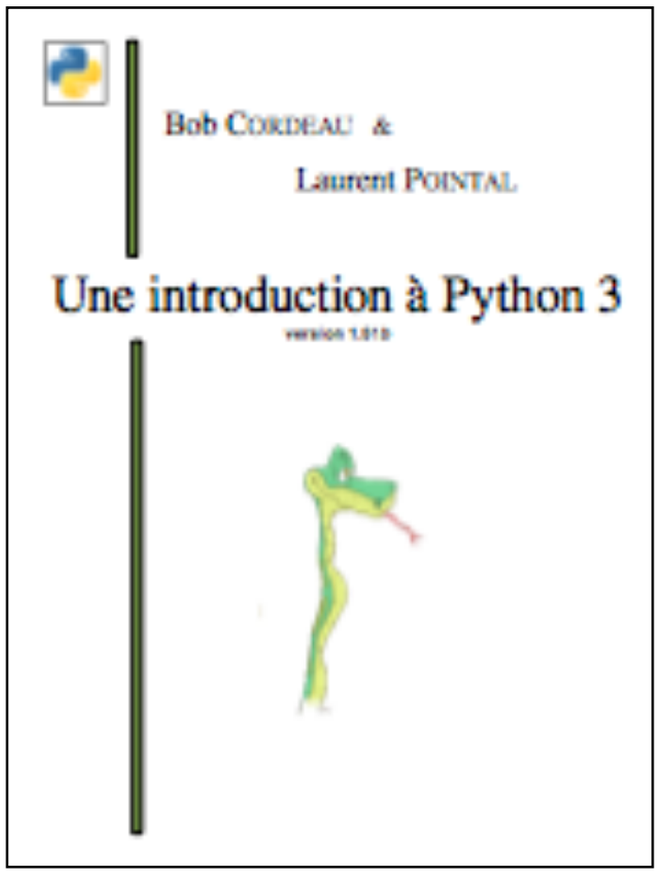
http://deptinfo.unice.fr/~elozes

Introduction au langage Python3

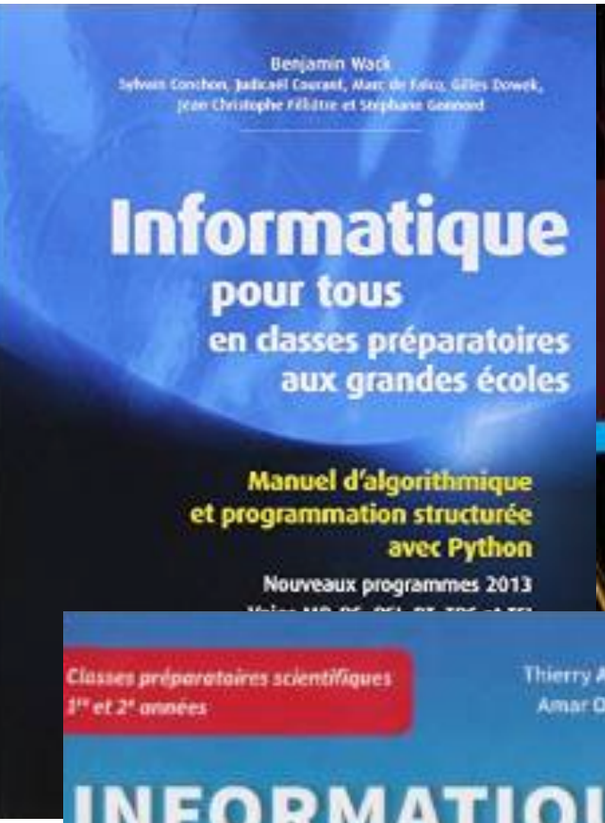
http://docs.python.org/py3k

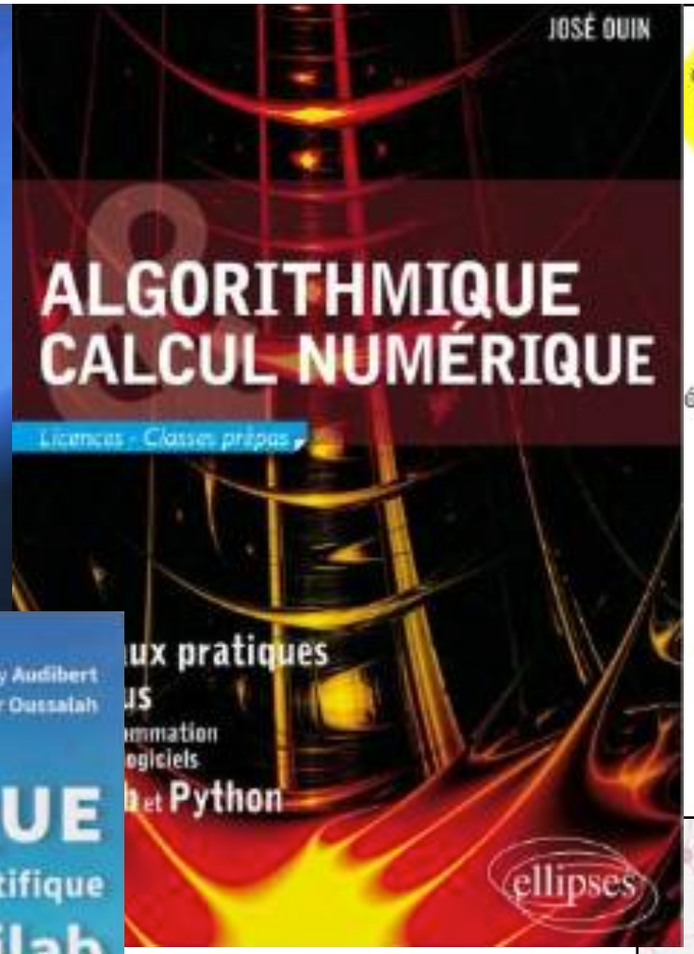
La maison mère

livre gratuit en ligne,
payant en librairie

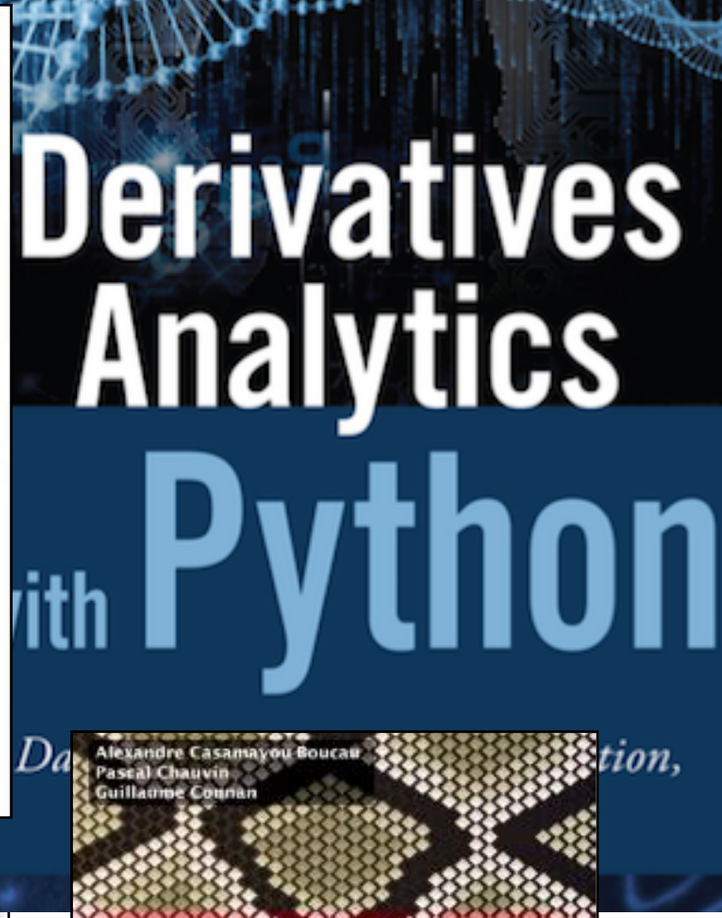


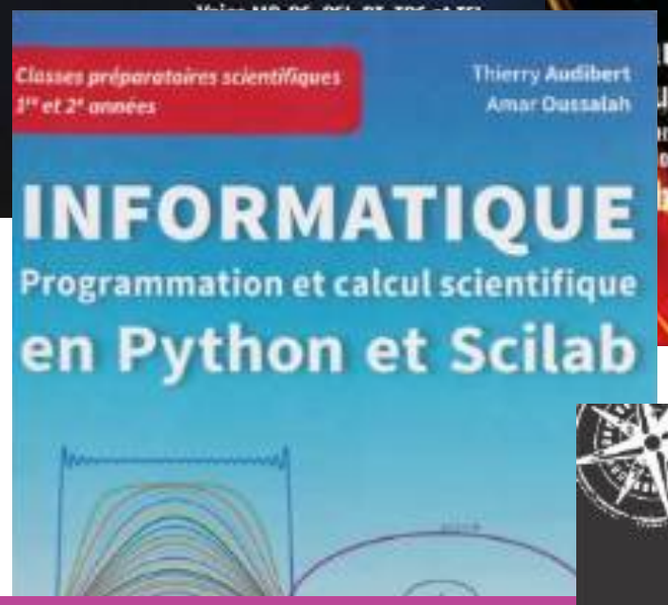
polycopié Orsay


Benjamin Wack
Sylvain Conchon, Judicaël Courant, Marc de Falco, Gilles Dowek,
Jean Christophe Filliâtre et Stéphane Gonnord
Informatique
pour tous
en classes préparatoires
aux grandes écoles
Manuel d'algorithmique
et programmation structurée
avec Python
Nouveaux programmes 2013

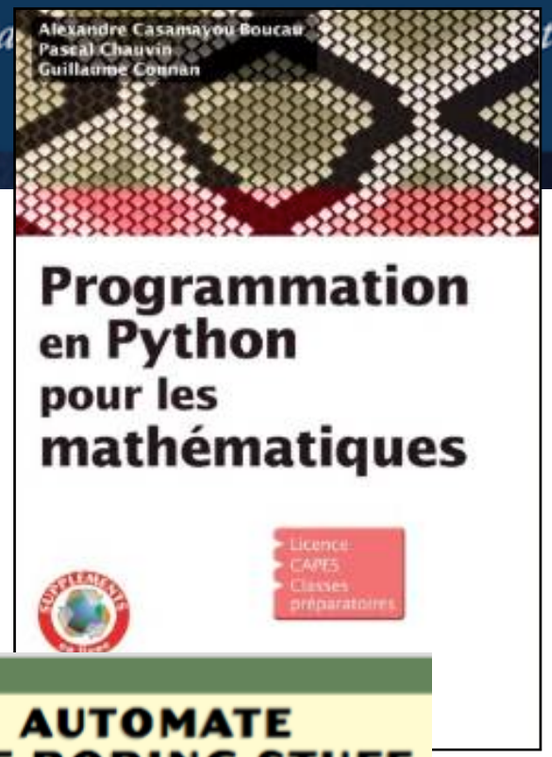

JOSÉ DUIN
ALGORITHMIQUE
CALCUL NUMÉRIQUE
Licences - Classes prépas
ellipses

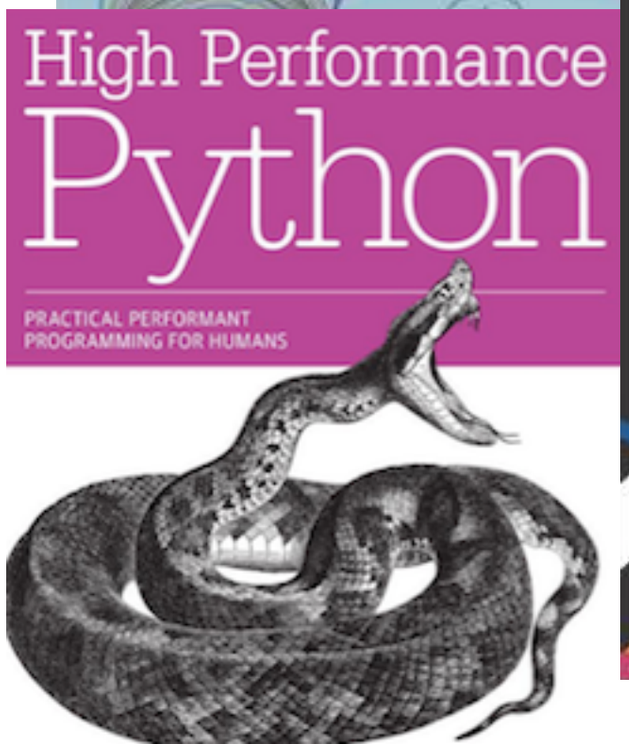

Vincent Rohart
CLASSES PRÉPAS MPSI
Les mathématiques dévoilées
Cours complet avec démonstrations
éthodes, erreurs fréquentes, recettes de cuisine
Exercices corrigés progressifs
Programmes en PYTHON
ellipses


Derivatives
Analytics
with Python


Classes préparatoires scientifiques
1re et 2e années
Thierry Audibert
Amar Oussalah
INFORMATIQUE
Programmation et calcul scientifique
en Python et Scilab

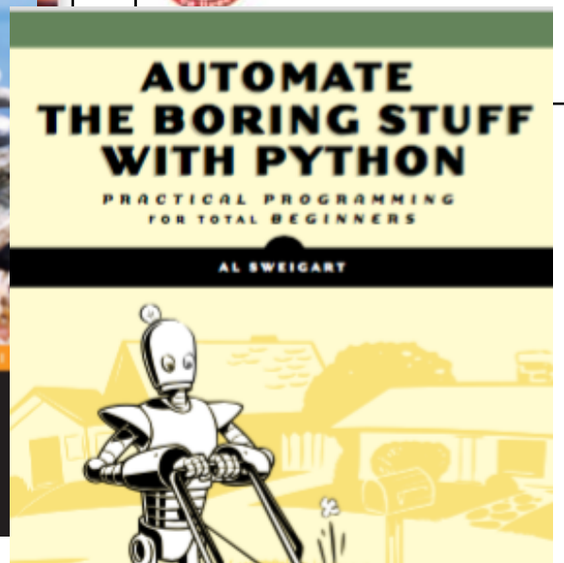

1001 cod
Python
pour la
modélisation
SPÉCIAL CLASSES PRÉPAS


Alexandre Casamayou Boucau
Pascal Chauvin
Guillaume Connan
Programmation
en Python
pour les
mathématiques
Licence
CAPES
Classes préparatoires


High Performance
Python
PRACTICAL PERFORMANT
PROGRAMMING FOR HUMANS


Classes préparatoires
scientifiques, toutes filières
Informatique
Initiation à l'algorithmique
en Scilab et Python
Éric Lr Nogard
PEARSON

Functional Python
Programming


AUTOMATE
THE BORING STUFF
WITH PYTHON
PRACTICAL PROGRAMMING
FOR TOTAL BEGINNERS
AL SWEIGART

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 **3.7**

- **Le cours sur MOODLE** http://lms.unice.fr

- La page publique du cours http://deptinfo.unice.fr/~elozes

- La documentation exhaustive du langage chez la maison mère :

http://docs.python.org/3.7/

Autres informations pratiques

Modalités de contrôle des connaissances

- 1/3 contrôle continu
- 1/3 partiel
- 1/3 examen

Le contrôle continu

Des QCMs très régulièrement en TP.

VÉRIFIEZ VOTRE ACCÈS AU COURS MOODLE AVANT LE PREMIER TP.

Le partiel

Ce sera lundi 18/03 de 17:30 à 19:30

NOTEZ LA DATE TOUT DE SUITE!

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...

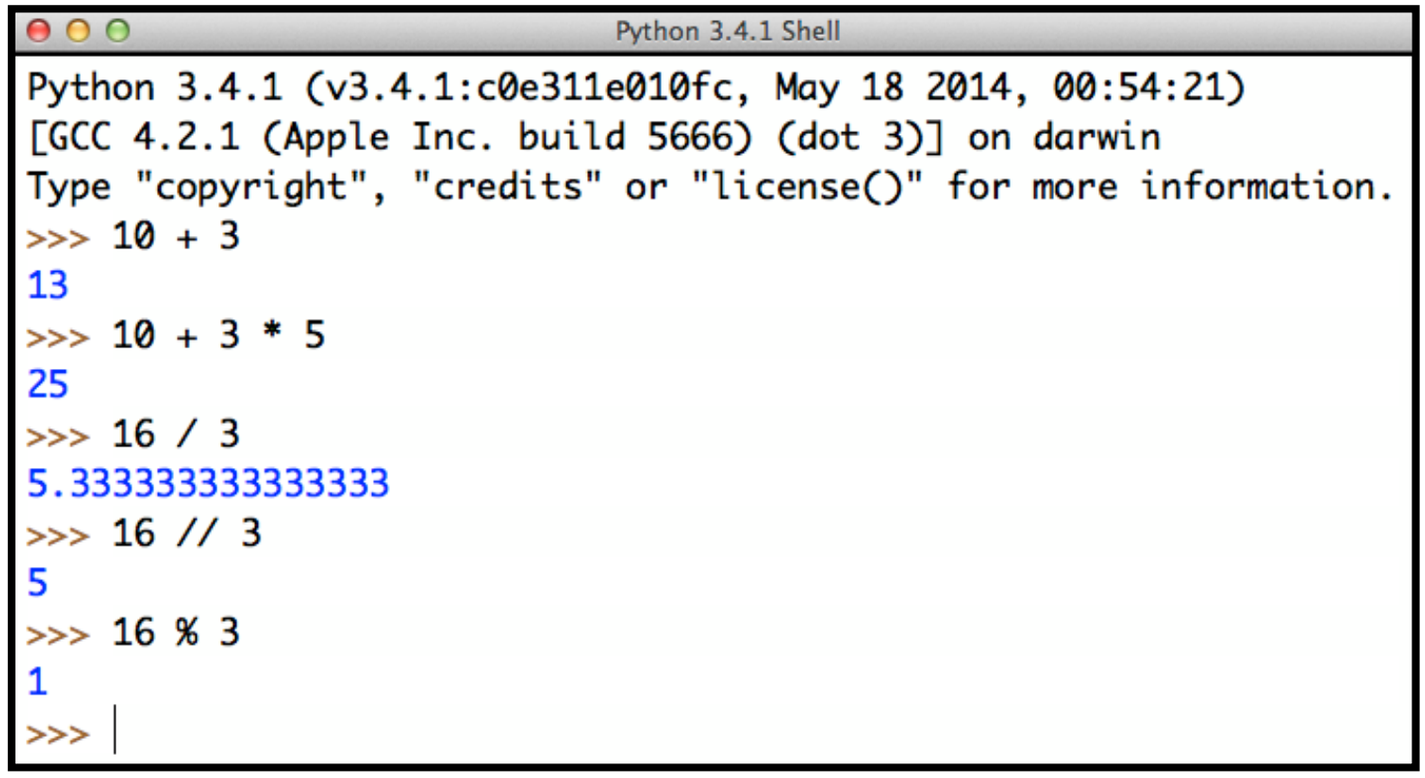
- On pourrait presque dire : calculer le Monde ?

- 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
```

après #, commentaires

- 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))
```

- Dernière règle : entre opérateurs de même priorité, on applique l'**associativité à gauche**

```
>>> 15 - 8 - 1
6
```

(15 - 8) - 1

~~15 - (8 - 1)~~

Attention avec l'**associativité à gauche** lorsqu'on mélange // et *

```
>>> 2 * 3 // 6
1
```

```
>>> 2 // 6 * 3
0
```

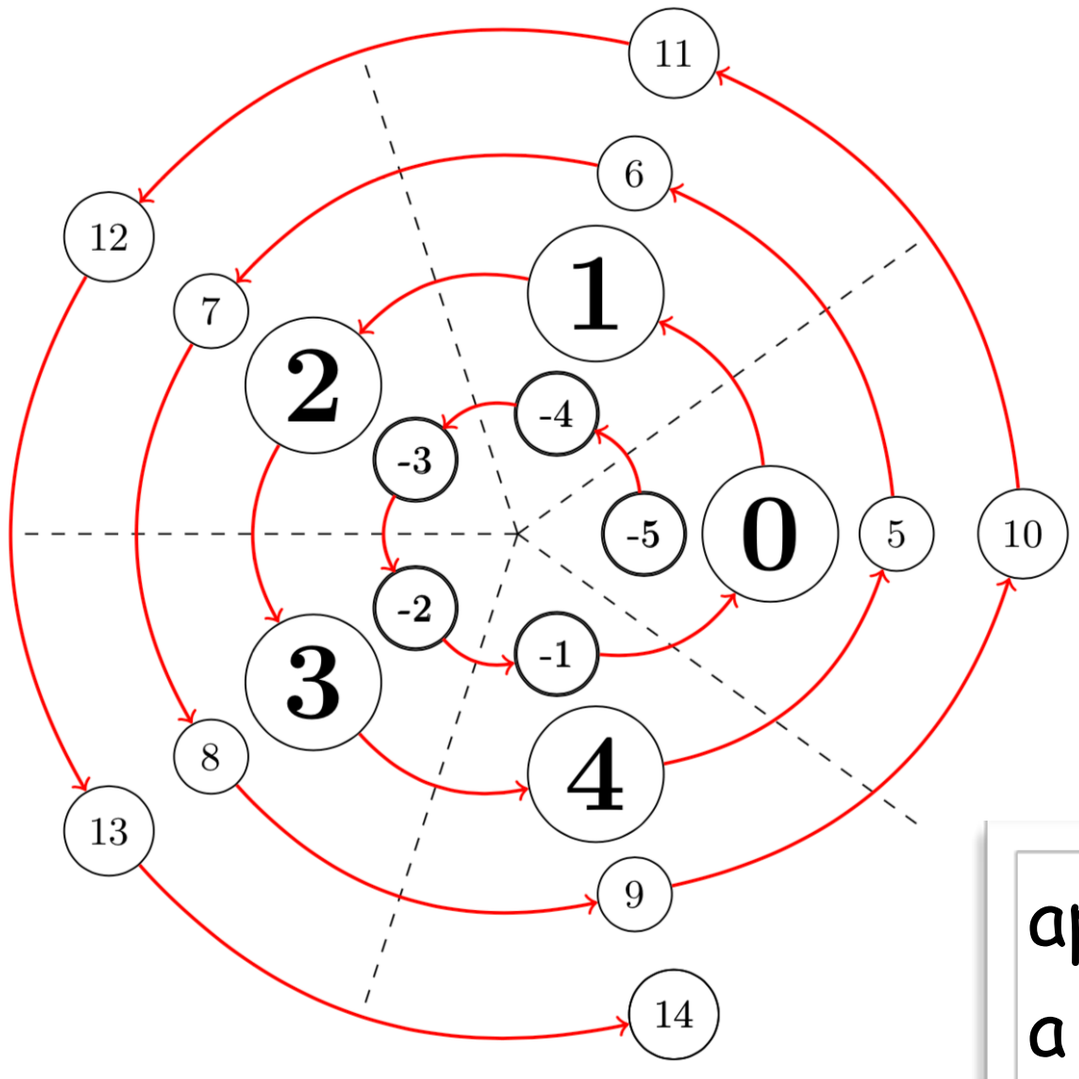
Bien sûr, rien n'interdit de rajouter des parenthèses « inutiles » pour être plus lisible

```
>>> (3 * (3 + 1)) // 6
2
```

en rouge inutile, en noir nécessaire

Une vue imagée des modulos

raisonner modulo b, c'est placer les nombres sur l'une des b « directions » possibles et les numéroter de 0 à b-1




```
>>> (-3) % 5
2
```

```
>>> (-3) // 5
-1
```

car -3 = 5 * (-3 // 5) + -3 % 5

application classique (à connaitre par coeur!):
a est un multiple de b ssi a % b = 0

hors cas a=b=0 (le seul multiple de 0 est 0)

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

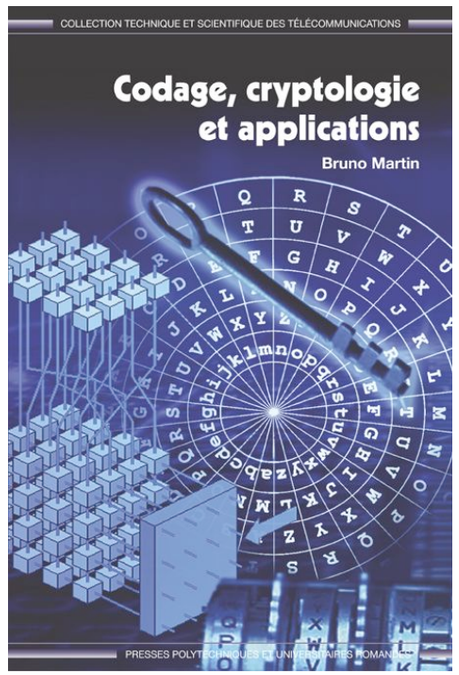
```
>>> 87412192161147838437159141 9 ** 10
260447454985660585245180084757921014509252146181223003
455270044550605023694309556482234857745408080127776885
578607664767325495386096105286268494775055260671252317
663749619931275002342815836733596266389781703659821356
427940431697254607426485624871372306773584664397463801
```

- 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 :

```
152734782063723562326183089878176004074 1
```

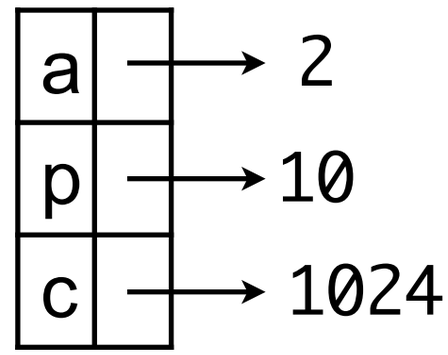
en produit de facteurs premiers !
Et pourtant indication : il n'y en a que deux...



Programmer avec des variables

Une variable est une boite dans laquelle on peut enregistrer 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 symétrique 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
```

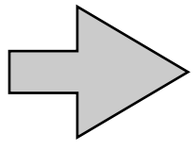
```
>>> c = a * x ** 2
>>>
```

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

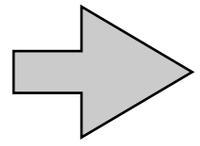
```
>>> a = 2
>>> b = a     # la valeur de a est calculée puis c'est elle qui est transmise à b
>>> a = a + 1       # qui se lit :  a  devient égal à  la valeur de a + 1
>>> b
2                   # et non 3, ok ?
```

- 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`. Bon à savoir pour comprendre ses erreurs, mais à **éviter.**

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

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

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

Que peut-on mettre dans une variable?

N'importe quelle valeur!

Attention une valeur n'est pas nécessairement un entier…

```
>>> a = 2
>>> b = 3.14
>>> c = False
>>> d = 'mon petit message à afficher'
>>>
```

Notion de programme

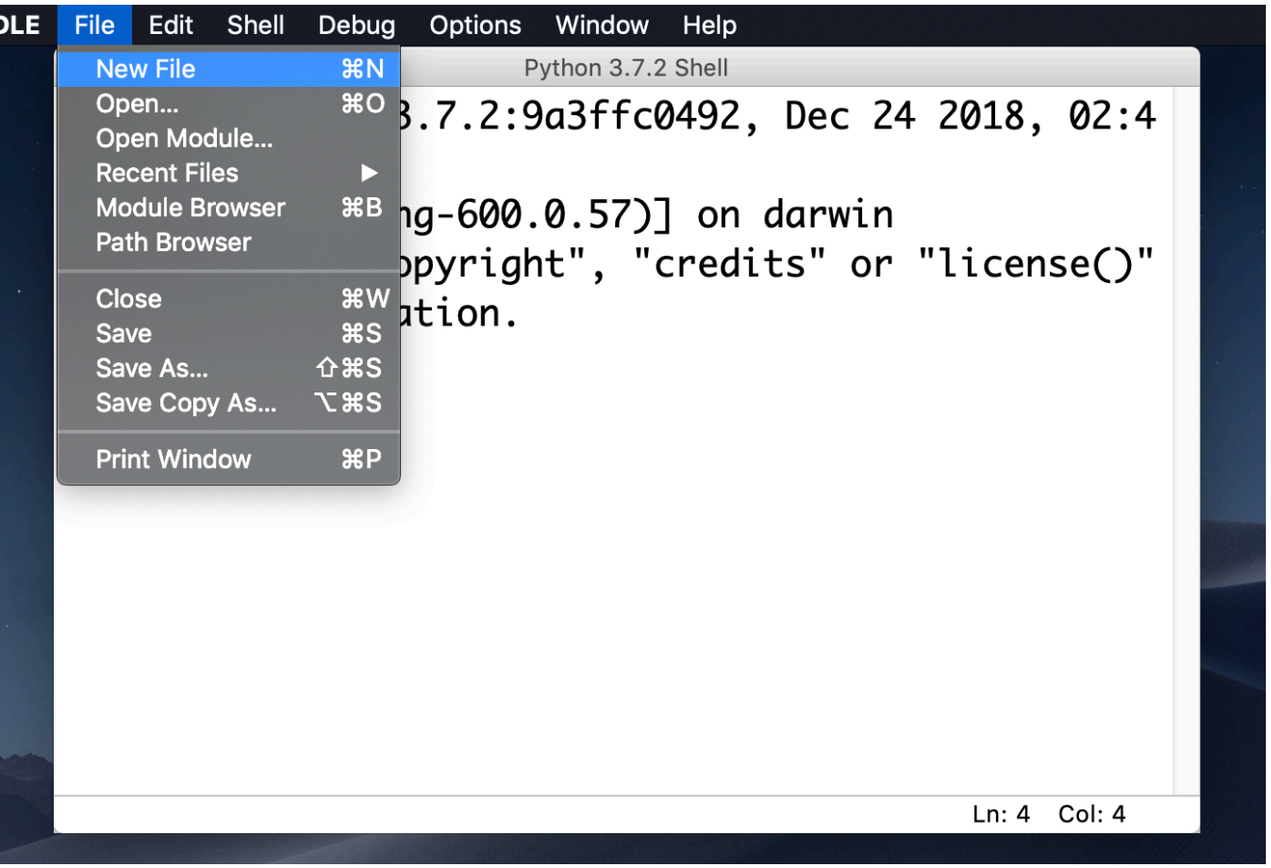
- Le travail interactif au ***toplevel*** est pratique pour de petits calculs et pour se familiariser avec Python.

- Mais un « vrai » programme, ce n'est pas ça!!

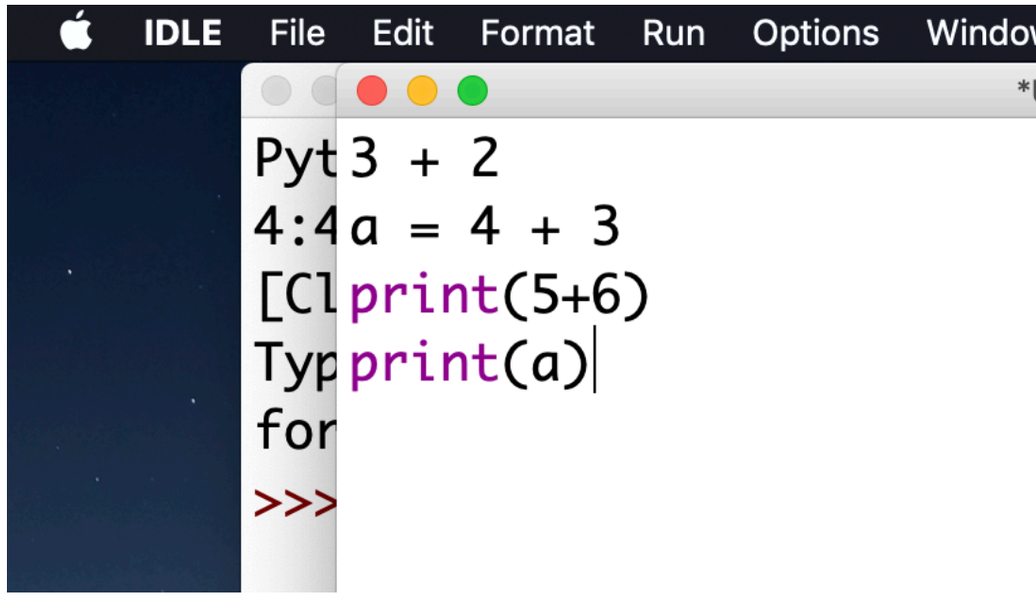
- Un programme = une suite d'instructions qui sont exécutées sans interruption et sans afficher de résultats. Il faut forcer l'affichage en utilisant l'instruction **print**.

PROGRAMME ≠ TOPLEVEL

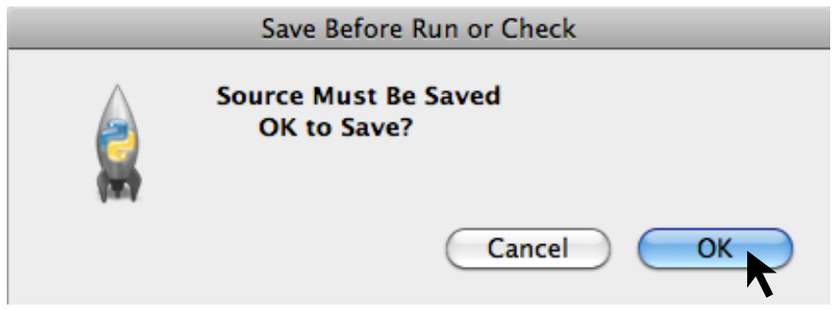
Écrire un programme dans l'éditeur IDLE



New File

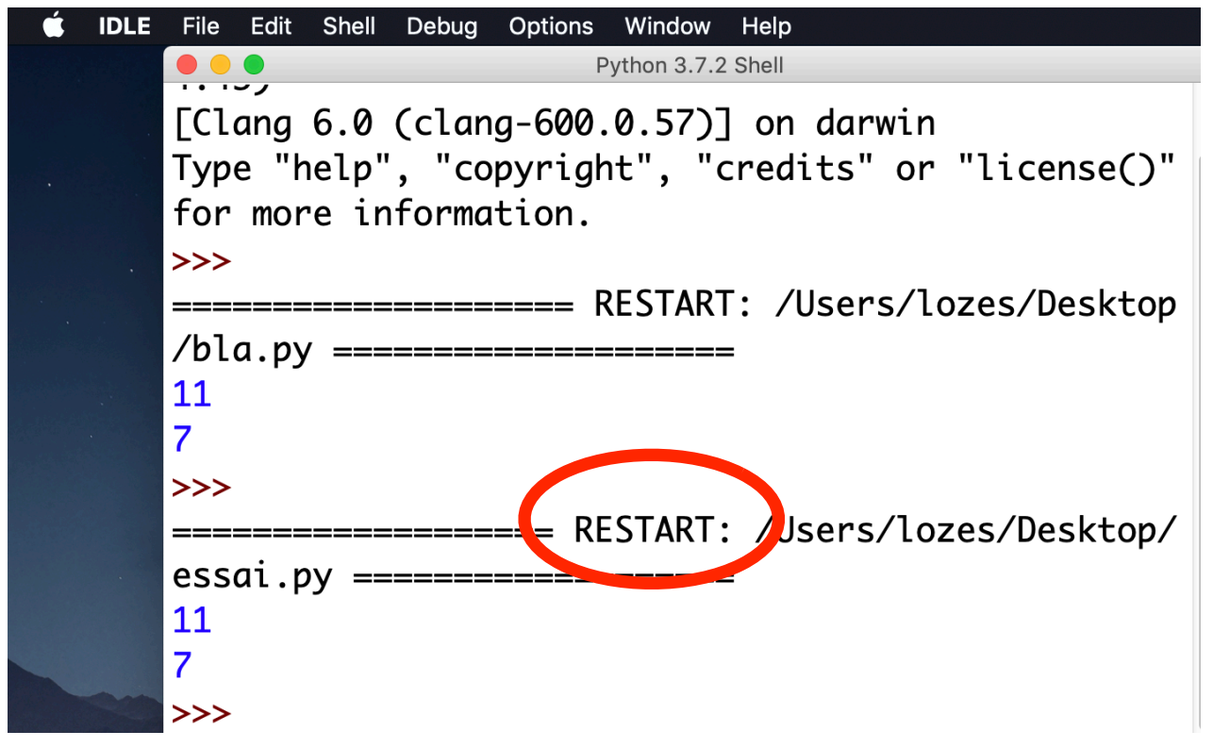


Run Module



OK

essai.py



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 <u>résultat</u> de la fonction print, mais <u>**l'effet**</u> de cette fonction. La fonction print n'a aucun résultat ! Ou plutôt son résultat est **None**, qui ne s'affiche pas au toplevel…

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

l'effet de print(5)

la valeur de print(5) ne s'affiche pas

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

Les différents types de valeurs

La valeur d'une expression ou d'une variable a un type. Jusqu'à présent nous avons rencontré les types suivants.

LES ENTIERS

valeurs possibles: 1, 2492042932330932, -23, etc

expressions possibles : 13 + 3928, 34 * 2 + 10 // 3 % 5, etc

LES BOOLEENS

valeurs possibles : True et False

expressions possibles : 0 == 0, 8+1 == 2 * 3, 13 >= a, etc.

LES CHAINES DE CARACTÈRES

valeurs possibles : 'Hello, world', 'la valeur de a est', '234' , etc.

LE TYPE DES INSTRUCTIONS (= LE TYPE DE « RIEN »)

valeurs possibles : None

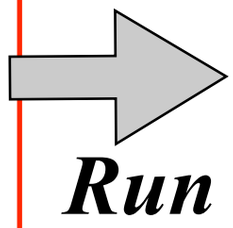
expressions possibles : print('hello'), x = a + b, etc

Conditionnelle : la prise de décision 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')
```




```
-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')
```

ligne vide, OVER, à toi !

```
-2 est négatif ou nul
```

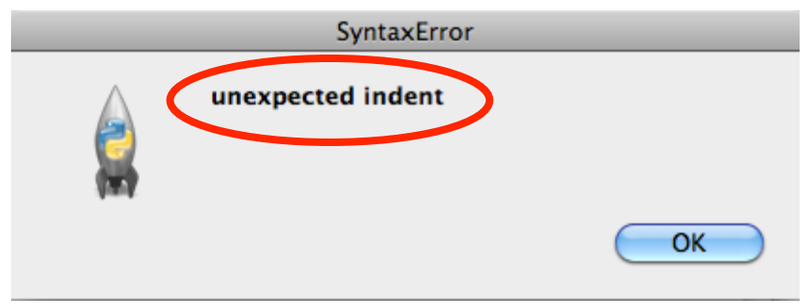
• La bonne distance à la marge d'une ligne (**indentation**) doit être respectée. Elle permet de structurer et de comprendre un programme :

```
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 (une affectation et un if).

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

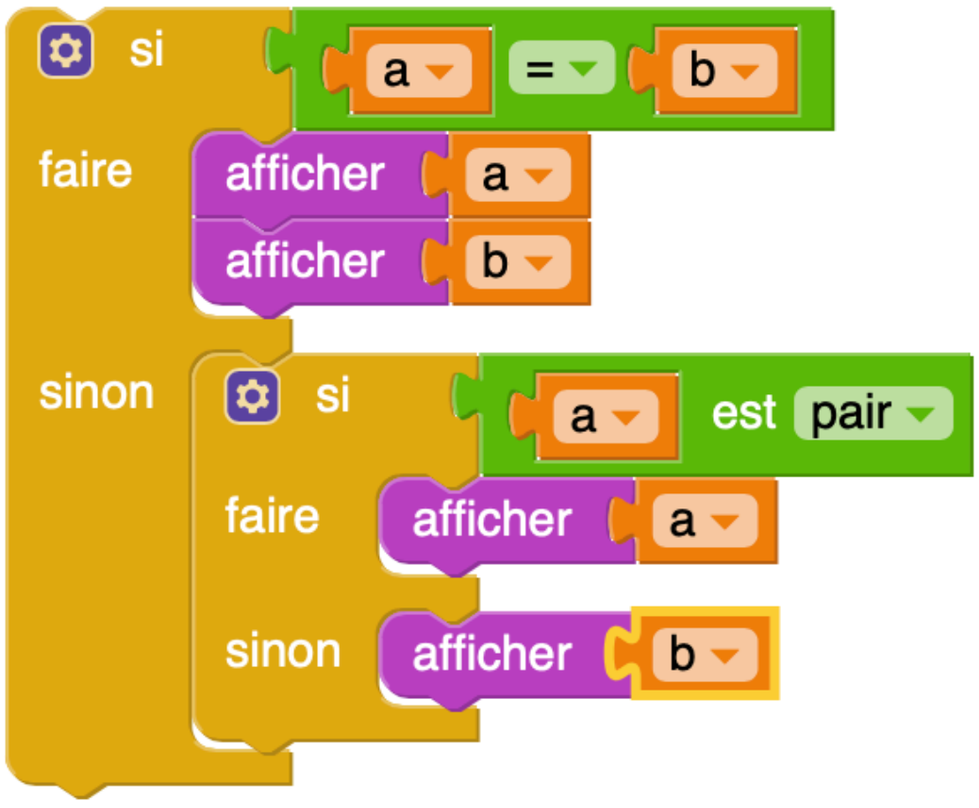
Run



• **UNE BONNE INDENTATION EST OBLIGATOIRE EN PYTHON.**

La structure en blocs d'un programme

Un programme est structuré en blocs imbriqués contenant des instructions, elles-mêmes formées de blocs d' expressions



Un programme **Pyblock**

```
if a == b :
⟷ print(a)
⟷ print(b)
else :
⟷ if a % 2 == 0 :
⟷ ⟷ print(a)
⟷ else :
⟷ ⟷ print(b)
```

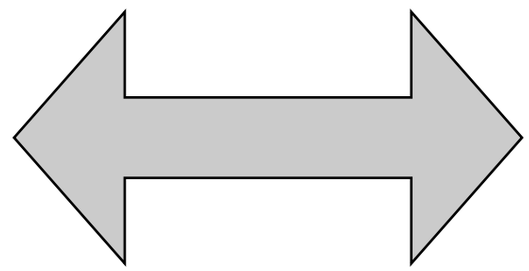
Code Python
automatiquement généré

http://mathematiques-medias.discipline.ac-lille.fr/PyBlock/

Les choix multiples

On peut faire des tests les uns à la suite des autres en utilisant la construction **if elif else**.

```
if a == b :
⟷ print(a)
⟷ print(b)
elif a % 2 == 0 :
⟷ print(a)
else :
⟷ print(b)
```



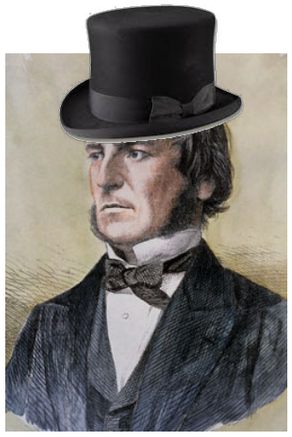
```
if a == b :
⟷ print(a)
⟷ print(b)
else :
⟷ if a % 2 == 0 :
⟷ ⟷ print(a)
⟷ else :
⟷ ⟷ print(b)
```

Forme générale d'un bloc if :

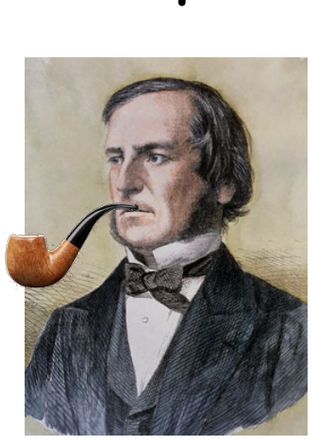
- 1 if et son sous-bloc
- 0, 1 ou plusieurs elif et leurs sous-blocs
- 0 ou 1 else et son sous-bloc

Les opérateurs booléens

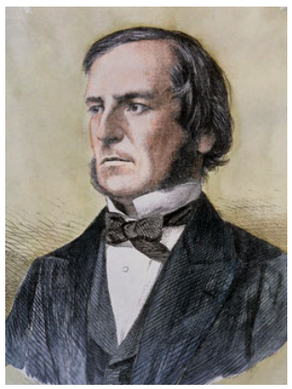
De même qu'avec les entiers on peut additionner, soustraire, multiplier…
il existe des opérations sur les booléens True et False



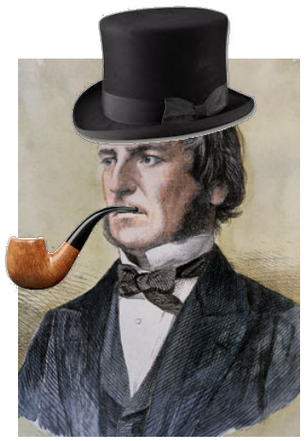
p



q

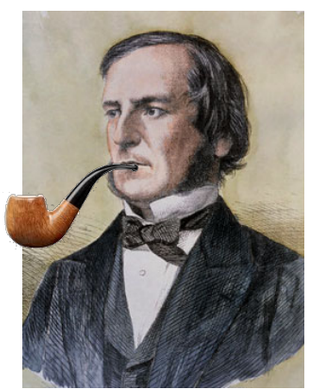


p
and
q



p
or
q

not p



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

and

or

p	True	False
not p	False	True

not

Opérateurs paresseux

En Python, and et or sont paresseux

```
>>> 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`
- Idem pour le mot-clé `or` :

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

car `True or ? == True`

- 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 !
- `!=` désigne la négation de `==`

```
if not a == b :
```
⇔
```
if a != b :
```

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 !

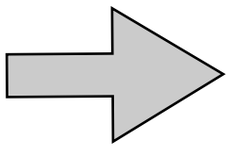
```
>>> gcd(18,12)                    #  je veux calculer un PGCD
NameError: name 'gcd' is not defined
>>> import fractions              #  importation du module fractions
>>> fractions.gcd(18,12)          #  mais le mot gcd reste inconnu !
6
>>> from fractions import gcd     #  importation du seul mot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




```
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
```

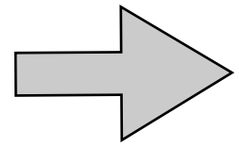
```
>>> f(5.2)
9.4
```

```
>>> 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

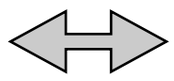



```
|-5| vaut 5
```

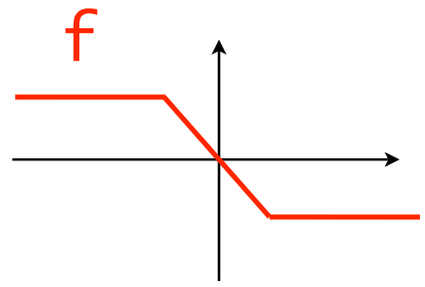
IMPORTANT : return provoque un échappement .

Après return, le reste du texte de la fonction est abandonné

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



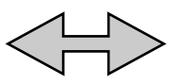
```
def f(x) :
    if x < -1 :
        return 1
    if x <= 1:
        return -x
    return -1
    return 0 # <- jamais exécuté!
```



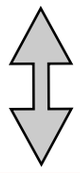
Comment savoir si deux entiers n'ont que 1 comme diviseur commun ?

IDLE

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



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



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

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

- Vous voyez qu'il existe différentes manières de coder une fonction.

Elles se distinguent par leur **efficacité**, mais aussi leur **élégance**.

Le jeu « pile ou face ? » en Python

Nous allons programmer le jeu « pile ou face »:

- l'humain fait un pronostic et le dit à l'ordinateur
- l'ordinateur « lance la pièce »
- l'ordinateur annonce le résultat

Il va nous falloir deux variables:

- la variable `pronostic` dont la valeur sera fixée par l'humain
- la variable `lancer` dont la valeur sera fixée par l'ordinateur

Demander une valeur à l'humain

On utilise la fonction prédéfinie **input**

exemple.py

```
nom = input('quel est ton nom, humain?')
print('enchanté,', nom)
```

Attention, la valeur retournée par input est une chaine de caractères
si on veut un nombre, il faut faire une conversion vers les integer

```
>>> '42' == 42
False
>>> int('42') == 42
True
```

exemple.py

```
age_texte = input('quel est ton age, humain?')
age = int(age_texte)
if age < 18 :
   print('gamin!')
```

Générer des entiers au hasard

- Il faudra 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]`.

…et une chaîne de caractères au hasard?

Il va falloir mettre en correspondance pile et face avec des nombres, par exemple

'pile' ⇔ 0 'face' ⇔ 1

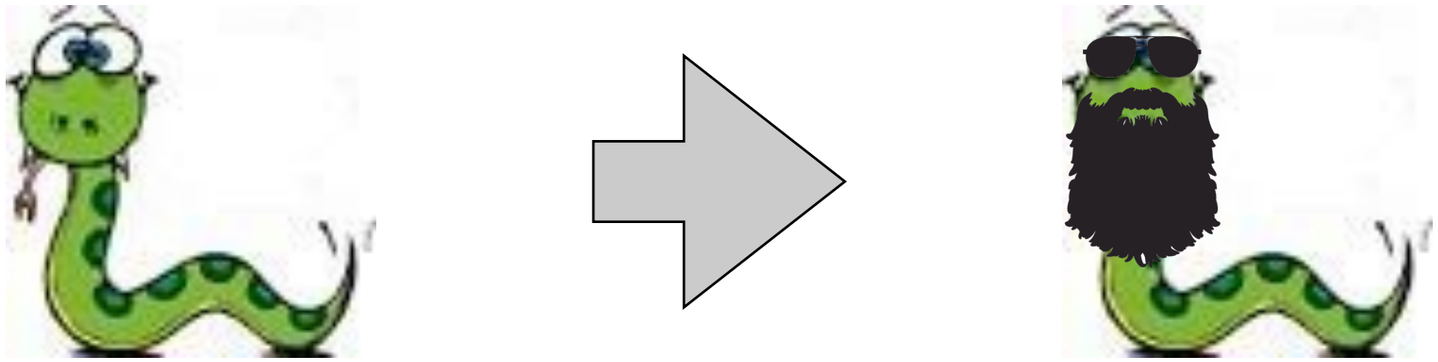
Le jeu « pile ou face ? », version 1

pile-ou-face.py

```
from random import randint

pronostic = input('pile ou face, humain?')
lancer = randint(0,1)
if pronostic == 'pile' and lancer == 0 :
   print('gagné!')
elif pronostic == 'face' and lancer == 1 :
   print('gagné!')
else :
   print('perdu!')
```

Le style Pythonesque

Ce programme marche est très bien, mais il manque de style



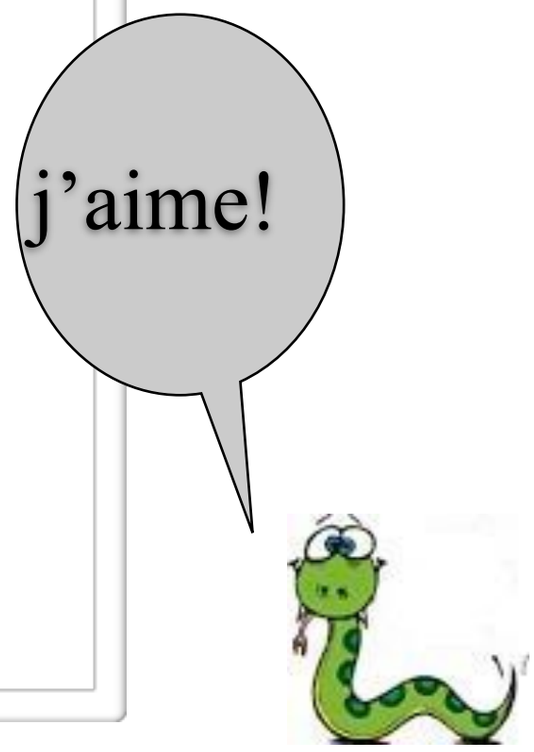
début du cours fin du cours

On a répété 2 fois print('gagné'); et puis les tests sont moches…
Et si dans lancer on mettait directement 'pile' ou 'face'?

Le jeu « pile ou face ? », version 2

pile-ou-face.py

```
from random import randint

pronostic = input('pile ou face, humain?')

if randint(0,1) == 0 :
   lancer = 'pile'
else :
   lancer = ‘face'

if pronostic == lancer :
   print('gagné!')
else :
   print('perdu!')
```



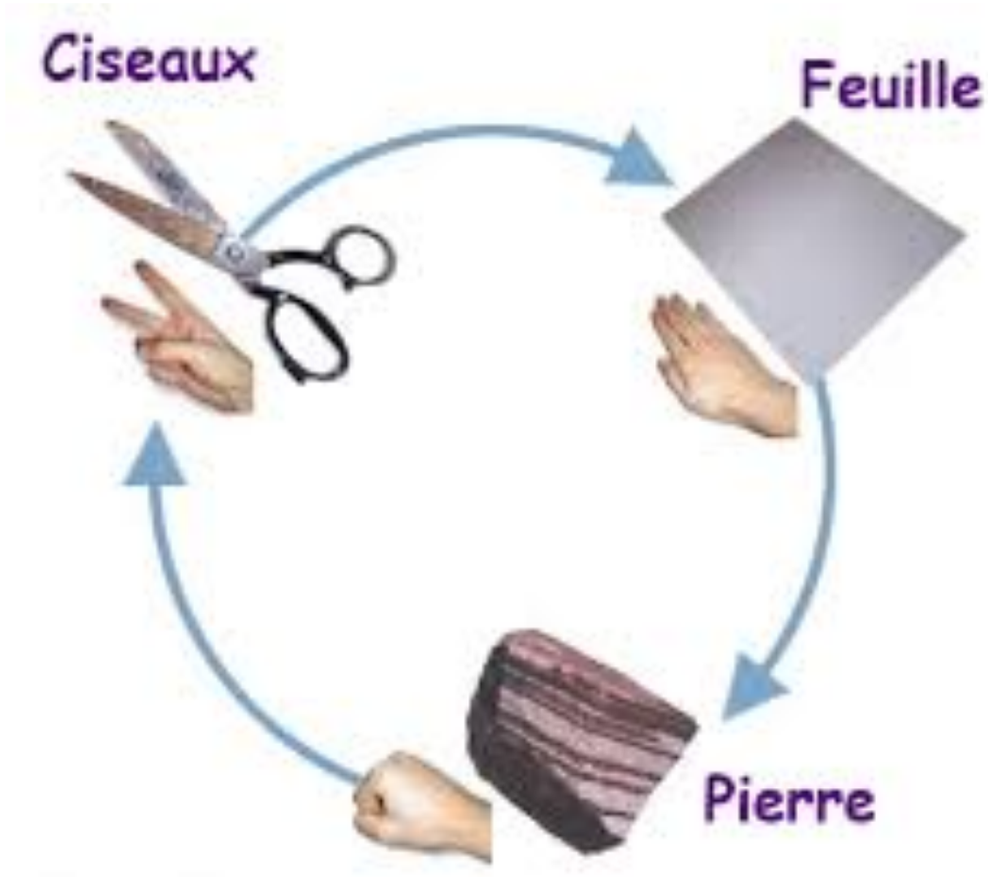
Le jeu « pile ou face ? », version 3

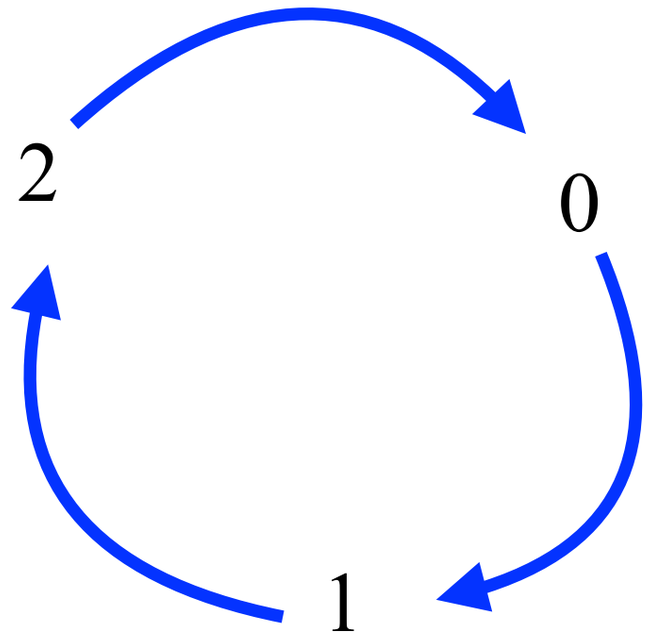
pile-ou-face.py

```
from random import randint

pronostic = input('pile ou face, humain?')

lancer = 'pile' if randint(0,1) == 0 else 'face'

if pronostic == lancer :
    print('gagné!')
else :
    print('perdu!')
```

expression conditionnelle

En TP : le jeu pierre-feuille-ciseaux





Indication: pour avoir du style, pensez aux modulos