LANGAGE DE PROGRAMMATION PYTHON

ASSOHOUN E. Stanislas

- *UFR MI (UFHB)*

PLAN

- **CHI – DESCRIPTION GÉNÉRALE DU LANGAGE**
- **CH2 – FONCTIONS ET MODULES**
- **CH3 – STRUCTURES DE DONNEES ET CLASSES**

Introduction

Ce support propose une initiation au langage Python. Il est destiné à un public étant familier avec au moins un langage de programmation, si possible orienté objet, et être familier avec les notions d'objet. Ce support n'est pas un cours de programmation, ni un cours complet sur le langage Python, il ne représente qu'une initiation à différentes choses que l'on peut faire avec Python. Toute remarque concernant ce support est toujours la bienvenue !

CHI

DESCRIPTION GENERALE DU LANGAGE PYTHON

INTRODUCTION

- Un peu d'histoire (1/2)

Python est un langage de programmation objet interprété. Son origine est le langage de script du système d'exploitation *Amoeba* (1990). Il a été développé par Guido Von Rossum au CWI, à l'Université d'Amsterdam et nommé par rapport au *Monthy Python's Flying Circus (une compagnie artistique d'Amsterdan)*.

Depuis, Python est devenu un langage de programmation généraliste.

INTRODUCTION

- Un peu d'histoire (2/2)

Un atout indéniable est sa disponibilité sur la grande majorité des plates-formes courantes (BeOS, Mac OS X, Unix, Windows). Python est un langage *open source* supporté, développé et utilisé par une large communauté : 300 000 utilisateurs et plus de 500 000 téléchargements par an.

I- CARACTERISTIQUES (1/2)

- très clair, syntaxe lisible
- de fortes capacités d'introspection
- expression naturelle de code procédural
- modularité totale, avec paquets hiérarchiques
- gestion des exceptions
- nombreux types de base : listes, chaînes, tables de hachage
- gestion transparente de la mémoire par référence comptage
- manipulation des références, pas des pointeurs
- lambda-calcul, fonctions anonymes …
- orienté objet intuitive : classes, héritage, exceptions

I- CARACTERISTIQUES (2/2)

- typage dynamique
- vastes bibliothèques standard et des modules de tierces parties
- extensions et modules en C, C ++ (ou Java pour Jython, ou langages .NET pour IronPython)
- intégrable dans les applications comme scripts d'interfaçage.

II- MODES D'EXECUTION

Un langage interprété

- Le programme python en mode interactif
- Le programme ipython : un shell Python évolué
- La fonction eval pour évaluer du code dans python

Un langage compilé

- Compilation en bytecode de machine virtuelle (comme Java)
- Les fichiers .pyc sont générés dynamiquement

III- STRUCTURE DU CODE PYTHON

Syntaxe d'un code python

La syntaxe d'un code python est composé d'instructions, de blocs, de modules, de packages

- ✓ Bloc : ensemble d'instructions
- ✓ Module : ensemble de blocs
- ✓ Packages: ensemble de modules

Régles

- Une instruction par ligne
- pas de séparateurs

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (1/3)

- Constantes numériques

Une constante littérale est l'expression écrite d'une valeur connue ; exemples : 12, -5, 3.5, 0, etc.

La donnée représentée par une constante a toujours un type qui détermine ses propriétés formelles (comme : quelles opérations la donnée peut-elle subir ?) et matérielles (comme : combien d'octets la donnée occupe-telle dans la mémoire de l'ordinateur ?). La manière la plus simple et la plus fiable de connaitre le type d'une expression consiste à poser la question à Python.

Exemples

>>> type(0) <type int>	>>> type(3000000000) <type long>	>>> type(-5.0) <type 'float'>	>>> type(6.0221417e+023) <type 'float'>

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (2/3)

- Variable(1/2)

Un identificateur est une suite de lettres et chiffres qui commence par une lettre et n'est pas un mot réserve (comme if, else, def, return, etc.). Le caractère _ est considéré comme une lettre. Exemples : prix, x, x2, nombre_de_pieces, vitesseDuVent, etc. Majuscules et minuscules n'y sont pas équivalentes : prix, PRIX et Prix sont trois identificateurs distincts.

Une variable est constituée par l'association d'un identificateur a une valeur. Cette association est créée lors d'une affectation, qui prend la forme :

variable = valeur

NB : Les Affectations multiples autorisées : a = b = 0 ;

Si un identificateur n'a pas été affecte son emploi ailleurs qu'au membre gauche d'une affectation est illégale et provoque une erreur.

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (3/3)

Variable(2/2)

- Les variables n'ont pas besoin d'être déclarées (c'est-a-dire préalablement annoncées), la première affectation leur tient lieu de déclaration.
- Les variables ne sont pas liées à un type (mais les valeurs auxquelles elles sont associées le sont forcement) : la même variable nombre a été associée à des valeurs de types différents (un int, puis un float).

Chaîne de caractères

Une donnée de type chaîne de caractères est une suite de caractères quelconques. Une constante chaîne de caractères s'indique en écrivant les caractères en question soit entre apostrophes, soit entre guillemets : 'Bonjour' et "Bonjour" sont deux écritures correctes de la même chaine.

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (3/3)

- Chaîne de caractères
  - Pour éviter les problèmes d'encadrement consiste a l'inhiber par l'emploi de \, comme dans la chaıne 'Il n\'a pas dit "oui"'.
  - L'encadrement par de triples guillemets ou de triples apostrophes permet d'indiquer des chaınes qui s'etendent sur plusieurs lignes.
    - Opérations sur chaines de caractères
- Concaténation: L'operateur + applique a deux chaınes de caractères produit une nouvelle chaıne qui est la concaténation (c'est-`a-dire la mise bout-`a-bout) des deux premières.

Exemple:

```
>>> "UFR"+" MI"
'UFR MI'
```

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (3/3)

- Opérations sur chaines de caractères

- Accès a un caractère individuel : On accède aux caractères d'une chaine en considérant celle-ci comme une séquence indexée par les entiers : le premier caractère a l'indice 0, le second l'indice 1, le dernier l'indice n – 1, n étant le nombre de caractères.

Exemple:

```
>>> s = 'Bonjour'
>>> s[6]
'r'
>>> s[-1]
'r'
```

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (3/3)

- Opérations sur chaines de caractères

- Nombre de caractères: Le nombre de caractères d'une chaıne s est donne par l'expression len(s). Pour accéder aux caractères a la fin d'une chaıne il est commode d'employer des indices négatifs : si i est positif, s[-i] est la même chose que s[len(s) - i].
- Tranches: On peut designer des tranches dans les chaınes ; la notation est chaıne[début : fin] ou début est l'indice du premier caractère dans la tranche et fin celui du premier caractère en dehors de la tranche. L'un et l'autre de ces deux indices peuvent être omis, et même les deux. De plus, ce mécanisme est tolérant envers les indices trop grands, trop petits ou mal places

IV- EXPRESSION ET OPERATEURS

IV-1- Expressions (3/3)

- Opérations sur chaines de caractères

✓ Tranches:

>>> s="Bonjour"	>>> s[3 :5] 'jo'	>>> s[3 :] 'jour'	>>> s[:5] 'Bonjo'
>>> s[:] 'Bonjour'	>>> s[6 :3] ''	>>> s[3 :100] 'jour'	

Les chaines de caracteres sont immuables : une fois creees il ne peut rien leur arriver. Pour modifier des caracteres d'une chaine on doit construire une nouvelle chaine qui, si cela est utile, on peut remplacer la precedente. Par exemple, proposons-nous de changer en 'J' le 'j' (d'indice 3) de la chaine precedente :

```
>>> s = s[ :3] + 'J' + s[4 :]
```

IV- EXPRESSION ET OPERATEURS

IV-2- Opérateurs

- Opérateurs arithmétiques

+ : addition	- : soustraction	* : multiplication	/ : division
% : reste de la division	** : puissance		

- Si une des opérandes est flottant, l'autre opérande est convertie si nécessaire en flottant et l'opération et le résultat sont flottants. Si, au contraire, les deux opérandes sont entiers, l'opération et le résultat sont entiers.

- La division peut produire des résultats surprenant du type : 3/2 vaut 1 et 2/3 vaut 0. Pour obtenir un résultat flottant il faut se débrouiller pour qu'au moins un des opérandes soit flottant : 3.0/2 ou float(3)/2 valent bien 1.5.

Attention : float(3/2) vaut 1.0, car la conversion en float intervient apres la perte des décimales.

IV- EXPRESSION ET OPERATEURS

IV-2- Opérateurs

✓ Priorité des opérateurs

Les operateurs multiplicatifs *, / et % ont une priorité supérieure a celle des operateurs additifs + et -. Cela signifie, par exemple, que l'expression a * b + c exprime une addition (dont le premier terme est le produit a* b). L'emploi de parentheses permet de contourner la priorité des operateurs. Par exemple, l'expression a * (b + c) représente bien une multiplication (dont le second facteur est l'addition b + c).

✓ Opérateurs de comparaison

==, != : égal, non égal.

<, <=, >, >= : inferieur, inferieur ou égal, supérieur, supérieur ou égal.

NB: la priorité des operateurs de comparaison est inferieure a celle des operateurs arithmétiques. L'expression a == b + c signifie bien a == (b + c).

IV- EXPRESSION ET OPERATEURS

IV-2- Opérateurs

- ✓ Booléens et opérateurs booléens

Le résultat d'une comparaison comme a == b est de type booléen ; ce type ne possède que deux valeurs, False (faux) et True (vrai).

Les opérateurs booléens ou connecteurs logiques sont

– not (négation) : not a est vraie si et seulement si a est fausse,

– and (conjonction) : a and b est vraie si et seulement si a et b sont toutes deux vraies,

– or (disjonction) : a or b est vraie si et seulement si au moins une deux

V- ENTREES-SORTIES

V-1- acquisition des données au clavier

Les fonctions **input** et **raw_input** sont utilisées pour les saisies au clavier :

– raw_input(invite)

– input(invite)

Exemple : *>>> s = raw_input("Entrer un nombre : ")*

ces fonctions sont souvent employées pour acquérir des données qui ne sont pas des chaînes (des nombres, etc.).Il faut alors la composer avec une fonction de conversion qui, en principe, a le même nom que le type visé (int, float, etc.).

Exemple : *>>> s = float(raw_input("Entrer un nombre : "))*

V- ENTREES-SORTIES

V-3- affichage formaté(1)

Il existe trois manières d'obtenir l'affichage d'une valeur à l'écran :

– l'affichage automatique du résultat que fait l'interpréteur lorsqu'on lui donne comme travail l'évaluation d'une expression

– la fonction print, certainement le moyen le plus pratique;

Exemple:

```
>>> s="bonjour les amis j\'ai un objectif, apprendre python"
>>> print(s)
```

– l'application à l'unité de sortie standard (l'écran) de fonctions de traitement de fichiers plus savantes, comme write, expliquées à la section traitant des fichiers.

V- ENTREES-SORTIES

V-3- affichage formaté(2)

la fonction print, a un usage plus souple présenté ci-dessous dans le dernier mode d'affichage.

Exemple:

```
nom = input("Votre nom : ")
prenom = input("Votre prénom : ")
age = int(input("Votre age : "))
print("Je m'appelle " ,nom, " " ,prenom, " et j'ai " ,age)
print("Je m'appelle " +nom,+" " +prenom,+" et j'ai « +str(age))
print("Je m'appelle {0} {1} et j'ai {2} ans.".format(prenom, nom, age))
```

Quant python exécute le print il replace {0}, {1}, …, {n} par la 1ere, 2nd, …, nieme variable passé à la méthode format

VI- STRUCTURES DE CONTROLE

- Python et indentation

L'indentation (c'est-`a-dire la marge blanche laissée a gauche de chaque ligne) joue un rôle syntaxique et permet d'économiser les accolades {...} et les begin...end d'autres langages. En contrepartie, elle obéit a des règles strictes :

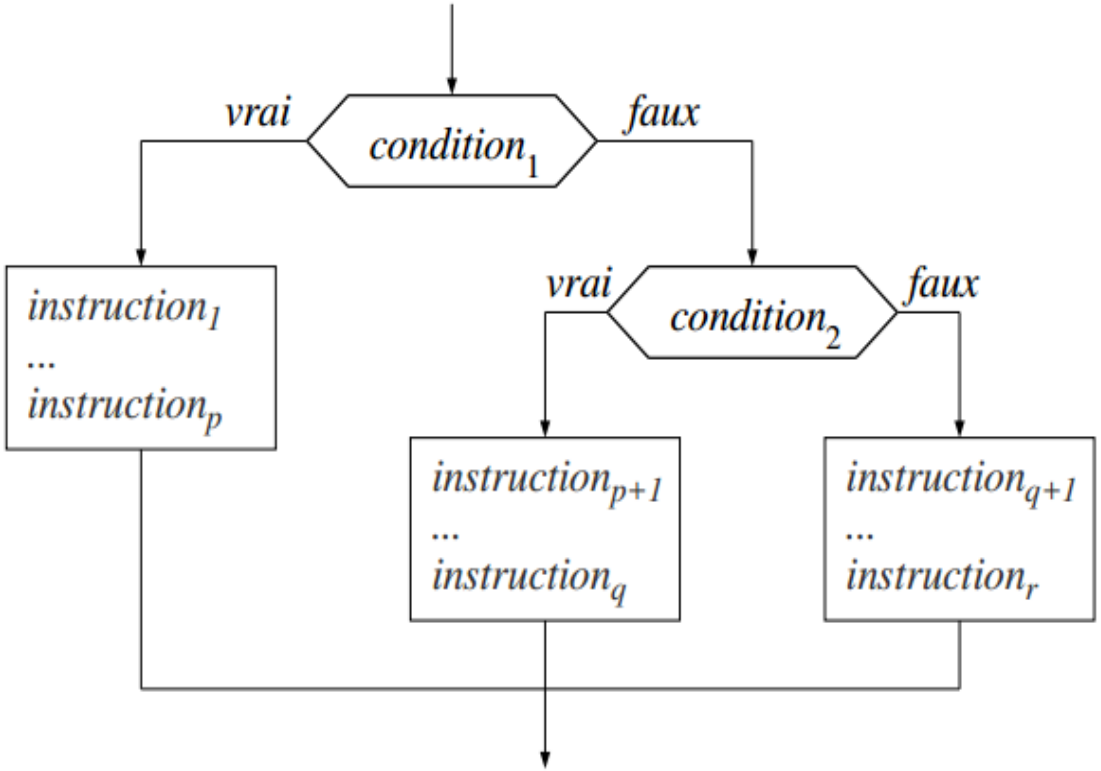
1. Une séquence d'instructions (instructions consécutives, sans subordination entre elles) est faite d'instructions écrites avec la même indentation.
2. Dans une structure de contrôle (instruction if, for, def) une sequence d'instructions subordonnees doit avoir une indentation superieure a celle de la premiere ligne de la structure de controle. Toute ligne ecrite avec la meme indentation que cette premiere ligne marque la fin de la sequence subordonnee.

VI- STRUCTURES DE CONTROLE

VI-1- Instruction conditionnelle

Syntaxe:

```
if condition1 :
    instruction1
    ...
    instructionp
elif condition2 :
    instructionp+1
    ...
    instructionq
else :
    instructionq+1
    ...
    instructionr
```



Remarque:

Le groupe de lignes de « elif condition2 : » `a « instructionq » peut apparaıtre un nombre quelconque de fois ou etre absent. Le groupe de lignes qui va de « else : » a « instructionr » peut être absent.

VI- STRUCTURES DE CONTROLE

VI-1- Instruction conditionnelle

Application: Résolution d'une équation du second degré dans IR

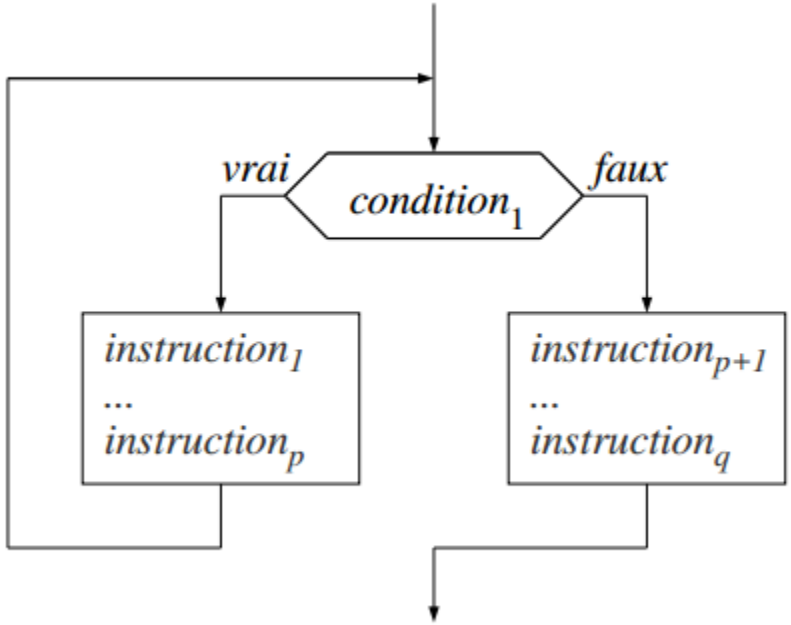
```
#####
from math import *
a = float(raw_input("a : "))
b = float(raw_input("b : "))
c = float(raw_input("c : "))
delta = b * b - 4 * a * c
if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        print ('deux solutions:', x1, x2)
elif delta == 0:
x = -b / (2 * a)
        print 'une solution:', x
else:
        print 'pas de solution reelle'
```

VI- STRUCTURES DE CONTROLE

VI-2- la boucle tant que

Syntaxe:

```
while condition1 :
    instruction1
    ...
    instructionp
else :
    instructionp+1
    ...
    instructionq
```



Remarque:

La partie qui va de « else : » a « instructionq » peut etre absente. Pratiquement, elle n'a d'intérêt que si une instruction break est présente dans la boucle.

VI- STRUCTURES DE CONTROLE

VI-1- la boucle tant que

Application: Etant donne un nombre positif ´ a, trouver le plus petit entier k vérifiant a ≤ 2k. Algorithme naïf (il y a plus efficient) : construire successivement les termes de la suite 2k (20 = 1, 21 = 2, 22 = 4, 23 = 8, 24 = 16...) jusqu'`a ce qu'ils ´egalent ou d´epassent la valeur de a.

```
#######
a = input("a : ")
k = 0
p = 1 # on veille `a avoir constamment p = 2**k
while p < a:
      p = p * 2
      k = k + 1
print( 'k:', k, ' 2^k:', p)
```

VI- STRUCTURES DE CONTROLE

VI-2- la boucle pour

Syntaxe:

```
for variable in séquence :
    instruction_1
    ...
    instruction_p
else :
    instruction_{p+1}
    ...
    instruction_q
```

Remarque:

La partie qui va de « else : » a « instructionq » peut être absente. Pratiquement, elle n'a d'intérêt que si une instruction break est présente dans la boucle.

VI- STRUCTURES DE CONTROLE

VI-3- la boucle pour

Les fonctions range et xrange:

La fonction range construit effectivement en mémoire la séquence demandée ; on peut regretter l'encombrement qui en résulte, alors qu'il nous suffirait de disposer successivement de chaque élément de la séquence. La fonction xrange fait cela : vis-à-vis de la boucle for elle se comporte de la même manière que range, mais elle ne construit pas la séquence.

######

```
>>> range(-35, +35, 5)
[-35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]
>>> xrange(-35, +35, 5)
xrange(-35, 35, 5)
>>> for i in xrange(-35, +35, 5): print i,
-35 -30 -25 -20 -15 -10 -5 0 5 10 15 20 25 30
```

VI- STRUCTURES DE CONTROLE

VI-2- Break et else dans les boucles

L'instruction break placée à l'intérieur d'une boucle while ou for produit l'abandon immédiat de la boucle et la continuation de l'exécution par la première instruction qui se trouve après la boucle. La clause else, si elle est présente, n'est alors pas exécutée.

######

acquisition de la chaîne s et du caractère c

```
for i in range(len(s)):
    if s[i] == c:
        break;
    else:
        i = -1
```

- ici, la valeur de i répond exactement à la question.
- Attention, piège! Notez que else est indenté comme for. L'aligner avec if aurait complètement changé le sens de ce bout de code et l'aurait rendu faux.