IF5a – Ecriture d'un programme

Vers 1830, Charles Babbage conçoit une machine capable de faire mécaniquement des opérations élémentaires grâce à des cartes perforées. C'est pour cette machine que Ada Lovelace réalise le premier programme permettant de calculer des séries de nombres dites de Bernoulli.

$$\sum_{k=0}^{n-1} k^m$$

Objectifs :
- Aborder la notion d'effets de bord.
- S'habituer à préciser le type de données en entrées et sorties.
- Concevoir des assertions pour vérifier certaines conditions.

I – Instructions

I-1) Instructions

Une instruction est un morceau de code minimal qui produit un effet. Une instruction est exécutée par une machine. Une instruction simple peut s'écrire sur une seule ligne. On peut en écrire plusieurs séparées par des points-virgules. Outre l'expression et l'affectation, quelques instructions simples sont :

- Affirmer avec *assert*, qui est suivi d'une expression, (les assertions sont précisées plus loin dans ce chapitre) ;
- Renvoyer avec *return*, qui est suivi d'une expression et s'emploie uniquement dans une fonction ;
- Arrêter avec *break*, mot isolé qui permet d'arrêter une boucle ;
- Importer avec *import*, suivi d'un nom de module, ou de fonction, appartenant à un module qui est précisé.

Une instruction composée est une instruction sur une ligne terminée par deux-points suivie d'une ou plusieurs instructions indentées : on dispose par exemple de :

- *if* qui peut être suivi de *elif*, de *else* pour exécuter des instructions selon une condition ;
- *for* et de *while* pour exécuter des instructions de manière répétée ;
- *def* pour définir une fonction.

Certaines instructions nécessitent une expression, comme avec le signe « = », les mots *assert, if, elif, while*, qui sont suivis d'une expression, donc d'une valeur. Ces choix de conception du langage Python ont des conséquences à connaître.

Si par exemple on écrit assert x = 1, ou while x = 0, ou if (assert x > 0), ou return x = 0, cela provoque une erreur de syntaxe…

Cependant Python permet quelques simplifications d'écriture comme :

- if x%2 au lieu de if x%2==1 pour tester si un nombre est impair ;
- while x plutôt que while x !=0.

```
Entrée [9]: x=2
            while x:
                x=x-1
                print(x)

            1
            0
```

I-2) Effet de bord

On dit qu'une fonction a un effet de bord « side effect » si son exécution modifie quelque chose en dehors de ce qui est défini dans le corps de la fonction, par exemple un de ses paramètres ou une variable globale définie dans le programme.

Un effet de bord, est un effet secondaire qui peut être désiré 😊 ou pas ☹. Une fonction a un objectif principal, et la suite des instructions écrites pour atteindre cet objectif peut avoir des effets sur les objets manipulés. C'est en particulier le cas avec les fonctions qui manipulent des objets mutables comme les listes. Dans le code qui suit, la première fonction a un effet de bord, pas la seconde.

```
Entrée [55]: liste=[1,2,3] ; liste2=[1,2,3] ; x=4

             def ajoute(liste,x):
                 liste.append(x)
                 return liste

             def ajoute2(liste2,x):
                 liste2=liste2+[x]
                 return liste2

Entrée [56]: ajoute(liste,x)
             ajoute(liste,x)

  Out[56]: [1, 2, 3, 4, 4]

Entrée [57]: ajoute2(liste2,x)
             ajoute2(liste2,x)

  Out[57]: [1, 2, 3, 4]
```

On voit que « liste » est modifiée globalement alors que « liste2 » est modifiée localement. Ce qui entraîne un effet de bord dans notre fonction « ajoute ».

II – Annotation

II-1) Spécification d'une fonction

Une spécification permet d'informer les utilisateurs de la tâche effectuée par la fonction, de préciser les contraintes imposées pour les paramètres et ce qui peut être attendu des résultats. Elle peut aussi préciser les messages d'erreurs affichés en cas de mauvaise utilisation.

La spécification est destinée à l'utilisateur. Elle n'est pas utilisée par l'interpréteur « Python ». On la représente par des triple guillemets.

Si on souhaite obtenir des informations de l'interpréteur, on utilisera la fonction help.

```
help(divmod)
```

```
Help on built-in function divmod in module builtins:

divmod(x, y, /)
    Return the tuple (x//y, x%y).  Invariant: div*y + mod == x.
```

La spécification d'une fonction est écrite, comme l'est un commentaire dans un programme, à l'attention des utilisateurs qui ont besoin de savoir comment l'utiliser. L'objectif est de les éclairer, de les aider à saisir rapidement le rôle d'une ou plusieurs instructions. Un choix pertinent dans les noms de variables et de fonctions participe aussi à la compréhension d'un code. Voici comment définir une fonction avec sa spécification et quelques annotations (représentées ici par #) :

```
Entrée [67]: def permute(liste):
                 """liste est de type list
                 la fonction permute le premier et le dernier élément
                 et renvoie la nouvelle liste"""
                 n=len(liste)
                 copie=liste[:] #copie de la liste originale
                 copie[0],copie[n-1]=copie[n-1],copie[0] #permutation des valeurs
                 return copie

             help(permute)
             liste2=[1,2,3]
             permute(liste2)

             Help on function permute in module __main__:

             permute(liste)
                 liste est de type list
                 la fonction permute le premier et le dernier élément
                 et renvoie la nouvelle liste


  Out[67]: [3, 2, 1]
```

Il faut bien comprendre qu'une spécification est une sorte de contrat entre l'utilisateur et l'auteur du code. L'auteur garantit un résultat sous réserve d'une utilisation correcte précisée par la spécification.

II-2) Annotations et commentaires

Un programme doit pouvoir être lu et relu facilement par l'auteur mais aussi par quelqu'un qui découvre le programme. Il est important pour cela d'annoter certaines lignes de code ou des blocs d'instructions afin de préciser leur rôle. On utilise pour cela un commentaire qui est une ligne de texte précédée du symbole #.

Comme c'est le cas pour une spécification, un commentaire n'est pas utilisé par l'interpréteur Python. Un choix de structure ou de méthode par exemple peut aussi être précisé et expliqué à l'aide de commentaires.

III – Assertion

Une spécification permet d'éclairer sur les données en entrées, le type des valeurs autorisées, la plage de valeurs acceptées. On peut ajouter des instructions qui vont arrêter le programme en cas de mauvaises utilisation. Une assertion est l'affirmation qu'une propriété est vraie. Elle est composée du mot *assert* suivi d'une expression dont la valeur est interprétée comme une valeur booléenne. Si l'expression a la valeur True il ne se passe rien, sinon le programme est interrompu et un message d'erreur s'affiche *AssertionError*. Voici un exemple simple :

```
Entrée [2]: def inverse(x):
                """x est un nombre non nul de type float/int, on renvoie l'inverse de x"""
                assert x!=0
                return 1/x

            inverse(3)

   Out[2]: 0.3333333333333333

Entrée [3]: inverse(0)

            ---------------------------------------------------------------------------
            AssertionError                            Traceback (most recent call last)
```

Dans cet autre exemple, on vérifie que k est bien une clé présente dans le dictionnaire.

```
def fonction(dico,k):
    """dico est un dictionnaire, k est une clé de ce dictionnaire"""
    assert k in dico
```