Stage LIESSE Python/BD

Introduction à Python et prise en main de l'environnement

Matthieu Moy (Matthieu.Moy@imag.fr)

Ensimag, Grenoble INP

juin 2014

Sommaire

Sommaire

Sources utilisées

- Tutoriel officiel : `http://docs.python.org/3/tutorial/index.html`
- `http://www.korokithakis.net/tutorials/python/`
- `http://hebergement.u-psud.fr/iut-orsay/Pedagogie/MPHY/Python/courspython3.pdf`
- `http://www.korokithakis.net/tutorials/python/`
- Transparents de Olivier Richard, UJF (stage LIESSE mai 2013)

Python : premier contact

```
>>> 2 + 2
4
```

Python : premier contact

```
>>> 2 + 2
4
```

`>>>` le « prompt », ou « invite de commande »
$\Rightarrow$ « Bonjour cher utilisateur, que dois-je faire ? »

`2 + 2` instruction Python, entrée par l'utilisateur

`4` réponse de l'interpréteur Python après exécution.

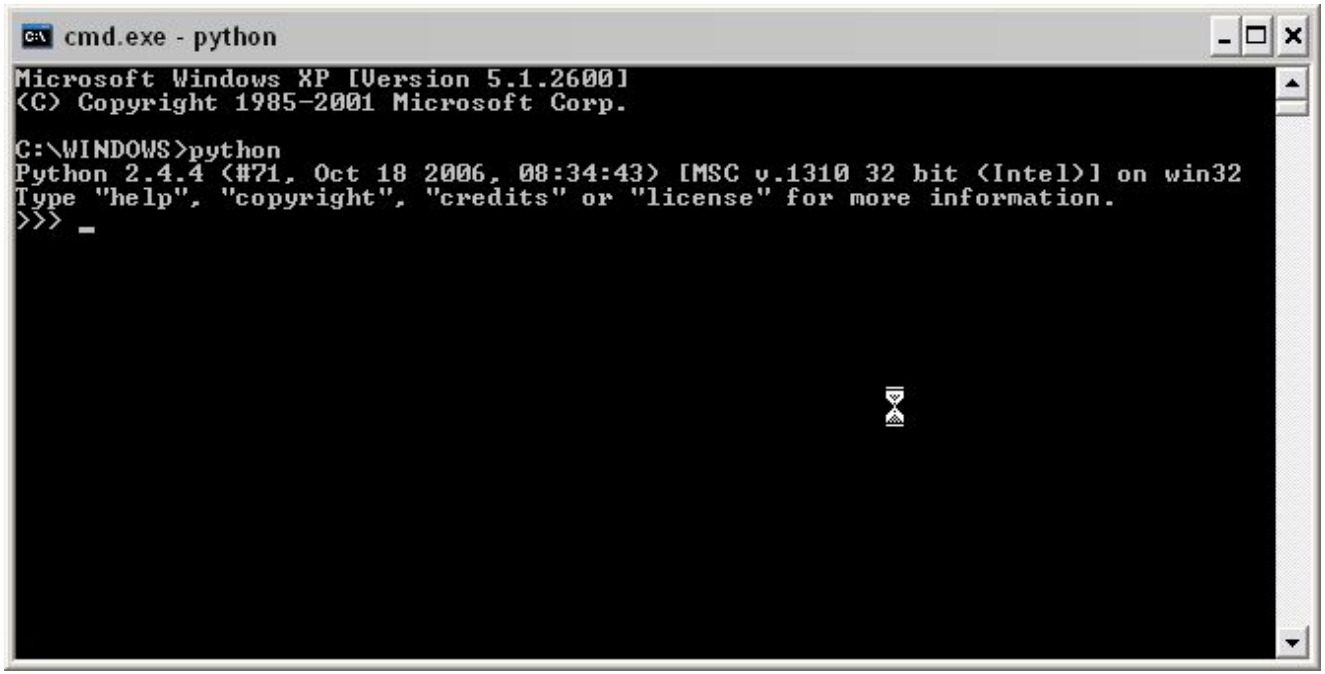
Mémoire : les variables

```
>>> x = 42
>>> x = x + 1
>>> x
43
>>> x = x + 1
>>> x
44
```

- `x = 42` : « x prend la valeur 42 »
  (mémorisé pour la session en cours)

Concretement, c'est quoi un interpréteur ?

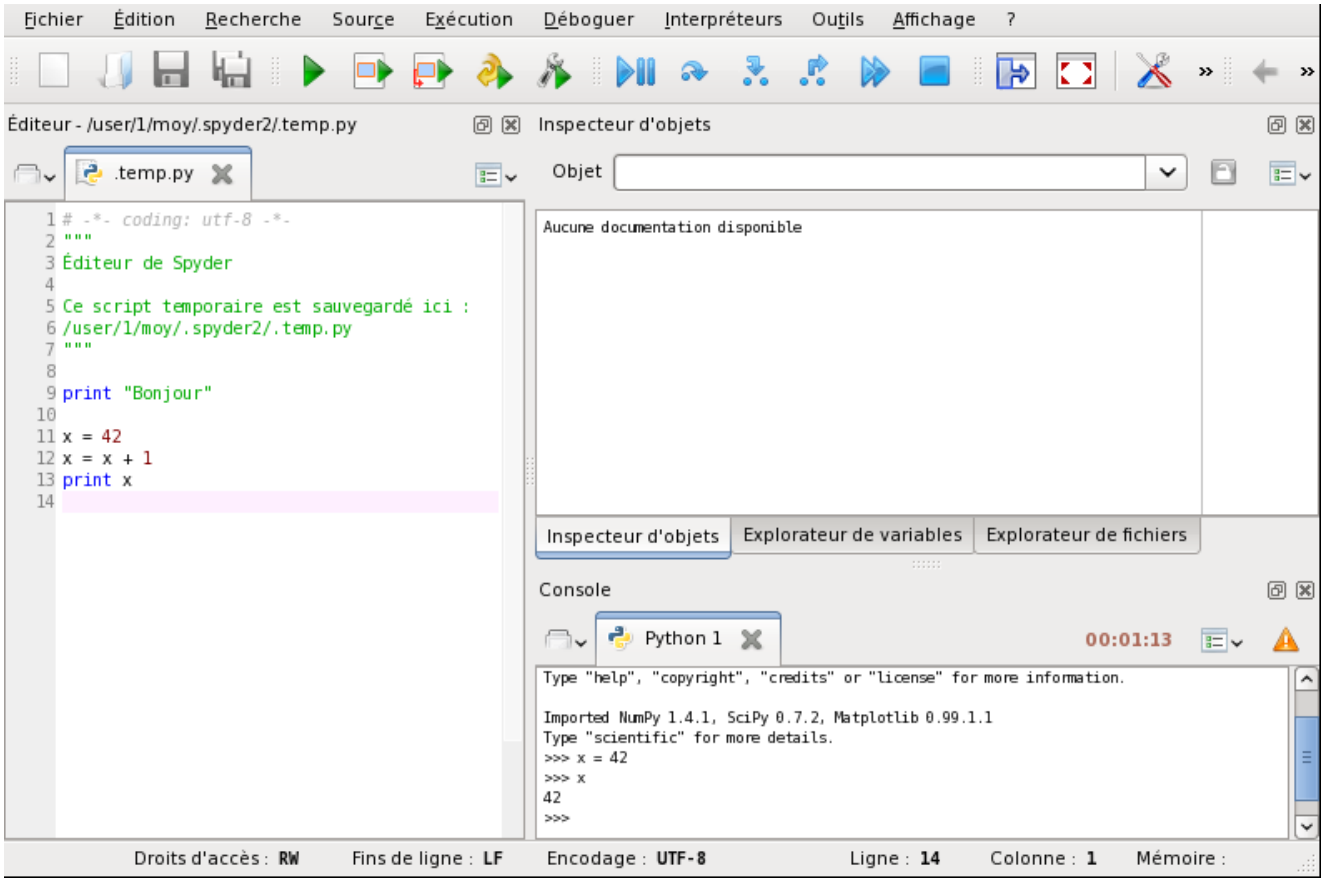
Option 1 : ligne de commande



- On entre du texte, l'interpréteur répond avec du texte
- Simple, mais peu convivial

Concretement, c'est quoi un interpréteur ?

Option 2 : L'IDE Spyder



- Toujours un interpréteur, toujours Python
- Plus d'interactivité

À vous de jouer !

- `http://www-verimag.imag.fr/~moy/cours/liesse/spyder/lancement/` (lien depuis EnsiWiki)
- Essayez quelques calculs simples, par exemple :
  - `2 + 2`
  - `2 - 2`
  - `2 + 3 * 4`
  - `(2 + 3) * 4`
  - `10 / 3`
  - `10 % 3`

Types de données de base

- Entiers : `0`, `-4`, `42`, `12345678900000`
- Flottants : `0.0`, `0.5`, `.5`, `-1.`, `1.2e+20`

   pas « `0,2` » !
- Chaînes de caractères :
  - `"Bonjour"`
  - `'Au revoir'`
  - `"""Rebonjour"""`
  - `"Je vous dis \"bonjour\""`
  - `'Je vous dis "au revoir"'`
- Booléens : `True` et `False`

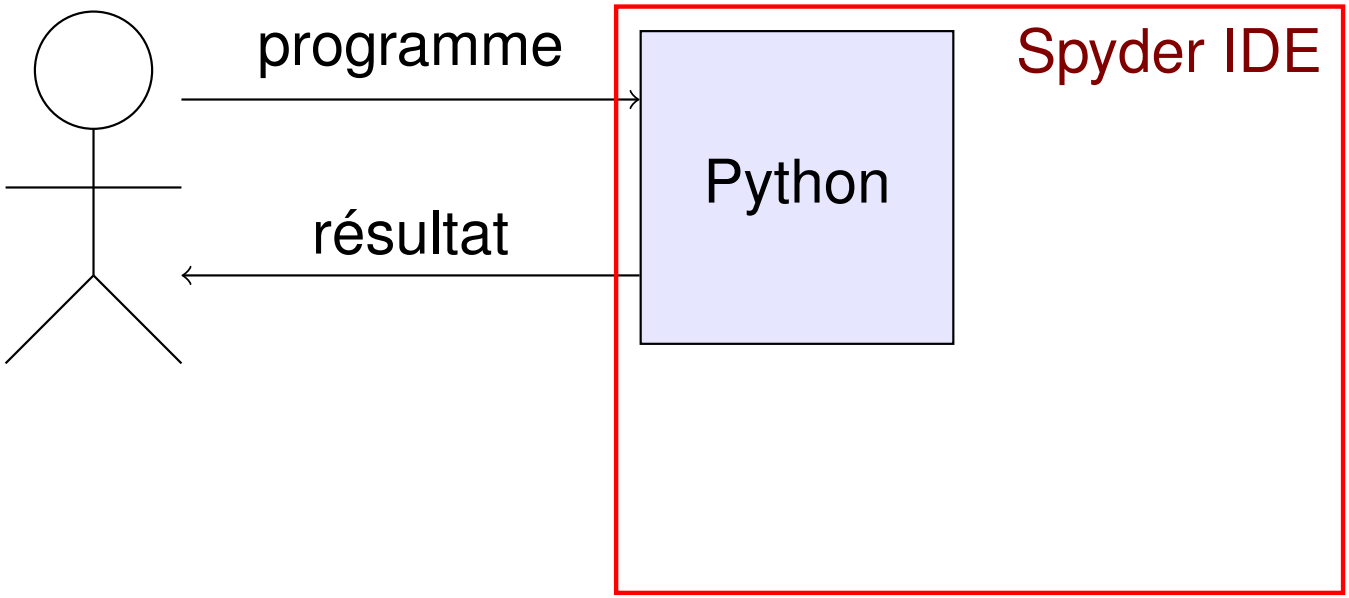
Petits exercices

- Évaluez les expressions suivantes :
  - `1.0 + 1`
  - `"Bonjour " + "a tous"`
  - `"Bonjour" + 10`
  - `2 == 2`
  - `2 == 3`
  - `(1 == 2) == (3 == 4)`

Sommaire

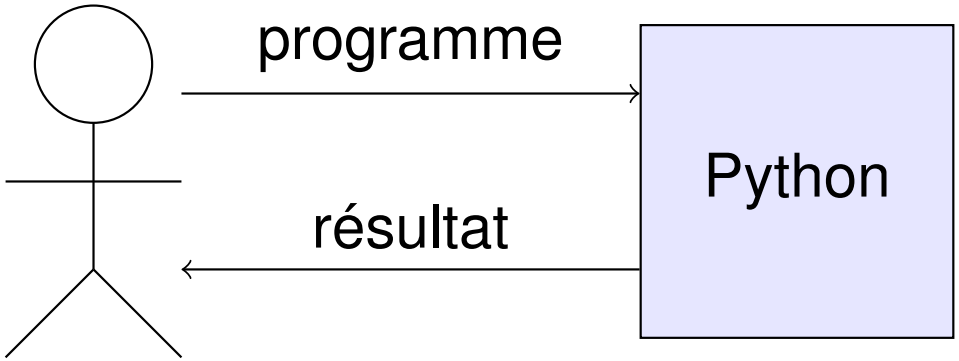
Principe de l'interpréteur

- Jusqu'ici :

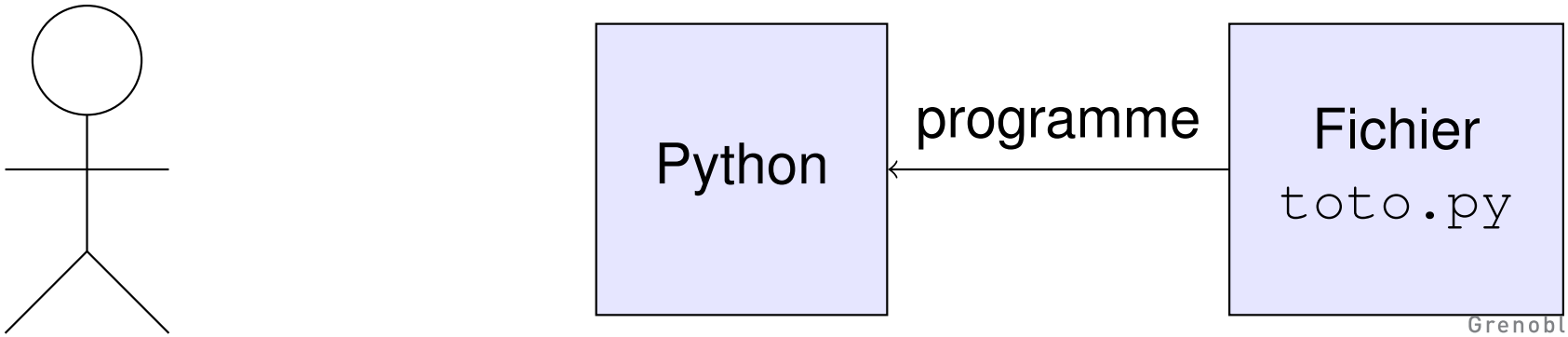


Principe de l'interpréteur

- Jusqu'ici :

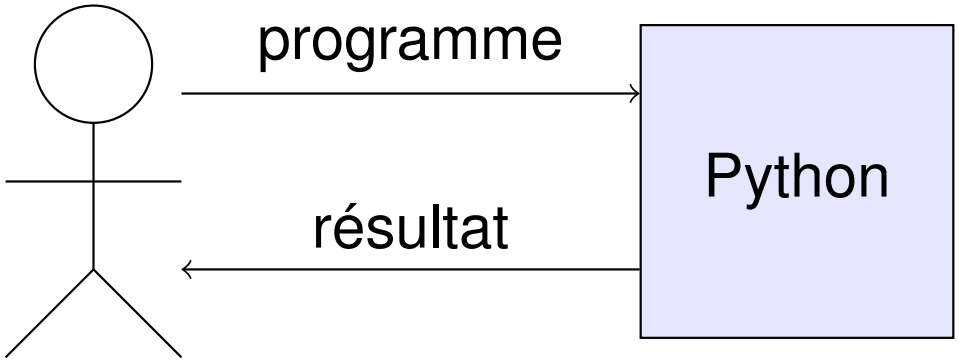


- Interprétation depuis un fichier :

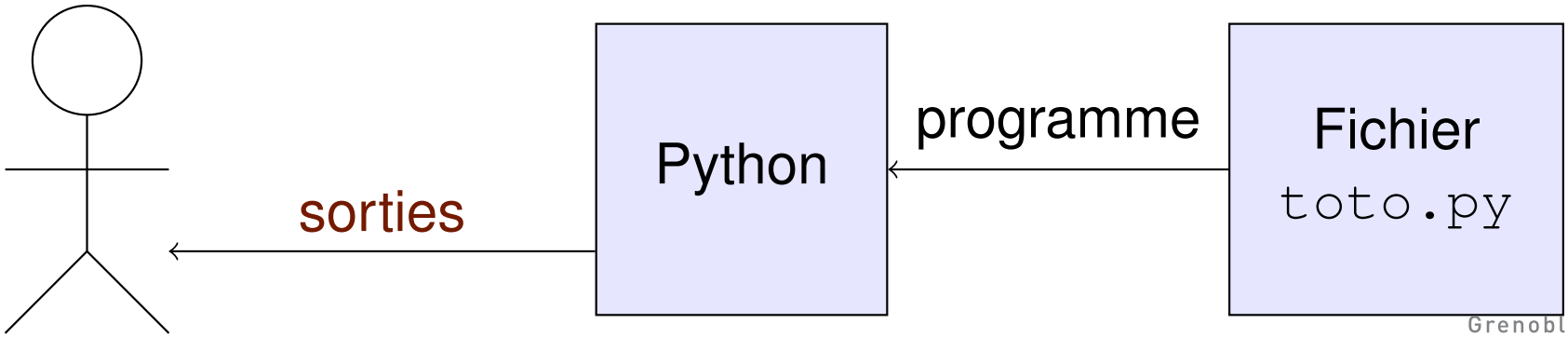


Principe de l'interpréteur

- Jusqu'ici :

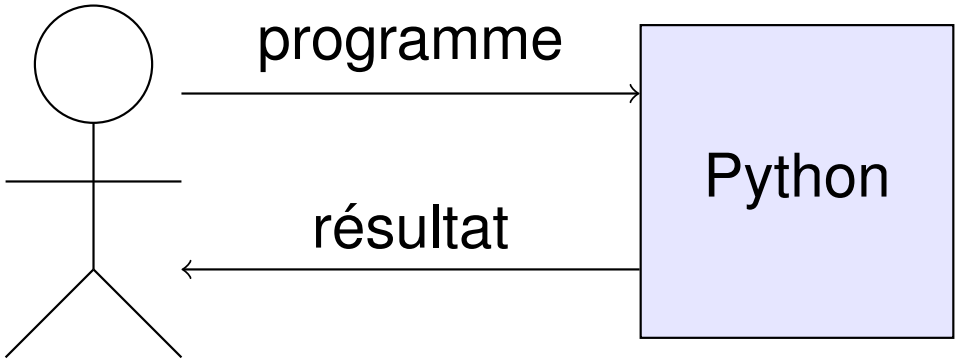


- Interprétation depuis un fichier :

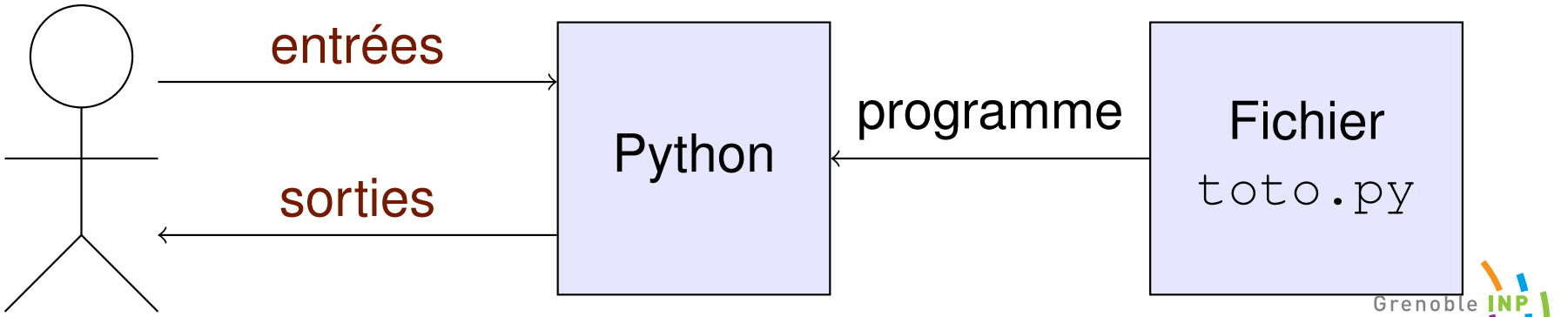


Principe de l'interpréteur

- Jusqu'ici :



- Interprétation depuis un fichier :



Premier programme avec entrées-sorties

- `http://www-verimag.imag.fr/~moy/cours/liesse/spyder/editeur/` (lien depuis EnsiWiki)
- Entrez le programme suivant dans l'éditeur :

```
x = input('Quel est votre nom ? ')
print('Bonjour, ' + x)
```

- Exécutez-le
- Ré-exécutez-le

`print` fait une sortie à l'écran, à ne pas confondre avec l'affichage du résultat qui est fait par défaut dans l'interpréteur.

Répéter une action : la boucle while

- Essayez le programme suivant :

```
x = 0
print("Je vais compter")
while x <= 10:
    print(x)
    x = x + 1
print("C'est fini")
```

L'indentation compte (même indentation avant `print(x)` et avant `x = x + 1`, de préférence 4 espaces)

- Essayez de remplacer `print(x)` par simplement `x`
  ⇒ ça n'affiche plus rien !
- Essayez d'indenter la ligne `print("C'est fini")` de 4 espaces
  ⇒ le `print` rentre dans la boucle.

Les commentaires

- Tout ce qui suit un # sur une ligne est ignoré par l'interpréteur :

```
x = 0  # initialisation de x
print("Je vais compter")
while x <= 10:     # blablabla
    print(x)       # encore du blabla
    x = x + 1
print("C'est fini")
```

- Très utile pour s'y retrouver dans le code

Les erreurs à l'exécution

- Mettez en commentaire ou supprimez l'initialisation de `x` :

```
# x = 0
print("Je vais compter")
while x <= 10:
    print(x)
    x = x + 1
print("C'est fini")
```

- Il peut se passer deux choses :
  - Notre éditeur de texte peut nous avertir que quelque chose ne va pas (les version récentes de Spyder le font, pas celle de l'Ensimag).
  - Si on exécute le programme, on obtient une erreur à l'exécution (après le premier affichage) :

```
Je vais compter
Traceback (most recent call last):
  File "/tmp/hello.py", line 9, in <module>
    while x <= 10:
NameError: name 'x' is not defined
```

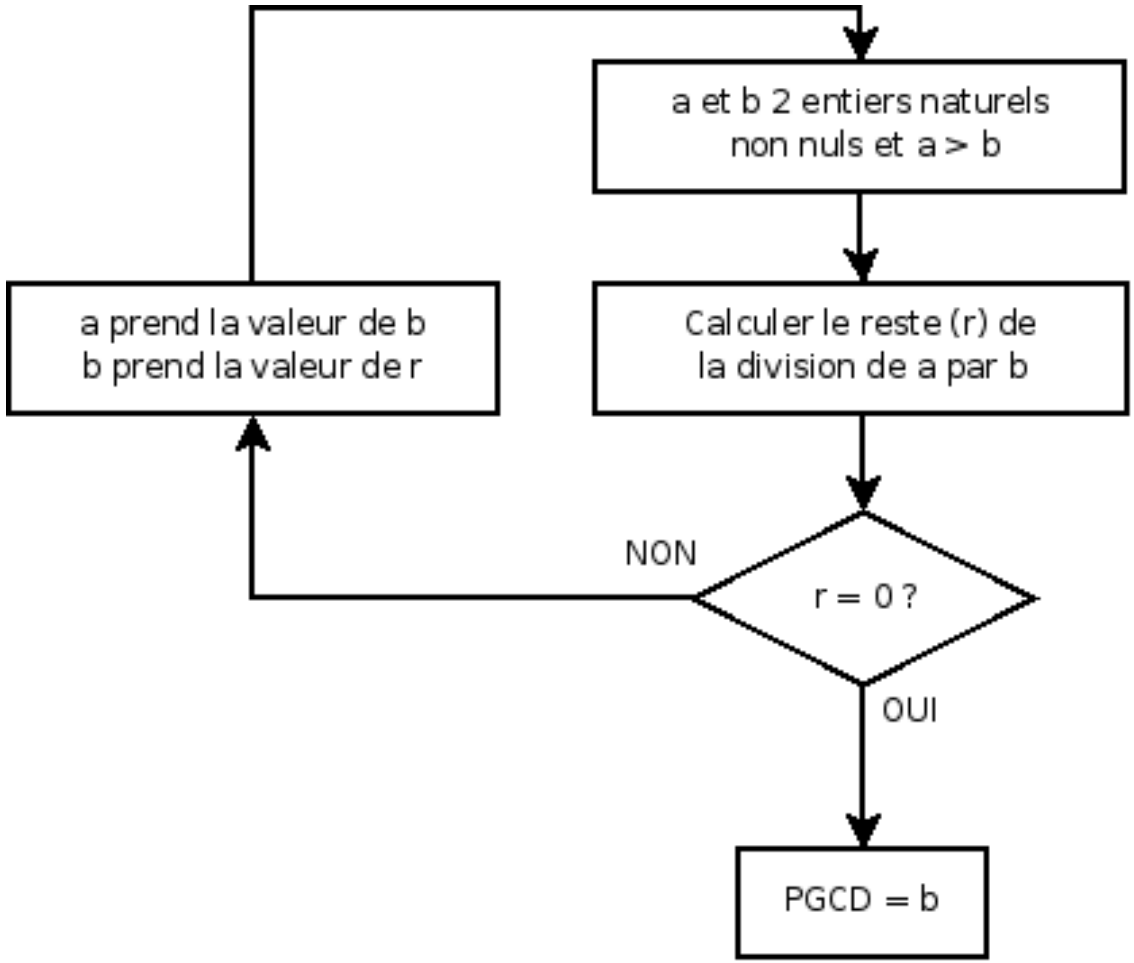
Les erreurs de syntaxe

- Supprimez maintenant les « : » de la boucle `while` :

```
x = 0
print("Je vais compter")
while x <= 10   # plus de :
    print(x)
    x = x + 1
print("C'est fini")
```

- Cette fois-ci, on obtient l'erreur avant le début de l'exécution (même `"Je vais compter"` n'est plus affiché).

Exercice : l'algorithme d'Euclide (PGCD)



http://commons.wikimedia.org/wiki/File:PGCD.png

Algorithme d'Euclide : version 1 (`euclide-1.py`)

```
a = int(input("Entrer a : "))
b = int(input("Entrer b : "))

if a < b:
    tmp = b; b = a; a = tmp

a_orig = a
b_orig = b
done = False

while not done:
    r = a % b
    if r == 0:
        done = True
    else:
        a = b
        b = r

print("Le PGCD de", a_orig, "et", b_orig, "est", b)
```

Algorithme d'Euclide : version 2 (`euclide-2.py`)

```
a = int(input("Entrer a : "))
b = int(input("Entrer b : "))

if a < b:
    tmp = b
    b = a
    a = tmp

a_orig = a
b_orig = b

while True:
    r = a % b
    if r == 0:
        break  # Sort de la boucle while
    a = b
    b = r

print("Le PGCD de", a_orig, "et", b_orig, "est", b)
```

Algorithme d'Euclide : version 3 (euclide-3.py)

```
#! /usr/bin/env python3

a = int(input("Entrer a : "))
b = int(input("Entrer b : "))

if a < b:
    b, a = a, b  # Echange a et b en une fois

a_orig = a
b_orig = b

while True:
    r = a % b
    if r == 0:
        break
    a, b = b, r

print("Le PGCD de", a_orig, "et", b_orig, "est", b)
```

Sommaire

Python : en quelques points

- Un langage script
- **Langage interprété** ($\neq$ langage compilé)
- **Typage Dynamique** ($\neq$ typage statique)
- **Indentation significative**
- **Orientation objet**
- Gestion automatique de la mémoire (*garbage collector*)

`http://fr.wikipedia.org/wiki/Python_(langage)`

Bibliothèque standard Python

- `http://docs.python.org/3/library/` : plus de 200 packages (approche *“batteries included”*)
  - Structures de données, manipulations de chaînes, ...
  - Manipulation de fichiers, bases de données, compression ...
  - Mathématiques (un peu dans la bibliothèque standard, beaucoup d’autres modules optionnels)
  - Internet (web, email ...)
  - Interfaces graphiques, multimedia ...
  - ...

Utilisateurs de Python

- Web : Google, Yahoo, Mozilla ...
- Calcul scientifique : LHC, NASA ...
- Langage de script pour étendre un logiciel (plugins) : Blender, vi, ...
- Des « success stories » : `http://brochure.getpython.info/`

Points forts, points faibles

- Dans la vraie vie :
  - Langage de haut niveau : on peut faire beaucoup avec peu de code
  - Typage dynamique $\Rightarrow$ lent et gourmand en mémoire (contournable)
  - Écosystème très fourni
  - Facile à apprendre, mais intéressant aussi pour des experts
- Pour la pédagogie :
  - Démarrage en douceur
  - Typage dynamique : discutable
  - Indentation obligatoire : vos étudiants seront obligés de rendre du code propre (ou presque)

Version 2.x vs 3.y

- Pourquoi est-ce important ?
  - Le code écrit en Python 2.7 ne marche pas toujours en 3.0
  - Certains modules et bibliothèques ont mis du temps à être portés en Python 3 (Spyder pour Python 3 n'est pas sorti officiellement)
  - Pour débuter, 2.7 est suffisant, mais autant démarrer avec Python 3
  - `http://docs.python.org/3/whatsnew/3.0.html`

Version 2.x vs 3.y

- Pourquoi est-ce important ?
  - Le code écrit en Python 2.7 ne marche pas toujours en 3.0
  - Certains modules et bibliothèques ont mis du temps à être portés en Python 3 (Spyder pour Python 3 n'est pas sorti officiellement)
  - Pour débuter, 2.7 est suffisant, mais autant démarrer avec Python 3
  - `http://docs.python.org/3/whatsnew/3.0.html`
- Faire du Python 3.0 avant l'heure

```
# au debut de chaque programme:
from __future__ import print_function
from __future__ import division
```

- Si on ne le fait pas :

```
>>> print(2/3, 42)
(0, 42)
```

Sommaire

Exécuter les exemples de code

- Télécharger et dé-zipper un fichier :

`http://www-verimag.imag.fr/~moy/cours/liesse/spyder/unzip/`

- Exécuter des portions de code les unes après les autres :

`http://www-verimag.imag.fr/~moy/cours/liesse/spyder/pas-a-pas/`

- Utilisation de l'inspecteur d'objets :

`http://www-verimag.imag.fr/~moy/cours/liesse/spyder/inspecteur/`

Les chaînes de caractères

- Les chaînes de caractères sont **non modifiable** (immutable)
- Opérations sur les chaînes par *fonction* ou *méthode*

```
a = "Ensimag"
len(a)   # 7
a.upper()   # ENSIMAG (nouvelle chaine)
a[0]     # 'E'
a[2:4]   # 'si'
```

Les entrées / sorties

- La saisie clavier :

```
# Lire une chaine
str = input("Entrez une chaine : ")
print(str)

# Lire un entier
# (lecture de chaine puis conversion)
nb = int(input("Entrez un nombre : "))
print(nb)
```

- Affichage :

```
print("toto")
print(42)
print("toto " + 42)
print("toto ", 42)
```

Listes en Python

- Listes = ensemble ordonné d'éléments :

```
a = ['spam', 'eggs', 100, 1234]
print(a)
# ['spam', 'eggs', 100, 1234]
print(a + ['python', 'eggs'])
# ['spam', 'eggs', 100, 1234, 'python', 'eggs']
```

- Tranche (*slice*) :

```
a[1]
# 'eggs'
a[1:2]
# ['eggs', 100]
a[1:-1]
# ['eggs', 100]
3 * a[:3] + ['Boo!']
# ['spam', 'eggs', 100, 'spam', 'eggs',
#  100, 'spam', 'eggs', 100, 'Boo!']
```

Fonctions disponibles sur les *Listes*

- *list.nom_fonction()* :

```
a = [66.25, 333, 333, 1, 1234.5]
print a.count(333), a.count(66.25), a.count('x')
# 2 1 0
a.insert(2, -1)
a.append(333)
print(a)
# [66.25, 333, -1, 333, 1, 1234.5, 333]
a.index(333)
# 1
a.remove(333)
print(a)
# [66.25, -1, 333, 1, 1234.5, 333]
a.reverse()
print(a)
# [333, 1234.5, 1, 333, -1, 66.25]
a.sort()
print(a)
# [-1, 1, 66.25, 333, 333, 1234.5]
```

Condition : if/then/else

- If :

```
x = 43
if x == 42:  # ne pas oublier le ':'
    print("x vaut 42")
    print("et pas autre chose")
# Fin de l'indentation = fin du if
print("Suite du programme")
```

- If/else

```
if x == 42:
    print("x vaut 42")
else:
    print("x vaut autre chose")
```

Condition : if/then/else

- If/elif/else :

```
if x == 42:
    print("x vaut 42")
elif x == 43:
    print("x vaut quarante trois")
else:
    print("x vaut autre chose")
```

Boucles

- While :

```
while x <= 42:
    x = x + 1
```

- For (en général, plus pratique à utiliser que while)

```
words = ['cat', 'window', 'defenestrate']
for w in words:
    print w, len(w)
```

Cas particulier avec les boucles for

- Énumérer les nombres de 0 a $N-1$:

```
range(10)
# Renvoie [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Parcourir les nombres de 0 a $N-1$:

```
for i in range(10):
    print(i)
# Affiche les nombres de 0 a 9 inclus
```

- Parcourir les nombres de M à $N-1$:

```
for i in range(7, 10):
    print(i)
# 7
# 8
# 9
```

Disposition du code

- Les fin de lignes sont importantes

```
x = 42   # OK
x =      # Interdit!
    42
```

- Coupage possible à l'intérieur des parenthèses :

```
x = (1 + 2 + 3 + 4 + 5 +   # OK
     6 + 7 + 8 + 9 + 10)
```

- Mettre plusieurs instructions sur une ligne (à éviter) : « ; »

```
x = 42; print(x)
```

Exercice : somme des N premiers entiers

- Écrire un programme qui calcule la somme des entiers de 0 à N (sans utiliser $N(N+1)/2$, ça serait de la triche !)

Exercice : somme des N premiers entiers

- Écrire un programme qui calcule la somme des entiers de 0 à N (sans utiliser $N(N+1)/2$, ça serait de la triche !)

```
n = int(input("Entrer N : "))

sum = 0
for i in range(n + 1):
    sum = sum + i

print("La somme des entiers de 0 a",
      n, "est :", sum)
```

Sommaire

Définition et appel de fonction

- Sous-programme sans valeur de retour :

```
def dire_bonjour(interlocuteur):
    print("Bonjour, " + interlocuteur)

dire_bonjour("Matthieu")
dire_bonjour("Ahmed")
```

- Renvoi de valeur :

```
def addition(a, b):
    return a + b

x = addition(42, 3)
x = addition(x, 1)
```

Exemple : factorielle

```
def fact(n):
    res = 1
    while n > 0:
        res = res * n
        n = n - 1
    return res

print(fact(3))
```

Exemple : factorielle récursive

- Fonction récursive = fonction qui se rappelle elle-même
- (Les détails cet après-midi)

```
def fact(n):
    if n <= 1:
        return 1
    else:
        return n * fact(n - 1)
```

Variables locales, variables globales (1/2)

- Une affectation crée une variable locale :

```
def x_egal_42():
    x = 42
    print("dans la fonction :", x)

x = 0
x_egal_42()
print("apres la fonction :", x)   # affiche 0
```

- Les variables globales sont visibles :

```
def affiche_x():
    print("dans la fonction :", x)

x = 42
affiche_x()
print("apres la fonction :", x)
```

Variables locales, variables globales (2/2)

- On peut modifier une variable globale avec `global` :

```
def x_egal_42():
    global x  # <-- Ici
    x = 42
    print("dans la fonction :", x)

x = 0
x_egal_42()
print("apres la fonction :", x) # affiche 42
```

Exercice : recherche de maximum dans une liste

- Le but :

```
>>> max([1, 2, 42])
42
>>> max([1, 42, 1])
42
>>> max([42, 1, 2])
42
```

- À vous de jouer !
- Nommer ses fichiers :

`http://www-verimag.imag.fr/~moy/cours/liesse/spyder/fichier/`

Solution : recherche de maximum dans une liste

```
def max(l):
    current_max = l[0]
    for elem in l:
        if elem > current_max:
            current_max = elem
    return current_max
```

Sommaire

Liens utiles

- Tutoriel officiel : `http://docs.python.org/3/tutorial/index.html`
- Antisèches : `http://www.cheat-sheets.org/saved-copy/PQRC-2.4-A4-latest.pdf`
- Documents divers pour enseignant CPGE `http://goo.gl/AVV5t`.
- Beginners' Guide
- Cours sur Python 3 de Bob CORDEAU.

Conclusion

- Python un représentant important des langages scripts dynamiques
- **Rapidité de développement**
- **Un très grand nombre de bibliothèques disponibles**

Sommaire

Exercice : découpage de liste

- Écrire une fonction qui prend en argument une liste, et renvoie deux listes : la première avec les éléments positifs ou nuls, la seconde avec les éléments négatifs.
- Aide : insertion en queue de liste = `list.append(elem)`

Découpage de liste : solution

```
def split(l):
    nonneg = []
    neg = []
    for e in l:
        if e >= 0:
            nonneg.append(e)
        else:
            neg.append(e)
    return nonneg, neg

l = [-4, 12, 42, 0, -12, 14]
print(split(l))
```

Exercice : Calcul de racine carrée par méthode de Newton

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{avec} \quad f(x) = x^2 - a$$

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n}$$

$$x_{n+1} = \frac{x_n + a/x_n}{2}$$

Racine carrée, solution 1

Nombre d'itérations fixe

```
import math  # Pour pouvoir utiliser math.sqrt()

def sqrt(a):
    g = 1.0
    for i in range(10):
        g = (g + (a / g)) / 2.0
        # decommenter pour voir la progression
        # print(g)
    return g

def test(x):
    print(sqrt(x), "devrait etre egal a", math.sqrt(x))

test(42.0)
test(4.0)
```

Racine carrée, solution 2

Contrôle de la précision

```
import math  # Pour pouvoir utiliser math.sqrt()

def sqrt(a):
    g = 1.0
    while abs(g ** 2 - a) > 0.0001:
        g = (g + (a / g)) / 2.0
        # decommenter pour voir la progression
        # print(g)
    return g

def test(x):
    print(sqrt(x), "devrait etre egal a", math.sqrt(x))

test(42.0)
test(4.0)
```