16.1 Validation d'un programme

NSI terminale - JB Duthoit

16.1.1 Coder de façon lisible et claire (spécifications)

Votre code doit être facile à lire par une autre personne. Et cette autre personne peut très bien être-vous deux ans plus tard....on est parfois surpris de ne pas comprendre ce que l'on a bien pu faire - :) !
Pour bien comprendre l'intérêt, il faut garder à l'esprit que vous passerez plus de temps à relire votre code qu'à l'écrire !

Généralités

Votre code doit :

- Beau (c'est mieux que laid :-))
- Explicite
- Simple (c'est mieux que complexe)
- Complexe mieux que compliqué
- Aéré
- Documenté. En Python les commentaires commencent par un # .

En python

Vous trouverez tous les détails sur le site officiel de Python.
En résumé :

- En python ne mélangez jamais tabulations et espaces
- Une ligne de code ne devrait pas dépasser 79 caractères.
- Encoder en Utf-8
- Importer les bibliothèques en début de programme sur des lignes séparées.
- Il y a de nombreuses conventions sur les espaces :
  - On met un (et un seul) espace avant et après les affectation, comparaisons, booléens et opérations
  - On ne met pas d'espace pour le reste (, { ,[...
- Pour les fonctions, variables et méthodes tout en minuscule avec _ pour séparer les mots.
- Pour le nom des classes : majuscule pour chaque nouveau mot et mots collés. (Exemple : MaClasse)
- En ce qui concerne la documentation à l'intérieur des fonctions :

- Lorsque vous créez une fonction, vous devez la documenter. C'est le rôle du docstring. Le docstring se place juste après la création de la fonction par def. Il commence et termine par trois guillemets "
- Votre 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
- On accède au docstring en tapant :

```
nom_de_ma_fonction.__doc__
```

16.1.2 Pourquoi faire des tests ?

☞ Parce que l'erreur est humaine !
Il est bien sûr préférable de détecter les bugs au stade du développent ! Cela évite :

- Une perte de temps
- Un perte de l'image de marque
- un coût de développement (reprise du code, compréhension, correction)

16.1.3 Comment et quand vérifier un programme ?

Quand ?

Il existe deux moments particuliers :

- A l'écriture du code
- A l'exécution

Comment

A l'écriture du programme, il convient de commenter son code.
A l'exécution : ce moment est malheureusement trop souvent ignoré ou négligé... Détaillons !

16.1.4 Les assertions

Une assertion permet de vérifier une condition . Si cette condition est vraie, alors l'assertion sera muette. Si elle est fausse, alors une erreur sera produite.
Voici un exemple :

```
def racine_carree(a):
        assert a >= 0, "Le nombre doit être positif"
        return a ** 0.5
```

```
>>> racine_carree(-4)
```

renvoie le message d'erreur

Remarque

Même si la condition est évidente, il est bien d'utiliser les assertions. Au contraire, plus la condition est évidente, plus on a intérêt à l'utiliser ! Si votre programme plante et que vous n'avez pas placé d'assertions, vous risquez d'aller chercher l'erreur ailleurs pendant un certain temps.
Si l'assertion signale l'erreur, on le sait tout de suite :-)

16.2 Tester son programme

NSI terminale - JB Duthoit

16.2.1 Avec un jeu de tests pertinents

16.2.2 avec une fonction