Python ? Vous en avez déjà entendu parler, vous savez peut-être qu'il s'agit d'un langage de programmation très en vogue, vous souhaitez le découvrir pour savoir s'il peut répondre à vos besoins ? Voyons cela ensemble. L'objectif de ce document n'est pas de faire de vous un expert en Python ni de vous former à la programmation de logiciels. Nous allons plutôt voir quelques bases de ce langage et des clés qui vous permettront d'aller plus loin si vous êtes mordu par Python.

Caractéristiques et domaines d'application

Le langage Python, dans ces premières versions était plutôt un langage de script dont l'objectif était, grâce à une syntaxe simple, la réalisation de tâches plus facilement qu'avec le Shell ou Perl. Dans ces dernières versions, il se rapproche beaucoup plus d'un langage de programmation complet et peut rivaliser aussi bien avec Java qu'avec PHP tout en gardant sa simplicité.

Parmi les caractéristiques importantes de Python on notera :

- Multiplateforme : Python peut être utilisé sur la plupart des systèmes d'exploitation (GNU/Linux, Microsoft Windows, DOS, MacOS, FreeBSD, AIX, OS/400, OS/390, VMS, BeOS, Palm OS, PlayStation, Symbian OS...)
- Licence compatible GNU/GPL (et même beaucoup moins restrictive)
- Orienté Objet : comme Java ou C++, toutes les caractéristiques d'un langage orienté objet sont présentes. Il peut donc être utilisé dans de gros projets informatiques où de nombreux développeurs doivent intervenir
- Extensible : suivant la plateforme sur laquelle il est utilisé, de nombreuses librairies sont disponibles. Ainsi, malgré sa portabilité, Python reste efficace. Par exemple, sous Windows, on pourra accéder à la base de registre, sous Symbian (téléphone Nokia) on pourra consulter les contacts ou les SMS reçus.
- Embarqué : les sources de Python ont été pensés pour rendre ce langage "embarquable" dans un autre logiciel. Ainsi, il est possible d'utiliser ce langage pour manipuler des objets propres à un logiciel. The Gimp en est un exemple intéressant.

Ces caractéristiques font de Python un langage de programmation de plus en plus utilisé dans des domaines très variés comme, par exemple, un serveur de messagerie instantanée, une application Web ou encore un logiciel de gestion intégré (TinyERP).

Installation

Sous Debian et distributions dépendantes (Ubuntu...)

Sélectionnez le paquet Synaptics **`python`**.
ou
En ligne de commande : **`$ sudo apt-get install python`**

Sous Windows

Téléchargez Python ici :
http://www.python.org/ftp/python/2.5/python-2.5.msi
et lancez son exécution.

Autres

Toutes les instructions sont là :
http://www.python.org/download/

Notre premier programme

Pour combler les impatients, voici enfin le premier programme en Python.

Avec votre éditeur de texte préféré (gedit, kate, notepad, vim...), créer un fichier que nous appellerons bonjour.py dont le contenu sera :

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  strPrenom = raw_input("Quel est ton prénom : ")
5  print "Bonjour", strPrenom, "!"
6
7  raw_input()
```

Exécutez ce programme en double cliquant dessus dans votre gestionnaire de fichier. Si nécessaire, choisissez "Lancer dans un terminal".

Analysons chaque ligne
Les lignes commençant par un # sont des commentaires, c'est donc le cas des lignes 1 et 2. Ces lignes ne sont pas exécuté à proprement parlé mais ce sont des commentaires particuliers qui respectent une certaine syntaxe.

La ligne 1 est une ligne standard pour les fichier de script Unix (ou Linux), elle indique où va se trouver l'interpréteur nécessaire à l'exécution du script. Elle est ignorée par l'interpréteur Python sous Windows et ne gène donc pas.

La ligne 2 précise quel est le jeu de caractères utilisé dans le script. Bien sûr il faut préciser une valeur en rapport avec le contenu du fichier. En règle générale, on peut dire que sous Linux, on indiquera "utf-8" et sous Windows "iso-8859-15"

La ligne 4 est composée de deux instructions : un appel de fonction et une affectation de variable. Une *variable* est un mot utilisé dans le programme pour symboliser une valeur qui peut changer au cours de l'exécution du programme. Une affectation de variable se fait à l'aide du signe égal (=).
Exemples :

```
maVariable = "Bonjour"
```

le symbole maVariable représente désormais la chaîne de caractères `Bonjour`. Dans le programme, je pourrai utiliser le symbole maVariable afficher le mot Bonjour.

Une fonction désigne un sous programme réutilisable dédié à une tache spécifique. Elle est utilisée en écrivant son nom suivi d'une paire de parenthèses pouvant contenir des paramètres. Les paramètres d'une fonction sont utilisés par la fonction pour adapter l'exécution du sous programme en suivant les besoins du programme appelant. Si plusieurs paramètres sont nécessaires, ils sont séparés par des virgules. Leur ordre est important.
Ici, la fonction raw_input() affiche la chaîne de caractères qu'il reçoit en paramètre et attend que l'utilisateur du programme saisisse des caractères au clavier et qu'il fasse "Entrée". Une fonction peut être utilisée dans une affectation de façon à ce que son résultat soit affecté à une variable :

```
strPrenom = raw_input("Quel est ton prénom : ")
```

la variable strPrenom représente alors le texte saisie par l'utilisateur du programme.

La ligne 5 est une instruction, c'est à dire un ordre simple donné au programme. Contrairement à la fonction, l'instruction ne prend pas de parenthèse et ne renvoie pas de résultat.

```
print "Bonjour", strPrenom, '!'
```

Affichage, les unes à la suite des autres, des chaînes de caractères. Ici, on remarque que les chaînes peuvent être des *constantes* comme "Bonjour" ou '!' mises entre guillements ou apostrophes ou des *variables* comme strPrenom.

La ligne 7 attend tout simplement que l'utilisateur fasse "Entrée" au clavier" avant que le programme ne se termine.

Quelques éléments du langage

Comme tout langage de programmation, Python demande au programmeur de respecter strictement une syntaxe. Et dans ce domaine, Python est assez exigeant, c'est ce qui lui permet d'être très lisible et très court (de 2 à 5 fois plus court qu'un programme équivalent en C++ ou Java)

Les variables

Comme vu dans notre premier programme, une variable représente une valeur qui peut changer au cours de l'exécution du programme. Le nom d'une variable doit respecté la règle suivante : il doit commencé par une lettre (sans accent) ou un souligné (_) puis éventuellement d'autres lettres, de chiffre ou de soulignés. Les variables commençants par un souligné ont une utilisation particulière, il est donc préférable de commencer par une lettre. Les majuscules et les minuscules peuvent être utilisées, mais doivent être respectées tout au long du programme (`uneVariable` et `unevariable` sont deux variables différentes).

Exemples : `ma_variable, uneAutreVariable, i, valeur1, valeur_2`...

Les variables font référence à des valeurs, ces valeurs sont *typées*, c'est à dire de types différents. Voici ces types :

- Chaîne de caractères : "bonjour", 'salut', "phrase avec un retour à la ligne \n"
- Booléen : True, False, 1 et 0
- Nombre entier : 1, 2007, 984646, 0, -273
- Nombre à virgule (le point est utilisé) : 3.14, 10.5, -4654.579823, 100.0
- Nombre complexe : 50j, 1.754J, -6.333j

Il est bon de prendre pour habitude de nommer correctement une variable en fonction du type de sa valeur. Je vous propose de préfixer le nom de vos variables respectivement par str, b, i, f et j. Cela permet de mieux comprendre le fonctionnement d'un programme.

Exemples : strNomDeFamille, bMarie, iAge, fTaille...

Conversion de type :

- `iNombre = int(strChaine)`
- `fVirgule = float(strChaine)`
- `strChaine = str(iNombre)`

Opérations

Il est bien sûr possible d'effectuer des opérations sur ces valeurs ou les variables qui les représentent.

Opérations mathématiques :
Addition +, soustraction -, multiplication *, division /, puissance **, modulo %
Exemples : `b=a+10, c=50*b-20, d=c/10, carre=d**2, reste=a%b`

Opérations sur les chaînes :
Concaténation : `phrase = mot1 + " " + mot2`
Longueur : `iLong = len(phrase)`
Formatage : "modèle"%(valeurs,...)
Exemple : "%s a acheté %d bonbons à %.2f euro" % (strPrenom, iBonbons, fPrix)

Découpage : `strVariable[indiceDebut:indiceFin]`

```
mot= "  A   B   C   D   E   F   G   H  "
      0   1   2   3   4   5   6   7   8  (indice positif)
     -8  -7  -6  -5  -4  -3  -2  -1      (indice négatif)
```

- premiereLettre = `mot[0]` => "A"
- sixiemeLettre = `mot[5]` => "F"
- troisPremieresLettres = `mot[0:3]` => "ABC"
- avantDerniereLettre = `mot[-2]` => G
- troisDerniereLettres = `mot[-3:]` => "FGH"
- longueur = `len(mot)` => 8
- lesDeuxLettresDuMilieu = `mot[len(mot)/2-1:len(mot)/2+1]` => "DE"

Instruction et bloc de contrôle

Notre premier programme s'exécutait de façon séquentielle ligne à ligne. Mais il n'est pas toujours possible de faire un programme de ce type, on doit parfois utiliser des conditions, des boucles, des tentatives...

L'instruction de condition : if

Elle permet de conditionner suite à un test, l'exécution d'un bloc d'instruction. Le bloc est matérialisé par une indentation (retrait à l'aide d'espaces ou de tabulation, ce retrait doit être le même pour toutes les lignes du bloc)

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  strPrenom = raw_input("Quel est ton prénom : ")
5  if strPrenom=="Marcel":
6      print "Bonjour Marcel !"
7      print "Je suis content de te revoir"
8  elif strPrenom=="Mauricette":
9      print "Bonjour Mauricette !"
10     print "Je t'attendais avec impatience"
11 else:
12     print "Bonjour", strPrenom, "!"
13     print "Enchanté de faire votre connaissance"
14 raw_input()
```

On remarquera que l'opérateur de test pour l'égalité est le double égal (==) très différent de l'affectation (=). Les autres opérateurs de comparaison sont :

- différent : a!=10 ou bien a<>10
- supérieur à : a>10
- supérieur ou égale à : a>=10
- inférieur à : a<10
- inférieur ou égale à : a<=10
- compris entre : 5<=a<=10
- strictement compris entre : 5<a<10

Ils peuvent être utilisé avec des chaînes de caractères, la comparaison porte alors sur l'ordre alphabétique.

Il est possible de faire plusieurs tests à la fois :

a<10 and b>50
a<10 or a>b
(a>10 and b>50) or (a<10 and a>b) and (strNom>="n")

L'instruction de boucle : while

Permet d'exécuter plusieurs fois un bloc d'instruction jusqu'à la véracité d'un test.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  iTentative = 0
5  strPrenom = raw_input("Quel est ton prénom : ")
6  while strPrenom!="Marcel":
7      iTentative = iTentative+1
8      if iTentative==10:
9          print "Nombre de tentatives maximum atteint"
10         break
11     print "Je ne te connais pas", strPrenom
12     strPrenom = raw_input("Quel est ton prénom : ")
13 else:
14     print "Bonjour Marcel !"
15 raw_input()
```

l'instruction break permet de sortir d'une boucle même si la condition du while n'est pas vérifiée. Dans ce cas, le bloc else n'est pas exécuté.

L'instruction de boucle : for

Permet d'exécuter un bloc d'instruction un nombre déterminé de fois.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  iMois = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
5  strMois = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai',
6  'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre',
7  'Novembre', 'Décembre']
8  m=0
9  for unMois in strMois:
10     print unMois, iMois[m]
11     m=m+1
12 raw_input()
```

for peut être utiliser avec plusieurs type d'objets : des tableaux, des fichiers...

Gestion des erreurs durant l'exécution (exceptions)
Il se peut que certaines instructions provoque des erreurs pendant leur exécution. Ces erreurs arrête le programme. Parfois, il peut être normal que ces erreurs surviennent dans la logique du programme sans qu'il soit nécessaire de l'interompre.

Exemple avec la division par zéro :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

iPoidsDuGateau = int(raw_input("Poids du gâteau : "))
iNombreDePersonnes = int(raw_input("Nombre de personnes : "))
try:
    iPart = iPoidsDuGateau/iNombreDePersonnes
    print "Chacun a droit à", iPart
except ZeroDivisionError:
    print "Personne ne veut du gâteau !"

raw_input()
```

Pour trouver le nom de l'exception (ici `ZeroDivisionError`), il suffit de faire un programme qui génère l'erreur (ou de ne pas mettre l'instruction try), le nom s'affiche alors au moment où le programme s'arrête.

Les modules

Python est fournit avec des modules complémentaires, la présence de chaque module dépend de la plateforme sur laquelle le programme s'exécute.

Pour utiliser les fonctions d'un module, il suffit de l'*importer*.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import random

iHasard = random.randint(0,100)
print "Voici un nombre entier compris entre 0 et 100", iHasard

raw_input()
```

Récréation

A vous de comprendre comment fonctionne ce programme :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Chargement de module
import random
import time

# Demande à l'utilisateur de son prénom
strPrenom = raw_input("Votre prénom : ")
print "Bonjour", strPrenom

# Initialisation du nombre à rechercher
iNbATrouver = random.randint(0,100)
bTrouve = False

print "Vous devez trouver un nombre entre 0 et 100 que j'ai caché !"
iTentative = 0

# Boucle demandant les différentes tentatives de l'utilisateur
while bTrouve==False:
	iTentative = iTentative+1
	iProposition = int(raw_input("Essai n°%d : " % iTentative))
	if iProposition==iNbATrouver:
		bTrouve=True
	elif iProposition>iNbATrouver:
		print "Trop grand !"
	else:
		print "Trop petit !"

# Réponse correcte
print "Bravo %s !"%strPrenom
print "Vous avez trouvé la réponse en %d coups" % iTentative

# Chargement des scores précédents
tScores = [];
try:
	fScore = open("score.txt", "r")
	for line in fScore:
		tScores.append(line[0:-1])
	fScore.close()
except IOError:
	# Cas d'impossibilité d'ouvrir ou de lire le fichier
	print "Création d'un nouveau fichier des meilleurs scores"

# Ajout du score de cette partie
horo = time.localtime()
strScorePartie = "%03d : %02d/%02d/%04d %02d:%02d : %s" % (iTentative,
horo.tm_mday, horo.tm_mon, horo.tm_year, horo.tm_hour, horo.tm_min, strPrenom)
tScores.append(strScorePartie)

# Tri des scores
tScores.sort()

# Affichage des scores
print "*** Meilleurs scores ***"
for line in tScores:
	print line
print "************************"

# Stockage des 5 meilleurs scores
```

```
# (ou du nombre de scores disponibles)
iNbScores = len(tScores)
iNbScores = min(5, iNbScores)
fScore = open("score.txt", "w")
for i in range(iNbScores):
      fScore.write(tScores[i])
      fScore.write("\n")
fScore.close()

# on attend que l'utilisateur tape Entrée
raw_input()
```

Aller plus loin

Ce petit cours vous aura peut-être donné envie d'approfondir le sujet. Si c'est le cas, vous trouverez avec ce même document d'autres documents libres de diffusion.

- Apprendre_a_programmer_avec_Python.pdf s'adresse à ceux qui débutent dans la programmation. Python est un très bon langage pour commencer. Vous découvrirez aussi toutes les possibilités offertes par la programmation orientée objet.
- Tutoriel_Python.pdf permet d'explorer le langage et tous ses concepts par l'exemple.
- Plongez_au_coeur_de_Python.pdf est destiné aux programmeurs qui pratiquent un autre langage et qui souhaitent connaître à fond les spécificités de Python.
- Python_Quick_Reference[english].html, malheureusement qu'en anglais, est un condensé complet de la syntaxe du langage Python. Il permet de retrouver rapidement comment utiliser telle ou telle fonction, structure... C'est le document que vous garderez sous le coude quand vous maîtriserez Python.

Licences

Le logo et la marque Python sont déposés par la Python Software Foundation

Jean-Pierre Morfin
G3L