python-jupyter-tp0a

May 2, 2017

1 Jupyter, Python : présentation

** à faire sur vidéo-projecteur, en salle de TD ? **

1.1 Python

Historique : https://fr.wikipedia.org/wiki/Python_(langage)

Nous utilisons pour ces TP `python3` - attention il n'y a pas de compatibilité ascendante entre `python2` et `python3`...

1.2 Jupyter

Jupyter aup,aravant nommé *iPython* pour *i*nteractive *Python*, est à la fois un serveur WEB et un serveur de calcul qui permet une expérimentation facile de certains langages - et tout d'abord Python. En particulier on peut intercaler dans les *notebook*s (que nous traduisons par *calepin*) des cellules de calcul et des cellules de texte - plus exactement des cellules `Markdown`.

1.3 dans nos salles de TP

La distribution installée est dénommée *anaconda* que vous pouvez aussi obtenir sur https://www.continuum.io/downloads (privilégiez la version `python3` que nous avons installée en salles de TP).

Les noyaux de calcul installés dans nos salles TP sont `python2`, `python3`, `bash`, `R`. D'autres noyaux sont disponibles, dont `Javascript`, `Java9`...

1.4 pourquoi utiliser Python à l'IUT

- Facilité apparente pour un débutant... et pour un prof de maths
- Supporte de nombreux styles de programmation : impératif, objet, fonctionnel...
- Présence de très nombreuses bibliothèques : il est facile d'interfacer des outils de calculs avec Python, par exemple en maths ou en sciences de l'ingénieur
  - `iPython` et son successeur `Jupyter` pour une interface confortable dans un cadre pédagogique
  - `sympy` pour le calcul symbolique (ou calcul formel)
  - `numpy` pour le calcul numérique en particulier pour le calcul matriciel
  - `pandas` pour l'analyse de données...

- Python est aussi très largement utilisé dans des domaines très variés de l'informatique, par exemple par des administrateurs système ou pour la création de serveurs d'applications WEB (Zope, Pyramid, Django… ce dernier par exemple chez Instagram)

1.5 pourquoi ne pas utiliser Python à l'IUT

- Trop de facililités… et donc risque de prendre de très mauvaises habitudes : il est très facile de mal programmer en Python.
  ## mise en garde N'oubliez pas que lors des TD et des contrôles, vous devrez être **capable d'utiliser votre calculette pour résoudre les exercices et problèmes proposés**.

2 Premiers calepins Jupyter

Les calculs sous jupyter sont enregistrés dans des documents appelés *calepins* ou *notebooks* (suffixe `.ipynnb`).

Dans un terminal lancer la commande `jupyter notebook` et choisissez le langage (bash, R, python3, python2…)

3 Activité 1 : un notebook `bash`

Lancez un notebook `bash` puis : - exécutez une cellule de calcul avec la commande `pwd` - on exécute le cellule courante en cliquant sur ▷ ou, au clavier, par "majuscule-retour" (on crée alors une nouvelle cellule de calcul, par "commande-retour" ou "contrôle-retour", on exécute alors la cellule actuelle sans en crér de nouvelle selon les modèles de PC. - précédez d'une cellule Markdown commentaire - exécutez dans une nouvelle cellule `cd ..` puis ré-exécutez `pwd` : que constatez-vous ? - renommez votre calepin, sauvegardez-le, fermez-le, rouvrez-le…

Dans un calepin `python`, vous pouvez exécuter des commande `bash` en commençant la cellule par la "commande magique" `%%bash` :

```
In [1]: %%bash
        pwd

/Users/msc/hdDocuments/en_cours


In [2]: %%bash
        cd ..
        pwd

/Users/msc/hdDocuments


In [3]: %%bash
        cd
        pwd

/Users/msc
```

4 activité 2 : un premier calepin python3

Créez un nouveau calepin python3, donnez-lui un nom, puis partez à la découverte de Python.

4.1 Les opérations courantes entre entiers

(quand on exécute une cellule, le dernier objet calculé est affiché)

```
In [4]: 123*345
```

```
Out[4]: 42435
```

```
In [5]: from sympy import Symbol
        x=Symbol('x')
        x**2
```

```
Out[5]: x**2
```

La commande suivante permet d'afficher les résultas des calculs en LaTeX — en fait en jsmath. Nous vous recommandanons de ne pas l'utiliser dans un premier temps

```
In [6]: #init_printing()
        x**2
```

```
Out[6]: x**2
```

```
In [7]: 123**345
```

```
Out[7]: 104055118129527077397314921919093102798089635254987537019282179164748863859
```

```
In [8]: a,b=1789,100
        # Notez la différence entre dividsion entière te division flottante.
        a,b,a+b,a-b,a*b,a/b,a//b,a%b
```

```
Out[8]: (1789, 100, 1889, 1689, 178900, 17.89, 17, 89)
```

4.2 typage dynamique

On a créé ci-dessus deux variables en parallèle, on aurait aussi pu écrire :

```
In [9]: a=1789
        b=100
        a,b,a+b,a-b,a*b,a/b,a//b,a%b
```

```
Out[9]: (1789, 100, 1889, 1689, 178900, 17.89, 17, 89)
```

Cette affectation en parallèle permet de réaliser élégamment un échange entre variables :

```
In [10]: a,b=1789,100
         print(a,b)
         a,b=b,a
         print(a,b)
```

```
1789 100
100 1789
```

Les objets sont typés lors de leur création, mais on n'a pas à déclarer le type des variables. Par contre on ne peut utiliser une variable non déclarée :

```
In [11]: a=3
         b=45.67
         c="je suis un texte écrit en unicode"
         print(a,b,c)
```

```
3 45.67 je suis un texte écrit en unicode
```

```
In [12]: print(d)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-12-9909575fff82> in <module>()
----> 1 print(d)


NameError: name 'd' is not defined
```

Voici comment créer des sorties textes un peu plus élégantes :

```
In [13]: print("a est l'entier %s, b est le flottant %s et c est la chaîne '%s'"%(a
```

```
a est l'entier 3, b est le flottant 45.67 et c est la chaîne 'je suis un texte écri
```

```
In [14]: print("a est l'entier %s, b est le flottant %6.6f et c est la chaîne '%s''
```

```
a est l'entier 3, b est le flottant 45.670000 et c est la chaîne 'je suis un texte
```

4.2.1 structures courantes

Les structures les plus utilisées sont les listes, les tuples et les dictionnaires

```
In [15]: laliste=[a,b,c]
         for i in laliste :
             print(i,i*2)
         print("="*50)
         letuple=(a,b,c)
         for i in letuple :
             print(i,i*2)
```

```
3 6
45.67 91.34
je suis un texte écrit en unicode je suis un texte écrit en unicodeje suis un texte
================================================
3 6
45.67 91.34
je suis un texte écrit en unicode je suis un texte écrit en unicodeje suis un texte
```

Quelle différence alors ? les tuples sont *immutables* càd qu'ils peuvent êtres utilisés comme clefs de dictionnaires...

4.3 listes définies en compréhension

ON utilisera souvent cette manière de créeer des listes :

```
In [16]: ll=[14,7,1789,14,9,1515]
         print(ll)
         print([x for x in ll])
         print([x**2 for x in ll])
         print([x for x in ll if x<1000])
         print([x**2 for x in ll if x<1000])

[14, 7, 1789, 14, 9, 1515]
[14, 7, 1789, 14, 9, 1515]
[196, 49, 3200521, 196, 81, 2295225]
[14, 7, 14, 9]
[196, 49, 196, 81]
```

L'itérateur `range` permet de créer des listes d'entiers à pas constant. N'hésitez pas à exécviuter `range?` dans une cellule pour obtenir une aide rapide.

```
In [17]: print([i for i in range(10)])
         print([i for i in range(3,10)])
         print([i for i in range(3,10,2)])
         print([i for i in range(10,3,-1)])

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 4, 5, 6, 7, 8, 9]
[3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4]
```

```
In [18]: from sympy import isprime
         print([i for i in range(1000) if isprime(i)])

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79
```

Nous utiliserons dans la suite les fonctions `sum` et `prod` qui calculent la somme et le produit des éléments d'une liste :

```
In [19]: print(sum(ll))
         print(prod(ll))

3348


         ---------------------------------------------------------------------------

         NameError                                 Traceback (most recent call last)

         <ipython-input-19-c227a7722b74> in <module>()
           1 print(sum(ll))
     ----> 2 print(prod(ll))


         NameError: name 'prod' is not defined
```

Caramba, ça ne marche pas, la fonction `prod` n'est disponible que dans des bibliothèques externes comme `sympy`...

```
In [20]: from sympy import prod
         print(sum(ll))
         print(prod(ll))

3348
33467216580
```

4.3.1 comment obtenir de l'aide en ligne ?

```
In [21]: prod?

In [22]: ll=[i for i in range(1,11)]
         print(ll,sum(ll),prod(ll))
         from sympy import factorial
         print(factorial(10))

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 55 3628800
3628800
```

Remarquez que dans un langage plus classique on écrirait quelque chose comme :

```
In [23]: s,p=0,1
         for i in range(1,11):
             s+=i
             p*=i
         print(s,p)

55 3628800
```

En détaillant un peu plus :

```
In [24]: s,p=0,1
         print("%8s %3s %8s"%("i","s","p"))
         print("%8s %3s %8s"%("avant",s,p))
         for i in range(1,11):
             s+=i
             p*=i
             print("%8s %3s %8s"%(i,s,p))

       i   s        p
   avant   0        1
       1   1        1
       2   3        2
       3   6        6
       4  10       24
       5  15      120
       6  21      720
       7  28     5040
       8  36    40320
       9  45   362880
      10  55  3628800
```

4.3.2 dictionnaires

```
In [25]: ledico={1:'un',2:'deux',7:'sept',70:'septante'}
         print(ledico)
         print(ledico[70])

{1: 'un', 2: 'deux', 7: 'sept', 70: 'septante'}
septante


In [26]: print(ledico[90])

         ---------------------------------------------------------------------------

         KeyError                                  Traceback (most recent call last)
```

```
        <ipython-input-26-a4c45f58ce1e> in <module>()
    ----> 1 print(ledico[90])


        KeyError: 90
```

```
In [27]: ledico[70]='soixante-dix'
         ledico[90]='quatre-vingt-dix'
         print(ledico[70])
         print(ledico[90])
```

```
soixante-dix
quatre-vingt-dix
```

4.3.3 boucle `for` et indentation

C'est l'**indentation** qui crée les blocs d'instructions (rôle des accolades en `Java` ou en `C`)...
Veuillez produire l'affichage suivant :

```
couplet 1
refrain
couplet 2
refrain
couplet 3
refrain
clap clap clap
```

```
In [28]: # essai 1
         for i in range(4) :
             print(i)
         #
         print("="*50)
         # essai 2
         for i in range(1,4) :
             print(i)
         #
         print("="*50)
         # essai 3
         for i in range(1,4) :
             print("essai %s"%i)
         #
         print("="*50)
         # essai 4
         for i in range(1,4) :
             print("essai %s"%i)
             print("refrain")
         print("clap "*3)
```

```
0
1
2
3
==================================================
1
2
3
==================================================
essai 1
essai 2
essai 3
==================================================
essai 1
refrain
essai 2
refrain
essai 3
refrain
clap clap clap
```

On dispose d'autres structures de contrôle : `while`, `if`, ...

4.3.4 fonctions

Deux rédactions : `lambda` et `def`, par exemple pour créer la fonction $f = x \mapsto x^2 + x + 4$:

```
In [29]: f = lambda x : x**2+x+4
         for i in range(3) :
             print('f(%s) = %s'%(i,f(i)))

f(0) = 4
f(1) = 6
f(2) = 10
```

```
In [30]: def f(x) :
             y = x**2+x+4
             return y
         for i in range(3) :
             print('f(%s) = %s'%(i,f(i)))

f(0) = 4
f(1) = 6
f(2) = 10
```

ou encore :

```
In [31]: def f(x) :
             return x**2+x+4
         for i in range(3) :
             print('f(%s) = %s'%(i,f(i)))

f(0) = 4
f(1) = 6
f(2) = 10
```

On peut avoir plusieurs paramètres et retourner plusieurs valeurs :

```
In [32]: somme_et_produit = lambda x1,x2 : (x1+x2,x1*x2)
         for (i1,i2) in [(1,2),(3,4),(5,6)] :
             print ((i1,i2),somme_et_produit(i1,i2))

(1, 2) (3, 2)
(3, 4) (7, 12)
(5, 6) (11, 30)
```

```
In [33]: def somme_et_produit(x1,x2) :
             return (x1+x2,x1*x2)
         for (i1,i2) in [(1,2),(3,4),(5,6)] :
             print ((i1,i2),somme_et_produit(i1,i2))

(1, 2) (3, 2)
(3, 4) (7, 12)
(5, 6) (11, 30)
```

On peut aussi accepter un nombre variables d'arguments, mais c'est un peu plus compliqué...
Les variables locales à une fonction sont bien entendu invisibles à l'extérieur de la fonction :

```
In [34]: def f(x) :
             y = x**2+x+4
             return y
         #####
         y="trucmuche"
         print(y)
         for i in range(3) :
             print('f(%s) = %s'%(i,f(i)))
         print(y)

trucmuche
f(0) = 4
f(1) = 6
f(2) = 10
trucmuche
```

4.4 classes

TODO

Python — et plus particulièrement `python3` est un **langage à objets** à part entière, permettant le multi-héritage et la suRcharge des opérateurs, entre autres caractéristiques agréables au mathématiocien ou à l'ingénieur… Nous laissons toutefois pour l'instant de côté ce point imortant mais qu'on ne peut traiter trop rapidement.

In [ ]:

In [ ]:

In [ ]: