

Mise en garde - versions

Algorithmique et programmation 1 - Cours 4

Premiers pas avec Python

L1 M-I-SPI – Université de Lorraine
Marie Duflot-Kremer
avec l'aide des collègues de Nancy et Metz

Transparents disponibles sur la plateforme de cours en ligne

- Ce cours a été conçu pour **Python 3**
- Pour programmer sur votre ordinateur, vérifiez bien la version de Python disponible
- Le cas échéant installer une version 3.x
- Quelques différences notables existent entre les versions 2.7 et 3

Présentation

- Langage **simple d'utilisation** :
 - rapide à apprendre, rapide à programmer
 - ~> permet de se concentrer plus sur l'algorithme
- utilisable en mode **interactif**,
- **généraliste** : couvre presque tous les domaines d'utilisation,
- utilisé dans certains lycées, dans les classes prépa,
- langage **interprété** : une seule commande pour exécuter le programme
- utilisé par Google pour ses sites web,
- permet de faire du calcul scientifique (sage/numpy),
- inclus dans de grands logiciels comme langage de scripts (ex : Maya 3D),
- première version publique, février 1991, Python 3 sorti en 2008.

Prise en main

- Mode interactif :
 - dans un terminal,
 - on lance Python :
 - ~> `python3` + Entrée
 - on saisit ses commandes et on voit les affichages dans la même fenêtre.
- Dans un fichier :
 - on utilise un éditeur de texte pour écrire et sauvegarder le programme,
 - puis on l'exécute.

Mode interactif

Invite du shell

on lance Python3

```
C'est a vous $ python3
Python 3.3.2 (v3.3.2 :d047928ae3f6, May 13 2013, 13 :52 :24)
[ GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Bonjour!")
Bonjour!
>>> x=13
>>> print("Au revoir",x)
Au revoir 13
>>> exit()
C'est a vous $
```

Instruction Python

Invite de Python

Instruction de sortie

Affichage

Avec un éditeur

- On tape le programme dans un éditeur de texte intelligent,
- on le sauvegarde dans un fichier avec l'extension `.py`,
- on l'exécute
 - soit en tapant `python3 monprogramme.py` dans un terminal
 - soit directement dans l'éditeur s'il le permet

Pourquoi l'extension `.py` ?

- nous aide à retrouver nos programmes
- permet à l'éditeur de texte de reconnaître le langage
 - il peut aider à l'indentation
 - il fait de la coloration syntaxique

Indentation

- Dans tous les langages : augmente la lisibilité
- En Python : sert à délimiter les blocs d'instructions

```
# bout d'algorithme
Bloc
|  Instruction1
|  Instruction2
Finbloc
Instruction3
```

```
# bout de programme
Bloc :
    Instruction1
    Instruction2
Instruction3
```

- Pas de balise de fin de bloc
 - Si on n'indente pas l'ordinateur ne sait pas où finit le bloc
~> erreur
- Voir exemples de conditionnelles et boucles

Les variables en Python

Ce qui est possible

Pas tout à fait comme pour l'algorithme :

- variable définie au moment où on lui donne une valeur,
- possible d'affecter n'importe quoi à une variable.

MAIS c'est source d'erreur et donc :

Dans ce cours

- déclaration des variables (en commentaire, au début)
- on affecte toujours à une variable une valeur du type déclaré

Types en Python

Types de base en Python :

- `int` : entier
- `float` : (valeur approchée d'un) réel
- `str` : chaîne de caractères
- `bool` : booléen
- et quelques autres que l'on verra plus tard si besoin
- pas de type spécial caractère ni entier naturel

Attention

En Python (et dans beaucoup d'autres langages) les nombres non entiers s'écrivent avec un point, par exemple 12.3. La virgule est réservée comme séparateur.

Déclaration des variables - exemple

```
# Programme declare.py
# Declare juste des variables
# Variables
# tas1, tas2 : int
# moy : float
# mot_avant, mot_apres : str
# fini, encore : bool
```

Règles pour les noms de variables :

- lettres non accentuées et chiffres,
- commence par une lettre,
- signe `_` (souligné) pour remplacer l'espace,
- choisir des noms explicites pour les variables,
- en général, on préfère les lettres minuscules.

Affectation

Notation

- `←` se traduit par `=`
- on note donc `x = 12`

Attention : notation non symétrique

```
x = 13
y = 2
x = y
y = x
```

```
x = 13
y = 2
y = x
x = y
```

Valeurs à la fin :

- `x` vaut et `y` vaut • `x` vaut et `y` vaut

Variables - règles et dangers

- Pas d'accents dans les noms de variables,
- certains mots clefs réservés : *if, and, else,...*
 `~>` ne peuvent pas être utilisés comme variables
- ne pas affecter n'importe quoi à n'importe quelle variable
 - ... même si l'interpréteur ne génère pas d'erreur
- Si on se trompe de nom, on crée une nouvelle variable

```
lim = 12                                      # initialise la variable lim à 12
lim = 11                                      # modifie bien la valeur de lim
limm = 10                                    # crée une variable limm, initialisée à 10
```

- **Attention** : pas de message d'erreur pour nous prévenir

Opérateurs - entiers

- +, -, * : addition, soustraction, multiplication
- //, % : quotient, reste de la division entière
- / : division réelle (**attention** : résultat de type float)
- ** : puissance
- ==, <, >, <=, >=, != : comparaisons
 - == : test d'égalité, ne pas confondre avec l'affectation
 - != signifie \neq

Opérateurs - flottants

- +, -, *, /, ** : idem entiers
- // : quotient de la division, mais de type float (ex : 3.0)
- % : le reste de la division entière est aussi un réel
- ==, <, >, <=, >=, != : comparaisons
 - **Attention** : sur des nombres arrondis, l'égalité/l'inégalité n'a plus vraiment de sens
 - on testera si la valeur absolue de leur différence est assez petite

Attention aux arrondis

```
>>> 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1==1
>>> 1+0.0000000000000001==1
>>> 1+0.0000000000000001==1
```

Opérateurs - priorités

Comme à l'école :

- D'abord la puissance,
 - puis multiplication et division,
 - puis addition et soustraction,
 - pour deux opérateurs de même priorité on commence par la gauche,
 - les parenthèses permettent de choisir l'ordre d'évaluation,
- ~> si on n'est pas sûr de l'ordre... on ajoute des parenthèses,
- ~> pour rendre l'expression plus claire... on ajoute des parenthèses.

Opérateurs - exemples

```
>>> 2 + 3 * 4                                # facile
>>> 3 * 1 ** 4                               # avec puissance
>>> (3 * 1) ** 4                             # on force l'ordre
>>> 4 / 3 * 3
>>> 4 // 3 * 3                                #
>>> 3 * 4 // 3
>>> 3 * (4 // 3)
```

Opérateurs - booléens

- deux valeurs : **True** et **False**
- **and** : et logique
- **or** : ou logique
- **not** : négation logique
- **not** prioritaire sur les deux autres
- si pas de priorité ni parenthèses, évalués de gauche à droite

Quelques exemples

```
>>> x=True
>>> y=False
>>> x and y

>>> not x and y
>>> not (x and y)
```

Opérateurs - exemples (2)

```
>>> toto="schtroumpf"
>>> print(toto[1])

>>> print(toto[9], toto[5], toto[6], toto[4])

>>> tata="12"
>>> n=3
>>> print(tata+n)

>>> print(int(tata)+n)
>>> print(int(toto))
```

Opérateurs - chaînes de caractères

- Délimitées par des apostrophes 'blabla' ou des guillemets "bilibli"
- \n impose un retour à la ligne
- toto[i] désigne le i+1^{ème} caractère de la chaîne toto
- ~> **Attention**, caractères numérotés à partir de 0!!
- len() donne la longueur d'une chaîne
- + : concaténation
- int() et float() convertissent une chaîne en nombre quand c'est possible

Et si on en veut plus

Doit-on tout reprogrammer avec ces opérateurs ?

- non
- beaucoup de choses existent déjà
- possibilité de charger des modules contenant plein de fonctions intéressantes
 - interfaces graphiques
 - transformation d'images
 - cryptographie
 - bases de données
 - réseau
 - conception de jeux video
 - ... et beaucoup d'autres.
- Se chargent dans un programme avec la commande **import**

~> `>>> import turtle`

Manipuler les types

```
# Variables
# a, b, c, somme : int

a = 4
b = 6
c = 7
somme = a+b+c
somme = somme / 3
print("La moyenne vaut",somme)
```

```
# Variables
# a, b, c, somme : int
#
a = 4
b = 6
c = 7
somme = a+b+c
```

- Exécuté sans erreur par Python ?
- Acceptable en AP1 ?
- Pourquoi ?
- Solution ?

Affichage

Se fait en Python 3 avec la fonction `print()`

- affiche un ou plusieurs arguments, séparés par une espace
 - affiche les constantes comme les valeurs des variables
- ↪ si `x` et `y` contiennent respectivement 12.3 et 2.0,
l'instruction `print("x", x, 1+2, 12.3, "x+3", x-2*y,"x"+"3")`
va afficher

Remarque

Comme on l'a vu, en mode interactif, si on tape une expression, Python affiche le résultat :

```
>>> x
True
>>> 2 ** 3
8
```

Cela ne fonctionne pas pour un programme dans un fichier !

Affichage - exemples

Par défaut,

- les expressions à afficher sont séparées par une espace,
- le tout est terminé par un retour à la ligne,

mais on peut le changer :

```
>>> print(toto[9], toto[5], toto[6], toto[4])
f o u r
>>> print(toto[9], toto[5], toto[6], toto[4], sep=" ")
>>> print(toto[9], toto[5], toto[6], sep="==", end=" et c'est fini \n")
```

Saisie

Se fait en Python 3 avec la fonction `input()`

- C'est une fonction, donc nécessite des parenthèses
 - ↪ `x = input()`
- On peut préciser un message à afficher
 - ↪ `y = input("Quel est votre prénom ? ")`
- Par défaut, récupère une chaîne de caractères
 - Mais on peut la convertir
 - ↪ `z = int(input("Quel age avez-vous ? "))`
 - ↪ `t = float(input("Votre moyenne au bac ? "))`
 - Si la valeur entrée n'a pas le bon type, génère une erreur

Ca ne marche pas!!!

Trois principaux types d'erreur :

- erreur de syntaxe :
 - le programme ne se lance pas, on a un message d'erreur
 - on n'a pas respecté les règles d'écriture d'un programme
 - voir dans le message où est l'erreur... et la corriger
- erreur sémantique :
 - pas de message d'erreur, mais résultat erroné
 - l'ordinateur fait ce qu'on lui demande...
 - ... mais on ne lui demande pas la bonne chose!!
 - revoir l'algorithme/le programme
- erreur à l'exécution
 - le programme commence à s'exécuter, puis rencontre un problème
 - voir dans le message d'où vient l'erreur... et la corriger

Ca ne marche pas - exemples

Erreur de syntaxe :

```
# programme test1
# 14 aout 2013
# Variables
# x : int

x = 3
print x)
```

On lance l'exécution :

```
C'est a vous $ python3 test1.py
File "test1.py", line 7
    print x)
          ^
SyntaxError : invalid syntax
```

Erreur sémantique :

```
# programme test2
# 14 aout 2013
# Variables
# x : int

x=int(input("Entrez un entier : "))
print("Valeur absolue : ", -x)
```

On l'exécute :

```
C'est a vous $ python3 test2.py
Entrez un entier : -3
Valeur absolue : 3
```

Et encore :

```
C'est a vous $ python3 test2.py
Entrez un entier : 4
Valeur absolue : -4
```

Ca ne marche pas - exemples (2)

Rappel : tests en Python

Erreur à l'exécution :

Le programme :

```
# programme test3
# 14 aout 2013
# Variables
# x : int

x=int(input("Entrez un entier x : "))
print("4/x vaut",4/x)
```

On l'exécute :

```
C'est a vous $ python3 test3.py
Entrez un entier x : 2
4/x vaut 2.0
```

```
C'est a vous $ python3 test3.py
Entrez un entier x : 0
Traceback (most recent call last) :
File "test3.py", line 8, in <module>
    print("4/x vaut",4/x)
ZeroDivisionError : division by zero
```

... mais pas pour tous!

Fonctionne pour un exemple...

- Pas les mêmes opérateurs,
- signe `=` réservé pour l'affectation,
- n'utilise que des caractères standards et donc pas `≤`, `≠`, `≥`.

Pseudo code	Python
<code>x = 3</code>	
<code>y ≤ 5</code>	
<code>(x ≠ 2) ou (y < 3)</code>	
<code>non (a et b)</code>	

Instruction if en Python

Syntaxe :

```
if condition:
    Instructions
```

Ce qui change :

- Si devient
- Alors devient
-

↪ c'est bien l'indentation qui dit quand l'instruction conditionnelle se termine

Indentation

Voici deux programmes :

```
if x>=2:
    print("x est grand")
    print("c'est fini")
print("au revoir")
```

```
if x>=2:
    print("x est grand")
print("c'est fini")
print("au revoir")
```

Qu'affichent-ils si au départ x vaut

4 ?

•

•

Même question si au départ x vaut -2.

•

•

Instruction if... else en Python

Syntaxe :

```
if condition:
    InstructionsA
else:
    InstructionsB
```

Ce qui change :

-
- le reste est pareil que le if

Indentation (2)

```
if x>=2:
    print("x est grand")
else:
    print("x est petit")
print("c'est fini")
```

```
if x>=2:
    print("x est grand")
print("c'est fini")
else:
    print("x est petit")
print("c'est fini")
```

```
if x>=2:
    print("x est grand")
else:
    print("x est petit")
print("c'est fini")
```

Qu'affichent ces programmes si au départ x vaut 4

•

•

•

Même question si au départ x vaut -2

•

•

•

Elif

```
x = int(input(" Combien avez-vous d'ordinateurs ? "))
if x < 0:
    print(" Menteur-euse !")
elif x == 0:
    print(" Allez en salle info")
elif x <= 2:
    print(" OK")
else:
    print(" Vous êtes un geek")
```

Intérêt

- `elif` remplace `else if`
- un seul niveau d'indentation
- permet de simuler l'instruction Selon... Finselon

Fonctions en Python

- Assez semblable au pseudo langage
- mot clef `def` pour dire que c'est une définition de fonction
- paramètres entre parenthèses (types déclarés après, en commentaire)
- deux points en fin de ligne
- corps de fonction indenté (quand l'indentation s'arrête, la fonction est finie)
- mot clef `return` pour retourner une valeur et quitter la fonction

```
def val_abs(x):
    """ param : x : int, résultat : int
    Calcule et retourne la valeur absolue d'un entier"""
    if x < 0:
        return -x
    else:
        return x
```

Fonctions en Python - documentation

- Les fonctions sont faites pour êtres réutilisées
- Important de décrire ce que fait une fonction
 - paramètres attendus, ce qu'ils représentent et leur type
 - résultat retourné (son type)
 - à quoi sert la fonction (ce qu'elle calcule)
- en Python,
 - juste après la signature de la fonction,
 - délimitée par des triples guillemets
 - en mode interactif `help(toto)` donne la doc de toto.

```
def moyenne(n1, n2, n3, c1, c2, c3):
    """ param : n1, n2, n3 : int, c1, c2, c3 : float, résultat : float
    Calcule et retourne la moyenne de trois notes n1, n2 et n3
    avec coefficients respectifs c1, c2 et c3"""
    return (c1 * n1 + c2 * n2 + c3 * n3) / (c1 + c2 + c3)
# ne pas oublier de diviser par la somme des coefficients
```

Sources



G.Swinnen, *Apprendre à programmer avec Python 3 (3ème édition)*, Eyrolles, 2012, Disponible en ligne à l'adresse <http://inforef.be/swi/python.htm>.



M. Lutz and D. Ascher, *Introduction à Python*, O'Reilly, 2000.