ENSG 1A – Remarques pour préparation de la colle info

Nous vous proposons quelques remarques pour vous aider à préparer la colle info. Ces remarques sont inspirées des erreurs rencontrées régulièrement dans vos programmes.

Regarder les corrections des exercices postés sur Arche

1. Essayer de refaire les exercices les plus courts.
2. Tenter de comprendre les autres et de les comparer à vos versions aux regards des remarques exprimées plus loin dans ce document.
3. N'hésitez pas à nous solliciter s'il y a des éléments que vous ne comprenez pas.

Savoir lire les messages d'erreur affichés par l'outil d'édition utilisé (IDLE ou pycharm ou …)

Une question dans ce sens pourrait tout à fait être prévue dans l'examen.

Les concepts fondamentaux à ne pas oublier

Fonctions

- Ne pas lire les paramètres dans une fonction comme ci-dessous :

```
def extractionAdresses(chaine, sep):
    chaine = input("Entrer la chaine a decryper : ")
    sep = input("Entrer le separateur utilise dans cette chaine : ")
```

- Aucune instruction n'est exécutée derrière un *return* :

```
def essai():
      print("je passerai par ici")
      return 1
      print("et je passerai par là")
```

On ne passera jamais par là et le print ne sera jamais exécuté….

3 grands types de fonctions/méthodes

- Les fonctions/méthodes qui retournent un ou plusieurs résultats :

  ⇒ **un return** (un seul)

  ⇒ récupération des résultats lors de l'appel de la fonction

```
def fct (par1, par2, …):
     …
     return res1, res2

# appel de la fonction
r1, r2, .. = fct(p1, p2 …)
```

- Les fonctions/méthodes qui ne font que de l'affichage (*print* ou *dessin turtle*)

  → pas de return

```
    def affiche(self):
        """ affiche un complexe """

        if self.pim_ > 0:
            print(self.pr_, "+i.", self.pim_)
        else:
            print(self.pr_, "-i.", -self.pim_)
```

- Les méthodes qui modifient des attributs d'un objet (*programmation objet vu au semestre 2*) **→ pas de return**

```
    def multiplie(self, facteur):
        """ modifie le complexe courant en le multipliant par facteur """

        self.pr_ *= facteur
        self.pim_ *= facteur
```

Appel d'une fonction/méthode

Un appel de fonction/méthode sans paramètre doit tout de même avoir des parenthèses :

```
     res = fct()              # par exemple pour une fonction sans argument
ou
     res = self.fct() # par exemple en programmation objet (semestre 2)
```

Ne pas faire le même travail plusieurs fois

C'est à dire ne pas appeler plusieurs la même fonction avec les mêmes paramètres …

Ecrire :

```
    def normalise(num, den):
        """
        :return: normalise la Fraction
        """

        # on s'appuie sur le pgcd pour trouver la forme normalisée
        le_pgcd = pgcd(num, den)

        num = num // le_pgcd
        den = den // le_pgcd
```

plutôt que la version ci-dessous, dans laquelle la fonction pgcd est exécutée deux fois :

```
    def normalise(num, den):
        """
        :return: normalise la Fraction
        """

        # on s'appuie sur le pgcd pour trouver la forme normalisée

        num = num // pgcd(num, den)
        den = den // pgcd(num, den)
```

Retourner une valeur pour tous les cas de figure

Par exemple, lorsqu'une fonction doit retourner *True* ou *False,* il faut systématiquement prévoir un return *True* **et** un return *False*.

Instructions conditionnelles

Ne pas tester plusieurs fois la même condition (ou son inverse)

Ecrire

```
if a != b :
     suite d'instructions
else:
     suite d'instructions
```

Plutôt que :

```
if a != b :
     suite d'instructions
elif a==b:
     suite d'instructions
```

Un else n'est pas obligatoire

Ecrire

```
if a != b :
     suite d'instructions
```

Plutôt que :

```
if a != b :
     suite d'instructions
else:
     continue
# ou encore pire
     False
```

Instructions itératives

Préférez quand c'est possible un for à un while

Ecrire

```
for i in range(n):
     suite d'instructions
```

Plutôt que :

```
while i<n:
     suite d'instructions
     i = i+1
```

Les limites d'un for doivent être entières :

```
for i in range(int(sqrt(n)):
     suite d'instructions
```

Les listes et matrices

Ajout d'un élément dans une liste

Ecrire : *ma_liste.append(v)*
Et non *ma_liste = ma_liste + [v]*

La seconde écriture est beaucoup trop longue car elle reconstruit à chaque fois une nouvelle liste.

Dimensions

Attention :

- les indices d'une liste l sont compris entre 0 et len(l)-1
- les indices d'une matrice M sont compris :
  - pour les lignes entre 0 et len(M)-1
  - pour les colonnes entre 0 et len(M[0])-1

Test d'appartenance

La suite d'instructions suivante qui permet de rechercher elt dans une liste l:

```
for elt2 in l:
     if elt == etl2:
          suite d'instructions
          break
```

peut être avantageusement remplacée par :

```
if elt in l:
     suite d'instructions
```

Initialisation d'une matrice

Quand on connaît les dimensions d'une matrice, on peut l'initialiser ainsi :

```
Matrice = [[0]*nbc  for i in range[nbl]]
```

Pour le semestre 2

Manipulation des fichiers

Ne pas confondre le nom du fichier et son "descripteur"

```
# si on écrit
fdata = open(filename, "r")

# dans la suite du programme, on n'utilise plus que fdata
# filename n'est utilisé dans dans le open
```

Etapes indispensables pour faire une lecture dans un fichier :

- Ouverture du fichier en lecture
- Lecture des lignes (*readline*() ou *for ligne in fichier*)
- Découpage de la ligne : *a, b, c … = lignesplit()*
- Conversion des variables lues dans le type adéquat, par exemple : *a = float(a)*

Programmation orientée objet

Les conventions de nommage :

- Majuscules pour un nom de classe
- att_ pour les noms d'attributs

Héritage

- Ne pas confondre deux classes qui héritent entre elles : attributs et/ou méthodes communes et mutualisées :
  - Par exemple un *Rectangle* est une *Forme_Geometrique*
- Et des classes qui dépendent l'une de l'autre :
  - Par exemple un *Cercle* a un centre qui est un *Point*

Affichage

- Quand on fait un simple *print(instance)* on n'affiche pas les valeurs des attributs mais seulement de quelle classe est issue l'instance et à quelle adresse elle est stockée.