
```

```
{1, 2, 'a'}
```

- types quelconques (mélanges possibles)
- elements uniques et triés
- immutable

DICTIONNAIRES

```
d = {'a':1, 2:3.14}
```

```
{2:3.14, 'a':1}
```

```
d['a']
```

```
1
```

- types quelconques (mélanges possibles)
- Ensemble de paires clé:valeur
- trié par clés
- mutable

STRUCTURES DE CONTRÔLE

- if / else
- while
- for
- iterate

TESTS

if… elif… else…

```
if a > 1:
    print('ok')
elif a < -1:
    print('ko')
else:
    pass
```

BOUCLES

while

```
a = 0
while a <= 4:
    print(a)
    a += 2
```

```
0
2
4
```

BOUCLES

for

```
for i in [7,3,8]:
    print(i)
```

```
7
3
8
```

```
for j in range(3):
    print(j)
```

```
0
1
2
```

BOUCLES

enumerate(...)

```
for idx, lettre in enumerate('chaine'):
    print(idx, lettre)
```

```
0 c
1 h
2 a
3 i
4 n
5 e
```

PYTHONISMES

is

```
if a is b:
        print("a et b sont le même objet.")
```

Range:

```
range(0,10,3)
```

```
[0, 3, 6, 9]
```

COMPREHENSIONS

List comprehensions:

```
squares = [nb**2 for nb in range(0,10)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Dictionnary comprehensions:

```
odd_squares = {nb:nb**2 for nb in range(0,10) if nb**2%2}
```

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

EXCEPTIONS

It's easier to ask forgiveness than it is to get permission.

- raise
- try / except

RAISE

```
denominateur = 0
if denominateur == 0:
    raise ValueError, "Le dénominateur ne peut être nul."
```

```
---------------------------------------------------------------
ValueError     Traceback (most recent call last)
 in ()
      1 if denominateur == 0:
----> 2    raise ValueError, "Le dénominateur ne peut être nul."
      3

ValueError: Le dénominateur ne peut être nul.
```

TRY... EXCEPT...

```
numerateur = 3.14
denominateur = 0.0
try:
    resultat = numerateur / denominateur
except ZeroDivisionError:
    print("Le dénominateur est nul.")
else:
    print("La division vaut", resultat)
finally:
    print("Calcul terminé.")
```

```
Le dénominateur est nul.
Calcul terminé.
```

ENTRÉES / SORTIES

- input
- print
- format
- fichiers

PRINT

- Affiche la chaine de charactère passée en argument

```
print("Bonjour!")
```

```
Bonjour!
```

Py2: *print* n'y est pas une fonction

```
print "Bonjour!"
```

```
Bonjour!
```

Utiliser *from __future__ import print_function* pour avoir la fonction en python 2.x

INPUT

- Permet de poser une question
- Renvoie une chaine de charactères
- Conversion nécessaire ensuite pour les autres types

```
nom=input("Quel est votre nom?")
```

```
Quel est votre nom?Terry
```

```
print(nom)
```

```
Terry
```

Py2: Équivalent à la fonction *raw_input*

FORMAT

Mise en forme d'une chaine de charactères

```
print('Bonjour {} !'.format(nom))
```

```
Bonjour Terry !
```

```
print('Bonjour {} {} !'.format('Sir',nom))
```

```
Bonjour Sir Terry !
```

```
print('Bonjour {ti} {name} ou {name} !'.format(ti="Sir",name=nom))
```

```
Bonjour Sir Terry ou Terry!
```

FICHIERS TEXTE

Lecture

```
with open('monfichier.txt','r') as myfile:
    for line in myfile:
        print(line)
```

```
with open('monfichier.txt','r') as myfile:
    contenu = myfile.readlines()
```

Écriture

```
with open('monfichier.txt','w') as myfile:
    myfile.write('mon texte à sauvegarder.\n')
```

FONCTIONS

- Définition et Arguments
- Valeur de retour
- Documentation
- Décorateurs
- Lambdas

ARGUMENTS

```
def calcul_milieu(pt_a, pt_b, precision='int', affiche_pts=False):
    …
```

```
calcul_milieu((3.5, 2.9), (-1.04, -8.32))
```

```
(4, 7)
```

```
calcul_milieu((3.5, 2.9), (-1.04, -8.32), affiche_pts=True)
```

```
A: (3.5, 2.9) B: (-1.04, -8.32) M: (4, 7)
```

```
calcul_milieu((3.5, 2.9), (-1.04, -8.32), 'float')
```

```
(4.02, 7.060000000000005)
```

VALEURS DE RETOUR

Retourner une valeur:

```
def dis_non():
    return "non"
dis_non()
```

```
'non'
```

Retourner un ensemble de valeurs:

```
def calcul_milieu(point_a, point_b):
    return (point_a[0]-point_b[0])/2. , (point_a[1]-point_b[1])/2.
calcul_milieu((0.,1.),(1.,0.))
```

```
(-0.5, 0.5)
```

DOCUMENTATION

```
def calcul_milieu(point_a, point_b):
    """
    Compute the middle of the (a,b) segment.
    :param point_a: tuple or list (xa, ya)
    :param point_b: tuple or list (xb, yb)
    :return: tuple (xm, ym)
    """
    return (point_a[0]-point_b[0])/2. , (point_a[1]-point_b[1])/2.

calcul_milieu?
```

```
Signature: calcul_milieu(point_a, point_b)
Docstring:
Compute the middle of the (a,b) segment.
:param point_a: tuple or list (xa, ya)
:param point_b: tuple or list (xb, yb)
:return: tuple (xm, ym)
File:      ~/Documents/Cours/Formation_Python/
Type:      function
```

DÉCORATEURS

```
@decorator
function(arg)
```

```
decorator(function(arg))
```

- Permet de faire une composition de fonction simplement
- ou de modifier l'execution d'une fonction

LAMBDA

fonction anonyme

```
square = lambda x:x**2
```

Permet de définir localement une fonction simple (dans un appel de fonction par exemple)

OBJETS ET CLASSES

En python, tout est objet

```
l = ['a', 2, 1, 2]
l.sort()
print(l)
```

```
[1, 2, 2, 'a']
```

```
l.__class__
```

```
list
```

VOCABULAIRE

Classe: Objet: Attribut: variable propre à un objet
Méthode: fonction propre à une classe

En python il n'y a pas de différence privé/publique formelle. L'usage de doubles underscores dans le nom indique un élément privé: *l.__class__*

SYNTAXE

```
class point:
    def __init__(self, coord_x,coord_y):
        self.x = coord_x
        self.y = coord_y

    def __repr__(self):
        return "X:{} Y:{}".format(self.x, self.y)
```

```
pt_a=point(3.4,7.8)
print(pt_a)
```

```
X:3.4 Y:7.8
```

INTERACTIONS GRAPHIQUES

JUPYTER NOTEBOOK

- Narrative computations
- Interactions

NARRATIVISME

Créer un document regroupant narration et calculs en utilisant les capacités du notebook: fusionner un cours et des exercices en un seul document, proposer une documentation exécutable…

DEMO

INTERACTIONS

Utilisation de widgets pour interagir avec le code

- Entrées utilisateur multiples
- Exploration interactive de données

PYQT: ÉLÉMENTS GRAPHIQUES

Bindings Qt4 pour python

- MessagesBox et Dialogs
- Vraies interfaces (ex surf_view)

MESSAGEBOX

Affiche un message ou pose une question simple

- QMessageBox.information()
- QMessageBox.question()
- QMessageBox.warning()
- QMessageBox.critical()

```
reply = QtGui.QMessageBox.question(self, 'Message',
"Are you sure to quit?", QtGui.QMessageBox.Yes |
QtGui.QMessageBox.No, QtGui.QMessageBox.No)
```

QDIALOGS

Permet de récupérer une information

- QInputDialog
  - .getInt()
  - .getDouble()
  - .getText()
  - .getItem()
- QFileDialog
  - .getExistingDirectory()
  - .getOpenFileName()
  - .getSaveFileName()

EXEMPLE MINIMAL D'INTERFACE

```
import sys
from PyQt4 import QtGui

def main():

    app = QtGui.QApplication(sys.argv)
    w = QtGui.QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Simple')
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

Python pour les sciences

BONUS: REGEXP

Regular expression: Expression rationnelle/régulière, description d'un motif correspondant à un ensemble de chaine de caractères possibles

- Rechercher / Replacer dans le code
- Traitement de données textuelles

BONUS: REST

Restructured text : langage de mise en forme de texte rapide, cousin du Markdown (md)

- Écriture des docstrings, voir de documentations complètes (voir demain après-midi)