Licence Sciences du Langage **Linguistique Informatique – L5NS6M2A**
Université Bordeaux 3 **2012 – 2013**

TD N^{o}2 Premiers pas en Python

---

1. Après vous être identifié sur la machine virtuelle (login : `etu` - mdp : `relai`) et avoir démarré le terminal, lancez l'interpréteur Python avec la commande `python3.2`.
   (a) Créez quatre variables différentes, contenant respectivement un nombre, une chaîne de caractères, une valeur de vérité, et une liste de plusieurs valeurs quelconques.
   (b) Augmentez de 1 (*incrémentez*) la valeur de la première variable.
   (c) Ajoutez la chaîne "aze" à la fin de la seconde variable, et stockez la chaîne obtenue dans une nouvelle variable.
   (d) Créez deux variables `premier` et `second` qui contiennent les deux premiers éléments de la dernière variable.
2. Quittez l'interpréteur Python avec la séquence `CTRL + D`. Créez ensuite un nouveau fichier vide (avec la commande `touch nom-du-fichier`) dont le nom se termine par `.py`.
   Lancez ensuite l'éditeur de texte `geany` avec la commande éponyme. Ouvrez le fichier que vous venez de créer, et entrez-y les instructions en langage Python qui effectuent les tâches demandées dans l'exercice 1.
   (a) Ajoutez à la fin du fichier une (ou plusieurs) instructions `print(...)` pour afficher la valeur de toutes les variables déclarées.
   (b) Définissez une fonction `exercice1()` qui exécute toutes les instructions entrées jusque là.
   (c) Effacez si nécessaire les instructions qui sont en dehors de la fonction, puis ajoutez une ligne à la fin de votre programme qui appelle cette fonction.
3. Enregistrez votre fichier, puis entrez dans le terminal l'instruction suivante :

   ```
   python3.2  nom-de-votre-fichier.py
   ```

   Cette méthode exécute d'une traite toutes les instructions de votre programme, et vous permet de modifier facilement des éléments de celui ci grâce à un éditeur de texte.

4. Ecrivez en haut à gauche de la feuille de TD le mot "papier". Ecrivez ensuite en bas à gauche le mot "papier". Réfléchissez ensuite aux questions suivantes :
   (a) Pouvez-vous déterminer si le mot écrit en haut à gauche est le même que celui en bas à gauche ?
   (b) Pour une machine (ou un philosophe), la réponse n'est pas si évidente. On pourrait par exemple arguer que l'un des mots est écrit en haut, et l'autre en bas, ce qui les distingue l'un de l'autre, et les rend différents ; ou encore que le trait qui les représente n'est pas rigoureusement identique. Que signifie réellement le fait d'être égal, pour deux mots ?
   (c) On adoptera pour la suite la définition la plus courante, à savoir que deux mots sont égaux si et seulement si ils sont formés par les mêmes lettres dans le même ordre. Sans remettre en question la notion habituelle d'égalité entre deux lettres, pouvez-vous décrire une procédure (un *algorithme*) pour savoir si deux mots sont égaux en suivant cette définition ?

5. Ajoutez à votre programme (juste avant l'appel de `exercice1()`) la définition de fonction suivante (faites attention à l'indentation - les barres verticales sont là pour vous aider, mais ne font pas partie de la fonction) :

```
# Cette fonction teste si deux mots sont égaux.
def mots_egaux (mot1, mot2):
| pos = 1
| while pos < len(mot1):
| | if mot1[pos] == mot2[pos]:
| | | pos = pos + 1
| | else:
| | | return False
| return True
```

   Comme l'indique le commentaire au début, cette fonction doit tester si ses deux arguments (supposés être des chaînes de caractères) sont égaux ou non.

   (a) Comment cette fonction détermine-t-elle si deux mots sont égaux ?

   (b) Testez cette fonction avec plusieurs paires de mots différents. Vous parait-elle fonctionner ?

   (c) La fonction provoque un bug si le second mot est plus court que le premier. Commençons par remplacer l'expression `len(mot1)` par `min(len(mot1), len(mot2))`. Cette expression choisit la longueur la plus courte pour ne pas "déborder" de l'autre mot. Essayez de deviner quelles peuvent être les conséquences de cette 'correction'.

   (d) Testez la fonction avec les mots "mouche" et "louche". Que se passe-t-il ? Corrigez la fonction pour réparer cette erreur. (Il suffit de changer un nombre !)

   (e) Testez la fonction corrigée avec les mots "sapin" et "sapinette". Que se passe-t-il ? Corrigez à nouveau la fonction. (Il y a plusieurs manières de faire, mais ajouter une ou deux lignes doit suffire.)

6. On considère la suite de termes suivante : "a", "cc", "non", "elle", "radar", "selles", "retâter", ... ; qu'ont-ils en commun ? [1]
   Les mots de ce type sont des palindromes. Essayez de trouver une définition précise et exacte pour la notion de mot palindrome.

   (a) Écrivez une fonction `palindrome (mot)` qui retourne `True` si la chaîne de caractères passée en argument est un palindrome, et `False` dans le cas contraire.

   (b) Testez votre fonction sur les mots de la liste précédente, et quelques autres pour faire bonne mesure. Pouvez-vous arriver à la "piéger" (trouver un mot pour lequel elle retourne une réponse fausse) ? Si oui, corrigez-la !

(Bonus) La phrase "Engage le jeu, que je le gagne." est considérée comme un palindrome. (Dans le cas des phrases, on ignore les majuscules, ponctuations, accents et espaces.) Est-elle reconnue comme telle par votre fonction ? Essayez de compléter votre fonction pour qu'elle accepte les palindromes au sens large...

1. Indice : essayez de les lire dans l'autre sens.

Annexe

L'interpréteur Python

L'interpréteur Python est une autre interface en ligne de commande. Il fonctionne de façon similaire au terminal UNIX, hormis le fait qu'il interprète les instructions qu'on lui donne d'après la syntaxe du langage Python. (Et non comme des commandes Linux.)

Assignation de variables

Pour enregistrer une valeur dans une variable (on parle d'*assigner* une variable), on écrit le nom de la variable qu'on veut créer ou modifier, suivi du symbole `=`, puis de la valeur qu'on veut donner à cette variable, comme dans les exemples suivants :

```
x = 4  # x prend la valeur 4
y = x  # y prend la valeur de x (donc 4)
x = x + 3  # x prend la valeur x+3 (donc 7)
x = x + 3  # x prend la valeur x+3 (donc 10)
```

Types de données

Les variables en Python peuvent contenir des données de n'importe quel type, mais l'ordinateur établit en interne une distinction entre ceux-ci. Par exemple :

`zero`	Est une variable qui porte le nom de 'zero'.
`"zero"`	Est la chaîne de caractères composée de 'z', 'e', 'r' et 'o'.
`0`	Est le nombre zéro.
`[zero, "zero", 0]`	Est une liste qui contient les trois éléments précédents.

Les principaux types de données reconnus par Python sont les nombres (0, 1, -5, 2.8, etc...), les chaînes de caractères ("a", "abc", "Bonjour tout le monde!", etc...), les valeurs de vérité (True et False) représentant une assertion vraie et fausse respectivement, et les listes (entre crochets) d'éléments, séparés par des virgules ([1, 2, 3], [42, [1, 2, 3], "fin de liste"], etc...). Il en existe d'autres, mais ceux la sont les plus importants.

Opérateurs

Python est capable d'effectuer diverses opérations et calculs sur les types de données existants, grâce à des opérateurs. En particulier (il en existe d'autres) :

+, -, *, /	Effectuent les opérations correspondantes sur des nombres.
+	Concatène (ajoute bout à bout) deux chaînes de caractères.
==	Retourne True (vrai) si ses opérandes sont égales, sinon False.
!=	Calcule l'inverse du précédent. (C'est une notation pour $\neq$.)
<, >, <=, >=	Teste si la valeur de gauche est inférieure/etc... à celle de droite.
not, and, or	Opérateurs logiques sur des valeurs de vérité.

Indices

Lorsqu'une de vos variables contient une chaîne de caractères ou une liste d'éléments, vous pouvez accéder à ses caractères ou ses éléments individuellement grâce à un opérateur spécial. La valeur du n-ième élément s'obtient en rentrant le nom de la variable, suivi de n entre crochets, comme illustré ci-dessous :

```
>>> mot = "bonjour"
>>> liste = [1, 2, 3, 4, 5]
>>> mot[0]
"b"
>>> mot[1]
"o"
>>> mot[2]
"n"
>>> mot[6]
"r"
>>> mot[-1]
"r"
>>> liste[3]
4
>>> liste[-3]
3
>>> liste[0] = 42
>>> liste
[42, 2, 3, 4, 5]
```

Notez que, bien que le système soit le même pour les chaînes et les listes, ces deux types restent très différents. (En particulier, vous ne pouvez pas changer la n-ième lettre d'une chaîne de caractères comme vous changeriez le n-ième élément d'une liste.)

Fonctions utiles

print()

La fonction `print` écrit à l'écran la valeur (ou les valeurs, séparées par des virgules) qui est donnée entre parenthèses. Dans l'interpréteur Python, elle a le même effet que d'écrire simplement le nom de la variable, mais lorsque vous exécutez un programme, le seul moyen d'afficher la valeur de vos variables est d'utiliser cette fonction.

len()

La fonction `len` calcule la longueur (length) de l'argument donné entre parenthèses, qu'il s'agisse d'une liste ou d'une chaîne de caractères.

Indentation

Le fait d'ajouter un nombre fixe (par exemple 4) d'espaces (ou une tabulation) avant le début d'une ligne s'appelle "indenter" une instruction. Un groupe d'instructions indentées forme un bloc, ce qui permet de marquer les instructions rattachées à une fonction, une boucle ou une condition.

Attention à garder une indentation cohérente dans vos programmes, car ils ne pourront pas être interprétés correctement dans le cas contraire.

Déclaration de fonctions

En Python, on peut définir des fonctions avec le mot-clé `def`, comme illustré ci-dessous :

```
def fonction ():
  instruction1
  instruction2
```

```
def autre_fonction (x, y):
  instruction3
  instruction4
  return x

instruction0
fonction()
a = autre_fonction(1, 2)
b = autre_fonction(3, 4)
instruction5
```

Le programme ci-dessus commence par déclarer deux fonctions (nommées `fonction` et `autre_fonction` respectivement). Il exécute ensuite l'instruction 0, puis le contenu de la première fonction (les instructions 1 et 2), puis il exécute une première fois les instructions 3 et 4 (avec 1 comme valeur de `x` et 2 comme valeur de `y`) et enregistre dans la variable `a` la valeur finale de `x`, et les exécute à nouveau aussitôt après (en changeant les valeurs de `x` et `y` par 3 et 4, et en enregistrant `x` dans `b`) ; enfin, il exécute l'instruction 5 finale.

Une fonction peut être appelée et réutilisée à tout moment après avoir été déclarée ; les valeurs listées entre parenthèses lors de l'appel d'une fonction sont affectées aux variables listées lors de sa définition, et le mot-clé `return` suivi d'une valeur remplace l'appel de la fonction par la valeur qui le suit. (Appelée valeur de retour.)

Conditions

Un programme Python peut choisir d'effectuer ou non une ou plusieurs instructions regroupées en bloc avec les mots-clé `if` (si) et `else` (sinon) en fonction d'une valeur de vérité comme suit :

```
if x == 4:
  print ("x vaut 4")
else:
  print ("x ne vaut pas 4")

if True:
  print ("Vrai est vrai. C'est une tautologie.")
```

Boucles

On peut répéter un bloc d'instructions en Python avec l'instruction `while` ; l'instruction teste la valeur de vérité qui suit et répète son bloc d'instructions aussi longtemps qu'elle reste vraie.

Une autre instruction, `for`, permet de répéter un bloc pour chaque élément d'une liste ou d'une chaîne :

```
x = 0
while x <= 10:
  print ("On compte jusqu'à 10 : ", x)
  x = x + 1

x = 1
mot = "bonjour"
for lettre in mot:
  print ("La ", x, "-ième lettre de ", mot, " est : ", lettre)
  x = x + 1
```