Initiation à Python

Étienne MIQUEY
emiquey@fing.edu.uy

1 Lexique

1.1 Variables, procédures, type

Python est un langage de programmation dit *impératif*, ce qui signifie que les programmes correspondent à des séquences d'instructions exécutées une par une par l'ordinateur. Cela correspond à l'idée intuitive d'un programme qui s'exécute pas à pas. Les deux objets principaux que nous allons manipuler sont des *variables* et des *procédures*. Une procédure correspond à l'idée de ce qu'est une fonction mathématique : c'est un programme qui prend des arguments, effectue des calculs et renvoie un résultat. Une variable représente un objet stocké dans la mémoire, qui peut être un nombre, une chaîne de caractères, une liste, une matrice, etc... À noter qu'en Python, la valeur d'une variable peut être modifiée au cours du temps, ce qui est tout aussi pratique que dangereux !

On déclare en général variables et fonctions de la manière suivante :

```
1 nom_variable=valeur_variable
2
3 def procedure(arg1,arg2):
4         instruction_1
5         instruction_2
6         return resultat
```

Par exemple, l'instruction `a=10` crée une variable `a` contenant l'entier 10, `b=3.` crée une variable `b` contenant le réel 3, et `mot="Bonjour"` crée une variable `mot` contenant le mot `'Bonjour'`.

En Python, l'indentation est cruciale, par exemple, dans le code suivant

```
1 def␣proc1(a):
2 ␣␣a=a*2
3 a=a+1
```

l'instruction `a=a+1` (ligne 3) n'appartient par à la procédure `proc1`. De fait, elle sera executée une seule fois (lorsque l'interpréteur arrive à cette ligne), mais ne s'exécutera pas lors d'un appel à `proc1`. Il faut donc toujours être extrêmement vigilant à l'indentation de votre code !

Question 1. Essayez de comprendre le comportement de chacune des instructions suivantes, puis vérifiez [1] votre réponse avec Python.

```
1  a=6.
2  b=5
3  c=2
4  a;b;c
5  def proc1(arg1, arg2):
6    c=arg1+arg2
7    c=c/2
8    return c
9
10 m1=proc1(a,b)
11 m2=proc1(b,c)
12
13 def proc2(arg1):
14    arg1=0
15
16 proc2(a)
17 a+b
```

1. *La déclaration de variable étant muette, pour connaître le contenu d'une variable (par exemple* `a`*), il faut en demander le contenu, simplement par la commande* `a` *(comme à la ligne 4 de cette exemple)*

1.2 Chaînes de caractères

Python est un langage dit "orienté objet", c'est-à-dire qu'un certain nombre de *classes* sont prédéfinies avec des procédures spécifiques appelées *méthodes*. L'idée est que chaque fois que l'on définit un *objet* (un élément) d'une classe, celui-ci hérite automatiquement des méthodes en question. La syntaxe pour appeler une méthode est la suivante : `objet.methode()`, qui appelle la procédure `methode` avec pour argument `objet`.

Le premier exemple que nous allons voir est celui des chaînes des caractères. Par exemple, dans la classe `string` sont prédéfinies des fonctions pour mettre en minuscule, mettre en majuscule, trouver une sous-chaîne, etc. Chaque fois que l'on définit une chaîne de caractères `mot`, celle-ci est automatiquement un objet de `string`, et possède donc toutes ses méthodes.

Question 2. Essayez les instructions suivantes afin de vous familiariser avec les chaînes de caractères, puis écrivez une procédure qui prendra en argument deux chaînes de caractères `mot1,mot2`, puis renvoie la chaîne formée de `mot1` en majuscule, un espace, 'et', un espace et `mot2` trois fois à la suite en minuscule.

```
a='Bonjour'
b='monsieur'
a[2];a[2:4];len(a)
b.capitalize(); a.upper()
a+b; a*3
'ou' in a; 'ou' in b
a.find('o'); a.find('ou')
a.replace('o','e')
```

1.3 Listes

Le deuxième exemple de classe que nous allons voir est celui des listes. Une liste correspond à une suite d'éléments, qui peuvent (en Python) être de différents types. Tout comme les chaînes de caractères, les listes sont donc des objets possédant plusieurs méthodes.

Question 3. Que font les instructions suivantes ? Vérifiez votre réponse à l'aide de Python.

```
l=[1,2,3,4,5]
len(l)
l[0]; l[5]
l[3]='a'
l+l; l
l=l*3; l
l.index(3)
l.append('Bonjour'); l
l.insert(3,[0,1]);l
l.remove(1);l
```

Question 4. Quelle(s) différence(s) notez-vous entre la liste `['B','o','n','j','o','u','r']` et la chaîne de caractères `'Bonjour'` ?

1.4 Type de données

En programmation, on associe à chaque objet un *type*, qui définit les valeurs qu'il peut prendre. Par exemple, le type booléen `bool` contient deux objets (`True` et `False`), le type "entier" `int` contient tous les entiers $0, 1, 2, ...$, le type des réels `float` contient les réels, etc... En Python, on a aussi des types pour chaque structure de données, par exemple `string` pour les chaînes de caractères, et `list` pour les listes. De manière générale, la commande `type(var)` vous donne le type de `var`.

Le point important est que la plupart des fonctions/procédures ne sont compatibles qu'avec certains types, et la définition peut changer suivant le type des arguments. Par exemple, l'addition est définie à la fois sur les entiers, les réels, les listes et les chaînes de caractères, mais dans les deux premiers cas elle correspond à l'addition usuelle et dans les deux derniers à la concaténation. En revanche, la division n'est définie que sur les entiers et les réels (et ne fait pas la même chose dans les deux cas : vous pouvez tester la différence entre `5/2` et `5./2.`), et Python renvoie donc une erreur si l'on essaie de diviser deux mots.

Python est un langage très souple dans le sens où il permet à l'utilisateur d'écrire beaucoup de choses sans jamais vérifier les types. Par exemple, on peut mettre à la fois dans une liste des entiers, des réels et des mots. Ce qui signifie aussi que c'est au programmeur d'être précautionneux, pour ne pas écrire des instructions qui n'ont pas de sens ou ne pourront pas s'exécuter.

2 Récursif vs itératif

2.1 Récursivité

Question 5. Sauriez-vous deviner ce que fait la fonction suivante ?

```
def mystere(a,b):
  if b==0:
    return 0
  else:
    return a+mystere(a,b-1)
```

On appelle *récursive* une procédure qui fait appel à elle-même. La récursivité correspond en programmation à l'idée de récurrence en maths : on explique à la procédure comment passer d'une étape à une autre et comment traiter les cas limites. Vous noterez au passage l'utilisation de l'instruction conditionnelle

```
if condition:
  instruction_1
else:
  instruction_2
```

Question 6. Écrire une procédure récursive `factorielle_rec` qui prend en argument un entier n et calcule $!n$.

Question 7. Écrire une procédure récursive `fibonacci` qui prend en argument trois entiers u_0,u_1 et n et renvoie les n premiers termes de la suite de Fibonacci [2] initialisée par u_0 et u_1, puis calculer `fibonacci(1,1,40)`.

2.2 Itératif

Question 8. Sauriez-vous deviner ce que font les fonctions suivantes ? Que signifient `while` et `for` ?

```
def mystere2(a,b):
  c=1
  for i in range(b):
    c=c*a
  return c

def mystere3(a,b):
  while a>0 :
    b=b+1
    a=a-1
  return b
```

Question 9. Écrire une procédure `factorielle_it` qui prend en argument un entier `n` qui calcule itérativement $!n$, en utilisant une boucle `for` ou une boucle `while`.

Question 10. Écrire une procédure `supp_num` qui prend en argument une liste d'entiers `l` et un entier `c`, et qui supprime toutes les occurrences de `c` dans `l`.

Question 11. Écrire une procédure `remplace` qui prend en argument un mot `m`, deux caractères `a` et `b`, et qui renvoie le mot `m` dans lequel `a` est remplacé par `b`, sans utiliser [3] aucune méthode des chaînes de caractères ni des listes.

3 Applications

3.1 Crible d'Ératosthène

Le crible d'Ératosthène est un algorithme destiné à lister tous les nombres premiers plus petits qu'une certaine valeur N. Le principe est le suivant : on suppose que l'on dispose d'un tableau qui contient tous les entiers de 2 à N, dont on va parcourir les cases par ordre croissant. Lorsque l'on arrive à la case i, si la case n'est pas rayée, on raye toutes les cases contenant des multiples de i (sauf la case i elle-même). Puis, dans tous les cas, on passe à la case $i+1$, et ainsi de suite jusqu'à N. Lorsque l'on a terminé, les cases non rayées sont exactement les nombres premiers plus petits que N

2. *Rappel : la suite de Fibonacci est définie par ses deux premiers termes u_1 et u_0 et la relation de récurrence $u_{n+2} = u_{n+1}+u_n$.*
3. *N'hésitez pas à demander des indices.*

Question 12. Sauriez-vous justifier la correction de cet algorithme ? A-t-on réellement besoin d'aller jusqu'à N ?

Question 13. Implémentation du crible :

1. Écrire une procédure `creer_liste` qui prend en argument un entier `n` et crée une liste contenant tous les entiers de 0 à n.
2. Écrire une procédure `elim_mult` qui prend en arguments un entier `n`, une liste `liste` de longueur $n+1$ et un entier i, et qui place un 0 dans les cases de `liste` correspondant à des multiples de i
3. Écrire une procédure `supp_elt` qui prend en arguments une liste d'entiers `liste` et un entier, et supprime toutes les occurrences de `a` dans `liste`
4. À l'aide des questions précédentes, implémenter le crible d'Ératosthène.

3.2 Suite de Syracuse

On appelle *suite de Syracuse* une suite d'entiers naturels définie de la manière suivante : on part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur. Par exemple, à partir de 11, on construit la suite des nombres : $11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2...$ C'est ce qu'on appelle la suite de Syracuse du nombre 11. La conjecture de Syracuse dit que la suite de toute entier strictement positif atteint 1.

Question 14. Écrire une procédure `syracuse` qui prendra en argument deux entiers n et l et renvoie `True` si la suite de Syracuse de n atteint 1 en moins de l étapes.

Question 15. À l'aide de votre procédure `syracuse`, écrire une procédure `conjecture` qui prendra en argument deux entiers n et l et qui testera si la conjecture est validée pour tous les entiers plus petits que n en moins de l étapes.

Question 16. Modifier la procédure `syracuse(n,l)` pour qu'elle renvoie la suite de Syracuse engendrée par n.

Question 17. (Difficile) Reprendre votre procédure `conjecture`, et tester à partir de quelle limite de n l'ordinateur prend du temps pour répondre. En vous inspirant du crible d'Ératosthène, sauriez-vous modifier vos procédures pour en améliorer l'efficacité ?

4 Appendice : quelques règles de base de la programmation

Brouillons Tout comme en maths ou en philo, il est (très) souvent utile de faire un brouillon de son code, à la main, avec un papier et un stylo. Il est très rare que l'on gagne du temps en se jetant sur le clavier sans avoir les idées claires. Un pseudo-code (c'est-à-dire du code "informel", comme nous l'avons fait jusque là ensemble) est un minimum vital.

Noms Vous allez avoir besoin de définir un certain nombre de fonctions, variables, etc... Si vous désirez pouvoir un jour être relu, par vous ou par qui que ce soit d'autre, pensez à utiliser des noms explicites et lisibles pour vos variables et fonctions. Une fonction nommée `ChercheUnChemin` ou une variable nommée `sommet` sont infiniment plus compréhensibles qu'une fonction `bli` ou une variable `toto`.

Commentaires Vous avez la possibilité d'écrire des commentaires au milieu de votre code, en commençant la ligne par #. Cela peut vous être utiles pour comprendre plus tard ce que vous avez fait ou indiquer ce qu'il manque, etc... :

```
def procedure(arg1,arg2):
  instruction1  #commentaires
  instruction2
  #on appelle la fonction auxiliaire
  return proc_aux(arg)
```

Débogage Lorsque votre code ne fonctionne pas, vous pouvez vous aidez de la fonction d'impression `print` pour afficher des valeurs intermédiaires. N'hésitez pas à en user et en abuser pour nettoyer votre code, c'est souvent assez utile.