Python : langage de script

YuGiOhJCJ

14 septembre 2008

Table des matières

1 Avant propos...

Cette documentation a été rédigée par YuGiOhJCJ. Vous lisez actuellement la version 20080914 qui est gratuite. Si vous souhaitez utiliser une partie de cette documentation pour vos créations, veuillez d'abord me contacter à yugiohjcj@1s.fr. La version la plus récente de ce document est disponible à l'adresse http ://yugiohjcj.1s.fr/. Cette publication peut contenir certaines erreurs. N'hésitez pas à me les rapporter pour que j'effectue une correction.

2 Introduction

Le langage de script Python est un langage interprété. C'est à dire que le code source n'a pas besoin d'être compilé (contrairement au langage de programmation C). La source est directement lue par un interpréteur.

3 L'interpréteur

Dans le fichier source, vous pouvez préciser en première ligne l'interpréteur qui sera utilisé pour lire votre code.

Pour définir l'interpréteur utilisé, on procède ainsi :

```
#!/usr/bin/python
```

Bien sûr, l'emplacement de votre interpréteur *python* n'est peut être pas « /usr/bin/python ». Vous devrez donc peut-être remplacer ce chemin par un autre (exemple : « /usr/local/bin/python »). Si vous avez installé le programme *which* vous pourrez connaître l'emplacement exacte de votre interpréteur en tapant cette commande :

```
which python
```

L'indication de l'interpréteur n'est pas obligatoire mais seulement pratique. En effet, vous pouvez omettre cette ligne et dans ce cas, vous devrez invoquer l'exécutable *python* à chaque exécution de votre script :

```
python /home/toto/mon_premier_script_python.py
```

4 L'encodage

Par défaut, l'interpréteur Python ne gère pas les caractères spéciaux (é, à, è, ...). Pour que votre interpréteur supporte ces caractères non-ASCII, vous pouvez ajouter cette ligne après la déclaration de votre interpréteur :

```
# -*- coding: iso-8859-15 -*-
```

Bien sûr, vous pouvez préciser d'autres jeux de caractères que « iso-8859-15 ». Si vous avez installé le programme *utrac* vous pourrez connaître l'encodage actuel de votre fichier source en tapant cette commande :

```
utrac -p /home/toto/mon_premier_script_python.py
```

5 Les commentaires

Il est possible d'ajouter des commentaires dans la source. Ceux-ci n'ont aucun effet sur le programme ; ils sont là pour donner des indications dans le fichier source.

Un commentaire en Python :

```
# Exemple de script Python
```

6 L'indentation

En Python, l'indentation du code permet de préciser où se situent les blocs. Ainsi, contrairement à d'autres langages de programmation qui utilises des accolades pour délimiter les blocs, l'interpréteur *python* vérifie la cohérence entre les espaces qui délimitent les différents blocs.

7 Les variables

On peut associer des valeurs à des noms. Ce sont des variables. Inutile, contrairement au langage C, de préciser le type de vos variables. En effet, que vos variables soient destinées à contenir des nombres ou des chaînes de caractères, le code à taper est identique.

Voici un exemple qui affiche le contenu de variables :

```
# Un entier
int = 100
# Un caractère
char = 'b'
# Une chaîne de caractères
string = "Bonjour"
# Un nombre à virgule
float = 100.123

# Afficher le contenu des variables
print int
print char
print string
print float
```

8 Les structures conditionnelles

Comme beaucoup de langages de programmation, on dispose de structures conditionnelles :

```
if int < 100:
 print "Ce texte ne s'affiche pas car la condition est fausse."
else:
 print "Étant donné que la condition du if est fausse, \
 c'est le bloc de else qui est exécuté."
liste = range(1,4)
for n in liste:
 print "On passe %s fois dans cette boucle for." %n
while 1 == 1:
 print "Ce texte s'affiche en continu car la condition du while est vraie."
```

9 Les fonctions

Il existe un grand nombres de fonctions. Par exemple, la fonction *range* est faite pour afficher une séquence de nombres. Si l'on souhaite générer une séquence de nombres de 0 à 9, il faut faire un appel de fonction :

```
range(10)
```

Parfois, il est nécessaire de créer sa propre fonction. Celle-ci peut posséder des arguments et peut retourner une valeur. Voici une fonction qui additionne deux valeurs :

```
def additionner(premier, deuxieme):
 somme = premier + deuxieme;
 return somme;
```

Il est maintenant possible de l'appeler en ajoutant les lignes suivantes :

```
a = 2
b = 3
solution = additionner(a, b)
print solution
```

10 Un exemple

```
#!/usr/bin/python
# -*- coding: iso-8859-15 -*-
# Exemple de script en Python
# YuGiOhJCJ <yugiohjcj@1s.fr>
# http://yugiohjcj.1s.fr
print "Hello World!\n"
```