“

Les tests peuvent servir à montrer la présence d’erreurs, pas leur absence.

”

Edsger Dijkstra

SPECIFICATIONS ET MISE AU POINT DE PROGRAMMES

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Prototyper une fonction
- Décrire les préconditions sur les arguments
- Décrire des postconditions sur les résultats
- Utiliser des jeux de test.

1. Généralités et bonnes pratiques

En tant que développeur débutant, il est indispensable de prendre les bonnes habitudes dès le départ. De nombreuses conventions se sont imposées au fil du temps devant la complexité, la taille croissante du code mais aussi parce que le partage du travail devient de plus en plus indispensable. L'accroissement de la mémoire a aussi permis d'améliorer la lisibilité du code. Et oui, il fut un temps où chaque octet était important et une variable à une lettre moins couteuse qu'une de 20, et où un commentaire prenait trop de place dans le fichier...

Aujourd'hui, la longueur du nom d'une variable ou d'un commentaire ne doit plus vous effrayer.

Votre code doit être facile à lire par une autre personne. Et cette autre personne peut très bien être-vous 2 ans plus tard.... on est parfois surpris de ne pas comprendre ce que l'on a bien pu faire !

Pour lutter contre cela, **il faut définir des noms de variables et de fonctions de manière explicite**. Les commentaires également sont essentiels. **En Python les commentaires commencent par un #.** Commenter ce n'est pas commenter chaque ligne mais plutôt indiquer les grandes étapes lorsque le code s'allonge et expliquer les lignes qui vous paraissent techniques. Il faut bien penser à mettre à jour vos commentaires si vous modifiez le code !

Il existe des commentaires qui ont un rôle spécifique dans une fonction. C'est le rôle du ***docstring***. Le *docstring* se place juste après la création de la fonction. Il commence et termine par trois guillemets. **Le *docstring* doit décrire le rôle de la fonction, puis les paramètres passés en arguments (type et rôle), ainsi que le type de ce qui est retourné.**

Exemple :

```
1  def mise_au_carre(x):
2      """
3      ----------------------------------------------------------------------
4      La fonction mise_au_carre(x) renvoie le carré du nombre x
5
6      paramètre en entrée :
7          x de type int ou float
8
9      en sortie : le résultat est de type int ou float suivant celui de x
10
11     ----------------------------------------------------------------------
12     """
13     return x**2
```

L'intérêt de ce *docstring* se trouve dans le fait que l'on peut l'appeler dans le code ou la console. On l'appelle avec le nom de la fonction suivi de la méthode **.__doc__**

Exemple :

```
16 print(mise_au_carre.__doc__)
```

Résultat dans la console :

```
In [5]: runfile('C:/Users/J.C. Bonnefoy/Desktop/COURS/NSI/1ere/17_specification/
carre.py', wdir='C:/Users/J.C. Bonnefoy/Desktop/COURS/NSI/1ere/17_specification')

    ----------------------------------------------------------------------
    La fonction mise_au_carre(x) renvoie le carré du nombre x

    paramètre en entrée :
        x de type int ou float

    en sortie : le résultat est de type int ou float suivant celui de x

    ----------------------------------------------------------------------
```

2. Mise au point de programmes

Un programme fonctionne très rarement du premier coup, il faut généralement le déboguer. Il y a plusieurs types de bugs. Le plus simple : une erreur de syntaxe : vous avez fait une faute de frappe et le programme s'arrête et vous indique où est l'erreur.

Parfois c'est plus grave, un dépassement dans une liste mais là aussi on sait au moins où l'erreur se produit

Le pire c'est lorsque l'erreur ne se produit qu'occasionnellement ou que le programme va bien jusqu'au bout mais en faisant n'importe quoi. Dans de tels cas, l'algorithme élaboré n'est pas bon et n'a pas tenu compte de toutes les contraintes.

Lorsque l'erreur est occasionnelle, la première difficulté et de reproduire l'erreur pour comprendre d'où elle vient. **Des jeux de tests variés et biens choisis peuvent permettre d'aider**. Cette partie fera l'objet d'un développement en classe de terminale.

Les environnements de développement ont des débogueurs intégrés. Ces derniers permettent de définir des points d'arrêt, d'observer les valeurs des variables (en passant la souris dessus), d'afficher la pile d'appels et d'exécuter le programme pas à pas (il existe des icones spécifiques).

Sinon, il reste la méthode à l'ancienne... (utilisée à de nombreuses reprises cette année 😉) : C'est-à-dire afficher les valeurs des variables ou des messages aux endroits sur lesquels on a des soupçons. Il faudra juste penser à nettoyer votre code ensuite !