ISM PCSI, Informatique
Année 2025/2026 19/12/25

Corrigé du QCM n°4

1. On considère la fonction `f(L)` d'argument `L` une liste non vide de nombres :

```
def f(L):
    ind=0
    n=len(L)
    for k in range(n):
        if ... :
            ...
    return ind
```

Identifier les propositions cohérentes :

1. L5 : `L[k]>L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du maximum
2. L5 : `L[k]>L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du maximum
3. L5 : `L[k]>L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du minimum
4. L5 : `L[k]>L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du minimum

Quand on rencontre un élément strictement plus grand celui en position `ind`, on garde sa position en mémoire. Comme le test est strict, on obtient l'indice de première occurrence du maximum puisque on ne réaffecte pas `ind` à chaque occurrence du maximum.

2. On considère la fonction `f` précédemment définie. Identifier les propositions cohérentes :

1. L5 : `L[k]>=L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du maximum
2. L5 : `L[k]>=L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du maximum
3. L5 : `L[k]>=L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du minimum
4. L5 : `L[k]>=L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du minimum

Quand on rencontre un élément plus grand celui en position `ind`, on garde sa position en mémoire. Comme le test est large, on obtient l'indice de dernière occurrence du maximum puisque on réaffecte `ind` à chaque occurrence du maximum.

3. On considère la fonction `f` précédemment définie. Identifier les propositions cohérentes :

1. L5 : `L[k]<L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du maximum
2. L5 : `L[k]<L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du minimum
3. L5 : `L[k]<L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du minimum
4. L5 : `L[k]<L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du maximum

Quand on rencontre un élément strictement plus petit celui en position `ind`, on garde sa position en mémoire. Comme le test est strict, on obtient l'indice de première occurrence du minimum puisque on ne réaffecte pas `ind` à chaque occurrence du minimum.

4. On considère la fonction `f` précédemment définie. Identifier les propositions cohérentes :

1. L5 : `L[k]<=L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du minimum
2. L5 : `L[k]<=L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du minimum
3. L5 : `L[k]<=L[ind]`, L6 : `ind=k`, résultat : indice de première occurrence du maximum
4. L5 : `L[k]<=L[ind]`, L6 : `ind=k`, résultat : indice de dernière occurrence du maximum

Quand on rencontre un élément plus petit celui en position `ind`, on garde sa position en mémoire. Comme le test est large, on obtient l'indice de dernière occurrence du minimum puisque on réaffecte `ind` à chaque occurrence du minimum.

5. On considère la fonction `g(L)` d'arguments `elt` un objet et `L` une liste :

```
def g(elt,L):
    ...
    n=len(L)
    for k in range(n):
        if L[k]==elt:
            ...
    return res
```

Identifier les propositions cohérentes :

1. L2 : `res=False`, L6 : `return True`, résultat : `True` si `elt` appartient à `L` et `False` sinon
2. L2 : `res=False`, L6 : `res=True`, résultat : `True` si `elt` appartient à `L` et `False` sinon
3. L2 : `res=True`, L6 : `res=1`, résultat : `False` si `elt` appartient à `L` et `False` sinon
4. L2 : `res=True`, L6 : `return False`, résultat : `True` si `elt` appartient à `L` et `False` sinon

On initialise `res` à `False` et si on rencontre `elt` dans la liste `L`, soit on bascule `res` à `True`, soit on renvoie `True`.

6. On considère la fonction `g` précédemment définie. Identifier les propositions cohérentes :

1. L2 : `res=False`, L6 : `res=k`, résultat : indice de dernière occurrence de `elt` s'il appartient à `L` et `False` sinon
2. L2 : `res=False`, L6 : `return k`, résultat : indice de première occurrence de `elt` s'il appartient à `L` et `False` sinon
3. L2 : `res=False`, L6 : `return k`, résultat : indice de dernière occurrence de `elt` s'il appartient à `L` et `False` sinon
4. L2 : `res=False`, L6 : `res=k`, résultat : indice de première occurrence de `elt` s'il appartient à `L` et `False` sinon

On initialise `res` à `False`. Si on mémorise `k` dans `res` à chaque occurrence de `elt` dans `L`, alors on renvoie l'indice de sa dernière occurrence. Si on effectue un renvoi quand on rencontre `elt` dans `L`, le `return` casse la boucle et renvoie la première occurrence.

7. On considère la fonction `g` précédemment définie. Identifier les propositions cohérentes :

   1. L2 : `res=0`, L6 : `return k`, résultat : nombres d'occurrences de `elt` dans `L`
   2. L2 : `res=0`, L6 : `res+=1`, résultat : nombres d'occurrences de `elt` dans `L`
   3. L2 : `res=0`, L6 : `res+=k`, résultat : nombres d'occurrences de `elt` dans `L`
   4. L2 : `res=0`, L6 : `res=1`, résultat : nombres d'occurrences de `elt` dans `L`

Pour compter le nombre d'occurrence de `elt` dans `L`, on initialise `res` à zéro et on incrémente `res` à chaque nouvelle occurrence de `elt` dans `L`.

8. On considère la fonction `g` précédemment définie. Identifier les propositions cohérentes :

   1. L2 : `res=[]`, L6 : `res+=[k]`, résultat : liste des indices de toutes les occurrences de `elt` dans `L`
   2. L2 : `res=[]`, L6 : `return [k]`, résultat : liste des indices de toutes les occurrences de `elt` dans `L`
   3. L2 : `res=[]`, L6 : `res.append(k)`, résultat : liste des indices de toutes les occurrences de `elt` dans `L`
   4. L2 : `res=[]`, L6 : `res=[k]`, résultat : liste des indices de toutes les occurrences de `elt` dans `L`

Pour avoir la liste de toutes les occurrences de `elt` dans `L`, on initialise `res` par une liste vide et on ajoute l'indice de chaque nouvelle occurrence dans `res`, soit par `append`, soit par concaténation.