

# Rappel sur les bonnes règles d'écriture du code

NSI TERMINALE - JB DUTHOIT

## Coder de façon lisible et claire (spécifications)

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)
- Complexé 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__
```

# 0.1

## Affectation

NSI TERMINALE - JB DUTHOIT

### Exercice 0.1

L'indice de Masse Corporel est un nombre réel utilisé en médecine. Sa formule est pour une masse en kilos et une taille en mètres :

$$IMC = \frac{\text{masse}}{\text{taille}^2}$$

1. Écrire en langage Python une fonction `masse_corporelle()` qui :

- Demande la masse de l'utilisateur en kg (nombre réel)
- Demande la taille de l'utilisateur en cm (nombre entier).
- affiche l'IMC de l'utilisateur

2. Écrire en langage Python une fonction `masse_corporelle_2(masse,taille)` qui a comme paramètre `masse` en kg et `taille` en mètres. La fonction renvoie l'indice de masse corporelle.

Testez votre programme avec différentes valeurs.

### Exercice 0.2

Écrire une fonction nommée `dire_bonjour(nom)` ayant comme paramètre une chaîne de caractère `nom` et qui affiche bonjour suivi du mot.

Exemple : `dire_bonjour("Paul")` affiche "Bonjour Paul".

## Remarque

 Il ne faut surtout pas confondre :

- Une fonction qui **affiche** quelque chose, et dans ce cas on utilisera la commande **print**. Dans ce cas, la fonction est d'ailleurs plutôt appelée **procédure** (car elle ne renvoie rien - en réalité, elle renvoie **None**)
- Une fonction qui **renvoie** quelque chose, et dans ce cas on utilise la commande **return**

### Exercice 0.3

L'énergie cinétique d'un objet de masse `m` (en kg) qui se déplace à la vitesse `v` (en m/s) est donnée par la formule :

$$E_c = \frac{1}{2} \times m \times v^2$$

Écrire une fonction nommée `get_energie_cinetique(m,v)` ayant comme paramètres la masse `m` et la vitesse `v` qui renvoie l'énergie cinétique.

Quelle est l'énergie (en Joule) d'un objet de 3 kg se déplaçant à 1.56 m/s ?

### Exercice 0.4

Écrire une fonction `div_euclidienne(a,b)` qui prend en paramètre deux nombres entiers `a` et `b`, qui effectue la division euclidienne de `a` par `b` et qui renvoie le couple (`a,i`), respectivement le reste et le quotient de cette division euclidienne.

# 0.2

## Instructions conditionnelles

NSI TERMINALE - JB DUTHOIT

### Exercice 0.5

Écrire une fonction python `renverse(a,b)` qui prend en paramètre deux entiers `a` et `b` et qui retourne `b` et `a` (en inversant l'ordre).

Par exemple `renverse(a,b)` va renvoyer `(b,a)`

### Exercice 0.6

On considère la fonction suivante.

```
def examen(x,y,z,t):
    m = (2 * x + 2 * y + z + t) / 6
    if m >= 10:
        print("Le candidat est reçu")
    else :
        print("le candidat est refusé")
    return None
```

1. Documenter la fonction en ajoutant un docstring.
2. Proposer des pré-conditions écrites sur les variables `x`, `y`, `z` et `t` en utilisant l'instruction `assert` qui assurent le bon usage de cette fonction `examen`.

### Exercice 0.7

Écrire une fonction `signe_prod(a,b)` qui reçoit deux entiers `a` et `b` relatifs et qui renvoie `True` si le produit est positif et `False` sinon.

$\triangleleft$  On ne doit pas calculer le produit des deux nombres.

### Exercice 0.8

Écrire une fonction `categorie(age)` qui a pour paramètre l'âge de l'enfant (qui sera un entier) et qui renvoie sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

### Exercice 0.9

Écrire une fonction Python `est_voyelle(lettre)` pour tester si `lettre` est une voyelle ou non. La fonction renvoie un booléen.

### Exercice 0.10

Écrivez une fonction `triangle(a,b,c)` pour vérifier qu'un triangle est équilatéral, isocèle ou scalène.

Remarque :

Un triangle équilatéral est un triangle dans lequel les trois côtés sont égaux.

Un triangle scalène est un triangle qui a trois côtés inégaux.

Un triangle isocèle est un triangle avec (au moins) deux côtés égaux.

Production attendue :

triangle(6,8,12) renvoie "Triangle scalène"

# 0.3

## Les boucles

NSI TERMINALE - JB DUTHOIT

### Exercice 0.11

Écrire une fonction `carre(n)` qui lit en entrée un entier `n` et écrit tous les carrés des nombres inférieurs à `n`, dans l'ordre croissant. Par exemple, si l'entrée est 16, la sortie sera :

4  
9

### Exercice 0.12

Créer une fonction `somme(n)` de paramètre `n` et qui retourne la somme des entiers naturels jusqu'à `n`.

- avec une boucle pour
- avec une boucle tant que

### Exercice 0.13

Écrire une fonction `tablemulti(n)` qui reçoit en entrée un nombre entier `n` compris entre 1 et 10 et affiche en sortie la table de multiplication de ce nombre. Par exemple, si la fonction prend comme argument le nombre 7, il affichera la table de multiplication :

1 fois 7 = 7  
2 fois 7 = 14  
3 fois 7 = 21  
etc...

### Exercice 0.14

Écrire une fonction qui permet d'écrire une liste d'entiers aléatoires qui s'arrête dès que la somme des nombres dépasse une certaine valeur. La fonction `make_list(deb,fin,maxi)` prend en entrée 3 valeurs entières `deb`, `fin` et `maxi` correspondant respectivement à la plus petite valeur des nombres aléatoires, à la plus grande et à la valeur à ne pas dépasser en faisant la somme de ces nombres.

### Exercice 0.15

Écrire une fonction qui demande à l'utilisateur d'entrer des notes entières, sans que l'on sache à l'avance combien de notes il va saisir : l'arrêt de la saisie se produit dès que l'utilisateur saisit un nombre négatif. Le programme doit calculer et afficher la moyenne (Attention : le nombre négatif saisi ne doit pas compter dans la moyenne!). On nommera cette fonction `moy` elle ne prendra aucune entrée et donnera un élément de type float en sortie.

### Exercice 0.16

Pour commencer l'ordinateur va choisir au hasard un nombre entier compris entre 1 et 100.

L'utilisateur doit alors deviner ce nombre comme ceci : L'utilisateur propose un nombre. L'ordinateur lui dit s'il est trop petit ou trop grand, et ainsi de suite jusqu'à ce que l'utilisateur ait trouvé le bon nombre. Possibilité d'ajouter un compteur qui donnera le nombre d'essais utilisés.

## 0.4

**A propos des chaînes**

NSI TERMINALE - JB DUTHOIT

**Exercice 0.17**

Définissez une fonction `minuchaine(chaine)` qui renvoie le résultat de la conversion de chaine en minuscules.

**Exercice 0.18**

Définissez une fonction `minuchaine2(chaine)` qui retourne le mot avec la première lettre en majuscule et le reste en minuscule.

**Exercice 0.19**

Écrire une fonction `bien_trie(l)` qui reçoit une liste `l` de trois chaînes de caractères et qui renvoie `True` si les trois mots sont rangés par ordre alphabétique et `False` Sinon.

**Exercice 0.20**

Écrire une fonction qui prend une chaîne de caractère en entrée et qui retourne la même chaîne mais en ayant au préalable effacé le premier et le dernier caractère. (Vous vous assurerez que la chaîne a une longueur d'au moins 2.) Par exemple, pour l'entrée `Fairy`, une fonction correcte écrira `air`.

**Exercice 0.21**

Écrire une fonction qui prend en entrée une chaîne et qui retourne en sortie la même chaîne mais avec le premier et le dernier caractères échangés. (Vous vous assurerez que la chaîne a comme longueur au moins 2.)

Par exemple, pour l'entrée `Fairy`, une fonction correcte écrira `yairF`.

☞ Indice : utilisez votre solution de l'exercice précédent comme point de départ.

**Exercice 0.22**

Écrire une fonction qui prend en entrée une chaîne de caractères, et qui retourne la chaîne dans l'ordre inverse.

**Exercice 0.23**

Écrire une fonction qui prend en entrée une chaîne de caractères et qui renvoie `True` si la chaîne de caractères est un palindrome, `False` sinon.

☞ Il est possible d'utiliser la fonction précédente !

**Exercice 0.24**

Une sous-chaîne est une séquence consécutive de caractères à l'intérieur d'une autre chaîne.

La même sous-chaîne peut se trouver plusieurs fois à l'intérieur de la même chaîne : par exemple :

- 'expertiser' a la sous-chaîne 'er' 2 fois
- 'banane trans-panaméenne' a la sous-chaîne 'an' 4 fois

Écrire une fonction `sous_chaine(aiguille,bottedefoin)` qui prend deux lignes d'entrées, dont la première s'appelle `aiguille` et la seconde `bottedefoin` et qui renvoie nombre de fois que `aiguille` se produit comme une sous-chaîne de `bottedefoin`.

|

## 0.5

## Les listes

## 0.5.1 Copie superficielle et copie profonde

**Exercice 0.25**

Analyser les lignes suivantes, et prévoir le résultat à l'appel de l1,l2,l3,l4,l5 et l6.

```
# copie superficielle
l1 = [5,6]
l2 = l1
l1.append(7)
l3 = [5,8]
l4 = l3
l4.append(9)
# copie profonde
l5 = [7,8]
l6 = copy.deepcopy(l5)
l5.append(9)
```

## 0.5.2 Parcours de liste

**Exercice 0.26**

Écrire une fonction `get_max(liste)` qui prend en paramètre une liste d'entiers et renvoie le max des éléments de celle-ci, sans ordonner la liste.

**Exercice 0.27**

Écrire une fonction `get_max_position(l)` qui prend en paramètre une liste d'entiers `l` et qui renvoie le couple (le tuple) (`PG,IG`), respectivement la plus grande valeur de la liste et la position de cette valeur maximale dans la liste. Ajouter une post-condition sur `IG`.

**Exercice 0.28**

Écrire un programme `melanger(liste)` qui prend une `liste` en paramètre, et qui renvoie la liste mélangé. Bien évidemment, on n'utilisera pas ici la fonction native "shuffle(list)" de Python.

On pourra comparer la vitesse d'exécution de `melanger(liste)` avec la fonction `shuffle` de python.

# 0.6

## Module time

NSI TERMINALE - JB DUTHOIT

### Exercice 0.29

Cette fonction nommée `heure(t,d)` prend deux paramètres :

- un "temps de démarrage" exprimé dans une horloge de 24 heures avec des zéros, comme 08 :30 ou 14 :07.
- La deuxième ligne est une durée d de quelques minutes.

Le programme devra afficher la nouvelle heure, c'est-à-dire d minutes après l'heure de départ.

Par exemple, pour l'entrée `heure(12:30,47)`, la sortie correcte serait 13:17. Toutes les heures doivent être formatées sous forme de nombres entre 00 :00 et 23 :59, mais le délai peut aller au-delà de minuit.

Par exemple, sur l'entrée `(23 :59,13)`, la sortie correcte est 00 :12.

# 0.7

## Chiffrement

NSI TERMINALE - JB DUTHOIT

### 0.7.1 ASCII

#### Exercice 0.30

Écrire la liste de tous les caractères ASCII.

### 0.7.2 Chiffrement

Rappel :

- $\text{ord}('A')-65 = 0$
- $\text{ord}('B')-65 = 1$
- ...
- $\text{ord}('Z') - 65 = 25$

De même ,

- $\text{chr}(0+65)='A'$
- $\text{chr}(1+65)='B'$
- ...
- $\text{chr}(25+65)='Z'$

On peut ainsi établir une correspondance entre les caractères A,B,...Z et les entiers de 0 à 25 :

0	1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	E	F	G	H	I	J	K	L	M
13	14	15	16	17	18	19	20	21	22	23	24	25
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

#### Exercice 0.31

On rappelle qu'en Python,  $\text{ord}('h')$  par exemple permet de donner le code ASCII du caractère 'h'. A l'inverse,  $\text{chr}(77)$  va donner le caractère correspondant au code ASCII 77.

Écrire une fonction nommée chiffrement, qui prend en paramètre une chaîne de caractère , et qui renvoie le chiffrement en binaire,au format str, chaque caractère étant codé sur 1 octet.

On pourra vérifier en utilisant : assert chiffrement('Hello !') ==  
'01001000011001010110110001101100011011110010000000100001'

#### Exercice 0.32

L'information génétique présente dans nos cellules est portée par les molécules d'ADN. Les molécules d'ADN sont, entre autres, composées de bases azotées ayant pour noms : Adénine (représenté par un A), Thymine (représenté par un T), Guanine (représenté par un G) et Cytosine (représenté par un C).

L'information génétique est donc très souvent représentée par de très longues chaînes de caractères, composées des caractères A, T, G et C. Exemple : CTATTTCAGCAGTC...

Il est souvent nécessaire de détecter la présence de certains enchainements de bases azotées (dans la plupart des cas un triplet de bases azotées code pour 1 acide aminé et la combinaison d'acides aminés forme

une protéine). Par exemple, on peut se poser la question suivante : trouve-t-on le triplet ACG dans le brin d'ADN suivant (et si oui, en quelle position ?) :

```
CAAGCGCACAAGACGCGGCAGACCTTCGTTATAGGCCATGATT
CGAACCTACTAGTGGGTCTCTTAGGCCGAGCGGTTCCGAGAGAT
AGTGAAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAAAGCG
CGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCCAGT
GTAGAAGCTCCGGCGTCCCCTAACCCTACGCTGTCCCCGGTA
CATGGAGCTAATAGGCTTACTGCCAATATGACCCCGGCCGC
GACAAAACAATAACAGTTGCTGTATGTTCCATGGTGGCCAATC
CGTCTCTTCGACAGCACGGCCAATTCTCCTAGGAAGCCAGCT
CAATTCAACGAAGTCGGCTGTGAACAGCGAGGTATGGCGTCG
GTGGCTCTATTAGTGGTGAGCGAATTGAAATTGGCGCTTAC
TTGTACACAGCGATCCCTCCCACCATTCTTATGCGTCGTCTG
TTACCTGGCTTGGCAT
```

### 0.7.3 Chiffrement de César

'U' correspond à l'entier 20.

Le chiffrement de César consiste à ajouter un entier k (positif ou négatif) : prenons par exemple k = 21.

On a  $20 + 21 = 41$ . On cherche à quelle lettre correspond 41 en sachant que 25 correspond à Z, 26 correspond à A, 27 correspond à B ... etc...

$41 - 26 = 15$  donc 'U' sera codé 'P'

En python, 41% 26 donne 15

#### Exercice 0.33

- Créer une fonction codage, qui prend en paramètre un mot (str) et qui renvoie la liste des nombres associées. Par Exemple, codage('COUCOU') renvoie [2,14,20,2,14,20]
- Créer une fonction decodage, qui prend en paramètre une liste d'entiers entre 0 et 25, et qui renvoie le mot associé (str). Par Exemple, decodage([2,14,20,2,14,20]) renvoie ('COUCOU')
- Créer une fonction chiffrement\_cesar, de paramètre k (int) et mot (str), et qui renvoie le message codé avec le chiffrement de César et la clé k.
- En déduire une fonction dechiffrement\_cesar qui déchiffre un message qui a été chifré avec le chiffrement de César, avec la clé k.
- Déchiffrer le message suivant, qui a été codé avec un chiffrement de César :  
SIRMFMFJRMVQIVLJJZ

### 0.7.4 Chiffrement de Vignère

0	1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	E	F	G	H	I	J	K	L	M
13	14	15	16	17	18	19	20	21	22	23	24	25
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Nous avons vu que le chiffrement de César présente une sécurité très faible, la principale raison est que l'espace des clés est trop petit : il y a seulement 26 clés possibles, et on peut attaquer un message chiffré en testant toutes les clés à la main.

Principe du chiffrement : On choisit une clé constituée d'un mot. Prenons le mot "CLEF" dans cette exemple.

Pour coder le mot "BONJOUR" :

Mot à coder	B 1	O 14	N 13	J 9	O 14	U 20	R 17
Clé répétée	C 2	L 11	E 4	F 5	C 2	L 11	E 4
Somme	3	25	17	14	16	31	21
Somme modulo 26	3	25	17	14	16	5	21
Mot codé	D	Z	R	O	Q	F	V

Un des gros avantages de ce chiffrement est qu'une même lettre peut être codée de façon différente. Dans l'exemple, O est codé par Z et ensuite par Q...

### Exercice 0.34

- Créer une fonction `conversion_mot_en_liste_de_chiffres(chaine)` qui convertit un mot en liste de nombres entiers compris entre 0 et 25 :

```
assert conversion_mot_en_liste_de_chiffres('BONJOUR') == [1, 14, 13, 9, 14, 20, 17]
```

- Créer une fonction `conversion_liste_de_chiffres_en_mot(chaine)` qui réalise l'inverse :

```
assert conversion_liste_de_chiffres_en_mot([1, 14, 13, 9, 14, 20, 17]) == 'BONJOUR'
```

- Créer une fonction `repetition_de_cle(chaine1, chaine2)` qui répète la clé autant de fois que nécessaire :

```
assert repetition_de_cle('BONJOURATOUTLEMONDE', 'FIFI') == 'FIFIFIFIFIFIFIFIFIF'
```

- Écrire une fonction `addition` qui additionne deux entiers et qui renvoie l'entier correspondant entre 0 et 25 :

```
assert addition(19, 20) == 23
assert addition(19, -20) == 25
```

- Écrire une fonction `vignere` qui a pour paramètres un mot et une clé, et qui renvoie le mot chiffré selon le chiffrement de Vignère :

```
assert vignere('CRYPTOGRAPHIE', 'FIFI') == HZDXYWLZFXMQJ'
```

```
def vignere(mot, cle):
    assert type(mot) == str
    assert type(cle) == str
    assert len(cle) > 0
```

- Déchiffrer le message suivant, codé avec la clé NSI :

YWKUANSJMZWVGVMIAORFMEWMFL  
CAUZLHBBKGFLMZWALEMGJQDMM

## 0.8

### Dictionnaire

**Exercice 0.35**

Antoine a obtenu les notes suivantes :

```
notes = {'nsi':18,'Maths':16,'SP':13,'Français':14, 'svt':10}
```

1. Créer une fonction `moyenne(d)`, de paramètre un dictionnaire `d`, et qui retourne la moyenne obtenue.
2. Créer une fonction qui produit un affichage du type "relevé de notes" :

```
Antoine a 18 en NSI
Antoine a 16 en Maths
...
Antoine a 10 en SVT
Antoine à ... de moyenne
```

#

**Exercice 0.36**

Écrire un programme Python pour convertir le nom du mois en un nombre de jours. Accédez à l'éditeur  
Sortie attendue :

```
Liste des mois : janvier, février, mars, avril, mai, juin, juillet, août,, Septembre Octobre Novembre
Décembre
Entrez le nom du mois : février
Nombre de jours : 28/29 jours
```

**Exercice 0.37**

Écrire un programme Python `compteur(d1,d2)` pour combiner deux dictionnaires en ajoutant des valeurs pour les clés communes.

```
d1 = {'a': 100, 'b': 200, 'c': 300}
d2 = {'a': 300, 'b': 200, 'd': 400}
assert compteur(d1,d2) == {'a': 400, 'b': 400, 'd': 400, 'c': 300}
#
```

**Exercice 0.38**

Écrire une fonction `transformation(chaine)` Python pour créer un dictionnaire à partir d'une chaîne.  
Remarque : faire correspondre chaque lettre de la chaîne par son nombre de lettres de la chaîne.

```
transformation("mathématiques") ==
{'m':2,'a':2,'t':2,'h':1,'é':1,'i':1,'q':1,'u':1,'e':1,'s':1 }
#
```

# 0.9

## Tri

NSI TERMINALE - JB DUTHOIT

### Exercice 0.39

Écrire une fonction *tri\_concatenation(l1,l2)* qui prend en argument deux listes triées A et B, et qui renvoie la liste triée C, C résultant de la concaténation de A et de B.

```
def tri_concatenation(l1,l2):
    """
    l1 et l2 sont des listes triées
    la fonction retourne une nouvelle liste, qui est la concaténation triée de l1 et l2
    """

#
```

# 0.10

## Autres

NSI TERMINALE - JB DUTHOIT

### Exercice 0.40

Écrire un programme Python pour trouver les nombres qui sont divisibles par 7 et multiple de 5, entre 1500 et 2700 (tous deux inclus)

### Exercice 0.41

Écrire une fonction Python `tableau(m,n)` qui prend en entrée deux chiffres m (ligne) et n (colonne) et génère un tableau à deux dimensions. La valeur de l'élément dans la i-ème ligne et la j-ème colonne du tableau doit être  $i * j$ .

Remarque :

$i = 0, 1 \dots, m-1$

$j = 0, 1, \dots, n-1$ .

Données de test : Lignes = 3, Colonnes = 4 `tableau(3,4)` renvoie `[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]`

### Exercice 0.42

Réécrivez les nombres de l'entrée à la sortie. Arrêtez le traitement de l'entrée après avoir lu le nombre 42. Tous les nombres en entrée sont des entiers à un ou deux chiffres.

Entrée :

2

88

17

42

17

9

Sortie :

2

88

17