1 Intro Python

Il existe de multiples langages de programmation et cette année nous utiliserons principalement Python. Nous aborderons aussi un peu Javascript.
Un langage de programmation permet d'implémenter des algorithmes de façon concrète. On peut écrire le même algorithme dans différents langages de programmation.
Python est un langage interprété et il est donc nécessaire d'installer un interpréteur sur l'ordinateur pour faire fonctionner un programme Python, fichier avec l'extension *.py.
Nous utiliserons aussi beaucoup l'environnement des cahiers Jupyter (**notebook**, fichier d'extension *.ipynb) qui permettent de mélanger des cellules interprétées en Python et des cellules de texte au format markdown (*.md).

2 Écrire une fonction en Python

Un programme complexe est construit à partir de différentes fonctions qui effectuent chacune une tache précise, assez simple.
L'écriture d'une fonction repose sur le mot clef `def`.
Une fonction nécessite la plupart du temps des paramètres d'entrée qui sont les données sur lesquelles travailler.
Et le résultat de sortie d'une fonction est obtenue avec le mot clef `return`.

Exemple simpliste :

```
def somme(a, b):
  return a + b
```

Après avoir définit une fonction, on peut appeler ailleurs dans le programme cette fonction pour qu'elle effectue sa tache élémentaire.

Exemples : `somme(-3, 7.2)` vaut 4.2.

Après la suite d'instructions suivantes, `b` vaut 3.

```
a = somme(2, 4)
b = 9 - a
```

3 Spécification d'une fonction

La **spécification** d'une fonction consiste à définir ce que doit faire la fonction, ainsi que les **préconditions** sur les données d'entrée (leur type et leur domaine de valeur en particulier), et quelles sont les **postconditions** sur les données de sortie.

Exemple : la fonction `rapport` doit renvoyer le rapport de 2 nombres a et b.
- préconditions : les entrées sont 2 nombres flottants (définition à voir par ailleurs) a et b. Et le nombre b n'est pas nul.
- postcondition : la fonction renvoie le rapport a/b (nombre flottant).

Application :

```
def rapport(a, b):
  """ Renvoie le rapport a/b :
  - entrée : a et b sont deux nombres, b différent de 0
  - sortie : nombre a/b.
  """
  return a / b
```

On peut contrôler les préconditions ou postconditions d'une fonction avec le mot clef `assert`. Cela permet de lever une exception d'Assertion (AssertionError) si la condition se révèle fausse.

```
1 def rapport(a, b):
2   """ Renvoie le rapport a/b :
3   - entrée : a et b sont deux nombres, b différent de 0
4   - sortie : nombre a/b.
5   """
6   assert type(a) == float or type(a) == int, "a doit être un nombre"
7   assert type(b) == float or type(b) == int, "b doit être un nombre"
8   assert b != 0, "b ne doit pas être nul"
9   return a / b
```

L'ajout des ligne 6-7-8 permet d'obtenir un message d'erreur explicite lors d'un appel de la fonction qui ne respecte pas les préconditions sur a ou b (en particulier sa non nullité).

On peut aussi contrôler des postconditions sur les valeurs de sortie en **testant des appels particuliers pour les fonctions créées.**
Exemple : on peut vérifier le bon fonctionnement de la fonction `rapport` sur 2 ou 3 cas :

```
assert rapport(10, 2) == 5
assert rapport(1, -1) == -1
```

4 Utilité et limite des tests

Effectuer des tests est très important pour **essayer de contrôler le bon fonctionnement d'un programme**, mais il est **impossible de prouver la correction d'un programme seulement avec des tests**. En revanche, si un test échoue, cela assure que le programme a un problème !
La qualité d'un jeu de tests est importante pour essayer de prévoir tous les cas possibles qui pourraient intervenir dans l'utilisation d'un programme.

Remarque : comme on le verra par ailleurs, il est très dangereux d'effectuer des tests d'égalité sur les nombres flottants. On préférera toujours vérifier qu'un flottant est très proche d'un nombre.
Exemple : Voici comment tester que `rapport(1, 10)` vaut bien 0.1 :

```
precision = 0.000000001 # on peut choisir la précision désirée
assert rapport(1, 10) - 0.1 < precision
```