Exercices d'algorithmie – TD 8

Exercice 1 - Trace d'un algorithme. Soit l'algorithme ci-après. Fournir sa trace d'exécution pour `a=3` et `b=4`.

```
def algoMystere(a, b):
    a = a * 2
    b = b + 1
    max = (a*b) // 2
    i = 0
    while i < max:
        c = a + b
        a = a - 1
        b = b + a
        i = i + 1
```

Exercice 2 - Trace d'un algorithme. Soit l'algorithme ci-après. Fournir sa trace d'exécution pour `a=7` et `b=8`.

```
def autreMystere(a, b):
    for i in range(a):
        chaine = ""
        for j in range(i):
            chaine = str(j) + chaine
        print(chaine)
```

Exercice 3. On demande d'écrire un algorithme qui affiche la table de multiplication de `m` entre les valeurs `a` et `b`. Par exemple, `tableMult(5, 3, 6)` doit afficher :

```
5*3=15
5*4=20
5*5=25
5*6=30
```

Vous trouverez ci-dessous deux corrections : avec un `for` et avec un `while`. Dans ces corrections :

- Complétez les entrées, sorties, pré-conditions et post-conditions.
- Observez l'utilisation de `str`. Quel est son rôle ?
- Corrigez les erreurs éventuelles des algorithmes.

```
def tableMultFor(m, a, b):
    '''
    :entrée:
    :sortie:
    :pré-conditions:
    :post-conditions:
    '''
    for mult in range(a, b):
        print(str(m) + "*" + str(mult) + "=" + str(m*mult))
```

```
def tableMultWhile(m, a, b):
    '''
    :entrée:
    :sortie:
    :pré-conditions:
    :post-conditions:
    '''
    mult = a
    while(mult < b):
        print(str(m) + "*" + str(mult) + "=" + str(m*mult))
        mult = mult+1
```

Exercice 4. Même exercice que l'exercice 3, mais on souhaite maintenant afficher plusieurs tables de multiplication (entre *m* et *n*). Vous trouverez ci-dessous une correction possible. Questions :

- Complétez le contrat.
- Complétez l'algorithme.
- Proposez une solution avec un `while`.
- Que se passerait-il si vous n'aviez pas corrigé les algorithmes de l'exercice précédent ?

```
def tablesMult(m, n, a, b):
    '''



    '''
    for i in range (       ):
        tableMultWhile(    )
        print("----")
```

Exercice 5. On demande d'écrire l'algorithme qui permet d'afficher le triangle de Pascal *sans utiliser de structure de données (listes ou tableaux) ni les formules de calcul de coefficients* que vous connaissez.

L'algorithme ci-dessous est correct... dans une certaine mesure.

- Observez l'usage de `int`. À quoi sert-il ? Quel serait le comportement du programme s'il n'était pas là ?
- À partir de quelle ligne cesse-t-il de fonctionner, et pourquoi ?
- En déduire le contrat.
- En utilisant uniquement les concepts algorithmiques "autorisés", quelle solution pouvez-vous imaginer pour pallier le problème identifié ? (On ne demande pas de corriger l'algorithme).

```
def trianglePascal(hauteur):
    '''

    '''
    ligne = "1"
    for i in range(hauteur):
        print(ligne)
        nligne = "1"
        for j in range(len(ligne)-1):
            nligne = nligne + str(int(ligne[j]) + int(ligne[j+1]))
        nligne = nligne + "1"
        ligne = nligne
```

Exercice 6. Ci-dessous se trouve un algorithme qui calcule la longueur d'une chaîne de caractères avec la contrainte de ne pas utiliser `len`.

Questions :

- Peut-on écrire le même algorithme avec un `while` ? Justifiez.
- Comment se comporte l'algorithme si on l'appelle avec une chaîne vide ?
- En déduire le contrat.

```
def longueurChaine(chaine):
    '''
    '''
    compteur = 0
    for ch in chaine:
        compteur = compteur + 1
    return(compteur)
```

Exercice 7. Ci-dessous se trouve un algorithme qui renvoie `True` si un caractère est présent dans une chaîne, et `False` dans le cas contraire.

- Expliquez le choix fait pour l'initialisation de la variable `i`.
- Peut-on écrire un algorithme similaire en utilisant un `for` ? Est-ce souhaitable ? Pourquoi ?
- Comment se comporte le programme sur la chaîne vide ? Pouvait-on s'y attendre ? Et si le caractère est une chaîne vide ?

```
def appartient(chaine, caractere):
    '''
    :entree chaine: string
    :entree caractere: char
    :sortie present: booleen
    :post-cond: present est vrai uniquement si le caractère est trouvé dans la chaîne.
    '''
    present = False
    i = len(chaine) - 1
    while not present and i >= 0:
        if chaine[i] == caractere:
            present = True
        i = i - 1

    return(present)
```

Exercice 8. Ci-dessous se trouve un algorithme qui compte le nombre d'occurrences d'un caractère dans une chaîne. Écrire le contrat. Pensez bien à traiter tous les cas aux limites.

```
def nbOcc(chaine, caractere):
    '''

    '''
    compteur = 0
    for c in chaine:
        if c == caractere:
            compteur = compteur + 1
    return(compteur)
```

Exercice 9. Étant donné le contrat et l'algorithme suivant, écrire l'énoncé de l'exercice et l'algorithme répondant à la question. Observez les différences d'implémentation par rapport à l'exercice précédent.

```
def remplacer(chaine, origine, remplace):
    i = 0
    taille = len(chaine)
    nouvellechaine = ""
    while i < taille:
        if chaine[i] == origine:
            nouvellechaine = nouvellechaine + remplace
        else:
            nouvellechaine = nouvellechaine + chaine[i]
        i = i + 1
    return(nouvellechaine)
```

Exercice 10. L'algorithme suivant recherche si un mot contient deux lettres consécutives. Écrivez le contrat et complétez l'algorithme pour qu'il fonctionne. Comment se comporte l'algorithme sur la chaîne vide ? Qu'en déduisez-vous ?

```
def doubleLettre(chaine):
    double = False
    for i in range(len(chaine)-1):
        if chaine[i] == chaine[i+1]:
            double = True
    return(double)
```

Exercices pour aller plus loin...

Si vous avez fini tous les exercices de cette fiche, n'hésitez pas à piocher dans le recueil d'exercices. Par exemple, savez-vous calculer le nième terme de la suite de Fibonacci, ou bien la différences entre deux heures de la journée passée en paramètres ? Savez-vous coder et décoder une chaîne avec le codage de Cesar ? Et vérifier si une chaîne est inclue dans une autre ?