Devoir d'informatique

Exercice 0

Question 1 *Ecrire la fonction* `somme(L:[int])-> int` *permettant de renvoyer la somme des éléments d'une liste d'entiers* `L`.

Correction

```
def somme(L:[int]) -> int :
    res = 0
    for e in L :
        res = res + e
    return res
```

Question 2 *Ecrire la fonction* `mult(L1:[int],L2:[int])-> [int]` *permettant de renvoyer la liste des produits terme à terme de deux listes de même taille.*

```
def mult(L1:[int],L2:[int]) -> [int] :
    L3 = []
    for i in range(len(L)) :
        L3.append(L1[i]*L2[i])
    return L3
```

Exercice 1

On rappelle que pour $n \in \mathbb{N}^*$, la factorielle de n est donnée par $n! = \prod_{1 \leq i \leq n} i$. Par convention $0! = 1$. On donne la fonction suivante permettant de calculer $n!$.

```
def factorielle(n:int) -> int :
    p = 1
    i = 1
    while i<=n:
        p = p*i
        i = i+1
    return p
```

Question 3 *Donner 2 assertions permettant de vérifier que l'argument de la fonction est compatible avec le calcul de $n!$.*

Correction

```
def factorielle(n:int) -> int :
    assert type(n) == type(0)
```

```
    assert n >= 0
    p = 1
    i = 1
    while i<=n:
        p = p*i
        i = i+1
    return p
```

Question 4 *Donner l'évolution des variables* `p` *et* `i` *lorsque on calcule* `factorielle(4)`. *On pourra recopier et remplir le tableau de la forme suivante.*

Correction

	p	i	$i \leq n$
Entrée dans la boucle	1	1	$1 \leq 4 \Rightarrow$ True
Fin du premier tour de boucle	1	2	$2 \leq 4 \Rightarrow$ True
Fin du deuxième tour de boucle	2	3	$3 \leq 4 \Rightarrow$ True
Fin du troisième tour de boucle	6	4	$4 \leq 4 \Rightarrow$ True
Fin du quatrième tour de boucle	24	5	$5 \leq 4 \Rightarrow$ False

Question 5 *Montrer que $n-i$ est un variant de boucle.*

Correction Avant d'entrée dans la boucle : si n est un entier (condition nécessaire pour calculer $n!$), `i=1` est un entier. Il en résulte que `n-i` est un entier.

Considérons que qu'au ième tour de boucle, `n-i` est un entier positif.

À la fin du tout de boucle suivant, $i' = i+1$ est un entier. $n-i'$ est un entier. De plus $i' > i$; donc $n-i' < n-i$. la quantité $n-i$ est donc strictement décroissante.

$n-i$ est donc un variant de boucle et la fonction termine.

Question 6 *Montrer que la proporiété $p_k = k!$ est un invariant de boucle.*

Correction

En entrant dans la boucle, $i = 1$ et $p = 1$; donc $1 = 1!$.
A la fin de la kieme itération, considérons que $p_k = k!$ et $i_k = k+1$.
À la fin de la $k+1$ieme itération, $p_{k+1} = p_k \times i_k = k(k+1) = (k+1)!$ et $i_{k+1} = k+2$.
La propriété d'invariance est bien démontrée.

Question 7 *Déterminer la complexité algorithmique de cette fonction.*

Correction
Chacune des instructions à une complexité que l'on peut considérer constante. La boucle `while` s'exécute n fois. L'algorithme est en $\mathcal{O}(n)$.

Exercice 2 – Traitement d'une image

Question 8 *Ecrire la fonction* `taille(im:[[int]])-> int,int` *renvoyant le nombre de lignes et de colonnes de l'image* `im`.

Correction

```
def taille(im:[[int]]) -> (int,int):
    return len(im),len(im[0])
```

Question 9 *Ecrire les lignes de code permettant d'ajouter dans* `imc` *les* `n` *lignes permettant de former la partie*

supérieure du cadre. Donner la complexité de l'algorithme proposé.

Correction

```
    # On ajoute le cadre du haut
    L = []
    for i in range(co+2*n):
        L.append(0)
    for i in range(n):
        imc.append(L)
```

Question 10 *Ecrire les lignes de code permettant d'ajouter dans* `imc` *les lignes provenant de* `im` *en leur ajoutant les bords latéraux.*

Correction

```
    # On ajoute les bordures latérales :
    for i in range(li):
        L = []
        for j in range(n):
            L.append(0)
        for j in range(co):
            L.append(im[i][j])
        for j in range(n):
            L.append(0)
        imc.append(L)
```

Question 11 *Ecrire la fonction* `conversion_nb(im:[[int]], n:int)-> [[int]]` *renvoyant une image convertie en noir et blanc en utilisant un seuil* `n`*. Donner la complexité de l'algorithme proposé.*

Correction

```
def convertir_nb(im:[[int]], n:int) -> [[int]] :
    li,co = taille(im)
    for i in range(li) :
        for j in range (co) :
            if im[i][j] > n :
                im[i][j] = 255
            else :
                im[i][j] = 0
    return im
```

Question 12 *Ecrire la fonction* `moyenne(im:[[int]])-> float` *permettant de calculer la valeur moyenne des niveaux de gris de l'image.*

Correction

```
def moyenne(im:[[int]]) -> float :
    li,co = taille(im)
    m = 0
    for i in range(li) :
        for j in range (co) :
            m = m+im[i][j]
    return m/li/co
```

Une autre méthode pour calculer le seuil est de déterminer une valeur de gris qui permettrait d'avoir autant de pixels blancs que de pixels gris.

Question 13 *Écrire la fonction* `effectifs(im:[[int]])-> [int]`*, qui prend en paramètre une image et qui renvoie une liste de longueur 256. Chaque élément* `i` *de la liste devra contenir le nombre de pixels de couleur* `i`*.*

Correction

```
def effectifs(im:[[int]]) -> [int] :
    eff = 256*[0]
    li,co = taille(im)

    for i in range(li) :
        for j in range (co) :
            pix = im[i][j]
            eff[pix] += 1
    return eff
```

Question 14 *Ecrire la fonction* `valeur_de_partage(im:[[int]])-> float` *permettant de mettre en œuvre l'algorithme précédent.*

Correction

```
def valeur_de_partage(im:[[int]]) -> float:
    L = effectifs(im)
    """ L est une liste d'entiers naturels : pour tout indice i, L[i] est l'effectif de la ↙
        valeur i """
    n = len(L)
    i = 0
    j = n - 1
    S1 = L[i]
    S2 = L[j]
    while i+1 < j:
        if S1 < S2:
            i = i + 1
            S1 = S1 + L[i]
        else:
            j = j - 1
            S2 = S2 + L[j]
    # i est égal àj-1
    m = (i+j) / 2
    return m
```

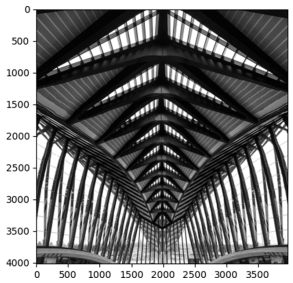
Image en niveau de gris.

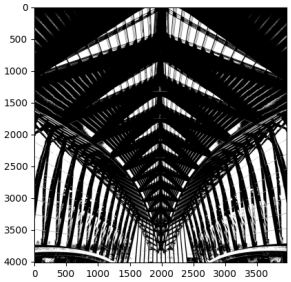
Image en noir et blanc – Seuils 127.

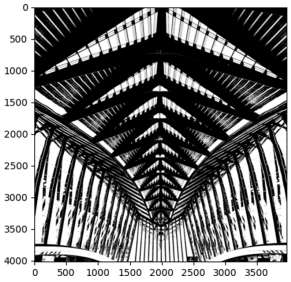
Image en noir et blanc – Seuil moyenne.

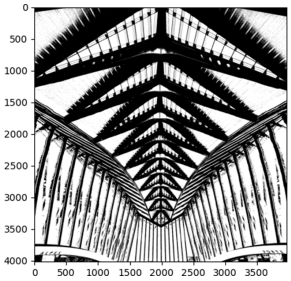
Image en noir et blanc – Seuil médiane.