Découvrir la programmation

Avec le langage Python

Sébastien Combéfis

UCLouvain ACM Student Chapter ASBL

16 avril 2010

UCLouvain
ACM Student Chapter

python™

Plan de la présentation

1. Concepts de base de la programmation
2. Types de données, variables et affectation
3. Expressions et opérateurs
4. Structure de contrôle de flux : conditions et boucles
5. Entrées/Sorties : manipulation de fichiers
6. Types de données avancés : tableaux, listes, ensembles et dictionnaires
7. Fonctions : définition, appel et passage des paramètres

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Un bref historique des langages

50's Approche expérimentale : Fortran, Lisp, Cobol ...

60's Langages universels : Algol, PL/1, Pascal ...

70's Génie logiciel : C, Modula-2, Ada ...

80's Programmation objet : C++, LabView, Eiffel, Matlab ...

90's Interprétés objet : Java, Perl, Tcl/Tk, Ruby, Python ...

Deux types de codes

- **Code compilé**

  Code source du programme compilé directement en un code machine, qui est directement exécuté par celle-ci

- **Code interprété**

  Code source est interprété, instruction par instruction, celles-ci étant ensuite exécutées sur la machine

Et en Python ?

- **Technique mixte :** compilation du code source en bytecode qui est ensuite interprété par une machine virtuelle

- Plus souple, mais codes générés moins performants

Modèle de programmation

- **Procédural**

  Raffinements successifs pour décomposer un problème complexe en sous-problèmes plus simples à résoudre. Dans ce modèle, on structure d'abord les actions

- **Orienté objets**

  On conçoit des fabriques (classes) qui permettent de créer des composants (objets) qui contiennent des données (attributs) et des actions (méthodes). On peut établir des relations entre classes : composition, héritage ...

➥ En Python, les deux approches sont possibles

Programme et Algorithme

- **Algorithme**

  Suite d'instructions qui décrivent des étapes à franchir pour résoudre un problème donné, en un nombre fini d'instructions

- **Programme**

  Traduction d'un algorithme dans un langage de programmation (une implémentation) qui sera exécutable sur un ordinateur

Code source d'un programme

- Le code source est destiné principalement à l'être humain, il doit donc être lisible : conventions d'écriture et commentaires
- Le caractère # indique le début d'un commentaire. Ce dernier se finit en fin de ligne

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Programme d'exemple
# Auteur  : Sébastien Combéfis
# Version : 10 avril 2010
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

print 9 + 2    # affiche la somme de 9 et 2 à l'écran
```

Programme Python

- **Via l'interpréteur Python**

  On va taper les instructions du programme une à une de manière interactive et elles seront directement exécutées

- **Via un fichier de script**

  On écrit toutes les instructions du programme dans un fichier texte qui sera lu instruction par instruction qui seront directement exécutées

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Types de données

- **Booléennes :** prennent deux valeurs possibles : vrai ou faux, représentées par `True` et `False`
- **Entiers :** nombres entiers (ensemble **Z**), par exemple `-12`, `0`, `42`, `12345` ...
- **Flottants :** nombres à virgule (sous-ensemble de **R**), par exemple `2.178`, `3e8`, `6.11e-13` ...
- **Complexe :** nombres complexes (sous-ensemble de **C**), par exemple `1j`, `(2+3j)` ...

Variable

- Une variable permet de stocker une donnée. Elle possède un nom et une valeur. Informatiquement, il s'agit d'une référence vers une zone mémoire
- Nom conventionnellement écrit en minuscule, commencent par `a-z_`, suivi de `a-z0-9_`. Doit être différent des mots réservés de Python est des constantes **`None`**, **`True`** et **`False`**

```
age
piR2
_nom
sea_sex_and_sun69
```

Mots réservés de Python 2.6

`and`	`def`	`finally`	`in`	`print`	`yield`
`as`	`del`	`for`	`is`	`raise`	
`assert`	`elif`	`from`	`lambda`	`return`	
`break`	`else`	`global`	`not`	`try`	
`class`	`except`	`if`	`or`	`while`	
`continue`	`exec`	`import`	`pass`	`with`	

+ les constantes `None`, `True` et `False`

Affectation

- On affecte une valeur à une variable en utilisant le signe = (rien à voir avec l'égalité mathématique)
- On peut ainsi définir ou changer la valeur d'une variable

```
age = 17          # affectation de la valeur entière 17
                  # à la variable age

age = 25          # la variable age reçoit la valeur 25

a = b = 0         # cibles multiples (de droite à gauche)

a, b = 1, 2       # affectation de tuple (par position)
```

Entrée et sortie standard

- On peut lire une valeur entrée sur le clavier par l'utilisateur en utilisant **input**
- On peut écrire une valeur vers l'utilisateur en utilisant **print**

```
age = input ("Quel est votre âge ?")

print "Vous avez", age, "ans"
```

Types et conversion

- On peut connaitre le type d'une valeur avec **type**
- On peut convertir une valeur avec **str**, **int**, **float** ...

```
value = input ("Entrez une valeur : ")
print type (value)
```

- On peut lire crument sur l'entrée standard avec **raw_input**

```
value = raw_input ("Entrez une valeur : ")
print type (value)

x = int (value)
print type (x)
```

Plan de la présentation

1. Concepts de base de la programmation
2. Types de données, variables et affectation
3. Expressions et opérateurs
4. Structure de contrôle de flux : conditions et boucles
5. Entrées/Sorties : manipulation de fichiers
6. Types de données avancés : tableaux, listes, ensembles et dictionnaires
7. Fonctions : définition, appel et passage des paramètres

Expressions et Opérateurs

- On va pouvoir écrire des calculs grâce à des opérateurs combinés avec des opérandes. On construit ainsi des expressions.
- Une expression possède une valeur et va pouvoir être affectée à une variable

```
cette_annee = 2010
annee = input ("Quel est votre année de naissance ?")
age = cette_annee - annee
vingt_ans = annee + 20

print "Vous avez", age, "ans"
print "Vous aurez 20 ans en", vingt_ans
```

Opérateurs booléens

- Opérateurs de comparaison : ==, !=, >, >=, < et <=

```
print 12 >= 22
print 1 <= 33 < 92
```

- Opérateurs logiques : **not**, **or**, **and**

```
print (3 == 3) or (9 > 14)
print (1 > 2) and (11 > 7)
print not (8 < 12)
```

Opérateurs arithmétiques

- Opérateurs arithmétiques usuels : `+`, `-`, `*`, `/`

```
print 21 + 32 * 10
print 2 * (3 - 10)
```

- Quotient et reste de la division entière : `/`, `%`

```
print 12 / 7
print 12 % 7
```

- Opérateur d'exponentiation : `**`

```
print 2 ** 10
```

Pour les flottants

- Même opérateurs arithmétiques que les entiers, sauf pour la division

```
print 1.0 / 2.0
print 1.0 // 2.0
```

L'opérateur `//` force la division entière

Retour sur les affectations

- Modifier la valeur d'une variable par affectation

```
x = 0
x = x + 1
```

- Affectation composées : raccourci d'écriture si la variable est initialisée

```
x = 0
x += 1
```

Priorité et associativité

- Les opérateurs sont classés par priorité
- La règle d'associativité règle l'évaluation d'expressions avec opérateurs de même niveau de priorité
- On utilise des parenthèses pour forcer la priorité

```
1 + 2 * 3
(1 + 2) * 3

1 - 2 - 3
(1 - 2) - 3
1 - (2 - 3)
```

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Instructions simples et composées

- Une instruction simple consiste en une ligne et correspond à une instruction

- Une instruction composée consiste en une entête terminée par deux-points et suivi d'un bloc d'instructions indenté au même niveau

Prendre une décision

- On peut prendre une décision avec l'instruction composée **if**

```
points = input ("Vos points sur 20 :")

if points >= 12:
    print "Vous avez réussi votre examen"

    # Affichage du pourcentage
    pourcent = points * 5
    print "Vous avez une moyenne de", pourcent, "%"
```

Contrôler une alternative

- L'instruction composée **if-else** permet d'effectuer une action si une condition est satisfaite, et une autre le cas échéant

```
points = input ("Vos points sur 20 :")

if points >= 12:
   print "Vous avez réussi votre examen"
else:
   print "Vous avez raté votre examen"

pourcent = points * 5
print "Vous avez une moyenne de", pourcent, "%"
```

Syntaxe compacte et plusieurs choix

- On peut utiliser une syntaxe compacte

```
if x < y:
    min = x
else:
    min = y
# est équivalent à
min = x if x < y else y
```

- Si on a plusieurs choix, on utilise `elif`

```
if 12 <= x <= 20:
    print "Réussi !"
elif 10 <= x < 12:
    print "Réussi, mais juste !"
else:
    print "Raté"
```

Répéter une portion de code

- On peut répéter du code avec l'instruction composée **while**

```
i = 1
while i < 5:  # la boucle se répète tant que i < 5
    print i
    i += 1
```

- Exemple : la fonction de Fibonacci ($F_n = F_{n-1} + F_{n-2}$, avec $F_1 = 1$ et $F_2 = 2$)

```
n = input ("n (> 2) ?")
a, b, i = 1, 2, 2
while i < n:
    tmp = b
    b = a + b
    a = tmp
    i += 1
print b
```

Instructions imbriquées

- On peut imbriquer des instructions composées, c'est-à-dire avoir une instruction composée qui fait partie du corps d'une autre instruction composée

```
i = 1
while i < 17:
    print i,
    if i % 2 == 0:
        print "est pair"
    else:
        print "est impair"
    i += 1
```

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Ouvrir et fermer un fichier

- On ouvre un fichier avec `open`, trois modes possibles (lecture, écriture et ajout)
- On ferme un fichier avec `close` (obligatoire pour que tous les changements soient écrits sur le disque)

```
file = open ("nomDeMonFichier", "r")        # en lecture
file.close()

file = open ("unAutreFichier", "w")       # en écriture
file2 = open ("encoreUnAutre", "a")          # en ajout
file.close()
file2.close()
```

Écrire dans un fichier

- Pour écrire dans un fichier, de manière séquentielle, on utilise `write` qui ajoute à la suite du fichier
- On ne peut écrire que des chaines de caractères, il faudra convertir le autres types

```
year = 1984

file = open ("monFichier", "w")
file.write ("Je suis né en")
file.write (str (year))
file.close()
```

Lire depuis un fichier

- Pour lire depuis un fichier, de manière séquentielle, on utilise `readline` qui lit la ligne suivante du fichier
- On peut lire tout le fichier avec `read`
- On peut utiliser la boucle `for` pour lire le fichier complètement, ligne par ligne

```
file = open ("monFichier", "r")
all = file.read()

for line in file:
   print line
file.close()
```

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Séquences

- En plus des types de base, Python propose des types de données avancés
- Une séquence est un conteneur ordonné d'éléments indicés par des entiers (en Python : chaines, listes et tuples)
- On peut agir sur une séquence via une fonction ou via une méthode

```
s = "Hello"
nbrcar = len (s)
up = s.upper()
```

Fonction et méthode

- **Fonction**

  On applique une fonction en utilisant son nom, en lui passant éventuellement des paramètres et en affectant éventuellement son résultat à une variable

```
day = input ("Quel jour sommes nous ?")
```

- **Méthode**

  On applique une méthode sur un objet, avec la notation pointée, en lui passant éventuellement des paramètres et en affectant éventuellement son résultat à une variable

```
x = "hello".upper()
```

Notation des chaines de caractères

- Entre guillemets doubles :

```
s = "Aujourd'hui, il faut beau !"
```

- Entre guillemets simples :

```
s = 'Elle est "bêbête"'
```

- Entre triples guillemets simples ou doubles :

```
s = """Texte cru
inséré exactement
tel quel :-)"""
```

Opérations sur les chaines de caractères

- Longueur d'une chaine : `len ("Hellow")`
- Concaténation de deux chaines : `"Bis" + "oux"`
- Répétitions d'une chaine : `"Bye! " * 2`
- Position d'une sous-chaine : `"Cordialement".find ("ale")`
- Convertir en minuscules : `"Marc".lower()`
- Supprime les blancs : `"  bla bla   ".strip()`

Indiçage des chaines de caractères

- Les caractères sont indicés en commençant à zéro
- On accède à un caractère en donnant son indice entre crochets
- Un indice négatif fait commencer par la fin
- On peut extraire une sous-chaine
- Les chaines de caractères ne sont pas modifiables

```
s = "Hello"
print s[0]
print s[2]
print s[len (s) - 1]
print s[-1]

print s[0:2]
print s[2:]
print s[:2]
```

Listes

- Collection hétérogène, ordonnée et modifiable d'éléments
- Notation entre crochets, séparés par des virgules
- Accès et modification par indices

```
finalistes = ["John", "Bob", "Catherine", "Stefen"]

print finalistes
print finalistes[1]

finalistes[1] = "Charles"
```

Créer une liste et appartenance

- Création de listes

```
l = []
m = [0] * 10
```

- Création de listes avec **range**

```
l = range (3)
m = range (3, 10)
n = range (3, 10, 3)
```

- Test de l'appartenance à une liste avec **in**

```
list = range (3, 8, 2)
if 8 in list:
   print "8 est dedans !!!"
```

Opérations sur les listes

- Ajouter un élément à la fin : `list.append (e)`
- Insérer un élément à un indice : `list.append (i, e)`
- Supprimer un élément : `list.remove (e)`
- Trouver l'indice d'un élément : `list.index (e)`

```
list = range (10)
list.append (99)
list.insert (0, 66)
list.remove (7)
i = list.index (9)
```

Techniques de « slicing »

- On peut donc utiliser un slice pour identifier une partie de tableau : `list[x:y]`
- En utilisant cette notation à gauche de l'affectation, on va pouvoir modifier une liste

```
list = ["Florence", "Patricia", "Tania"]
list[0:0] = ["Claire"]
list[3:3] = ["Sophie", "Stéphanie"]
list[1:2] = []
list[2:4] = ["Sarah"]
```

Tuples

- Collection hétérogène, ordonnée et **non**-modifiable d'éléments
- Notation entre parenthèses, séparés par des virgules
- Utilisation comme les listes, mais plus rapide

```
finalistes = ("John", "Bob", "Catherine", "Stefen")

print finalistes
print finalistes[1]

finalistes[1] = "Charles"    # Erreur d'exécution
```

Dictionnaires

- Collection de couples `clé : valeur`
- Notation entre accolades, séparés par des virgules
- Utilisation comme les listes, avec indice au lieu de clé

```
dico = {"je" : "I", "tu" : "you"}
dico["il"] = "he"

print dico["je"]
del dico["tu"]

print dico
```

Opération sur les dictionnaires

- Récupérer toutes les clés avec `keys` et toutes les valeurs avec `values`
- Récupérer tous les éléments avec `items`
- Savoir si une clé existe avec `has_key`

```
dico = {"je" : "I", "tu" : "you"}

k = dico.keys()
e = dico.items()

print dico.has_key ("il")
print "il" in dico
```

Parcourir les collections avec un itérateur

- On peut utiliser la boucle `for` pour parcourir les éléments d'une collection

```
list = [12, 82, 11, 9, 12]
for i in list:
    print i

dico = {"je" : "I", "tu" : "you", "il" : "he"}
for k in dico.iterkeys():
    print k
for v in dico.itervalues():
    print v
for e in dico.iteritems():
    print e
```

Plan de la présentation

1 Concepts de base de la programmation

2 Types de données, variables et affectation

3 Expressions et opérateurs

4 Structure de contrôle de flux : conditions et boucles

5 Entrées/Sorties : manipulation de fichiers

6 Types de données avancés : tableaux, listes, ensembles et dictionnaires

7 Fonctions : définition, appel et passage des paramètres

Fonction

- Une fonction est un ensemble d'instructions qui porte un nom et s'exécute à la demande
- Une fonction peut renvoyer un résultat
- Une fonction peut recevoir des paramètres

```
def max (a, b):
    if a > b:
        return a
    else:
        return b

print max (7, 12)
print max (81, 3)
```

↪ Attention, pour fonctions vides, il faut utiliser `pass`

Passage des paramètres et valeur par défaut

- Les paramètres sont passés par affectation
- On peut définir des valeurs par défaut

```
def max (a, b = 10):
    if a > b:
        return a
    else:
        return b

print max (8)
print max (12)
print max (9, b = 3)
print max (b = 13, a = 8)
```

Module

- Un module est une bibliothèque de fonctions
- On peut importer tout un module ou seulement certaines fonctions

```
import math

# -ou-

from math import sqrt
```

Alias et copies de liste

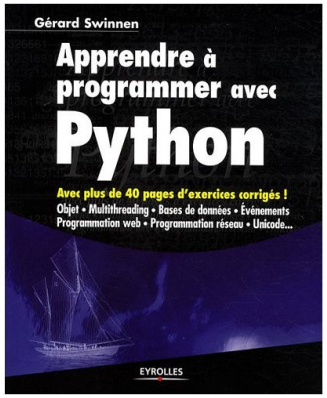
- Des alias, ce sont des références vers le même objet
- Pour faire de vraies copies, on utilise le module `copy`

```
from copy import copy

a = [1, 2, 3]
b = a
b[0] = 15
print a

a = [1, 2, 3]
b = copy (a)
b[0] = 15
print a
```

Ressources et Références



Apprendre à programmer avec Python (2009)

Gérard Swinnen

Eyrolles

978-2212124743

- **Site web officiel :** http://www.python.org/
- **Ressources diverses :** http://python.developpez.com/
- **Tutoriel en ligne :** http://diveintopython.org/