Python en 10 minutes

Poromenos
http://www.poromenos.org

Traduction: Matthieu Nouzille
http://www.oxyg3n.org

1 Introduction

Vous souhaitez vous mettre au langage de programmation Python mais vous n'arrivez pas à trouver un tutoriel concis qui couvre les principaux aspects du langage? Ce tutoriel va essayer de vous enseigner Python en 10 min. Ce n'est probablement pas un tutoriel à proprement parler, il se situe plutôt entre un tutoriel et une "cheatsheet". L'objectif de celui-ci est de vous expliquer les concepts qui vous permettront de vous lancer. De toute évidence, si vous souhaitez vraiment apprendre un langage il vous faudra pratiquer. Je vais supposer que vous avez déjà programmée et donc ne pas m'attarder sur les choses non spécifiques au langage. Les mots clés les plus importants seront surlignées, vous pourrez ainsi les repérer facilement. Faites attention cependant car certaines notions seront abordées directement dans des exemples, la plupart du temps accompagnées d'un commentaire.

2 Caractéristiques

Python est fortement typé (les types sont forcés), dynamiquement et implicitement typé (cad, vous n'avez pas à déclarer de variables), sensible à la casse (ex: var et VAR sont deux variables différentes) et orienté objet (tout est objet).

3 Obtenir de l'aide

L'aide en Python est accessible directement depuis l'interpréteur. Si vous souhaitez connaitre le fonctionnement d'un objet, vous devez simplement invoquer la méthode: **help(objet)**. Une autre fonction intéressante, **dir()**, celle-ci liste les méthodes d'un l'objet et affiche le "docstring" de celui-ci.

```
>>> help(5)
Aide sur un objet de type int:
(etc etc)

>>> dir(5)
'__abs__', '__add__', ...

>>> abs.doc
'abs(number) -> number\n\nRetourne la valeur absolue de l argument.'
```

4 Syntaxe

Python n'utilise pas de caractère spécifique pour délimiter les blocs d'instructions. Tout se fait par identation (ndt: la bonne pratique est d'utiliser 4 caractères "espace" pour identer un bloc de code). L'identation démarre un bloc d'instruction et le desidentation le termine. Les instructions qui ont besoin d'être identées se terminent par deux-points (:). Les commentaires d'une seule ligne commencent avec un #, et le commentaires qui s'étalent sur plusieurs lignes utilisent des chaines de caractères (strings) multi-lignes. Les valeurs sont assignées aux variables avec un un signe égal ("=") et les tests d'égalité se font avec un double égal ("=="). Vous pouvez incrémenter/décrémenter des valeurs en utilisant les opérateurs += et -=. Ceci fonctionne avec de nombreux types de données, les chaines inclues. Vous pouvez aussi affecter plusieurs variables sur une seule ligne. Exemple:

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4
"""Ceci est un commentaire multi-lignes.
Le code suivant concatene deux chaines."""
>>> mystring = "Hello"
>>> mystring += " world."
>>> print mystring
Hello world.
# Echange de deux variables en une seule ligne(!).
```

```
# Cela ne pose pas de problème de typage car les valeurs
# ne sont pas encores assignées.
>>> myvar, mystring = mystring, myvar
```

5 Type de donées

Les structures de données disponibles en Python sont: les listes, les tuples et les dictionnaires. Les "sets" sont disponibles via la librairie "sets" (fait partie du langage à partir de Python 2.5). Les listes ressemblent à des tableaux à une dimension (Les tableaux python peuvent-être de tout type, vous pouvez donc mélanger les entiers (int), les chaines (strings) dans les listes/dictionnaires/tuples). Pour toutes ces structures de données, l'index du premier élément est 0. Les nombres négatifs permettent de parcourir les tableaux de la fin vers le début; -1 représente la valeur précédente. Les variables peuvent également pointer des fonctions. L'utilisation est la suivante:

```
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again"
>>> mylist[-1] = 3.14
>>> mydict = {'Key 1': 'Value 1', 2: 3, 'pi': 3.14}
>>> mydict['pi'] = 3.15
>>> mytuple = (1, 2, 3) # une fois créé, un tuple ne peut être modifié.
>>> myfunction = len
>>> print myfunction(mylist)
3
```

Vous pouvez accéder a un ensemble de valeurs consécutives en séparant les index par deux-points (:). Lorsque le premier index est laissé vide, on parle implicitement du premier, lorsque l'index de fin est vide, on parle du dernier. Ci-dessous un exemple:

```
>>> mylist = ["List item 1", 2, 3.14]
>>> print mylist[:]
['List item 1', 2, 3.1400000000000001]
>>> print mylist[0:2]
['List item 1', 2]
>>> print mylist[-3:-1]
['List item 1', 2]
>>> print mylist[1:]
[2, 3.14]
```

6 Les Chaines

Le chaines peuvent utiliser les simples ou les doubles quotes. Vous pouvez inclure un type de quote dans des quotes d'un autre type (ex: "Il a dit 'Salut'"). Les chaines multilignes sont entourés de 3 doubles (ou simples) quotes. Python supporte directement Unicode avec la syntaxe suivante: u"Ceci est un String Unicode". Pour remplir un string avec des valeurs, vous devez utiliser l'opérateur % et un tuple. Chaque %s du String est remplacé par un item du tuple, ceci de gauche à droite. Vous pouvez également utiliser un dictionnaire de substitutions. Exemple:

```
>>>print "Nom: %s\nNuméro: %s\nChaine: %s" % (myclass.name, 3, 3 * "-")
Name: Poromenos
Number: 3
String: ---

strString = """Ceci est
une chaine
multilignes."""

# ATTENTION: ne pas oublier le s dans "%(key)s".
>>> print "Ceci %(verbe)s un %(nom)s." % {"nom": "test", "verbe": "est"}
Ceci est un test.
```

7 Instruction de contrôle de flux

Les instructions de contrôle de flux sont: **if, for** et **while**. Il n'y a pas de switch, il faut utiliser if à la place. Utiliser **for** pour parcourir les valeurs d'une liste. Pour obtenir une liste de valeurs, utilisez **range(nombre)**. La syntaxe des instructions est la suivante:

```
rangelist = range(10)

>>> print rangelist

0, 1, 2, 3, 4, 5, 6, 7, 8, 9
for number in rangelist:
    # Vérifie si 'number' est
    # présent dans le tuple

    if number in (3, 4, 7, 9):
        # "Break" termine la boucle for sans
        # exécuter le contenu de l'instruction "else".
        break
    else:
        # L'instruction else est optionnelle. Le contenu de celle-ci
```

```
        # est exécuté seulement si la boucle n'est pas stoppée par le "↩
            break"
        # "Continue" permet de passer à la prochaine itération
        # de la boucle. Il est plutôt inutile ici car c'est,
        # la dernière instruction de la boucle.
        continue
else
        pass # ne rien faire

if rangelist[1] == 2:
    print "deuxième item (la premier index d'une liste est 0) est 2"
elif rangelist[1] == 3:
    print "deuxieme item est (la premier index d'une liste est 0) est 3"
else:
    print "Je sais pas"

while rangelist[1] == 1:
        pass
```

8 Les Fonctions

Les fonctions sont déclarées avec le mot clé **"def"**. Les arguments possibles sont définis dans la déclaration de la fonction. Ceux-ci sont placés après les arguments obligatoires et sont initialisée avec des valeurs par défaut. Pour les arguments nommés, le nom de l'argument est assigné à la valeur. Les fonctions peuvent retourner un tuple. Une fonction lambda doit au minimum contenir une instruction. Les paramètres sont passés par référence mais les types non-mutables (ndt: tuples, int, strings etc. en programmation objet, les types non-mutables sont des objets dont on ne peut modifier l'état une fois qu'ils ont été créés) ne peuvent être changés. Ceci est du au fait que l'emplacement mémoire de l'élément est passé et lier un autre objet a une variable remplace l'ancien objet. Les types non-mutables sont donc remplacés. Par exemple:

```
# Même chose que: def f(x): return x + 1
functionvar = lambda x: x + 1
>>> print functionvar(1)
2

# an_int et a_string sont optionnels, ils ont des valeurs par défaut
# au cas ou l'une d'entre elle ne serait pas passée
#(respectivement 2 et "chaine par defaut").
def passing_example(a_list, an_int=2, a_string="chaine par defaut"):
    a_list.append("nouvel item")
    an_int = 4
    return a_list, an_int, a_string

>>> my_list = 1, 2, 3
>>> my_int = 10
```

```
>>> print passing_example(my_list, my_int)
([1, 2, 3, 'nouvel item'], 4, "chaine par defaut")
>>> my_list
[1, 2, 3, 'nouvel item']
>>> my_int
10
```

9 Classes

Python supporte une forme limitée d'héritage multiple entre classe. Des variables et des méthodes privées peuvent être déclarées (ceci est une convention, le langage Pyhton en lui-même n'impose rien) en ajoutant au minimum deux underscore avant le nom choisi (ex: "spam").

```
class MyClass:
    common = 10

    def __init__(self):
        self.myvariable = 3

    def myfunction(self, arg1, arg2):
        return self.myvariable


# Instanciation de la classe
>>> classinstance = MyClass()
>>> classinstance.myfunction(1, 2)
3
# Cette variable est partagée entre toutes les classes.
>>> classinstance2 = MyClass()
>>> classinstance.common
10
>>> classinstance2.common
10
# Notez l'utilisation du nom de la classe
# à la place de l'instance.
>>> MyClass.common = 30
>>> classinstance.common
30
>>> classinstance2.common
30
# Ceci ne modifiera pas la variable de la classe.
>>> classinstance.common = 10
>>> classinstance.common
10
>>> classinstance2.common
30
>>> MyClass.common = 50
# Ceci n'a pas changé car "MyClass.common" est
# une variable d'instance
>>> classinstance.common
```

```
10
>>> classinstance2.common
50

# Cette classe herite de MyClass. L'héritage
# est déclaré de la façon suivante:
# class OtherClass(MyClass1, MyClass2, MyClassN)
class OtherClass(MyClass):
    # L'argument "self" est passé automatiquement
    # et fait référence à l'instance de la classe. Vous pouvez donc
    # définir les variable d'instance comme ci-dessus mais depuis l'↩
        intérieur de la classe.

    def __init__(self, arg1):
        self.myvariable = 3
        print arg1

>>> classinstance = OtherClass("hello")
hello
>>> classinstance.myfunction(1, 2)
3
# Cette classe n'a pas de membre nommé .test. Nous pouvons
# tout de même en ajouter un à l'instance.
# Celui-ci sera membre de l'instance seulement.
>>> classinstance.test = 10
>>> classinstance.test
```

10 Exceptions

Les Exceptions en python sont gérées par des blocks de type try-except nom de l'exception. Exemple:

```
def some_function():
    try:
        # les division par zéro lèvent une exception
        10 / 0
    except ZeroDivisionError:
        print "Oops, division par zero!."
    else:
        # pas d'exception, tout va bien...
        pass

some_function()
Oops, division par zero!.
```

L'instruction finally: Le code contenu dans le bloc finally sera toujours exécuté, même si un exception est déclenchée dans le bloc try.

11 Imports

Les librairies externes sont utilisées à l'aide du mot clé import nom de la lib. Vous pouvez également utiliser: from libname import nom de la fonction pour importer seulement une fonction. Voici un exemple:

```
import random
from time import clock

randomint = random.randint(1, 100)
>>> print randomint
64
```

12 Lecture/écriture de fichiers

Python possède de nombreuses librairire. Comme exemple, voici comment la est utilisée la serialisation (conversion de structure de données avec la librairie pickle) avec des Entrées/Sorties fichiers.

```
import pickle
mylist = ["This", "is", 4, 13327]
# Ouvre le fichier C:\binary.dat en écriture. La lettre r avant le
# nom du fichier est utilisée pour empêcher l'échappement du
# caractère avec un "antislash".
myfile = file(r"C:\binary.dat", "w")
pickle.dump(mylist, myfile)
myfile.close()

myfile = file(r"C:\text.txt", "w")
myfile.write("Chaine exemple")
myfile.close()

myfile = file(r"C:\text.txt")
>>> print myfile.read()
'Chaine exemple'
myfile.close()
# Ouvre le fichier en lecture.
myfile = file(r"C:\binary.dat")
loadedlist = pickle.load(myfile)
myfile.close()
>>> print loadedlist
['This', 'is', 4, 13327]
```

13 Divers

- Les conditions peuvent être chaînées. $1 < a < 3$ vérifie que a est inférieur à 3 et supérieur à 1.
- Vous pouvez utiliser "del" afin d'effacer des variables ou des valeurs dans de tableaux.
- Vous pouvez manipuler et créer des listes de la manière ci-dessous (voir exemple), une instruction for initialise la liste.

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [3, 4, 5]
>>> print [x * y for x in lst1 for y in lst2]
3, 4, 5, 6, 8, 10, 9, 12, 15
>>> print [x for x in lst1 if 4 > x > 1]
2, 3
# Vérifie si un item à une propriété spécifique.
# "any" retourne true si une valeur de la liste correpond.
>>> any([i % 3 for i in [3, 3, 4, 4, 3]])
True
# Explication: 4 % 3 = 1, et 1 est true, donc any()
# retourne True.

# Compte le nombre d'item qui correspondent.
>>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
2
>>> del lst1[0]
>>> print lst1
[2, 3]
>>> del lst1
```

Le variables globales sont déclarées en dehors des fonctions avec le mot clé "global". Si vous ne le faîtes pas de cette manière, Python va affecter cet objet à une variable locale (attention à ça, ça peut vous faire perdre pas mal de temps). Exemple:

```
number = 5

def myfunc():
# Ceci affichera 5.
print number

def anotherfunc():
# Ceci lève une exception car la variable n'a pas été
# initialisée avant le "print". Python arrive à déterminer qu'une valeur
# sera affecté a cette variable plus tard dans le programme. Il crée donc ↩
    un nouvel
# objet de manière locale et l'utilise au lieu d'utiliser
# la variable globale
print number
```

```
number = 3

def yetanotherfunc():
global number
# Ceci changera la variable globale.
number = 3
```

14 Conclusion

Ce tutoriel n'est pas censé être une liste exaustive de tout ce que peut faire Python. Python dispose d'un grand nombre de librairies et vous découvrirez plein plein d'autres fonctionnalité en lisant d'autres livres, tels que l'excellent "Dive Into Python". J'espère avoir facilité votre transition vers Python. Merci de laisser vos commentaires si vous pensez que quelque chose peut être amélioré ou si vous souhaitez ajouter quelques chose à ce document (classes, gestion des erreurs ou autre...).