10.1 Spécification

10.1.1 Coder de façon lisible et claire

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__`

10.1.2 Les commentaires

Les commentaires doivent être des phrases complètes.
En Python les commentaires commencent par un # .
Pensez à mettre à jour vos commentaires si vous modifiez le code !
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.

10.1.3 La documentations dans les fonctions

Intéressons-nous maintenant à la documentation des fonctions.
Lorsque vous créez une fonction, vous devez la documenter. C'est le rôle du `docstring`, qui se place juste après la création de la fonction par `def`. Il commence et termine par trois guillemets " et se termine également 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.
Ce docstring, peut être lu en tapant : `nom_de_ma_fonction.__doc__`

```
def mettre_au_carre(x):
  """renvoie le carré de x

  paramètres nommés:
  x -- un nombre quelconque

  """
  return(x * x)
```

10.1.4 Nommage des objets

Pour différencier les objets des fonctions, variables et méthodes (des objets) on utilise deux conventions de nommage différentes :

- `nom_de_ma_fonction` pour les fonctions, variables et méthodes tout en minuscule avec _ pour séparer les mots.
- `MaClasse` pour le nom des classes(en terminale). Majuscule pour chaque nouveau mot et mots collés.
- Les constantes sont entièrement en majuscule : `NOM_DE_MA_CONSTANTE`. Une constante est une variable à laquelle on donne une valeur qui ne changera pas dans tout le programme. Par exemple, on code un jeu et au début, on préfère s'en tenir à deux joueurs. On peut

créer `NB_MAX_JOUEURS = 2` et on utilisera cette variable dans tout le programme sans jamais la changer. Imaginons que le code permette de jouer à trois joueurs. Il suffit de changer la valeur de la constante `NB_MAX_JOUEURS = 4`.

⚠ Il est indispensable de choisir des noms de variable explicite. On préférera `count = 0` plutôt que `x = 0` dans le cas où la variable désigne un compteur par exemple.

Exercice 10.1

La mesure d'un angle peut être donnée en degrés ou en radian. L'unité de calcul naturelle pour les angles est le radian.
✔ La bibliothèque math ne sait donc calculer le cosinus que d'un angle donné en radian.
`ma_fontion1`, convertit des degrés en radians car pi radians correspondent à 180 degrés.
Modifier les noms des fonctions, variables et créez le docstring selon les normes énoncées plus haut.

```
import math
def ma_fonction1(x):
  return(x*math.pi/180)

def ma_fonction2(x):
  return(math.cos(ma_fonction1(x)))
```

Exercice 10.2

Comprendre ce que fait le code suivant et corriger tous les problèmes de spécification.

```
def fonction(a,b):
  return(a*100/b)
```

Exercice 10.3

Comprendre ce que fait le code suivant et corriger tous les problèmes de spécification

```
import math
def fonction1(a,b,c,d):
  return(a+b+c+d)
def fonction2(a,b,c,d):
  return(fonction1(a,b,c,d)/4)
```

Exercice 10.4

Comprendre ce que fait le code suivant et corriger tous les problèmes de spécification

```
def fonction1():
  print("1 - jouer à la bataille navale")
  print("2 - jouer au puissance 4")
  print("3 - quitter")
  a=input("Taper votre choix 1 ou 2 ou 3:")
  while a!="1" or a!="2" or a!="3":
    a=input("Taper votre choix :")
  return(a)
```