Programmation avec Python - Automne 2016
Bruno Martin & Jean-Paul Roy
L1-Sciences

Cours n°1

http://deptinfo.unice.fr/~roy

Introduction au langage Python3



http://docs.python.org/py3k

La maison mère

COMPOSING PROGRAMS

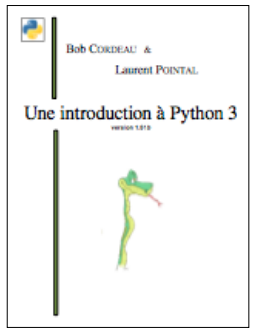
for those who have some background in programming and are theoretically inclined…

CS 61A

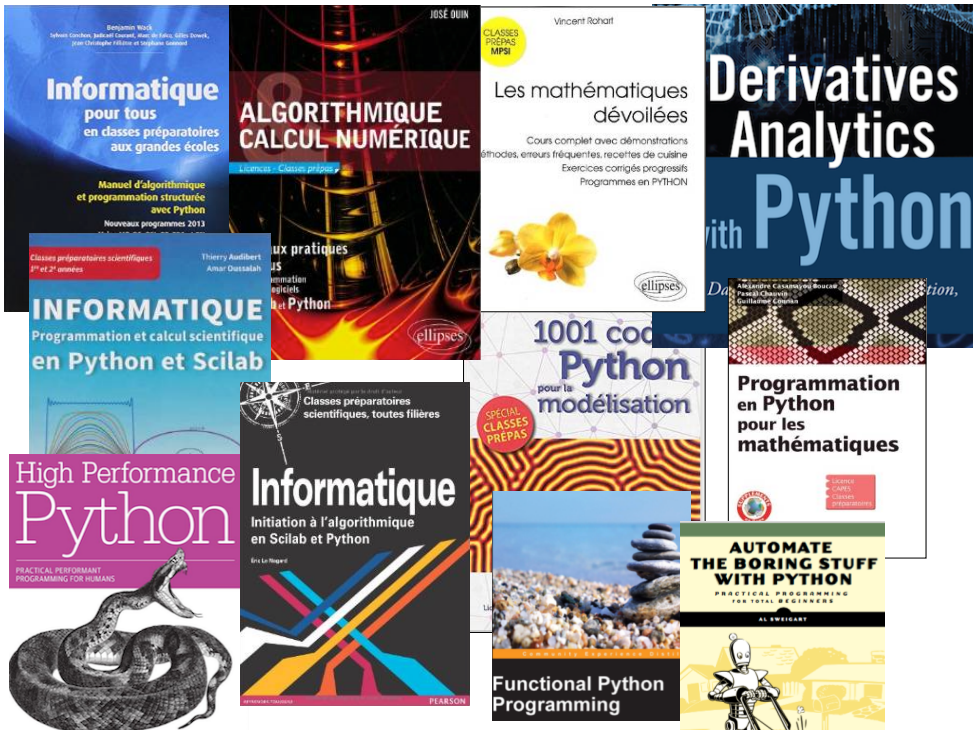
Structure and Interpretation of Computer Programs, Fall 2014

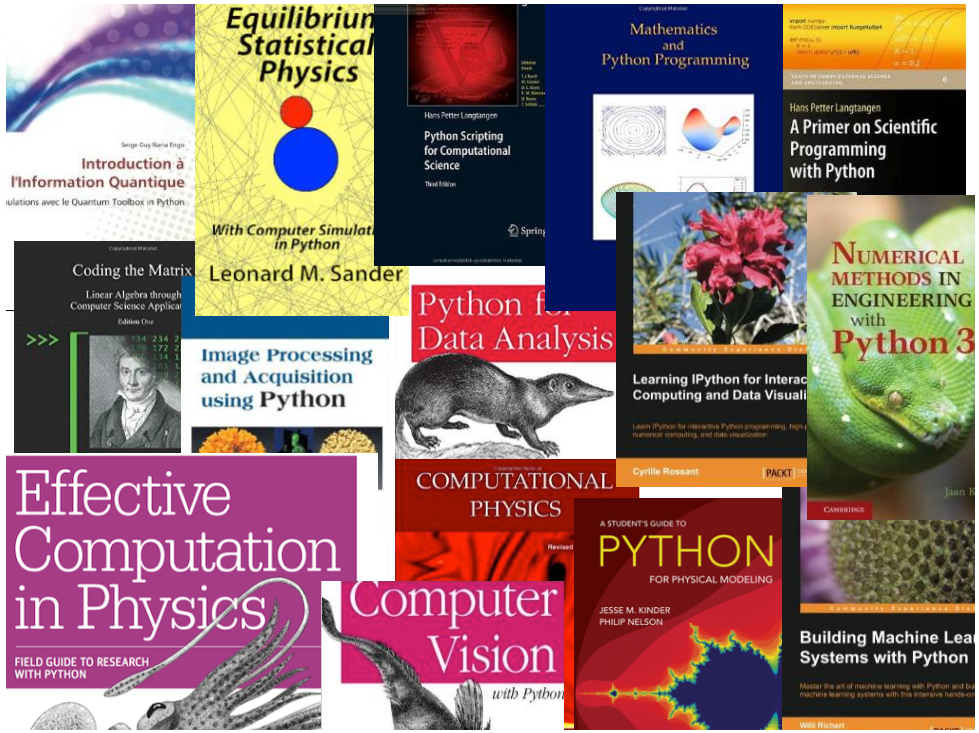
Instructor: John DeNero

(Berkeley Univ.)



polycopié Orsay





Ressources sur cet enseignement

- Les livres cités à la page précédente, entre autres. Achat non obligatoire. Mais il est très important de **LIRE DES LIVRES** !

- Le langage support est **Python**, dans sa version **3.6**

- La **page Web du cours**, que vous enregistrez dans vos signets :

  `http://deptinfo.unice.fr/~roy`

- La documentation exhaustive du langage chez la maison mère :

  `http://docs.python.org/3.6/`

- Les logiciels à télécharger : voir la page Web du cours...

Qu'est-ce que la Programmation ?

- L'art de rédiger des textes dans une langue artificielle.

  ↓ *la science ?* ↓ *les programmes* ↓ *un langage de programmation*

- Dans quel but ? Faire exécuter une tâche à un ordinateur.
- Il s'agit d'une activité de **résolution de problèmes**.

- Hum, on va faire des maths, alors ?

- Un peu quand même, mais pas trop, le monde est vaste. Nous allons tâcher d'en modéliser de petites portions pour les faire rentrer dans la machine. Des nombres, du texte, des images, le Web...

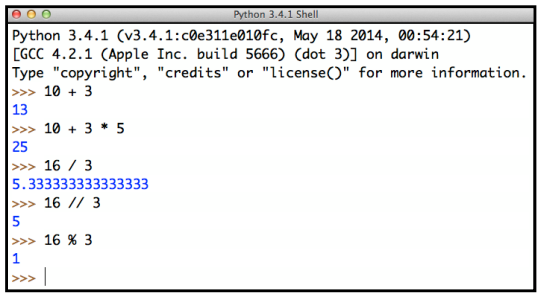
- On pourrait presque dire : calculer le Monde ?

- Oui, bravo, c'est cela, ***réduire le Monde à des objets sur lesquels on peut faire des calculs****.*

Jouer avec des nombres entiers

- Comme dans toutes les sciences, les nombres jouent un rôle important.

- Python se présente comme une calculette interactive à travers son **toplevel**.
Il présente son prompt >>> pour que vous lui soumettiez un calcul...



- **Opérateurs de base dans les entiers** : `+` `-` `*` `//` `%` `**`

  a // b fournit le **quotient** de l'entier a par l'entier b ≠ 0.

  a % b fournit le **reste** de la division de l'entier a par l'entier b ≠ 0.

- Si a et b sont entiers avec b ≠ 0, alors :

  `a == b * (a // b) + (a % b)` ***division euclidienne***

- L'opérateur ** permet de calculer une puissance :

```
>>> 2 ** 10          # le "kilo informatique"
1024
```

- Mal utilisée, une opération peut provoquer une ERREUR :

```
>>> 16 % 0
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    16 % 0
ZeroDivisionError: integer division or modulo by zero
```

- L'ordre du calcul d'une expression arithmétique tient compte de la **priorité** de chaque opérateur.

`+` `-`	`*` `//` `%`	`**`
même priorité (faible)	même priorité (haute)	(très haute)

```
>>> 5 - 8 + 4 * 2 ** 3
29
```

```
a - b + c * d ** e
(a - b)  +  (c * (d ** e))
```

- La taille des entiers n'est limitée que par la mémoire de la machine :

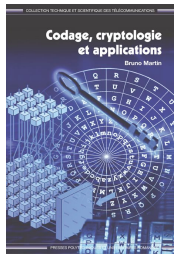
```
>>> 87412192161147838437159l419 ** 10
260447454985660585245180084757921014509252146181223003
455270044550605023694309556482234857745408080127776885
578607664767325495386096105286268494775055260671252317
663749619931275002342815836733596266389781703659821356
427940431697254607426485624871372306773584664397463801
```

- On dit improprement que l'on travaille avec des nombres entiers ***en précision infinie*** ! On dit aussi que ***le calcul entier est exact***.

- Cette propriété est importante pour les problèmes de **cryptologie** par exemple (codes secrets). Il est difficile de décomposer :

```
152734782063723562326183089878176004074l
```

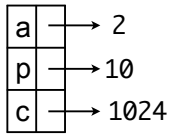
en produit de facteurs premiers !
Et pourtant, indication : il n'y en a que deux...



Programmer avec des variables

- Un peu comme les ***inconnues*** en Algèbre, sauf qu'ici une variable devra être ***connue*** et contenir une valeur...

```
>>> a = 2          # lire : a prend pour valeur 2
>>> p = 10         # p prend pour valeur 10
>>> c = a ** p     # c prend pour valeur celle de a^p
>>> c
1024
```



variable = expression

- **Ce signe = non commutatif n'a rien à voir avec celui des maths !**
- L'écriture `2 + 3 = a` n'aura donc aucun sens !!

```
>>> 2 + 3 = a
SyntaxError: can't assign to operator
```

```
>>> a = 2 + 3
>>> a
5
```

- On dit que `a = 2 + 3` est une **instruction d'affectation**.

- Ne confondez pas les **EXPRESSIONS** et les **INSTRUCTIONS**, qui toutes deux vont contenir des variables. Une expression sera ***calculée***, tandis qu'une instruction sera ***exécutée*** !

Expression (a vaut 5, x vaut 3)

```
a * x ** 2   ⟹   45
```

```
>>> a * x ** 2
45
```

Instruction (a vaut 5, x vaut 3)

```
c = a * x ** 2   ⟹   Aucun résultat
```

```
>>> c = a * x ** 2
>>>
```

- Une **variable** a le droit de **changer de valeur** !

```
>>> a = 2
>>> b = a      # la valeur de a est calculée puis c'est elle qui est transmise à b
>>> a = a + 1      # qui se lit :  a  devient égal à  la valeur de a + 1
>>> b
2              # et non 3, ok ?
```


- Exemple : **échange de deux variables**. On utilise une variable temporaire.

```
>>> a = 2
>>> b = 3
```

⟹

```
>>> temp = a
>>> a = b
>>> b = temp
```

⟹

```
>>> a
3
>>> b
2
```

- L'opérateur d'**égalité** mathématique se note ==

```
>>> a == 3
True
```

```
>>> b == 3
False
```

```
>>> True == False
False
```

- Les valeurs `True` et `False` sont les **valeurs booléennes** :

```
>>> a > 3
False
```

```
>>> a + 1  >= 3
True
```

```
>>> a + 1  < 3
False
```

- Dans le contexte d'une expression arithmétique, `True == 1` et `False == 0`. **Evitez d'utiliser ce genre de facilités très laides** !

```
>>> 5 + True
6
```

```
>>> True * False
0
```

```
>>> True == False + 1
True
```

Affichage de résultats avec print(...)

- La fonction print(...) permet d'afficher une suite d'expressions :

```
>>> a = 2
>>> print('Le carré de a vaut',a*a,'et non 5')
Le carré de a vaut 4 et non 5
>>>
```

espace automatique

- Ce qui est affiché en bleu n'est pas le résultat de la fonction print, mais **l'effet** de cette fonction. La fonction print n'a aucun résultat ! Ou plutôt son résultat est None, qui ne s'affiche pas...

```
>>> print(5) == None
5
True
>>> x = None
>>> x
>>>
```

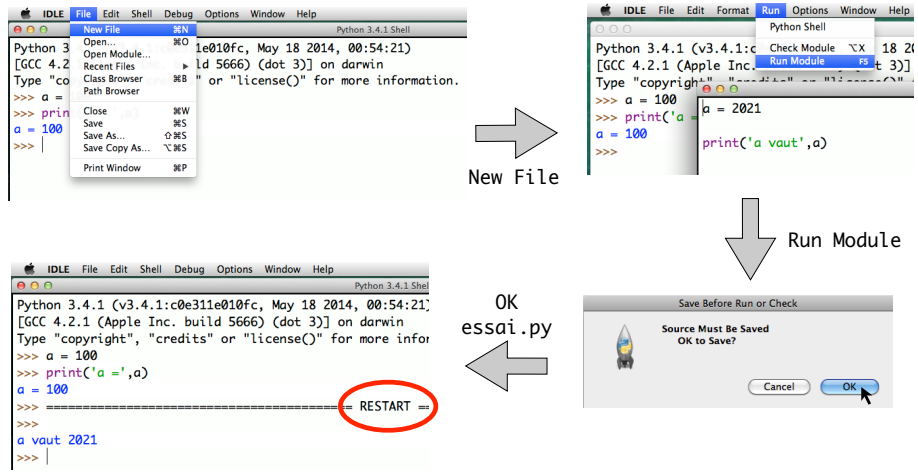
l'effet de print(5)

le résultat du test d'égalité

None est une valeur spéciale signifiant "*rien*"...

Rédaction de textes dans l'éditeur IDLE

- Le travail interactif au ***toplevel*** est pratique pour de petits calculs. Mais mieux vaut en général travailler dans ***l'éditeur intégré*** IDLE.



Conditionnelle : la prise de décisions if

- L'instruction conditionnelle if permet de prendre une décision :

IDLE

```
a = -2
if a > 0 :
    print(a,'est positif')
else :
    print(a,'est négatif ou nul')
```

Run

```
-2 est négatif ou nul
```

- Au toplevel, c'est moins agréable, il faut terminer par une ligne vide.

```
>>> a = -2
>>> if a > 0 :
    print(a,'est positif')
else :
    print(a,'est négatif ou nul')

-2 est négatif ou nul
```

ligne vide, OVER, à toi !

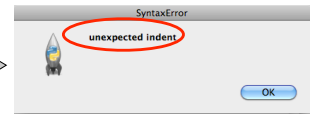
- La bonne distance à la marge d'une ligne (**indentation**) doit être respectée. Elle permet de structurer et de comprendre un programme :

```
a = -2
if a > 0 :
    print(a,'est positif')
else :
    print(a,'est négatif ou nul')
```

- Un **bloc** d'instructions est une suite d'instructions alignées à la verticale. Vous voyez ci-dessus un bloc formé de deux instructions.

```
a = -2
if a > 0 :
print(a,'est positif')
else :
    print(a,'est négatif ou nul')
```

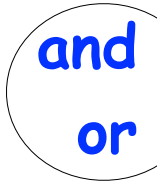
Run



- **UNE INDENTATION CORRECTE EST OBLIGATOIRE EN PYTHON.**

- Les mots-clés and et or ressemblent à ceux de la Logique :

p	True	True	False	False
q	True	False	True	False
p and q	True	False	False	False
p or q	True	True	True	False



- Mais en Python, ils sont **court-circuités** :

```
>>> a = -2
>>> x == 3
NameError: name 'x' is not defined
>>> (a > 0) and (x == 3)
False
```

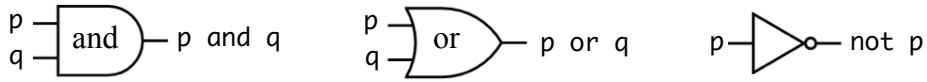
- L'expression `x == 3` n'a pas été évaluée car `False and ? == False`
- La priorité de and étant plus faible que celle des opérations arithmétiques, on aurait pu écrire : `a > 0 and x == 3`. Dans le doute, mieux vaut mettre des parenthèses !

- Idem pour le mot-clé or :

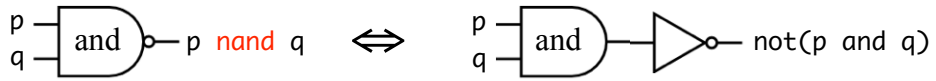
```
>>> a = -2
>>> (a < 0) or (x == 3)
True
```

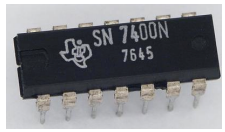
car `True or ? == True`

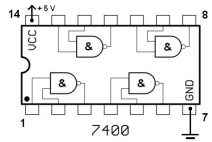
- En électronique numérique, on représente and, or et not par des **portes logiques** :



- Le mot-clé not inverse les valeurs True et False. C'est **l'inverseur**...
- Les constructeurs d'ordinateurs utilisent beaucoup la porte nand :







Les fonctions prédéfinies de Python

- Tous les langages de programmation fournissent un large ensemble de **fonctions** prêtes à être utilisées. Exemple dans les entiers :

```
>>> 5 * 2 ** 3     # les opérateurs arithmétiques sont des fonctions cachées
40
>>> abs(-5)        # la fonction "valeur absolue" est prédéfinie
5
```

- Certaines fonctions résident dans des **modules** spécialisés, comme `fractions`, `math`... Il faut consulter la documentation en ligne !

```
>>> gcd(18,12)                  # je veux calculer un PGCD
NameError: name 'gcd' is not defined
>>> import fractions            # importation du module fractions
>>> fractions.gcd(18,12)        # le mot gcd seul reste inconnu !
6
>>> from fractions import gcd   # importation du seul mot gcd
>>> gcd(18,12)                  # le mot fractions reste inconnu !
6
```


Comment définir une nouvelle fonction ?

- Soit à définir la fonction $f : n \mapsto 2n - 1$

IDLE

```
def f(n) :
    return 2 * n - 1

print('f(5) vaut',f(5))
```

Run

```
f(5) vaut 9
```

- Notez l'indentation (automatique en principe)...
- Le mot return signifie "***le résultat est...***". On dit que la fonction ***retourne un résultat*** (à celui qui a demandé le calcul).
- Il n'est nulle part dit que n est un entier. On peut aussi utiliser f sur les réels approchés voire même sur les complexes :

```
>>> f(5)
9
```

```
>>> f(5.2)
9.4
```

```
>>> f(2+3j)      # j au lieu de i
(3+6j)
```

- Comment définir la valeur absolue si elle n'existait pas ?

```
def val_abs(n) :
    if n > 0 :
        return n
    else :
        return -n

print('|-5| vaut',val_abs(-5))
```

IDLE

Run

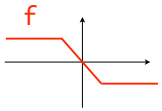
|-5| vaut 5

- Les choix multiples avec if ... elif ... elif ... elif ... else ... :

```
def f(x) :
    if x < -1 :
        return 1
    elif x <= 1 :
        return -x
    else :
        return -1
```

⇔

```
def f(x) :
    if x < -1 :
        return 1
    else :
        if x <= 1 :
            return -x
        else :
            return -1
```



- Comment savoir si deux entiers n'ont que 1 comme diviseur commun ?

```
def premiers_entre_eux(p,q) :
    if gcd(p,q) == 1 :
        return True
    else :
        return False
```

IDLE

⇔

```
def premiers_entre_eux(p,q) :
    return gcd(p,q) == 1
```

```
>>> premiers_entre_eux(21,6)
False
>>> premiers_entre_eux(21,8)
True
```

⇕

```
def premiers_entre_eux(p,q) :
    if gcd(p,q) == 1 :
        return True
    return False
```

Le mot clé return provoque un échappement. Le reste du texte de la fonction est abandonné !

- Vous voyez qu'il existe différentes manières de coder une fonction. Elles se distinguent par leur **efficacité**, mais aussi leur **élégance**.

Générer des entiers au hasard

- Il faut importer la fonction `randint(...)` du module `random` :

```
from random import randint
```

- Avec a ≤ b entiers, `randint(a,b)` retourne un entier aléatoire de l'intervalle [a,b].

pseudo-aléatoire !

- Ex: `2 * randint(0,10)` retourne un entier pair aléatoire de `[0,20]`.
- Une fonction `jet35()` sans arguments qui tire 3 ou 5 au hasard :

```
def jet35() :
    if randint(0,1) == 0 :
        return 3
    else :
        return 5
```

IDLE

```
>>> jet35()
3
>>> jet35()
5
>>> jet35()
5
......
```