Python et Unix

Yann Thierry-Mieg
Laboratoire d'informatique de Paris 6
EPSI

Organisation

- 5 séances de 4h, généralement 1h cours, 3h TP
- 2 notes de CC : TP troisième séance relevé et mini-projet par binômes
- 1 examen sur table = 60% de la note finale
- Documents "manuscrits" + polys de cours autorisés, pas de livres

Plan

- Introduction à Python
- UNIX : Les communications Inter-Processus
- Python : mécanismes objets et programmation avancée
- UNIX : Les modèles de Thread
- Python : les librairies haut niveau
- UNIX : configuration de services

Introduction

- Python : langage de script interprété
- Avantages :
  - Simplicité
  - Puissance
  - Robustesse
  - Libre
  - Objet
  - Librairies très haut niveau
  - Accès aux API système

Historique

- 1989: Guido van Rossum (père de python)
- 1991: première distribution publique
- A partir de 1997 engouement de la communauté

- Actuellement: Python 2.2.2, porté sur toutes les plate formes, centaines de libs, centaines de milliers d’utilisateurs…

Caractéristiques de Python

- Haut Niveau : 4eme génération:
  1. Assembleur
  2. Procédural (C,fortran…)
  3. Objet (C++, java, …)
  4. Interprétés (Python, perl, tcl…)
- Orienté Objet :
  - Héritage, structuration en classes
  - Données/Traitements regroupés

Caractéristiques de Python (2)

- Modulaire :
  - Facilité de définition de modules
  - Bonne réutilisation => projets de plus grande envergure que sh
- Extensible :
  - Intégration de code C, C++, java …(efficace)
  - Definition de module extension du langage

Caractéristiques de Python (3)

- Facile à apprendre :
  - Peu de mots clés
  - Syntaxe bien définie
  - Objet non obligatoire (cf. java)
- Facile à Lire :
  - Pas de declarations, pas de parenthèses blocs
  - Force a indenter
- Maintenable (cf. problème avec Perl)

Caractéristiques de Python (4)

- Pas de gestion de mémoire
- Compilé en byte-code interprété
- **LIBRE ET GRATUIT**

- Conclusion : Adapté au prototypage rapide de systèmes, au scripting cross-platform, a l'interaction entre plusieurs langages, très utilisé en Win32 (COM…)

Python vs. autres langages de script

- Perl :
  - assez similaire sur le concept (script + accès aux API système)
  - Syntaxe plus obscure
  - +++Analyse de chaînes de caractères
- Javascript :
  - assez proche de python (objet,syntaxe..)
  - pas d’accès au système
  - browser based

Démarrage

- python :
  - Démarre en mode interactif
- Options :
  - d : debug
  - v : verbose
  - python fich.py : execute le script
- Unix :
  - #! /usr/bin/python en 1ere ligne
  - chmod +x script.py

PLAN PARTIE I : SYNTAXE PYTHON

- Entrées/sorties
- Opérateurs de Base
- Types python :
  - numériques
  - string
  - list
  - tuple
  - dictionnaire
- Structures de controle (if, for ...)
- Définition de procédures

Entrées/Sorties (1)

- Sortie : print
  - possible de le lier a des fichiers quelconques
  - opérateurs de formatage (%s : string , %d : entier , %f float) comme printf() de C

>>>print "hello world"
hello world
	>>> **print** "%s number %d!" % ("Python", 1)
	Python number 1
	>>> string = "Hello"
	>>> print string, " world"
	Hello world

Entrées/Sorties (2)

- Entrée : raw_input
  - prompte sur stdin
  - récupère un string, typer pour en faire un int
  - il existe aussi "input" qui reconnaît le type de l'entrée, mais il faut gérer les entrées invalides

```
>>>nom = raw_input("Votre nom ? ")
Votre nom ? toto
>>>print "Bonjour ",nom
Bonjour toto
```

```
>>> num = raw_input("un chiffre ? ")
un chiffre ? 10
>>> print '2^chiffre : %d' % (2**int(num))
2^chiffre : 1024
```

Commentaires

- # jusque fin de ligne
- pydoc :
  - la premiere ligne suivant la declaration d'une fonction, classe, module est une description de l'objet, accessible par __doc__

```
>>>#un commentaire
>>>print "toto" #un autre
toto
```

```
>>>def hello () :
…           " imprime hello "
…           print "hello"
…
>>>print hello.__doc__
' imprime hello '
```

Opérateurs

+ (plus) – (moins) * (mult) / (div)
% (modulo) **(puissance)

- Attention, int / int = division entiere
- Python fait aussi calculatrice !
- Priorités usuelles réspectées:

```
>>>print –2*4+3**2
1
```

Opérateurs (2) : comparateurs

< <= > >= == != **and or not**

- Attention, == comparateur
- Valeur de vérité : 1 , 0 = false

>>> (2 < 4) **and** (2 == 4)
0
>>> (2 > 4) **or** (2 < 4)
1
>>> **not** (6.2 <= 6)
1
>>> 3 < 4 < 5
1
	and n'évalue le 2eme membre que si le 1er est **true**
	
	or n'évalue le 2eme membre que si le 1er est **false**
	
	négation
	
	équivalent a: (3<4) and (4<5)

Variables, Affectations : symbole =

- Pas de :
  - déclaration préalable,
  - de typage explicite,
- **Type et valeur initialisés à l'affectation**
- Identifiant valide :(a-zA-Z_)(a-zA-Z0-9_)*

```
>>> counter = 0
>>> miles = 1000.0
>>> name = 'Bob'
>>> counter = counter + 1
>>> kilometers = 1.609 * miles
```

int
float
string
incrément, il existe aussi x += 1 mais pas x++ ou ++x
float

Sensible a la casse: toto != ToTo

Types numériques

- int : entier signé
  - 345 ; -21 ; 0x80
- long : entier de longueur arbitraire
  - 9234914633468138136863**8L** ; -2625362323**l**
- float : double en C
  - 3.141593 ; 2.1E-3 ; 10. ; 6.022e23
- complex : intégré au langage
  - 1+1.2j ; 0+1j ; 2+0j

Type string (chaîne)

- Trois formes : "s1" 's2' """s3"""
- Accès indexé type tableau de caractères

>>> c1 = "c'est une chaine!"
>>> c2 = 'ceci est "une autre" chaine \n'
>>> c3 = """ une chaine
...avec des retours chariots """
	>>> c1[0]
	'c'
	>>>c1[-1]
	'!'

Type chaîne : opérations basiques (2)

- Concaténation : +
- Repetition : *

>>> "hello"+ "world"
'hello world'
 	
>>> num = 3
>>> "hello" + `num`
'hello3'
>>> "hello" * 3
'hellohellohello'
	le backquote ` transforme en string la variable num, équivalent à repr(num)

Type List

- Emploi similaire à un array C
- Contient des objets de types arbitraires
- **Accès indexé très puissant : slice**

list[a:b]

```
>>> maListe = [1, 2, 3, 4]
 >>> maListe
[1, 2, 3, 4]
 >>> maListe[0]
 1
>>> maListe[2:]
 [3, 4]
```

```
>>> maListe[:3]
 [1, 2, 3]
 >>> maListe[1] = "toto"
 >>> maListe
[1, 'toto', 3, 4]
```

Type Tuple

- Contient des objets de types arbitraires
- **Accès indexé avec slice tuple[a:b]**
- Pas de modification du contenu

```
>>> aTuple = ('robots', 77, 93, 'try')
>>> aTuple
('robots', 77, 93, 'try')
 >>> aTuple[0]
'robots'
>>> aTuple[2:]
(93, 'try')
```

```
>>> aTuple[:3]
 ('robots', 77, 93)
>>> aTuple[1] = 5
 Traceback (innermost last):
 File "<stdin>", line 1, in ?
TypeError: object doesn't
                support item assignment
```

Type Dictionnaire (Hash table)

- Paires (clé,valeur)
- Clé : souvent int ou string (efficacité)
- Valeur : objet quelconque

>>> monDict = {}	table vide
>>> monDict['host'] = 'earth'	insère la paire ('host','earth') dans monDict
>>> monDict['port'] = 80	insère la paire ('port',80) dans monDict
>>> monDict	affiche le contenu de Dict ,
{'host': 'earth', 'port': 80}	ordre non défini
>>> monDict.keys()	Rend une liste contenant les clés
['host', 'port']	
>>> monDict['host']	rend la valeur associée a 'host'
'earth'	(si elle existe)

Blocs d'instructions : INDENTATION

- Les blocs d'instructions sont délimités par l'indentation
- pas d'accolades {bloc}

- Résultat : augmente la lisibilité
- Utilisez des tabulations

- Pas de point-virgule en fin de ligne

Conditionnelle : if.. elif .. else ..

- Pas de switch/case
- Remplacé par "elif"

val = raw_input("action ?") if val == "quit" : print "bye" elif (val == "login") : doLogin() else : print "action inconnue",val	condition parenthèsage implicite de la condition appel de fonction cas par défaut

Itérations (1) : while

- Classique : tant que (condition) : action
- blocs définis par l'indentation
- **break** interrompt l'itération,
- **continue** remonte en début de boucle

```
iter = 0
while iter < 5 :
        print "itération :", `iter`
print "fini"
```

Execution :
itération : 0
itération : 1
itération : 2
itération : 3
itération : 4
itération : 5

Itérations (2) : for

- Proche du **foreach** d'autres langages
- Itération sur un objet sequence : **list, tuple ou string**
- **break** et **continue**

for fruit in ['banane','pomme','poire'] : print fruit, print Execution : banane pomme poire	fruit : variable de boucle la virgule en fin de ligne empêche l'ajout d'un '\n' Le print final flush la sortie
for i in range(6) print i	range(6)=[0,1,2,3,4,5] donne un comportement similaire à un for

Définition de fonctions : def

- def nomFonc (paramètres formels): bloc
- polymorphisme de l'appel: type des arguments et du retour déterminé à l'exécution

def addMe2Me(x):
'apply + operation to argument'
return (x + x)
	>>> addMe2Me(4.25)
	8.5
	>>> addMe2Me(10)
	20
	>>> addMe2Me('Python')
	'PythonPython'
	>>> addMe2Me([-1, 'abc'])
	[-1, 'abc', -1, 'abc']

Définition de fonctions (2) : def

- possibilité de fournir des arguments par défaut

```
def foo(debug=1):
    """determine if in debug mode
        with default argument """
    if debug:
        print 'in debug mode'
    print 'done'
```

```
>>> foo()
 in debug mode
 done
>>> foo(0)
done
```

Définition de modules

- **Un module = un fichier module.py**
- Utilisation :
  - **import** module
  - **from** module **import** n1,n2 …
  - **from** module **import** *

import modFoo module.foo()	importe les fonctions et variables du fichier modFoo appel en notation pointée
from modFoo import foo foo()	importe la fonction foo de modFoo.py appel sans qualification de nom

Bibliographie

- **Core Python Programming** ,Wesley J. Chun , Prentice Hall PTR
- **Python Essential Reference, Second Edition**,David M Beazley ,New Riders Publishing
- **Programming Python, 2nd Edition,** Mark Lutz , O'Reilly
- www.python.org/doc
- web.pydoc.org