Correction du TP noté

Julien Reichert

Version téléchargeable sous forme de fichier python : `http://www.jdreichert.fr/Enseignement/CPGE/IPTSUP/correction_TP_noté.py`.[1]

```
### Exercice 1

# Ma version :
def check_exam(arr1, arr2):
    return max(0, sum([5 * (arr1[i] == arr2[i]) - (arr2[i] != "") for i in range(len(arr1))]))

# Version étudiant :
def check_exam(arr1, arr2):
    somme = 0
    for i in range(len(arr1)): # Ce n'est pas forcément 4.
        if arr1[i] == arr2[i]:
            somme += 4
        elif arr2[i] != "": # Pas besoin de faire de ligne somme = somme sinon !
            somme -= 1
    if somme < 0: # Pas dans la boucle !
        return 0
    return somme

### Exercice 2

def switch_it_up(number):
    return ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"][number]
# C'est mieux que 21 lignes pleines de tests enchaînés, beurk !

### Exercice 3

def first_non_consecutive(arr):
    for i in range(1,len(arr)): # On peut décaler d'un et comparer avec l'indice i+1.
        if (arr[i]) != (arr[i-1]+1):
            return arr[i]
    # Pas besoin de return None, c'est fait par défaut.

### Exercice 4

def debordement(l, c):
    somme = 0
    for x in l: # Ou on fait un parcours sur les indices.
        somme += x
        if somme < 0 or somme > c:
            return False
    return True # Attention à l'emplacement !
```

1. Devient `http://www.jdreichert.fr/Enseignement/CPGE2021/IPTSUP/correction_TP_noté.py` à la fin de l'année scolaire 2020/2021.

```
### Exercice 5

def factorielle(n):
    res = 1
    for i in range(2, n+1): # Attention aux bornes !
        res *= i
    return res

def kparmin(n, k):
    return factorielle(n) // (factorielle(k) * factorielle(n-k) # On veut un entier !

### Exercice 6

def invert(lst):
    return [-x for x in lst]
# Peut se faire avec des append, une boucle sur les indices, etc.

### Exercice 7

# Ma version :
def shark(po_di, sh_di, you_sp, sh_sp, dol):
    return ["Shark Bait!", "Alive!"][(po_di / you_sp)/(1 + dol) < (sh_di / sh_sp) and po_di <= sh_di]

# Version étudiant :
def shark(pontoon_distance, shark_distance, you_speed, shark_speed, dolphin):
    if shark_distance <= pontoon_distance: # Pour la rigueur, mais aucun test n'en tient compte.
        return "Shark Bait!"
    if dolphin: # Pas besoin d'écrire else vu le return dans le test
        shark_speed /= 2
    if pontoon_distance / you_speed >= shark_distance / shark_speed:
        return "Shark Bait!"
    return "Alive!" # Même remarque !

### Exercice 8

# Ma version :
def next_id(arr):
    dict = {} # Plus rapide
    for x in arr:
        dict[x] = True
    i = 0
    while i in dict:
        i += 1
    return i

# Ma version plus compréhensible et pour ainsi dire aussi rapide :
def next_id(arr):
    n = len(arr)
    bools = [False for i in range(n)]
    for x in arr:
        if 0 <= x < n:
            bools[x] = True
    i = 0
    while i < n and bools[i]:
        i += 1
    return i
```

```
# Version étudiant :
def next_id(arr):
    i = 0
    while i in arr: # On peut faire une boucle jusqu'à len(arr), c'est logique.
        i+=1
    return i

### Exercice 9

def correspondances(l, ll):
    total = 0
    for i in range(min(len(l), len(ll))):
        if l[i] == ll[i]:
            total += 1
    return total

### Exercice 10

def correspondancesmax(l, ll):
    maxtotal = 0
    if len(l) < len(ll):
        l, ll = ll, l # Pour simplifier la suite
    for i in range(len(l)): # Tous les décalages possibles
        total = 0
        j = 0 # On va faire un while pour se faciliter la vie
        while j < len(ll) and i+j < len(l): # On parcourt ll jusqu'à éventuellement déborder de l.
            if ll[j] == l[i+j]:
                total += 1
            j += 1 # Inconvénient du while : on peut oublier ceci !
        if total > maxtotal:
            maxtotal = total # On peut aussi mémoriser i et le retourner avec maxtotal.
    return maxtotal # Ici on ne signale pas la localisation.
```