IMN-530

Reconstruction et analyse d’images médicales

-

Survol python – En préparation de vos TP

1. Linux et le terminal

3 sortes de systèmes d'exploitation

- Windows
- Mac
- Linux
  - Plus « geek ». Moins intuitif pour l'utilisateur.
  - Plus facile de contrôler ce qui se passe.

  - Plus adapté pour coder.
    - *Particulièrement en python!*
    - *Particulièrement en imagerie!*

      La plupart des programmes sont disponibles sur Linux. Parfois sur Mac. **Rarement sur Windows.**

3 sortes de systèmes d'exploitation

Prenez le temps de trouver une façon de travailler sur Linux!

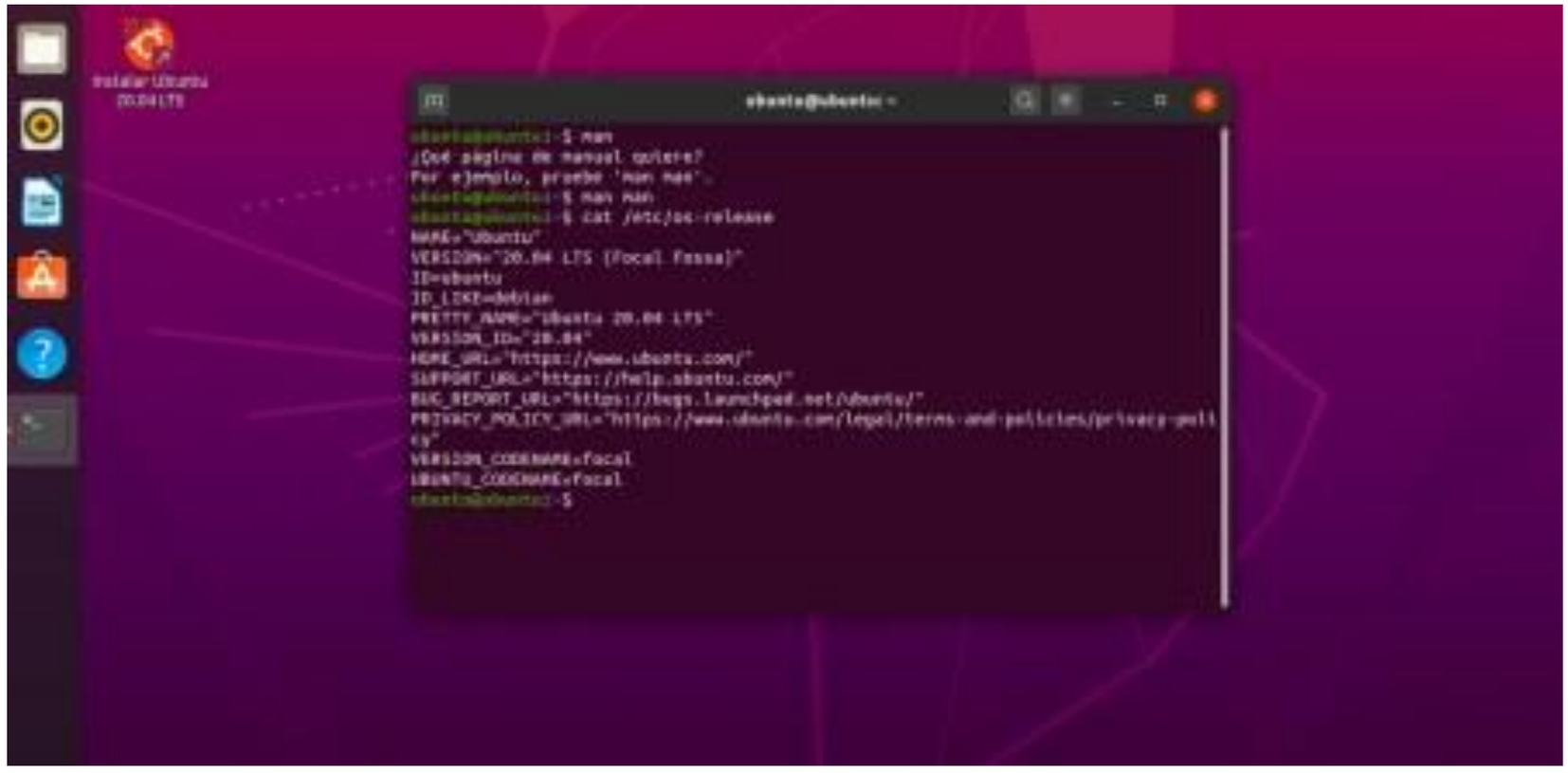
- Dual boot
- Machine virtuelle
- Ordis du labo

Vos devoirs vont être beaucoup plus faciles

Intro to Linux, par Pr. François Rheault:
https://scil-documentation.readthedocs.io/en/latest/intro_to/explore_os.html

Le terminal

- Démo en classe:
  - Ouvrir un terminal
  - Naviguer entre les dossiers
  - Lancer des commandes
  - Le langage bash



TP0 – Installation du nécessaire

- Installation de python et des programmes nécessaires pour le cours.
- **Ne sous-estimez pas!**
  - Un peu long et pénible, désolée :(
  - Mais une fois que c'est fait, on est « setté » pour toute la session!
- Souvent du cas par cas pour déboguer.
  - Je peux vous aider, mais ce n'est pas une promesse de réussite!
  - Vous pouvez aussi demander à Antoine Théberge, aide et correcteur pour le cours.
  - Posez vos questions sur notre groupe Teams, ça profitera peut-être à d'autres!

2. Python

Préalables souhaités:

IFT211 ou similaire: Programmation scientifique en Python

Sinon: Vous pouvez suivre le cours, mais c'est à vous d'apprendre les bases de python par vous-même. Aujourd'hui n'est qu'un survol.

A. Comment s'assurer que python existe sur votre ordinateur ?

Sur Linux:

- Python existe par défaut. Linux fonctionne sur une base de python
- Mais ce n’est peut-être pas la version de python que vous voulez.
- Même si c’est déjà la bonne version, vous ne voulez pas risquer de briser votre python « principal » en installant des libraires.

- Démo en classe: Un mot sur les environnements! Fortement recommandé dans le TP0!

B. Comment l'utiliser ?

Comment on « lance » python?

1. Interactif: directement dans le terminal

- Démo en classe:
  - ...on vs ipython

```
In [1] ...numpy
In [2] n...m.rand()??
```

Impossible pour ce cours!

- Lance une console python directe...tre terminal.
- Peut vous donner des explications sur des methodes.

- Mais si on quitte, tout est effacé

2. Jupyter notebook

- Aid familiariser avec le langage.

- Pour le lance cliquer dessus, ou, dans le terminal. Ex:

```
jupyter notebook ~/Documents/Enseigment/                IntroPython/0_basics.ipynb
```

- Mais difficile de créer des scripts co
  - Difficile de diviser en plusieurs fichiers = de un monstre.
  - Difficile d'envoyer des arguments
    - *Interdiction d'avoir des noms de fichiers « hard-coded »*

Non recommandé pour ce cours!

3. En écrivant un script

- Vous pouvez utiliser votre éditeur de texte préféré
  - Mon choix personnel: pycharm
  - Il en existe plusieurs autres.

  - Démo en classe

- Pour lancer le script:
  - Console de votre éditeur de texte
  - Directement dans le terminal
    - *Ce que je ferai pour tester vos devoirs*
    - *Démo en classe*

```
>>> cd my_path
>>> python my_script.py arg1 arg2
```

C. Comment coder en python

C. Comment coder en python

1. Éléments de base

Syntaxe

- Python: très lisible 
- Utilise l'indentation (au lieu de {…})

Qu'est-ce que ce script affiche?

```
def teacher_speach():
    print("HELLO CLASS!")
    for i in range(6):
        # Let's salute each student.
        print("   Hello student {}".format(i))

teacher_speach()
```

Syntaxe

- Commentaire: #

```
    # Let's salute each student.
    print("   Hello student {}".format(i))
```

- Mots clés: souvent anglais au lieu de symboles.

```
for i in range(6):
```

*p.s. équivalent C:

```
for (i = 0; i < 6; i++)
```

Opérateurs (voir 0_basics.ipynb)

- Opérateurs mathématiques habituels (+, -, *, /, //, %)
- Autre opérateurs similaires à d'autres langages. À étudier si vous ne les connaissez pas

```
print("For loop:")
for i in [0, 1]:
    print(i)

print("While loop:")
i = 0
while i < 3:
    print(i)
    i += 1

print("If / else:")
i = 3
if i == 1:
    print('Yes, 1!')
elif i == 2:
    print("Oups, 2.")
else:
    print("Ah, no: {}".format(i))
```

Qu'est-ce que ce script affiche?

Types de bases et assignations de valeurs

```
# Integers
a = 1
a = b = c = 1  # assignations multiples
print('a est un {} qui vaut {}.'
      .format(type(a), a))
print('a+2 = {},\na*2 = {},\na/2 = {},\na^2 = {}\n'
      .format(a + 2, a * 2, a / 2, a**2))

# Floats
a = 1.0
print('a est un {} qui vaut {}.'
      .format(type(a), a))
print('a+2 = {},\na*2 = {},\na/2 = {},\na^2 = {}\n'
      .format(a + 2, a * 2, a / 2, a**2))

# Strings
a = 'Hello'
print('a est un {} qui vaut {}.\n'
      .format(type(a), a))

# no such this as a char. Only strings.
a = 'a'
print('a est un {} qui vaut {}.\n'
      .format(type(a), a))


# Bools
a = True
print('a est un {} qui vaut {}.\n'
      .format(type(a), a))
```

Qu’est-ce que ce script affiche?

Types de bases et assignations de valeurs

- Pas besoin de déclarer le type des valeurs à l'avance. Python les devine au moment où on les initialise.

- Dans les prochaines slides:
  - Types que vous connaissez déjà si vous avez déjà codé, presque peu importe le langage:
    - *Float, int, str, double,*
  - Nouveaux types:
    - *List, numpy array*
    - *dict, itérateurs, sets, tuples, etc.*
    - *Type «vide», ou «rien»:* `None`. *Ex:* `a = None`

Listes, vecteurs, matrices (voir 0_basics.ipynb)

- Le type de base de python pour stocker plusieurs valeurs ensemble est soit une liste ou un tuple

```
# Une liste
a = [0, 1, 2, 3]
print("a est un {} qui vaut {}.\n"
      .format(type(a), a))


# Une liste peut contenir des types variables
# (ex, int et floats)
a = [0, 1.4, 2, 3]
print("a est un {} qui vaut {}.\n"
      .format(type(a), a))


# Une liste peut contenir d'autres listes'
a = [0, [1, 1.1, 1.2], 2, 3]
print("a est un {} qui vaut {}.\n"
      .format(type(a), a))
```

Qu'est-ce que ce script affiche?

Listes, vecteurs, matrices (voir 0_basics.ipynb)

- Tuples

```
# Tuple
a = (0, 1, 2)
print("a est un {} qui vaut {}"
      .format(type(a), a))
```

Qu'est-ce que ce script affiche?

- Type de retour d'une fonction pour plusieurs éléments.

```
def function():
    a = 1
      b = 2
    return a, b

result = function()
result_a, result_b = function()
```

Que valent result, result_a, result_b?

Listes, vecteurs, matrices

- Opérations sur les listes

Qu'est-ce que ce script affiche?

```
# Opérations sur les listes
a = [0, 1, 2, 3]
print("Initial a {}".format(a))
a.append(4)
print("After appending a number: {}".format(a))
a.extend([5, 6, 7])
print("After extending the list: {}".format(a))
a.append([8, 9, 10])  # oups.
print("After appending a list (oups): {}\n".format(a))


# Résultats "in place"
# i.e. il ne faut pas réassigner le résultat
a = a.append(5)  # OUPS
print("Returned result (OUPS): {}".format(a))
```

Listes, vecteurs, matrices

- `len(a_list)` = le nombre d'éléments.
- Attention! Le dernier élément d'une liste de longueur 100 est l'élément 99.

```
# Accéder les informations d'une liste
# ATTENTION. Les indices commencent à 0 en python!

a = [0, 1, 2, 3]
print(a[0])
print(a[-1])
print(a[-2])
print(a[:])
```

Qu'est-ce que ce script affiche?

Listes, vecteurs, matrices

- Les listes sont toujours 1D (des « vecteurs »).
- Pour utiliser des matrices, on utilise la librairie numpy.

```
import numpy as np

a = [1, 2, 3, 4]
b = np.asarray(a)
print("b est un {} qui vaut {}"
      .format(type(b), b))


# On peut créer une matrice à partir
# d'une liste de listes
a = [[0, 0, 0],
     [1, 1, 1],
     [2, 2, 2]]
b = np.asarray(a)
print("Une liste de listes: {}".format(a))
print("Une matrice: {}".format(b))
```

Qu'est-ce que ce script affiche?

Question: Comment accède-t-on à la première valeur de a? De b?

Numpy (voir 2_intro_to_numpy.ipynb)

- Numpy: plusieurs outils mathématiques
  - Sin, cos, tan, etc.
  - Opérations matricielles
  - Modifications de matrice (reshape, transpose, etc).
  - Nombres aléatoires.
  - Etc.

Il y a des forums pour tout! Google est votre ami.

C. Comment coder en python

3. Utilisation de librairies

Librairies

- Heureusement, beaucoup de choses sont déjà codées pour nous.
  - Afficher une image ou un graphique (matplotlib)
    - *Nécessaire pour le TP1!*
  - Utiliser des matrices (numpy)
  - Utiliser des images d'IRM de diffusion (dipy)
  - Faire des calculs mathématiques plus complexe (scipy)
  - Faire de l'intelligence articificielle (torch)
  - Etc.

Librairies

- On les utilise ainsi:

```
import some_library


some_library.do_something()
```

```
from some_library import do_something


do_something()
```

D. Écrire un script.

Bonnes habitudes et recommandations

Ou comment aider le prof à s'y retrouver dans votre TP

Mes consignes:

- Interdiction que tout soit d'un seul bloc!
- Séparer le main() dans une méthode.
- Le main() devrait contenir très peu de lignes. Ex:

```
def main():
    args = parse_args()
    img = load_image(args.my_path)
    img_clean = denoise_image(img)
    save_image(img_clean, args.my_saving_path)
```

- Presque tout le reste est « encapsulé » dans des sous-fonctions dans des fichiers séparés.
- Utiliser un Argument Parser.

Démo

- Démo en classe:

www.github.com/EmmaRenauld/Teaching_scriptExample

E. Eléments avancés

Si vous ne comprenez pas ce qui suit, vous vous en sortirez quand même, rassurez-vous! ☺

ÉVITER LES BOUCLES IMBRIQUÉES

- Sera particulièrement important pour travailler avec des images 3D.

- Prenez le temps de lire 3_challenges_numpy.ipynb

- Ex:

```
Total = matrice 1 + matrice 2
```

- ET NON

```
matrice_totale = np.zeros(size_x, size_y)
for i in range(size_x):
     for j in range(size_y):
          matrice_totale[i, j] = matrice1[i, j] +
matrice2[i, j]
```

Passage par référence

- Dans python:
  - Les objets immutables sont passés par copie.
  - Les objets mutables sont passés par référence.

Immutables: nombres, strings, tuples, ets.

```
def modif_x_float(x):
    x += 2
    return x


a = 1
b = modif(a)
print(a, b)  ---> 1, 3
```

Attention au passage par référence

Mutables: classes, dict, lists

```
class myClass():
    def __init__(self):
        self.x = 1


def modif_x_class(c):
    c.x += 2
    return c

a = myClass()
b = modif(a)
print(a.x, b.x)  ---> 3, 3
```

List comprehension (voir 1_advanced_basics.ipynb)

- Une des forces de python s'appelle les « list comprehension ». Généralement plus rapides qu'une boucle for.

```python
from datetime import datetime

# Accéder les informations d'une liste
# ATTENTION. Les indices commencent à 0 en python!

a = list(range(100))  # Une liste de 0 à 99.

# List comprehension. On le fait 10000 fois.
start = datetime.now()
for i in range(1000):
    b = [v ** 2 for v in a]
end = datetime.now()
print("Temps écoulé 1: {}".format((end - start
    ).total_seconds()))

# Boucle for. On le fait 10000 fois.
start = datetime.now()
for i in range(1000):
    b = []
    for v in a:
        b.append(v ** 2)
end = datetime.now()
print("Temps écoulé 2: {}".format((end - start
    ).total_seconds()))
```

```
Temps écoulé 1: 0.023562
Temps écoulé 2: 0.03209
```

Fonctions

```python
from typing import Union

def division(nb1: float,
             nb2: float,
             division_entiere: bool = False
             ) -> Union[int, float]:
    """
    Cette méthode sert à diviser deux nombres.

    Parameters
    ----------
    nb1: float
        Numérateur.
    nb2: float
        Dénominateur.
    division_entiere: bool
        Si vrai, le résultat de la division
        entière est retournée. Défault: False.

    Returns
    -------
    resultat: Union[int, float]
        Résultat.
    """
    if division_entiere:
        return int(nb1 // nb2)
    else:
        return nb1 / nb2

print(division(1.0, 2.0))
print(division(1.0, 2.0, division_entiere=True))

print(division(15.0, 2.0))
print(division(15, 2))
```

Qu’est-ce que ce script affiche?

Mettez des commentaires!

Fonctionne aussi (sans « type hints » ni « docstrings »). Mais moins beau à lire .

Classes

```python
from typing import List

# Je n'ai pas mis les doctstrings, type hints par
# soucis de concision. À ajouter.
class Person:
    def __init__(self, age, name, family_name):
        self.age = age
        self.name = name
        self.family_name = family_name

    def celebrate_birthday(self):
        self.age += 1

class Mom(Person):
    def __init__(self, age, name, family_name, kids: List[Person]):
        super().__init__(age, name, family_name)
        self.kids = kids

    def show_kids_ages(self):
        for kid in self.kids:
            print(kid.age)

Arthur = Person(age=1, name='Arthur', family_name='Boutin')
Zoe = Person(3, 'Zoe', 'Boutin')
Jane = Mom(32, 'Jane', 'Doe', kids = [Arthur, Zoe])

Zoe.celebrate_birthday()
Jane.show_kids_ages()  # Shows 1 and 4.
```