CHARTE DE CODAGE

Les conventions de codage visent essentiellement à améliorer la lisibilité du code : elles doivent permettre d'identifier du premier coup d'œil 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 générale à tous les langages utilisés.

A- Convention de nommage des fichiers :

- Chaque nom de fichier doit commencer par l'identifiant du groupe, suivi d'un underscore et du nom de votre fichier.
  Exemple : g8_synonym.
- Il est essentiel de choisir un nom de fichier **court** et **significatif.**
- Le nom du fichier doit être en **anglais** et écrit en lettres **minuscule.**
- Si le nom de votre fichier est composé, utiliser des underscores « _ » pour séparer les éléments.
- **Sur Github :**
  - Nommez tous vos fichiers comme dans l'exemple : g3_predict_vX.Y.py

  - Nommez 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

    Z : Révision de la version précédente
    Correction de bug, optimisation d'une fonction

  Attention, sur le serveur, il ne faudra mettre que la version finale.

B- Convention de nommage des répertoires :

- Les répertoires sont déjà créés sur le serveur.
  Chaque groupe doit placer ses fichiers dans le bon répertoire en se référant à l'arborescence ci-dessous.
- Pour déposer les fichiers, il faut suivre le chemin /var/www/html/projet2018 sur le serveur.
- Chaque groupe doit placer ses fichiers dans le bon répertoire en se référant à l'arborescence donnée en annexe à la fin de cette charte.

C- Conventions de nommage des variables, fonctions et classes :

- Veillez à ne coder qu'en une seule langue : **l'anglais**. Veillez donc à ne pas utiliser de lettres accentuées.
- Veillez à écrire vos variables en **minuscule**. Si une variable qui contient plusieurs mots, séparez-les avec des underscores « _ ».
- Evitez les variables trop longues, peu compréhensibles et quasi-semblables. Exemple à ne pas reproduire : ceci_est_une_variable et ceci_est_une_variable2.
- Veillez à écrire vos constantes en **majuscule**. Si une constante contient plusieurs mots, séparez-les avec des underscores « _ ».
- Veillez à écrire le nom de vos fonctions en **minuscule** et séparez-les avec des underscores « _ » si elles contiennent plusieurs mots.
- Veillez à commencer le nom de vos classes en **majuscule**. Si une classe qui contient plusieurs mots, séparez-les en commençant chaque mot par une majuscule.

D- Commentaires du code et des fonctions :

- Commentez votre code **en anglais.**
- Un commentaire doit se trouver avant la ligne qu'il commente.
- Evitez les commentaires inutiles.
- **Toutes vos 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.
- Indiquez 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).

E- En ce qui concerne le code :

- Chaque code doit être encodé en **utf-8**.
- Chaque code doit commencer par un commentaire indiquant votre groupe, ainsi que les initiales du membre du groupe qui a réalisé ce code.
- Veillez 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.
- Veillez à toujours bien **espacer** vos variables de vos opérateurs.
  Exemple : `var = var1 + var2`
- Veillez 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, découpez-la après les opérateurs.
- Préférez écrire du code simple qui est facile à comprendre. Le code le plus simple n'est pas nécessairement le plus petit, soyez prêt à faire des compromis de taille en faveur de la simplicité.
- Préférez les méthodes **simples** et les fonctions qui prennent un petit nombre de paramètres. Évitez les méthodes et les fonctions qui sont longues et complexes.
- Évitez de mettre beaucoup d'idées sur une seule ligne de code.
- Veillez à ce que toutes les variables que vous avez définies ainsi que les packages que vous avez importés soient utilisés.
- Ne déclarez pas une variable qui n'est utilisée qu'une seule fois.

II. HTML/CSS

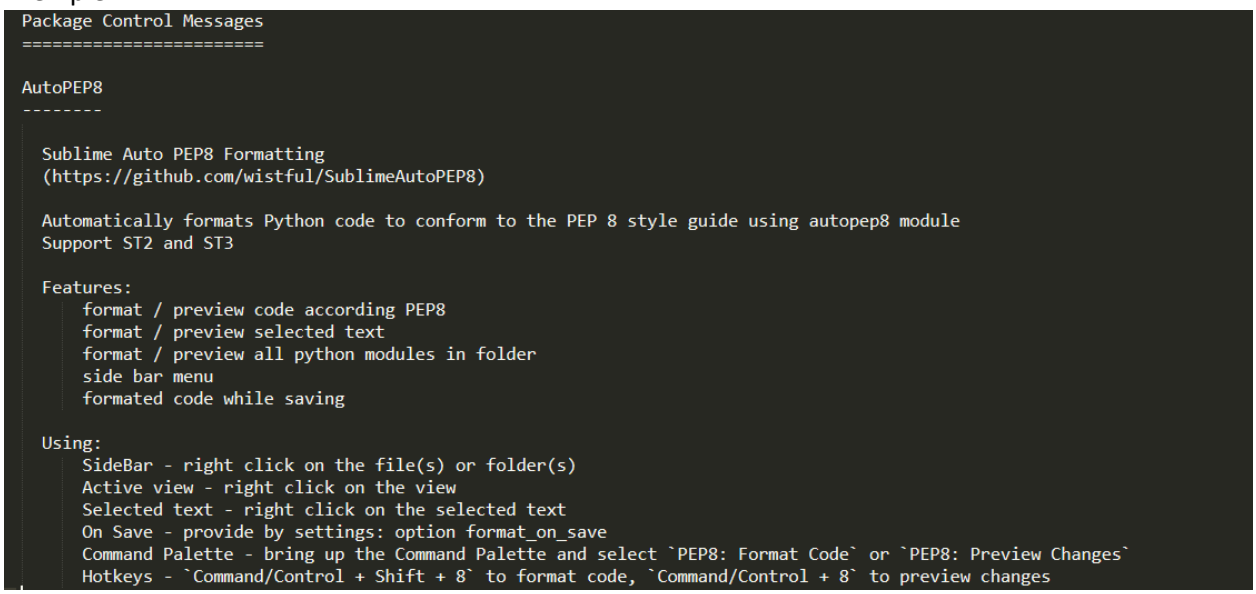
En ce qui concerne HTML :

- Veillez à commencer par un DOCTYPE sur chacune de vos pages pour que votre code HTML soit valide selon les règles de grammaire de ce dernier.
- Veillez à la correcte ouverture et fermeture des balises.
- Commentez votre code en utilisant la syntaxe suivante :
  < !--Voilà un commentaire>.
- Assurez-vous de l'unicité des valeurs de vos attributs « id » au sein d'une même page.

En ce qui concerne CSS :

- Séparez votre code CSS du code HTML.
- Veillez à ne pas nommer une classe en fonction de l'apparence qu'elle donne à l'élément. Car si vous changez l'apparence il faudra changer le nom aussi.
- Il est utile d'utiliser BEM (Bloc, Élément, Modifier) qui est une méthodologie de nommage qui tend à découper les entités de la page en blocs et éléments.

La validation W3C consiste à vérifier le code d'un site Web pour déterminer s'il respecte les normes de format. Vous pouvez installer sur Sublime text un package qui vous aide à le faire. Pour cela, sur Sublim Text -> CNTL+SHIFT+P -> Package Control : Install Package -> W3CValidators. Rendez-vous ensuite sur le document HTML ou CSS que vous voulez vérifier, dans le menu « Outils », accédez à W3C Validators, puis sélectionnez le type de votre document. Les résultats seront affichés dans un nouvel onglet.
Exemple :



III. Python

- Précédez-les fonctions et classes « protected » par « | » et celles « private » par «_».
  Exemple : def |hello_sid
  def _hello_sid
- Séparez les méthodes par une ligne blanche. Séparez les classes par deux lignes blanches.
- Préférez 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.
- Assurez-vous 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.
- Veillez à ce que les imports de plusieurs modules soient sur plusieurs ligne.
  Exemple :
  ```
  import sys
  import os
  ```
  Bien sûr, cela n'est pas valable pour « `from x import y,z` ».
- Pour importer vos modules, Préférez des chemins relatifs.
  Exemple : « `import sys` » et non « `from sys import *` ».
- Veillez à ne pas mettre d'espace avant les deux points et les virgules, mais après.
- Veillez à ne pas mettre d'espace à l'intérieur des parenthèses, crochets ou accolades.
- Evitez de comparer une variable Booléenne avec True/False. Utilisez directement la variable. Exemple :

```
if yes:
    #faites quelque chose
if no:
    #faites autre chose

#ou bien

if not yes:
    #faites quelque chose
```

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

IV. PHP

- L'utilisation des balises courtes d'ouverture **<? ?>** est déconseillée.
- Préférez l'utilisation des guillemets simples aux guillemets doubles.
- Utilisez True/False à la place de 0/1.
- Utilisez un point pour concaténer une chaîne de caractère, évitez de mettre un espace après le point.
- Veillez à revenir à la ligne après chaque déclaration, condition et boucle pour écrire votre instruction.
- Préférez les « if/else » aux « switch/case ».
- Ne pas utilisez d'espace :
  - Après les accolades, crochets et des parenthèses ouvrantes ;
  - Avant les accolades, crochets et des parenthèses fermantes ;
  - Avant les virgules, les points virgulent et les deux points ;
  - Avant la parenthèse ouvrante qui introduit la liste des paramètres d'une fonction ;
  - Avant le crochet ouvrant indiquant une indexation ou sélection.
- Utilisez un espace après chaque virgule.

V. Javascript

- Veillez à toujours déclarer vos variables avec « var » sinon la variable est attachée au contexte global de l'application, venant écraser des valeurs de variables potentiellement existantes.
- Veillez à ne pas oublier le point-virgule à la fin de chaque instruction.
- Ne déclarez pas de String sur plusieurs lignes, utilisez plutôt la concaténation de chaine à la place.
  Exemple :

```
var mystring = 'Ceci est un exemple ' +
    'pour illustrer notre point ' +
    'dans la charte de codage' +
    'et spécialement' +
    'en Javascript' +
    'Bonne lecture';
```

- Utilisez des guillemets simples pour les chaînes de caractères.
- Utilisez un espace avant la parenthèse ouvrante et après la parenthèse fermante.
- N'utilisez pas d'espace après une parenthèse ouvrante ou avant une parenthèse fermante.
- Revenez à la ligne après chaque accolade ouvrante, sauf si elle est immédiatement suivie d'une accolade fermante.
- Contrairement aux autres noms de fonction, commencez le nom des constructeurs par une majuscule.
- Pour nommer les accesseurs, utilisez des noms du type getValeur(), setValeur(val). Si la propriété est un booléen, nommez-les isValeur(), hasValeur().
- Lorsque vous devez déclarer plusieurs variables :
  - N'utilisez qu'une seule déclaration var ;
  - Déclarez chaque variable sur une nouvelle ligne ;
  - Déclarez en dernier les variables indéfinies.

- Ajoutez un underscore au début du nom de vos propriétés privées.
- Utilisez « _this » lorsque vous sauvegardez une référence à « this ».

VI. SQL

En ce qui SQL :

Attention ce langage n'est pas sensible à la casse, c'est à dire qu'il ne distingue pas les majuscules des minuscules.

- Veillez à ce que le nom des tables soit :
  - Toujours en minuscule ;
  - Toujours au singulier ;
  - Sans accents ;
  - Sans abréviations ;
  - Si c'est une table de jointure, l'écrire dans l'ordre alphabétique.
- Veillez à ce que le nom des tables ne soit pas un mot réservé de SQL.
- Veillez à ce que le nom des colonnes soit pour :
  - Une clef primaire : id_ + nom de la table ;
  - Un libellé : lib_ + nom de la table.
- Nommez les contraintes des clés étrangères en commençant par «fk_» suivi du nom de la table fille puis de la table mère.

- Nommez les contraintes des clés primaires en commençant par «pk_» suivi du nom de la table. Pour les autres contraintes, commencez par « ck_ ».
- Définissez les contraintes au niveau de la table et non des colonnes.
- Veillez à ce que tous les mots clé de SQL soit en majuscule.

ANNEXE : arborescence du répertoire.