Quelques exercices

Q 1 :

```
def positif_all (L):
    for elt in L :
        if elt < 0 :
            return False
    return True
```

Q 2 :

```
def cherche_lettre (chaine, x):
    indice = [ ]
    for i in range(len(chaine)):    #pour avoir les indice
        if chaine[i] == x:      #si l'élément est item, enregistrer l'indice
            indice.append(i)
    return indice
```

Q 3 :

```
def seuil(k) :
    u = 2
    m = 0
    while u<k : #s'arrête si u>=k
        u = 1 + 2*u**2
        m = m + 1  # bon indice car on test la valeur ensuite
    return m
```

Q 4 :

```
def nettoieListe (L , x):
    i = 0  # indice du départ pour recherche
        while i < len(L) and len(L) != 0 : # temps que l'indice est inférieur à l'indice max de la liste
            if x == L[i]: # on enlève élément de la liste
                del L[i]
            else :
                i = i + 1 # on prend indice suivant pour le test
    return L
```

Q 5 :

```
fr_en = {« je » : « I » , « aime » : « love » ,  « informatique » : « « computing »}
```

Q 6 :

```
def traduction(liste , dico) :
    liste_traduite = [ ]
    for elt in liste
        liste_traduite.append ( dico(elt))
    return liste_traduite
```

Q 7 :

```
def inverse_dico ( dico ) :
    dico_inverse = { }
    for k in dico :  # k correspond à chaque clé
        dico_inverse [dico(k)] = k  # on prend valeur en clé et la clé en valeur
return dico_inverse
```

Q 8 :

```
def inventaire(liste) :
  invent = { }
  for elt in liste :
    if elt in invent :
      invent [ elt ] += 1 # si l'élement est déjà dans l'inventaire on rajoute 1
    else :
      invent [elt] = 1 # sinon on l'ajoute à l'inventaire et on met sa valeur à 1
  return invent
```

10 km de la Saint Nicolas

Partie A : Gestion de la course et statistiques

Q9.

```
def vainqueur (dossard, temps) :
        t_min = temps [0]
        ind = 0          #on initialise le temps mini au temps du premier de la liste
        for i in range (len(temps)) :
                if temps[i] < t_min          #on cherche le temps mini , on enregistre ce temps
                        t_min = temps[i]
                        ind = i              # on enregistre l'indice du temps mini
        return dossard[ind]
```

Q10.

```
def temps_moy(temps) :                  #cela revient à faire la moyenne des éléments d'une liste
        somme = 0
        for t in temps :
                somme += t
        return somme/len(temps)
```

Partie B : Alerte météo

Q11 Dossard du gagnant : 66 (personne ne le précède)
Dossard du deuxième : 78 (celui suivant 66)
Ordre de la course : 66, 78, 85, 23, 12.

Q12.

```
def verifie(dossard, numero):
        for elt in dossard
                if numero == elt:
                        return True
        return False
```

Q13.

```
def cherche(L,x) :
        for i in range (len(L)) :
                if L[i] == x :                  #dès qu'on trouve l'élément x dans L
                        return i                #on sort de la fonction et renvoie l'indice
        return None
```

Q14.

```
def gagnant(dossard, preced) :
    ind = cherche(preced, None)        #on cherche l'indice du coureur qui n'a personne devant lui
    return dossard[ind]
```

Q15.

```
def perdant(dossard, preced) :
    for i in range(len(dossard)) :
        numero = dossard[i]
        if not verifie(preced,numero) : #si un numéro ne se trouve pas dans preced, c'est le dernier
            return (i,numero)
```

Q16. ind_prec = [1,4,None,2,3]

Q17.

```
def construit_ind_prec(dossard, preced) :
    ind_prec = [ ]                                  #on crée une liste vide
    for k in range (len(dossard)) :
        ind_prec.append(cherche(dossard,preced[k]))  #on ajoute le précédent de chaque dossard
    return ind_prec
```

Q18.

```
def classement(dossard,preced) :
    ind_prec = construit_ind_prec(dossard,preced)   #liste des indices précédents de chaque dossard
    ordre = [perdant(dossard,preced)[1]]            #crée une liste avec le perdant
    indice = perdant(dossard,preced)[0]         #on récupère l'indice du perdant
    for i in range (1,len(dossard)) :           #on commence à 1 car on a déjà « récupéré » le perdant
        indice = ind_prec[indice]                   #on récupère l'indice de celui précédent le perdant
        ordre.append(dossard[indice])               #on l'ajoute à la liste ordre
    return ordre
```

Q19. x et y sont des listes.

Q20. Cette fonction fait la différence des sommes des numéros de dossard des listes dossard (s) et preced (t). Elle renvoie le numéro du perdant.

<u>Partie C : Coureur peu lucide ou quelque peu tricheur</u>

Q21.

```
def erreur_vainqueur(preced) :
    for elt in preced
        if None = = elt :
            return False
    return True
```

Q21.

```
def cherche_doublon(preced) :
    for i in range(len(preced)-1) :
        for j in range(i+1,len(preced)) :
            if preced[i] == preced[j]:
                return preced[i]
    return None
```

Q22.

```
def fautif(dossard,preced,doublon) :        #on part de doublon et on remonte
    actuel = doublon
    while dossard_devant(dossard,preced,actuel) != doublon :
        actuel = dossard_devant(dossard,preced,actuel)        #on remonte les numéro de proche en proche
    return actuel        # on s'arrête parce que l'on a quelqu'un qui pointe vers le doublon alors qu'on a commencé à remonter à partir de doublon, c'est donc actuel qui est le fautif
```