1ère NSI — Exercices d'Entraînement

Algorithmique

(et un peu de traitement de données en table / dictionnaires)

Lycée Fustel de Coulanges, Massy

Marc Biver, février 2024, *v0.3*

Exercice 1: Docstring & Tests

Soit la fonction de calcul de puissance suivante (dont on a étudié l'algorithme en cours) :

```
1  def Puissance(x, n):
2      r = 1
3      for i in range(n):
4          r = r * x
5      return r
```

a. Rédiger la docstring d'une telle fonction ;

b. Déterminer un jeu de tests pour cette fonction – en d'autres termes, compléter le tableau suivant qui liste les couples (x, n) qu'il conviendra d'utiliser pour convenablement tester cette fonction.

x	n	Résultat attendu
2	2	4
.....	.....	.....

Exercice 2: Variant & invariant de boucle

Soit la fonction suivante :

```
1  def SommeEltLst(liste):
2      '''
3      Fonction qui prend en entrée une liste de nombres et qui renvoie la
4      somme des nombres qui la constituent.
5      '''
6      res = 0
7      for i in range(len(liste)):
8          res += liste[i]
9
10     return res
```

a. Donner un variant de boucle pour cette fonction et montrer qu'elle se termine systématiquement ;

> **Rappel:**
>
> On rappelle qu'un variant de boucle est une quantité entière positive qui décroît strictement à mesure qu'on passe dans la boucle, ce qui permet de justifier que la boucle, tôt ou tard, se terminera.

b. Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

Rappel:

On rappelle qu'un invariant de boucle est une quantité ou une propriété qui est vraie avant et après chaque itération de la boucle – et en particulier *avant* que l'on rentre dans la boucle, et *après* sa dernière itération. Il permet de justifier que le résultat voulu sera atteint. On procède pour cette technique en quatre étapes :

(a) On choisit l'invariant :
- Comprendre clairement le but de la boucle – qu'est-elle censée accomplir ? Quel est le résultat attendu ?
- Partir "de la fin", c'est-à-dire du résultat attendu et identifier quelle quantité est "construite" au fur et à mesure des itérations de la boucle pour constituer ce résultat.
- Ceci devrait vous mettre sur la voie de votre invariant – une propriété (somme d'éléments déjà traités, ordre d'éléments dans une liste...) qui ne change pas malgré les itérations de la boucle.

(b) On montre que l'invariant est vérifié avant la boucle (initialisation) ;

(c) On montre que si l'invariant est vérifié *avant* un passage dans la boucle, alors il est préservé *après* le passage dans la boucle ;

(d) On peut conclure sur la valeur finale à la sortie de la boucle.

Exercice 3: Variant & invariant de boucle — suite...

Soit la fonction suivante :

```python
def RechercheDiviseur(nb):
    '''
    Fonction qui prend en entrée un entier strictement supérieur à 1 nb
    et renvoie son plus petit diviseur autre que 1.
    Si elle n'en trouve pas (que le nombre est donc premier), elle
    renvoie 0.
    '''
    div = 0
    i = 2
    while i < nb:
        # On vérifie si nb est un multiple de i
        if nb % i == 0:
            # Dès qu'on en a trouvé un on le renvoie
            div = i
            return div
        i += 1
    # On atteindra ce point si on n'a pas trouvé de diviseur et on
    ↪ renverra donc la valeur initiale de div, soit 0.
    return div
```

a. Donner un variant de boucle pour cette fonction et montrer qu'elle se termine systématiquement ;

b. Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

> **Indice:**
> On rappelle qu'un invariant de boucle n'est pas nécessairement une formule mathématique – il peut aussi être une propriété que l'on pourrait exprimer par une phrase du genre (où bien entendu il faut remplir les trous) : "Quel que soit l'entier k tel que $2 \leq k \leq$ ___, k n'est pas___".

c. (question bonus 1) Cette boucle "`while`" semble assez artificielle – on dirait presque qu'elle n'est là que pour vous forcer à travailler sur la notion de variant de boucle... Quelle est la boucle "`for`" équivalente qu'on utiliserait plus logiquement dans un tel cas ?

d. (question bonus 2) Cette fonction n'est pas franchement optimale... Comment remplacer la condition "`while i < nb:`" pour lui faire faire nettement moins de tests tout en gardant le bon résultat ?

Exercice 4: Variant & invariant de boucle — encore un !

Soit la fonction suivante :

```
def RechercheMax(Lst):
    '''
    Fonction qui prend en entrée une liste de nombres positifs
    et renvoie la valeur du plus grand d'entre eux.
    '''
    res = 0
    for i in range(len(lst)):
        if lst[i] > res:
            res = lst[i]

    return res
```

Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

Exercice 5: Complexité – inversion de liste

Soit la fonction suivante :

```
def InverseListe(lst):
    '''
    Fonction qui prend en entrée une liste quelconque et qui renvoie la
    ↪  liste inversée.
    '''
    # On initialise la liste résultat - liste vide
    res = []
    # On la remplit en partant de la fin de la liste en entrée
    for i in range(len(lst)):
        res.append(lst[len(lst) - i - 1])

    return res
```

a. Montrer, en utilisant un invariant de boucle, qu'elle fait bien ce que sa docstring spécifie.

b. Compter son nombre d'opérations élémentaires, et en déduire sa complexité (préciser sa dénomination et sa notation en "$\mathcal{O}$").

Exercice 6: Complexité – note pondérée

Soit la fonction suivante :

```python
def EltPondere(lst_notes, indice):
    '''
    Fonction qui prend en entrée une liste de notes et un indice dans
    ↪ cette liste et qui renvoie la valeur pondérée de cette note (donc
    ↪ sa valeur divisée par le nombre de notes).
    '''
    res = lst_notes[indice] / len(lst_notes)
    return res
```

Déterminer sa complexité.

Exercice 7: Complexité – somme de produits ou produit de sommes ?

Soit les deux fonctions suivantes :

```python
def ProdSomme(n1, n2):
    '''
    Fonction qui prend en entrée deux entiers n1 et n2 et renvoie
    le produit des sommes de tous les i et j pour i < n1 et j < n2.
    '''
    res1 = 0
    res2 = 0
    for i in range(n1):
        res1 += i
    for j in range(n2):
        res2 += j

    res = res1 * res2

    return res


def SommeProd(n1, n2):
    '''
    Fonction qui prend en entrée deux entiers n1 et n2 et renvoie
    la somme de tous les produits i.j pour i < n1 et j < n2.
    '''
    res = 0
    for i in range(n1):
        for j in range(n2):
            res += i * j
            #On rappelle que "a += b" est équivalent à "a = a + b"

    return res
```

a. Démontrer (simplement) que si j'applique ces deux fonctions aux mêmes paramètres n1 et n2 leur retour sera identique – qu'en d'autres termes elles font le même calcul.

b. Quelles sont les nombres d'opérations élémentaires qu'effectuent chacune de ces deux fonctions (exprimés en fonction de n1 et n2) ? Laquelle des deux, du coup, vous semble la meilleure pour atteindre le résultat ?

Exercice 8: Tri à bulles - bidouillons-le un peu...

On rappelle le code Python du tri à bulles dans sa version la plus basique :

```
1  def TriBulles(liste):
2      '''
3      Fonction qui effectue le tri par permutations de la liste passée en
       ↪  entrée.
4      '''
5      n = len(liste)
6      for i in range(n):
7          for j in range(n-1):
8              if liste[j] > liste[j+1]:
9                  # Cette syntaxe permet d'affecter deux variables
                   ↪  simultanément et donc de ne pas passer par une
                   ↪  variable intermédiaire
10                 liste[j], liste[j+1] = liste[j+1], liste[j]
11     return liste
```

a. A quoi sert l'indice "i" dans cette fonction ?

b. Que se passe-t-il si on remplace la ligne 7 par `for j in range(n-1, -1, -1)` ? Cette syntaxe – qui n'est rien de plus qu'une utilisation de la syntaxe `range` habituelle mais dans le contexte d'une suite décroissante) permet de réaliser une boucle allant de n-2 (le premier paramètre) à 0 (arrêt juste avant d'atteindre le deuxième paramètre) par pas de -1 (le troisième paramètre). L'indice va donc décrire la suite (n-2), (n-3), ..., 1, 0 au lieu de 0, 1, ..., (n-3), (n-2).

c. Que se passe-t-il si on remplace la ligne 8 par `if liste[j] < liste[j+1]:` ?

d. La boucle interne ne fait-elle pas des opérations inutiles ? Comment la modifier pour la rendre plus efficace ?

Exercice 9: Devinette de nombre

Le code utilisé pour l'implémentation de la fonction de recherche dichotomique est le suivant :

```
1  def RechDich(T, x):
2      '''
3      Fonction qui effectue une recherche dichotomique de x dans T.
4      '''
5      debut = 0
6      fin = len(T) - 1
7      while fin >= debut:
8          milieu = (fin + debut) // 2
9          if T[milieu] == x:
10             return milieu
11         elif x > T[milieu]:
12             debut = milieu + 1
13         else:
14             fin = milieu - 1
15     return -1
```

Appuyez-vous sur cette approche et développez le jeu évoqué en cours – vous choisissez un nombre entre 0 et 100 et l'ordinateur doit le deviner en vous posant des questions.

Exemple de séquence :

```
Pensez à un nombre entre 0 et 100...
Si il est plus grand que 50, tapez 1; s'il est plus petit, tapez 2;
s'il est égal à 50 tapez 3.
2
Si il est plus grand que 25, tapez 1; s'il est plus petit, tapez 2;
s'il est égal à 25 tapez 3.
3
J'ai trouvé! Votre nombre est 25.
```

Indices :

- Vous allez évidemment devoir utiliser la fonction "input()" pour recueillir les réponses de l'utilisateur ;
- Vous allez construire de nombreuses fois le même message, avec des valeurs différentes. Voici un exemple qui pourra vous aider :

```
>>> nb = 50
>>> msg = "Voici un message avec le nombre " + str(nb) + " dedans."
>>> print(msg)
Voici un message avec le nombre 50 dedans.
```

Version améliorée : l'ordinateur doit être capable de détecter si l'utilisateur triche (change de nombre en cours de route)...

Exercice 10: Recherche dichotomique

Donnez deux exemples d'exécution de recherche dichotomique où le nombre d'exécutions de la boucle est exactement 4 — un sans solution (retour de -1), une avec une solution. Dans les deux cas il vous est demandé de donner la liste considérée et le nombre recherché.

Exercice 11: Recherche dichotomique modifiée

a. Modifier l'algorithme de recherche dichotomique afin qu'il indique le nombre valeurs examinées dans un tableau donné pour localiser un élément dans un tableau trié ou indiquer qu'il n'est pas présent dans le tableau

b. Quels tests peut-on imaginer pour cet algorithme ?

Exercice 12: Jeu des 7 différences (enfin 2 en l'occurrence...)

Soit le code suivant :

```python
def RechDich(T, x):
    '''
    Fonction qui effectue une recherche dichotomique de x dans T.
    '''
    debut = 0
    fin = len(T) - 1
    while fin >= debut:
        milieu = (fin + debut) // 2
        if T[milieu] == x:
            return milieu
        elif x > T[milieu]:
            debut = milieu
        else:
            fin = milieu
    return -1
```

Voyez-vous les deux différences entre ce code et celui qu'on a développé en cours ? Quel problème est-ce que cela pose ? Que se passerait-il si je lançais la commande `RechDich([1, 2, 3, 4, 5], 5)` ?

Exercice 13: Une fonction tordue

On considère le code de fonction suivant :

```python
def NbrDup(t):
    '''
    Fonction qui fait quelque chose d'assez tordu.....
    '''
    n = len(t)
    n_dup = [0] * n
    for i in range(n):
        for j in range(i + 1, n):
            if t[i] == t[j]:
                n_dup[i] += 1

    return n_dup
```

a. A la main, déterminez le résultat de l'appel à cette fonction pour les cas suivants :
   (a) `t = [1, 2, 3]`
   (b) `t = [1, 1, 1]`
   (c) `t = [1, 1, 3]`
   (d) `t = [1, 2, 2]`

b. Comment, du coup, formuleriez-vous la `docstring` de cette fonction pour expliquer ce qu'elle fait ?

c. En termes de complexité (donc du nombre d'opérations effectuées par la fonction), lequel des 4 cas suivants est le pire ?

d. On se place dans un cas "au pire" : combien de fois, en fonction de n la taille du tableau `t`, l'instruction `n_dup[i] += 1` est-elle effectuée lorsque :
   (a) i vaut 0 ?
   (b) i vaut 1 ?
   (c) i vaut $n-1$?

e. En déduire, en justifiant bien, la complexité en fonction de n de la fonction `NbrDup`.

Exercice 14: Un peu de travail sur les dictionnaires

On considère le code de fonction suivant :

```
def Occ(t):

    res = {}
    for x in t:
        if x in res:
            res[x] += 1
        else:
            res[x] = 1

    return res
```

a. ***Encore*** une fonction où on a oublié de mettre la `docstring`!! Que proposez-vous ?

b. Ecrire une fonction `compare_tableaux(t, u)` qui prend deux tableaux `t` et `u` en entrée et détermine si ils contiennent les mêmes éléments, mais sans les trier.

Exercice 15: Ah tiens au fait...

Est-ce que la fonction `Occ` qu'on a vue à l'exercice précédent fonctionnerait si on lui passait en entrée une chaine de caractères au lieu d'une liste ? Que renverrait l'appel `Occ("abracadabra")` ?
Qu'en serait-il de `compare_tableaux` si on lui passait deux chaines de caractères en entrée ?

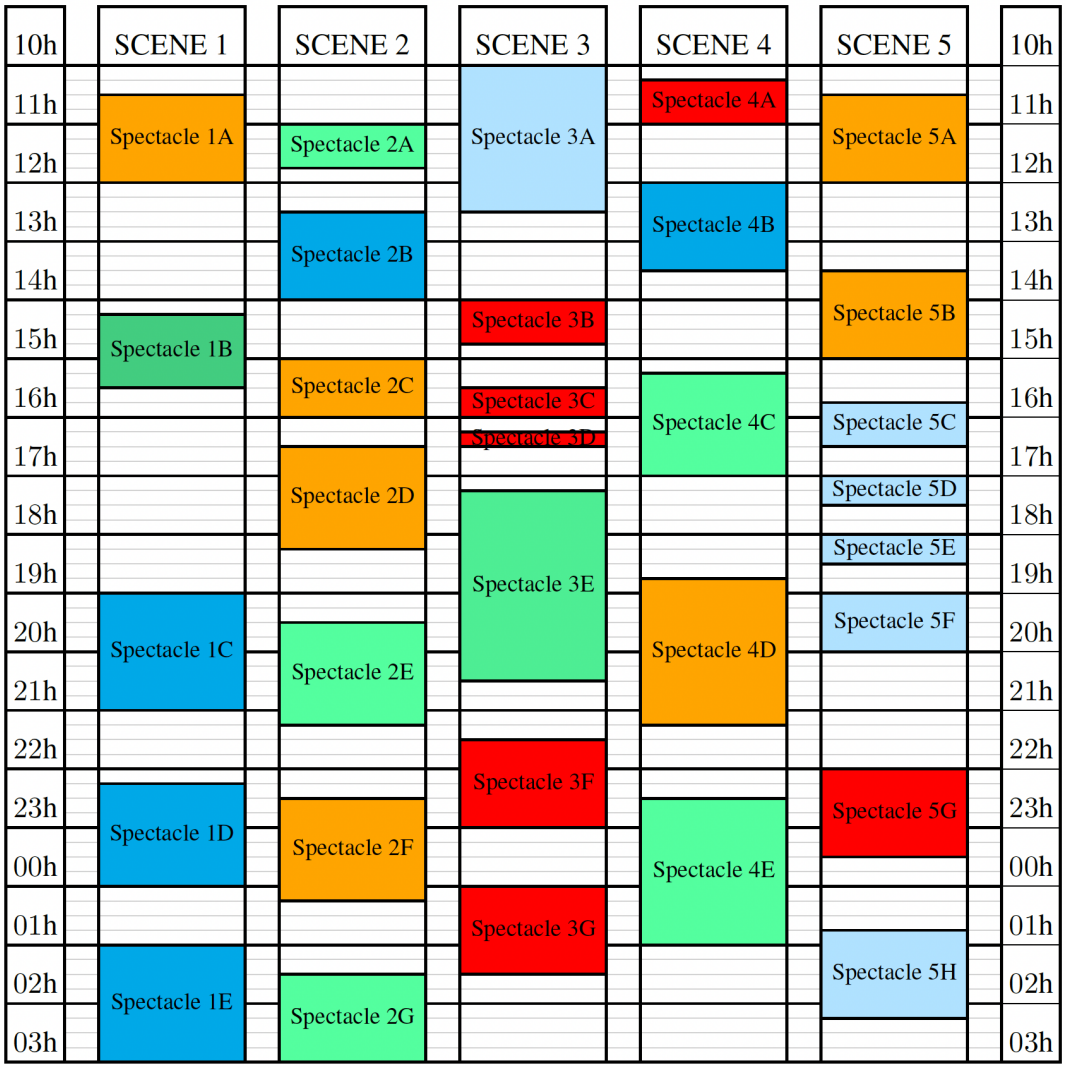
Exercice 16: Tris décroissants

a. Ré-écrire l'algorithme du tri par sélection tel qu'on l'a vu en cours pour qu'il produise un tri par ordre décroissant ;

b. Même chose avec l'algorithme de tri par insertion.

Exercice 17: Un festival glouton

Vous venez d'arriver au Festival d'Avignon et découvrez qu'il se déroule sur 5 scènes différentes, avec une programmation extrêmement complexe :



Brillant élève de NSI que vous êtes, vous vous décidez à choisir les spectacles auxquels vous allez assister en appliquant une méthode gloutonne. Les règles à respecter sont :

- Vous voulez voir le nombre maximal possible de spectacles dans la journée ;
- Lorsque vous commencez un spectacle vous voulez le voir en entier (du début à la fin) ;
- On suppose que le temps de trajet entre les scènes est nul.

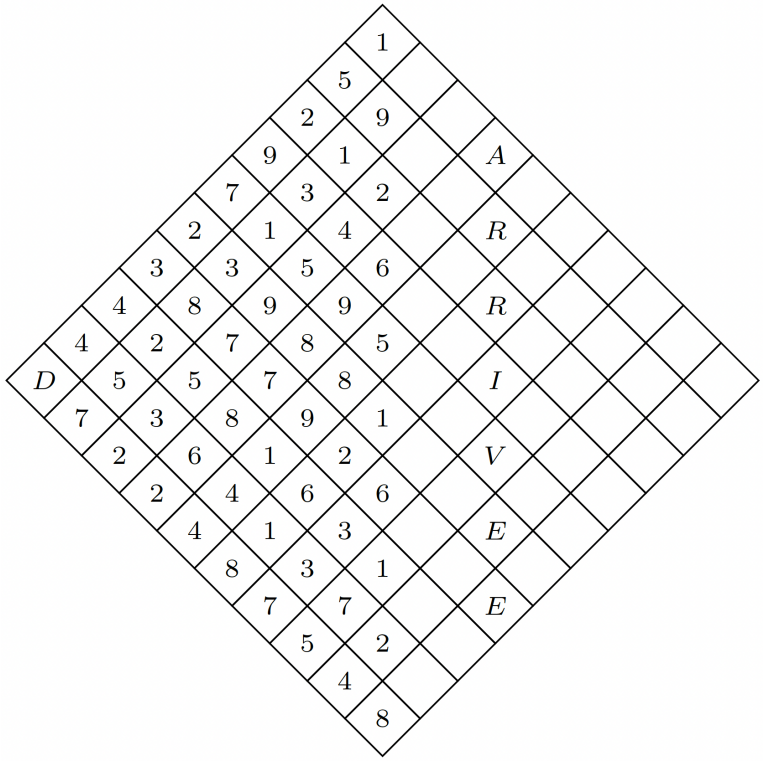
On va considérer deux algorithmes gloutons séparés : appliquez le déroulé de chacun d'entre eux et concluez sur celui qui fonctionne le mieux.

a. Algorithme A : moins on attend entre deux spectacles plus on en verra, donc à chaque étape on choisit le prochain spectacle qui débute le plus tôt.

b. Algorithme B : plus chaque spectacle que l'on voit finit tôt plus on aura de temps pour voir d'autres spectacles, donc à chaque étape on choisit le prochain spectacle qui finit en premier.

Exercice 18: Encore un peu d'algorithmes gloutons...

On considère la figure ci-dessous :



Le jeu sur cette grille consiste à partir n part de la case "**D**" à gauche et de se rendre sur une des cases vides à droite en se déplaçant de case en case. Lorsqu'on est sur une case on peut se déplacer sur une des deux cases voisines situées sur la droite, mais pas sur la case reliée uniquement par un sommet (donc de la case D on peut aller en 4 ou en 7 mais pas en 5).
On note S la somme de toutes les cases traversées. Par exemple on peut effectuer la trajectoire suivante : `D - 7 - 5 - 3 - 5 - 7 - 9 - 8 - 9 - 6` qui donne $S = 59$.
Le but du jeu est d'effectuer la trajectoire qui rend la somme S la plus petite possible.

a. Décrivez une approche gloutonne à la résolution de ce problème. Combien de calculs sont nécessaires pour choisir la trajectoire ?

b. En appliquant cette approche, quelle trajectoire et quelle somme S obtenez-vous ?

c. Sur cette grille, en cherchant bien, la trajectoire optimale donne une somme $S = 23$. Votre algorithme glouton a-t-il trouvé cette trajectoire optimale ? Si la réponse est non, est-ce que la trajectoire trouvée vous semble proche de la trajectoire optimale ?

d. Comment pourrait-on modifier l'algorithme pour l'améliorer ? Combien de calculs sont nécessaires pour choisir la trajectoire avec cette amélioration ?

e. Pour être sûr d'obtenir la solution optimale cependant, il n'y a d'autre choix que d'adopter une approche "force brute" – regarder toutes les trajectoires possibles et choisir celle au score le plus faible. Démontrer qu'on va devoir calculer 512 trajectoires. calculs.

Exercice 19: Et un petit exo sur les fichiers CSV pour finir

Soit le code suivant :

```
import csv
fichier = open('Truc.csv', 'r', encoding = 'utf-8')
table = list(csv.DictReader(fichier))
```

Et soit le fichier `Truc.csv` contenant les données suivantes :

```
Eleve,Age,Classe
Loubna,17,1G5
Olivier,16,1G2
Lenny,17,1G2
```

a. Rédigez un programme qui affiche à l'écran tous les prénoms d'élèves du fichier `Truc.csv` les uns à la suite des autres.

b. Rédigez une fonction `AgeMoy(classe)` qui renvoie l'âge moyen des élèves présents dans la classe passée en argument.