Quelques bonnes pratiques de programmation (Python)

Traduction d'un problème en programme

1. *Analysez le problème*

- Identifiez clairement ce que sont les **données fournies**, ainsi que les **résultats et types** attendus à l'issue du traitement.
- Formalisez une **démarche générale de résolution** par une séquence d'opérations simples.
- Vérifiez que vous **envisagez tous les cas** de figures (en particuliers les cas "limites").

2. *Découpez votre problème en fonctions*

- Chaque fonction doit réaliser ***une* tâche** clairement identifiée.
- Limitez les fonctions à **25 lignes** maximum, sauf dans des cas exceptionnels.
- **Eviter la redondance** dans le code (copier/coller). Si cela arrive, c'est qu'il manque soit une fonction, soit une boucle, soit que des tests conditionnels peuvent être regroupés.
- N'utilisez **pas de variables globales**.
- Veillez à ce que tous les paramètres et variables d'une fonction soient **utilisés** dans cette fonction.
- Pour une fonction qui renvoie un résultat, organisez le code pour qu'il ne contienne qu'**un seul** `return`, placé comme dernière instruction de la fonction. Pour une fonction qui ne renvoie pas de résultat, ne mettez pas de `return` (il y en aura un implicitement à la fin de l'exécution de la fonction).
- Ne modifiez pas les *paramètres*.□ Exemple: Si vous recevez une borne inférieure `first` et une supérieure `last` et que vous devez itérer de la première à la dernière, n'incrémentez pas `first` dans la boucle, car la signification n'en serait plus claire ; créez plutôt une variable locale `pos` initialisée à `first`.
- Sauf si la fonction a comme but de modifier la (structure de) données reçue en paramètre ; dans ce cas la fonction ne renvoie pas de valeur.

3. *Testez le code au fur et à mesure du développement*

- Créez des **scénarios de test**, pour lesquels vous choisissez les données fournies et vous vérifiez que le résultat de la fonction est conforme à ce que vous attendez.
- Vérifiez les cas particuliers et les *conditions aux limites*. □ Exemples: Pour le calcul d'une racine carrée, que se passe-t-il lorsque le paramètre est un nombre négatif ?

Programmation

1. *Style de programmation*

- N'utilisez pas les instructions `break` ou `continue`
- Utilisez la forme raccourcie `if(is_leap_year(2008))` plutôt que la forme équivalente `if(is_leap_year(2008)==true)`
- Utilisez la forme `return <expression booléenne>` plutôt que la forme équivalente

```
if <expression booléenne>:
   res = true
else:
   res = false
return res
```

- N'exécutez pas plusieurs fois une fonction alors qu'une exécution suffit en retenant le résultat.
- Précisez le domaine de validité des paramètres.

Programmation (suite)

2. *Quelques erreurs classiques*

- Vous essayez d'utiliser une variable avant de l'avoir **initialisée**.
- L'**alignement des blocs** de code n'est pas respecté.
- Vous oubliez de fermer un fichier que vous avez ouvert.

Nommage de variables, fonctions, *etc.*

1. *Utilisez une convention de nommage*

`joined_lower`	pour les **variables** (attributs), et **fonctions** (méthodes)
`ALL_CAPS`	pour les **constantes**

2. *Choisissez bien les noms*

- Donner des noms de variables qui expriment leur contenu, des noms de fonctions qui expriment ce qu'elles font (cf. règles de nommage ci-dessus).
- Évitez les noms trop proches les uns des autres.
- Utilisez aussi systématiquement que possible les mêmes genres de noms de variables.
  □ Exemples: `i`, `j`, `k` pour des indices, `x`, `y`, `z` pour les coordonnées, `max_length` pour une variable, `is_even()` pour une fonction, etc.

Style et documentation du code

1. *Soignez la clarté de votre code*

... c'est la première source de documentation.

- Utilisez les `docstrings` dans chaque fonction pour :
  - brièvement décrire ce que fait la fonction, **pas comment elle le fait**, et préciser ses entrées et sorties.
  - Décrire les arguments des fonctions.
- Soignez les indentations (2 à 4 espaces chacune) et la gestion des espaces et des lignes blanches (deux lignes blanches avant et entre chacune des définitions de fonction globales ; une ligne blanche pour mettre en évidence une nouvelle partie dans une fonction),
- Il faut commenter le code à bon escient et avec parcimonie. Évitez d'indiquer le *fonctionnement* du code dans les commentaires. □ Exemples: Avant l'instruction "`for car in line:`", ne pas indiquer qu'on va boucler sur tous les caractères de la `line`...
- Évitez de paraphraser le code. N'utilisez les commentaires que lorsque la fonction d'un bout de code est difficile à comprendre.

Structure d'un programme Python

1. *Voir verso*