Charte de codage

Les conventions de codage visent essentiellement à améliorer la lisibilité du code : elles doivent permettre d'identifier du premier coup d'oeil un maximum de choses dans le code, de se repérer facilement et de savoir où trouver les choses. Vous trouverez donc ici un ensemble de règles et de conseils à adopter durant le projet.

I. Conventions générales

Cette partie est à tous les langages utilisés.

A. Convention de nommage des fichiers :

- Pour chaque fichier il est essentiel de choisir un nom de fichier court et significatif.
- Le nom du fichier écrit en lettres minuscules.
- Si le nom de votre fichier est composé, on utilisera des underscores « _ » pour séparer les éléments.
- Nous nommerons vos versions de la façon suivante vX.Y. où X,Y∈ {1,..,9} :
  X : changement majeur de version Ajout, suppression d'une fonctionnalité ou restructuration du programme
  Y : changement mineur de version Ajout, suppression d'une fonction

B. Conventions de nommage des variables, fonctions et classes :

- Nos variables seront en minuscule. Si une variable qui contient plusieurs mots, nous les séparons avec des underscores « _ ».
- Nous éviterons les variables trop longues, peu compréhensibles et quasi-semblables. Exemple: this_is_a_variable et this_is_a_variable2.
- Le nom des fonctions seront en minuscule et seront séparé avec des underscores « _ » si elles contiennent plusieurs mots.

C. Commentaires du code et des fonctions :

- **Evitez les comm**entaires inutiles.
- Toutes les fonctions doivent être commentées, de façon à indiquer ce que prend la fonction en entrée et ce qu'elle retourne, suivie d'un petit résumé de ce qu'elle effectue.
- Les commentaires ne doivent pas dépasser 30% de la longueur du code.
- Nous indiquerons dans les commentaires ce que fait le code, pas comment il le fait (le code
  doit être suffisamment clair pour que le « comment » se lise tout seul).

D. En ce qui concerne le code :

- Chaque code doit être encodé en **utf-8**.
- Nous veillerons à bien indenter votre code ! Une bonne indentation du code permet de mieux s'y retrouver et d'éviter souvent de nombreuses erreurs.
- Aérez votre code avec des retours à la ligne. Cela améliore sa structure.
- Nous veillerons à toujours bien espacer vos variables de vos opérateurs.
- Nous veillerons à ne pas écrire des lignes de code trop longues. Sur une ligne, 80 caractères semblent être la limite actuelle pour ne pas utiliser la barre de défilement afin de voir la fin du code.
- Dans le cas d'une ligne qui dépasse 80 caractères, découpons-la après les opérateurs.
- Nous écrirons un code simple qui est facile à comprendre.
- Il faut préférer les méthodes simples et les fonctions qui prennent un petit nombre de paramètres. Évitons les méthodes et les fonctions qui sont longues et complexes.
- Évitons de mettre beaucoup d'idées sur une seule ligne de code.
- Veillons à ce que toutes les variables que nous avons définies ainsi que les packages que nous avons importés soient utilisées.
- Ne déclarons pas une variable qui n'est utilisée qu'une seule fois.

II. Python

- Nous devons sauter une ligne après chaque méthode.
- Nous devons sauter deux lignes après chaque classe.
- Préférons l'utilisation de la programmation fonctionnelle dans vos scripts. Il s'agit d'utiliser le plus possible des fonctions pour les actions qui sont répétées plusieurs fois.
- Assurons-nous qu'une fonction retourne quelque chose dans tous les cas. Attention que le seul « return » ne se situe pas dans un « if» dont la condition n'est pas toujours vérifiée.
- Veillons à ce que les imports de plusieurs modules soient sur plusieurs ligne.
- Pour importer nos modules, préférons des chemins relatifs.
  Exemple : « import sys » et non « from sys import * ».
- Veillons à ne pas mettre d'espace avant les deux points et les virgules, mais après.
- Veillons à ne pas mettre d'espace à l'intérieur des parenthèses, crochets ou accolades.
- Evitons de comparer une variable Booléenne avec True/False. Utilisez directement la variable.

Avec Spyder, nous pouvons afficher des alertes lorsque votre code viole une directive PEP8. Pour les activer, il faut aller dans Outils -> Preferences -> Editeur -> Introspection et analyse de code et cochez la case à côté de Real-time code sytle analysis (PEP8).

III. SQL server

En ce qui concerne SQL server :
Attention ce langage n'est pas sensible à la casse, c'est à dire qu'il ne distingue pas les majuscules des minuscules.

Le nom des tables doit :

- ✓ Toujours en minuscule ;
- ✓ Toujours au singulier
- ✓ Sans accents
- ✓ Sans abréviations
- ✓ Si c'est une table de jointure, l'écrire dans l'ordre alphabétique.

Le nom des tables ne doit pas être un mot réservé de SQL.

Le nom des colonnes doit être pour :

- ✓ Une clef primaire : id_ + nom de la table
- ✓ Un libellé : lib_ + nom de la table.

Nommons les contraintes des clés étrangères en commençant par «fk_» suivi du nom de la table fille puis de la table mère.

Nommons les contraintes des clés primaires en commençant par «pk_» suivi du nom de la table. Pour les autres contraintes, commencez par « ck_ ».

Définissons les contraintes au niveau de la table et non des colonnes.

Veillons à ce que tous les mots clé de SQL soit en majuscule.

IV. R

- S'assurer de travailler avec la dernière version de R et des packages que l'on emploie
- Utiliser un éditeur de code R, par exemple RStudio
- Tous les fichiers doivent être indépendants.
- Les librairies doivent être en début de ficher
- Veillons à ce que les appels au Library soient sur plusieurs lignes.
- Séparez les blocs d'instructions ou les fonctions par une ligne.
- Ne pas utiliser d'abréviations pour FALSE et TRUE
- Evitons de comparer une variable Booléenne avec True/False. Utilisez directement la variable.
- Assurons-nous qu'une fonction retourne quelque chose dans tous les cas.