Algorithmique

Table des matières

Première partie
Cours

1 Quelques rappels

1.1 Qu'est ce qu'un algorithme ?

Définition :

Un algorithme est un enchainement des étapes ou instructions à effectuer dans un certain ordre et donc la réalisation va permettre la résolution d'un problème donné.

Remarque :

Le mot **algorithme** viens de nom d'un mathématicien perse **Al - Kawarizmi**, qui à vécu au 9ème siècle.

1.2 Qu'est ce qui constitue un algorithme ?

- Un début
- Une fin
- Des instructions
- Des variables

Exemple 1 :

Variables : a : entier
1 **début**
2 | $a \leftarrow 5$
3 | $a \leftarrow a + 2$
4 | Afficher (a)
5 **fin**

```
1 a = 0  # Déclarer et initialiser la variable 'a'
2 a = 5  # Assigner la valeur 5 à 'a'
3 a = a + 2  # Ajouter 2 à la valeur actuelle de 'a'
4 print(a)  # Afficher la valeur de 'a'
```

Exemple 2 :

Variables : a , b : entiers
1 **début**
2 | $a \leftarrow 5$
3 | $b \leftarrow a + 2$
4 | $a \leftarrow b + 5$
5 | Afficher (a)
6 | Afficher (b)
7 **fin**

```
1 a = 0  # Déclarer et initialiser la variable 'a'
2 b = 0  # Déclarer et initialiser la variable 'b'
3 a = 5  # Assigner la valeur 5 à 'a'
4 b = a + 2  # Assigner la valeur de 'a' + 2 à 'b'
5 a = b + 5  # Assigner la valeur de 'b' + 5 à 'a'
6 print(a)  # Afficher la valeur de 'a'
7 print(b)  # Afficher la valeur de 'b'
```

Exercice 1 :

Écrire un algorithme qui demande à l'utilisateur deux nombres réels et qui affiche leur somme et leur produit :

Entrées : Deux nombres réels a et b
Sorties : La somme s et le produit p
1 **début**
2 | $a \leftarrow$ Saisir ("Entrez le premier nombre réel : ")
3 | $b \leftarrow$ Saisir ("Entrez le deuxième nombre réel : ")
4 | $s \leftarrow a + b$
5 | $p \leftarrow a \times b$
6 | Afficher ("La somme est s")
7 | Afficher ("Le produit est p")
8 **fin**

```python
a = float(input("Entrez le premier nombre réel : "))
b = float(input("Entrez le deuxième nombre réel : "))

somme = a + b
produit = a * b

print("La somme est", somme)
print("Le produit est", produit)
```

Exercice 2 :

Écrire un algorithme qui demande à l'utilisateur deux nombres réels et qui affiche le plus grand :

Entrées : Deux nombres réels a et b
Sorties : Le plus grand nombre max

```
début
    a ← Saisir ("Entrez le premier nombre réel : ")
    b ← Saisir ("Entrez le deuxième nombre réel : ")
    si a ≥ b alors
        max ← a
    sinon
        max ← b
    fin
    Afficher "Le plus grand nombre est max"
fin
```

```python
a = float(input("Entrez le premier nombre réel : "))
b = float(input("Entrez le deuxième nombre réel : "))

if a >= b:
    plus_grand = a
else:
    plus_grand = b

print("Le plus grand nombre est", plus_grand)
```

Exercice 3 :

Écrire un algorithme qui demande à l'utilisateur un entier naturel et qui calcul et affiche la somme de tous les entiers inférieurs ou égaux à cet entier :

Entrées : Un entier naturel n
Sorties : La somme s des entiers inférieurs ou égaux à n

```
début
    n ← Saisir ("Entrez un entier naturel : ")
    s ← 0
    pour i de 1 à n faire
        s ← s + i
    fin
    Afficher "La somme est s"
fin
```

```python
n = int(input("Entrez un entier naturel : "))

somme = 0
for i in range(1, n + 1):
    somme += i

print("La somme est", somme)
```

2 Les Boucles en Algorithmique

Les boucles sont des structures essentielles en programmation qui permettent d'exécuter un bloc de code de manière répétée.

2.1 Boucle `for` (Pour)

La boucle `for` est utilisée pour itérer sur une séquence de valeurs et exécuter un bloc de code pour chaque élément de la séquence.

```
pour i de 1 à 5 faire
    Afficher(i)
fin
```

```python
# Boucle for en Python
for i in range(1, 6):
    print(i)
```

2.2 Boucle `while` (Tant Que)

La boucle `while` est utilisée pour exécuter un bloc de code tant qu'une condition est vraie.

Variables : i : entier
début
  $i \leftarrow 1$
  tant que $i \leq 5$ **faire**
    Afficher(i)
    $i \leftarrow i + 1$
  fin
fin

```python
# Boucle while en Python
i = 1
while i <= 5:
    print(i)
    i += 1
```

2.3 Boucle `do while` (Faire Tant Que

La boucle `do while`, également connue sous le nom de boucle "Faire Tant Que" en algorithmique, exécute un bloc de code au moins une fois, puis continue tant qu'une condition est vraie.

Variables : i : entier
début
  $i \leftarrow 1$
  faire
    Afficher (i)
    $i \leftarrow i + 1$
  tant que $i \leq 5$
fin

```python
# Boucle do while en Python
i = 1
while True:
    print(i)
    i += 1
    if i > 5:
        break
```

2.4 Boucle `foreach` (Pour Chaque)

La boucle `foreach`, également connue sous le nom de boucle "Pour Chaque" en algorithmique, itère sur les éléments d'une collection, telle qu'une liste.

Variables : liste : liste d'entiers
début
  liste ← [1, 3, 5, 7, 9]
  pour chaque *element dans liste* **faire**
    Afficher (*element*) ;
  fin
fin

```python
# Boucle foreach en Python
liste = [1, 3, 5, 7, 9]
for element in liste:
    print(element)
```

Deuxième partie
Exercices

3 Conjecture

3.1 Préambule

- On rappelle qu'un entier naturel est premier s'il admet exactement deux diviseurs distincts (1 et lui-même).

  Exemples et contre-exemples :

  - 0 n'est pas premier car il admet une infinité de diviseurs (tous les entiers naturels non nuls).
  - 1 n'est pas premier car il n'admet qu'un diviseur (1).
  - 2 est premier car il admet exactement 2 diviseurs (1 et 2).
  - 4 n'est pas premier car il admet 3 diviseur (1, 2 et 4).

- On souhaite tester si la propriété mathématique suivante est vraie :
  « **Pour $n > 1$, les termes de la suite (S_n) définie par $S_n = 2^0 + 2^1 + \ldots + 2^n$ sont alternativement premiers et non premiers.** »

  Exemples :

  - $S_2 = 2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7$ (7 est un nombre premier)
  - $S_3 = 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15$ (15 n'est un nombre premier)
  - $S_4 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 1 + 2 + 4 + 8 + 16 = 31$ (31 est un nombre premier)

3.2 Fonction `mystere1`

```
1 Fonction mystere1(k : entier) :
  |  Entrées : Un entier k
  |  Sortie  : Un entier
  |  Data : j : entier
2 |  resultat ← 1;
3 |  pour j allant de 1 à k faire
4 |  |  resultat ← resultat * 2;
5 |  fin
6 |  return resultat;
```

```python
# Fonction mystere1 en Python
def mystere1(k):
    resultat = 1
    for j in range (1, k + 1):
        resultat = resultat * 2
    return (resultat)
```

3.3 Fonction `mystere2`

```
1  Fonction mystere2(N : entier) :
2  |  si (N<2) alors
3  |  |  res ← 0;
4  |  fin
5  |  sinon
6  |  |  res ← 1;
7  |  |  i ← 2;
8  |  |  tant que i < N faire
9  |  |  |  si (N mod i = 0) alors
10 |  |  |  |  res ← 0;
11 |  |  |  fin
12 |  |  |  i ← i + 1;
13 |  |  fin
14 |  fin
15 |  return res;
```

```python
# Fonction mystere2 en Python
def mystere2(N):
    res = 0
    i = 0
    if(N < 2):
        res = 0
```

```
7       else :
8           res = 1
9           i = 2
10          while(i < N):
11              if(N % i == 0):
12                  res = 0
13              i = i + 1
14      return (res)
```

3.4 Questions et réponses

A1. Quel est le résultat renvoyé par `mystere1(4)` ? Quel est le rôle de cette fonction ?

Le résultat renvoyé par `mystere1(4)` est 16.
Le rôle de cette fonction est de calculer la valeur de 2 élevé à la puissance k, où k est un nombre entier passé en paramètre à la fonction.

A2. Lister les variables utilisées dans la fonction `mystere2` et leur type.

Les variables utilisées dans la fonction `mystere2` et leur type sont :

- **N** : un entier (passé en paramètre).
- **res** : un entier.
- **i** : un entier.

A3. Quelles instruction faut-il ajouter dans la fonction `mystere2` à la place des points des suspension ? Que se passe-t-il si on ne l'ajoute pas ?

Pour compléter la fonction `mystere2`, il faut ajouter l'instruction $i \leftarrow i+1$ à la fin de la boucle `while`, juste avant la fin de la fonction.

Si on ne l'ajoute pas, la boucle `while` sera exécutée indéfiniment, ce qui est appelé une boucle infinie. Cela se produit parce que i ne change jamais de valeur, donc la condition de la boucle (i < N) est toujours vraie.

A4. Que signifie la condition *(N mod i = 0)* ?

La condition *(N mod i = 0)* ou *(N % i == 0)* teste si N est divisible par i sans reste. En d'autres termes, cela signifie que i est un diviseur de N.

A5. a) Quel est le résultat renvoyé par `mystere2(1)` ? par `mystere2(3)` ? par `mystere2(4)` ?

- Le résultat renvoyé par **`mystere2(1)`** est 0.
- Le résultat renvoyé par **`mystere2(3)`** est 1.
- Le résultat renvoyé par **`mystere2(4)`** est 0.

b) Expliquer le rôle de cette fonction.

La fonction `mystere2` permet de déterminer si un entier N est premier ou non.

Si la fonction renvoie 0 = le nombre n'est pas premier.
Si la fonction renvoie 1 = le nombre est pas premier.

A6. Écrire l'algorithme `testPremier` qui demande à l'utilisateur de saisir un entier strictement positif noté s et qui affiche un message indiquant si s est premier ou non. On pourra utiliser les fonctions `mystere1` et `mystere2`.

```
1  Fonction testPremier(s : entier) :
       Entrée : Un entier s
2      si s < 2 alors
3          Afficher(s, "n'est pas un nombre premier.") retourner
4      si mystere2(s) = 1 alors
5          Afficher(s, "est un nombre premier.")
6      sinon
7          Afficher(s, "n'est pas un nombre premier.")
8      fin
9  s ← Saisir ("Saisissez un entier strictement positif : ")
10 Afficher (testPremier(s))
```

```python
# Fonction testPremier en Python
def testPremier(s):
    if s < 2:
        print(s, "n'est pas un nombre premier.")
        return

    if mystere2(s) == 1:
        print(s, "est un nombre premier.")
    else:
```

```
10          print(s, "n'est pas un nombre premier.")
11
12  # Demande à l'utilisateur de saisir un entier positif
13  s = int(input("Saisissez un entier strictement positif :
        "))
14  testPremier(s)
```

A7. Écrire l'algorithme de la fonction `calcsn(n)`. On pourra utiliser les fonctions `mystere1` ou `mystere2`.

```
1  Fonction calcsn(n : entier) :
       Entrée : Un entier n
       Sortie  : Un entier
       Data : resultat : entier
2      somme ← 0;
3      pour i allant de 0 à n faire
4          somme ← somme + mystere1(i);
5      fin
6      si mystere2(somme) = 1 alors
7          resultat ← somme;
8      sinon
9          resultat ← -somme;
10     fin
11     return resultat;
```

```
1  # Fonction calcsn en Python
2
3  # Définition de la fonction calcsn
4  def calcsn(n):
5      # Initialisation de la somme
6      somme = 0
7
8      # Boucle pour calculer la somme des termes de la
           suite Sn
9      for i in range(n + 1):
10         # Appel à la fonction mystere1 pour calculer le
               terme 2^i
11         terme = mystere1(i)
12
13         # Ajout du terme à la somme
14         somme += terme
15
16     # Test si la somme est un nombre premier en
           utilisant mystere2
17     if mystere2(somme) == 1:
18         # Si la somme est premier, on la renvoie
19         resultat = somme
```

```
20      else:
21          # Sinon, on renvoie la somme avec un signe né
                gatif
22          resultat = -somme
23
24      # Renvoi du résultat
25      return resultat
```

Cet algorithme calcule la somme des termes de la suite S_n définie par $S_n = 2^0 + 2^1 + \ldots + 2^n$ en utilisant la fonction `mystere1`. Ensuite, il teste si la somme obtenue est un nombre premier en utilisant la fonction `mystere2`. Si la somme est un nombre premier, la fonction `calcsn(n)` renvoie la somme. Sinon, elle renvoie la somme avec un signe négatif.

4 Nombres Zigzag

4.1 Principe

On appelle **nombre zigzag** un nombre entier vérifiant les conditions suivantes :

- Le nombre est supérieur ou égal à 100 (donc il a au minimum trois chiffres) ;
- On parcourt le nombre par ses chiffres pris de gauche à droite ;
- À partir du second chiffre du nombre, ceux-ci sont alternativement strictement plus grand ou strictement plus petit qur leur prédécesseur.

Exemples :

- L'entier 51 627 est un nombre zigzag car, en partant de 5, on a : $1 < 5$, $6 > 1$, $2 < 6$, et $7 > 2$;
- Les entiers 131 ; 2 020 sont aussi des nombres zigzag ;
- 347 n'est pas un nombre zigzag car $4 > 3$ et $7 > 4$;
- Le nombre 344 n'est pas un nombre zigzag car deux chiffres consécutifs ne peuvent être égaux.

4.1.1 Questions et réponses

A1. Donner tous les nombres zigzag à trois chiffres commençant par 26 ?

Les nombres zigzag à trois chiffres commençant par 26 sont : 260 , 261 , 262 , 263 , 264 , 265.

A2. Parmi les nombres suivants, quels sont ceux qui sont Zigzag ?

$a = \mathbf{4\ 781}$; $b = \mathbf{703\ 154}$; $c = \mathbf{2\ 016}$; $d = \mathbf{49\ 376\ 801.}$

Parmi les nombres donnés, les nombres zigzag sont : b = 703 154 et d = 49 376 801.

4.2 Une fonction

On considère la fonction `Test` ci-dessous :

```
1 Fonction Test(a, b ,c : entiers) :
  |  Entrée : Trois entiers : a, b, c
  |  Sortie  : Un booléen : resultat
  |  /* a, b et c désignent trois chiffres consécutifs d'un
  |     nombre entier                                        */
2 |  si (b-a)*(c-b) < 0 alors
3 |  |  resultat ← Vrai
4 |  sinon
5 |  |  resultat ← Faux
6 |  fin
7 |  return resultat
```

4.2.1 Questions et réponses

A3. Préciser la valeur de sortie de chacune des instructions suivantes :

- `Test(3, 4, 1)`

  Vrai

- `Test(9, 5, 4)`

  Faux

- `Test(6, 0, 0)`

  Faux

- `Test(1, 7, 3)`

  Vrai

A4. Quel est le rôle de cette fonction `Test` dans le cadre des nombres zigzag ?

Le rôle de la fonction `Test` dans le cadre des nombres zigzag est de vérifier si les chiffres consécutifs a, b et c d'un nombre entier forment une séquence zigzag. En d'autres termes, la fonction vérifie si le chiffre du milieu (b) est soit strictement plus grand que ses voisins (a et c), soit strictement plus petit. Si c'est le cas, la fonction renvoie $Vrai$, indiquant que les chiffres

sont dans une configuration zigzag. Sinon, elle renvoie $Faux$. Cette fonction est utilisée pour déterminer si un ensemble de trois chiffres consécutifs d'un nombre satisfait la condition de nombre zigzag.

4.3 L'algorithme principal

On donne maintenant l'algorithme principal nommé `Zigzag_ou_pas` dans lequel on utilise la fonction `Test` précédemment définie.
Dans cet algorithme, on utilise également les deux fonctions suivantes qui permettent de transtyper des variables :

- `Entier_chaine(Entier : N)` qui convertit l'entier `N` en chaine de caractères ;
- `Chaine_entier(Chaine : mot_nb)` qui convertit la chaine de caractères `mot_nb` en entier.

Dans cet algorithme on utilise enfin la fonction `Long_Chaine()` qui envoie un entier :

- `Long_Chaine(Chaine : mot_nb)` permet d'obtenir le nombre de caractères de la chaine de caractères `mot_nb`.

On rappelle que dans une chaine de `n` caractères, les caractères sont numérotés de 0 à `n`-1.

Variables :

Entiers : *N, i, chiffre1, chiffre2, chiffre3*
Chaine de caractères : *mot_nb*
Booléen : *zigzag*

```
1  début
2  |  zigzag ← Vrai
3  |  i ← 0
4  |
   |  /* Block d'instruction 1                                    */
5  |  ⋮
   |  /* Fin du block d'instruction 1                            */
6  |
7  |  mot_nb ← Entier_Chaine(N)
8  |  tant que zigzag et i < Longe_Chaine(mot_nb) - 3 faire
9  |  |  chiffre1 ← Chaine_Entier(mot_nb[i])
10 |  |  chiffre2 ← Chaine_Entier(mot_nb[i + 1])
11 |  |  chiffre3 ← Chaine_Entier(mot_nb[i + 2])
12 |  |
   |  |  /* Block d'instruction 2                                 */
13 |  |  zigzag ← Test(···)
   |  |  /* Fin du block d'instruction 2                         */
14 |  |
15 |  |  i ← i + 1
16 |  fin
   |  /* Block d'instruction 3                                    */
17 |  si (··· alors
18 |  |  ⋮
19 |  fin
   |  /* Fin du block d'instruction 3                            */
20 |
21 fin
```

4.3.1 Questions et réponses

A5. `N` est la variable contenant le nombre entier à tester pour sa voir s'il est ou pas un nombre zigzag.
Dans l'algorithme principal, écrire le bloc d'instructions 1 permettant d'effectuer la saisie de la variable `N` et de tester si le nombre est à trois chiffres au moins et ne commence pas par zéro ; si ce n'est pas le cas, effectuer une nouvelle saisie jusqu'à ce qu'il convienne.

```
1 tant que N < 100 ou Long_Chaine(N) < 3 ou
    Chaine_Entier(N[0]) = 0 faire
2 |  N ← Saisir("Saisissez un nombre : ")
3 fin
```

A6. Dans le bloc d'instructions 2, écrire l'instruction d'appel de la fonction `Test`.

```
1 zigzag ← Test(chiffre1, chiffre2, chiffre3)
```

A7. Écrire le bloc d'instructions 3 permettant d'afficher si le nombre `N` est zigzag ou pas.

```
1 si zigzag = Vrai alors
2 |  Afficher(N, "est un nombre zigzag.")
3 sinon
4 |  Afficher(N, "n'est pas un nombre zigzag.")
5 fin
```

4.4 Questions et réponses

B1. Programmer dans le langage de votre choix l'algorithme principal `Zigzag_ou_pas`, ainsi que la fonction `Test`.

```python
def Entier_Chaine(N):
    return str(N)

def Chaine_Entier(mot_nb):
    return int(mot_nb)

def Long_Chaine(mot_nb):
    return len(mot_nb)

def Test(a, b, c):
    if (b - a) * (c - b) < 0:
        return True
    else:
        return False

def Zigzag_ou_pas():
```

```
17      N = int(input("Saisissez un nombre supérieur ou égal
            à 100 : "))
18      while N < 100:
19          N = int(input("Saisissez un nombre supérieur ou
                égal à 100 : "))
20
21      mot_nb = Entier_Chaine(N)
22      zigzag = True
23      i = 0
24
25      while zigzag and i < Long_Chaine(mot_nb) - 2:
26          chiffre1 = Chaine_Entier(mot_nb[i])
27          chiffre2 = Chaine_Entier(mot_nb[i + 1])
28          chiffre3 = Chaine_Entier(mot_nb[i + 2])
29
30          zigzag = Test(chiffre1, chiffre2, chiffre3)
31
32          i += 1
33
34      if zigzag:
35          print(N, " est un nombre zigzag.")
36      else:
37          print(N, " n'est pas un nombre zigzag.")
38
39  # Appel de la fonction Zigzag_ou_pas
40  Zigzag_ou_pas()
```

B2. Ce programme peut-il planter lors de la saisie du nombre à tester ?
Si non, justifier la réponse.
Si oui, comment peut-on y remédier ?

Oui, le programme peut planter lors de la saisie du nombre à tester. Si l'utilisateur entre une valeur non numérique (par exemple, du texte), le programme générera une exception lorsqu'il essaie de convertir l'entrée en un entier.

Pour remédier à cela, vous pouvez envelopper la saisie utilisateur dans une structure de validation pour vous assurer que l'entrée est un entier valide avant de l'utiliser dans le programme. Par exemple, vous pouvez utiliser une boucle avec des instructions `try` et `except` pour gérer les exceptions lors de la conversion de l'entrée en entier. Si une exception est levée, vous pouvez afficher un message d'erreur et demander à l'utilisateur de saisir à nouveau une entrée valide.

5 Fréquence d'une lettre

Pour coder un message, on peut effectuer une substitution de lettre. Jules César effectuait un décalage de trois lettres dans l'alphabet : il remplaçait les a par d, les b par e, ... Le message obtenu devient alors illisible pour celui qui ne connait pas les correspondances entre les lettres.

L'une des solutions pour trouver ces correspondances est de calculer les fréquences de chaque lettre. On sait en effet que c'est la lettre e qui est la plus fréquente. Ainsi, la lettre qui obtiendra la fréquence la plus élevée d'apparition dans le message codé correspondra au e.

5.1 Partie 1 - Algorithmique

On vous fournit l'algorithme suivant :

```
1  début
2  |  message ← Saisir("Saisissez votre message : ")
3  |
4  |  n ←longueur(message)
5  |  i ← 0
6  |  Nb ← 0
7  |
8  |  tant que i < n faire
9  |  |  si message[i] = "a" alors
10 |  |  |  Nb ← Nb + 1
11 |  |  fin
12 |  |  ......
13 |  fin
14 |  Afficher(Nb)
15 fin
```

5.1.1 Questions et réponses

1.1. Quel traitement réalise cet algorithme ?

Cet algorithme compte et affiche le nombre de lettres "a" dans le message.

1.2. Quel est l'instruction qu'il doit y avoir à la ligne 12 et pourquoi ?

12 $i \leftarrow i + 1$

L'objectif de cette ligne est d'incrémenter la variable i pour passer au caractère suivant dans la boucle.

1.3. Listez les variables utilisées et préciser leur type.

- `message` : Chaîne de caractères (string) pour stocker le message saisi par l'utilisateur.
- `n` : Entier (integer) pour stocker la longueur du message (nombre de caractères).
- `i` : Entier (integer) pour représenter l'index actuel lors de la boucle de parcours du message.
- `Nb` : Entier (integer) pour compter le nombre de lettres "a" dans le message.

1.4. À la lecture de l'algorithme, veuillez donner les valeurs que prendront les variables n et Nb pour les cas suivants : message = « Fsbrsn jcig aofrw aohwb » et de message = « Jwjs zsg Aohvsaohweisg »

- Pour `message = « Fsbrsn jcig aofrw aohwb »` :
  - `n` : La longueur du message est de 23 caractères.
  - `Nb` : Le nombre de lettres "a" dans le message est 2.
- Pour `message = « Jwjs zsg Aohvsaohweisg »` :
  - `n` : La longueur du message est de 22 caractères.
  - `Nb` : Le nombre de lettres "a" dans le message est 1.

1.5. Proposez une modification qui pourrait être effectuée sur la ligne 9 pour tenir compte de la casse.

Pour tenir compte de la casse et compter à la fois les lettres "a" majuscules et minuscules, nous pouvons modifier la ligne 9 comme suit :

9 **si** $message[i] = "a"$ *ou* $message[i] = "A"$ **alors**

Pour la suite du problème, on supposera que le message est écrit en minuscules.

1.6. Effectuez les ajouts nécessaires à l'algorithme afin de pouvoir connaître la fréquence en pourcentage d'apparition de la lettre "a".

```
1  début
2  |  message ← Saisir("Saisissez votre message : ")
3  |
4  |  n ←longueur(message)
5  |  i ← 0
6  |  Nb ← 0
7  |
8  |  tant que i < n faire
9  |  |  si message[i] = "a" ou message[i] = "A" alors
10 |  |  |  Nb ← Nb + 1
11 |  |  fin
12 |  |  i ← i + 1
13 |  fin
14 |
   |  /* Calcul de la fréquence en pourcentage                    */
15 |  fréquence ← (Nb/n) * 100
16 |
17 |  Afficher("Nombre de lettres 'a' : ", Nb)
18 |  Afficher("Fréquence en pourcentage : ", fréquence, "%")
19 fin
```

5.2 Partie 2 - Programmation

5.2.1 Questions et réponses

2.1. Traduisez votre algorithme dans le langage de votre choix.

```python
def calculate_frequency(message):
    n = len(message)
    i = 0
    Nb = 0

    while i < n:
        if message[i] == "a" or message[i] == "A":
            Nb += 1
        i += 1

    frequency = (Nb / n) * 100
    return Nb, frequency

```

```
message = input("Saisissez votre message : ")
Nb, frequency = calculate_frequency(message)

print(f"Nombre de lettres 'a' : {Nb}")
print(f"Fréquence en pourcentage : {frequency:.2f}%")
```

2.2. Modifier votre programme pour que chaque lettre et sa fréquence d'apparition soit stockée dans un tableau.

```
def calculer_frequence(message):
    n = len(message)
    frequences = {}

    # Parcours de chaque caractère dans le message
    for char in message:
        # Ignorer les caractères qui ne sont pas des lettres
        if char.isalpha():
            char_lower = char.lower()
            # Vérifier si le caractère existe déjà dans le dictionnaire de fréquences
            if char_lower in frequences:
                frequences[char_lower] += 1
            # Si le caractère n'existe pas, l'ajouter au dictionnaire avec un compteur initial de 1
            else:
                frequences[char_lower] = 1

    # Calcul des fréquences en pourcentage pour chaque lettre dans le dictionnaire
    for char in frequences:
        frequences[char] = (frequences[char] / n) * 100

    return frequences

message = input("Saisissez votre message : ")
frequences = calculer_frequence(message)

# Création d'une liste pour stocker les lettres, les fréquences et les nombres de fois
lettres_frequences_nombres = []

# Ajout de chaque lettre, sa fréquence et son nombre de fois à la liste
for char, frequence in frequences.items():
    lettres_frequences_nombres.append((char, frequence, message.lower().count(char)))
```

```
32
33 # Affichage de chaque lettre, sa fréquence et son nombre
       de fois
34 for lettre, frequence, nombre in
       lettres_frequences_nombres:
35     print(f"Lettre '{lettre}': Fréquence en pourcentage
           : {frequence:.2f}%, Nombre de fois : {nombre}")
```

6 Clé de Luhn

En algorithmique, nous utiliserons la syntaxe suivante pour les opérations de transtypage (modification du type d'une variable) :

- `str()` permet de transformer un entier en chaîne de caractères. Par exemple, `str(123)` renvoie la chaîne "123".
- `int()` permet de transformer une chaîne de caractères formée de chiffres en entier. Par exemple, `int(123)` renvoie l'entier "123".

Rappel : si `chaine` est une chaine de caractères, sa longueur est égale au nombre de caractères de cette chaine, de plus `chaine[0]` représente le caractère le plus à gauche. Par exemple, si `chaine` = "257", alors `longueur(chaine)` = 3, `chaine[0]` = "2", `chaine[1]` = "5", `chaine[2]` = "7".

Remarque : on note `""` la chaine vide, sa longueur est égale à 0.

6.1 Partie A - Algorithmique

A1. On considère l'algorithme ci-dessous :

Variables :
Entiers : *nombre, nombrebis, lg, i ,x*
Chaines de caractères : *chaine, chainebis*

```
1  début
2      nombre ← Saisir("Veuillez saisir un nombre entier à 7 chiffre : ")
3      chaine ← str(nombre)
4      lg ← longueur(chaine)
5      chainebis ← ""
6
7      pour i variant de 0 à lg-1 faire
8          si i%2 ≠ 0 alors
9              chainebis ← chainebis + chaine[i]
10         sinon
11             x ← int(chaine[i])
12             x ← 2 * x
13             si x > 9 alors
14                 x ← x − 9
15             fin
16             chainebis ← chainebis + str(x)
17         fin
18     fin
19     nombrebis ← int(chainebis)
20     Afficher(nombrebis)
21 fin
```

L'utilisateur saisit le nombre 1234567. Déterminer l'affichage produit par cet algorithme.

L'algorithme prend un nombre entier de 7 chiffres en entrée et effectue les opérations suivantes :

1. Il transforme le nombre en une chaîne de caractères.
2. Il calcule la longueur de la chaîne.
3. Il initialise une nouvelle chaîne vide appelée `‘chainebis‘`.
4. En parcourant chaque chiffre de la chaîne, il effectue les opérations suivantes :
   - Si l'indice `‘i‘` est impair (c'est-à-dire ‘i % 2!= 0‘), il ajoute simplement le chiffre à la chaîne `‘chainebis‘`.
   - Si l'indice `‘i‘` est pair, il multiplie le chiffre par 2 et le stocke dans la variable `‘x‘`. Si `‘x‘` est supérieur à 9, il soustrait 9 de `‘x‘`.
   - Enfin, il ajoute le résultat `‘x‘` à la chaîne `‘chainebis‘`.

5. Une fois la boucle terminée, il transforme la chaîne `‘chainebis‘` en un entier `‘nombrebis‘` et l'affiche.

Si l'utilisateur saisit le nombre 1234567, voici comment l'algorithme se déroulera :

- `‘chaine‘` = "1234567"
- `‘lg‘` = 7
- `‘chainebis‘` = ""

Maintenant, parcourons les chiffres de `‘chaine‘` :

- `‘chaine[0]‘` = "1", `i` = 0 (pair)
  - `x` = 1 * 2 = 2
  - `‘chainebis‘` = "2"

- `‘chaine[1]‘` = "2", `i` = 1 (impair)
  - `‘chainebis‘` = "2" + "2" = "22"

- `‘chaine[2]‘` = "3", `i` = 2 (pair)
  - `x` = 3 * 2 = 6
  - `‘chainebis‘` = "22" + "6" = "226"

- `‘chaine[3]‘` = "4", `i` = 3 (impair)
  - `‘chainebis‘` = "226" + "4" = "2264"

- `‘chaine[4]‘` = "5", `i` = 4 (pair)
  - `x` = 5 * 2 = 10 (`x` > 9, donc `x` = `x` - 9 = 1)
  - `‘chainebis‘` = "2264" + "1" = "22641"

- `‘chaine[5]‘` = "6", `i` = 5 (impair)
  - `‘chainebis‘` = "22641" + "6" = "226416"

- `‘chaine[6]‘` = "7", `i` = 6 (pair)
  - `x` = 7 * 2 = 14 (`x` > 9, donc `x` = `x` - 9 = 5)
  - `‘chainebis‘` = "226416" + "5" = "2264165"

Finalement, `'nombrebis'` = 2264165 et c'est ce qui sera affiché par l'algorithme.

A2. On considère la fonction `Sommechiffre` écrite ci-dessous de manière incomplète.
On vous demande de compléter l'écriture de cette fonction de façon qu'elle calcule la somme des chiffres du paramètre `nombre`.

```
1  Fonction Sommechiffre(nombre : entier) :
       Variables :
       Entiers : lg, i ,a, resultat
       Chaine de caractères : chaine

2      début
3          chaine ← str(nombre)
4          lg ← longueur(chaine)
5
6          .....
7
8          pour i variant de 0 à lg-1 faire
9              a ← int(chaine[i])
10
11             .....
12
13         fin
14         return resultat
15     fin
```

Voici l'algorithme `Sommechiffre` complété :

```
1  Fonction Sommechiffre(nombre : entier) :
       Variables :
       Entiers : lg, i ,a, resultat
       Chaine de caractères : chaine

2      début
3          chaine ← str(nombre)
4          lg ← longueur(chaine)
5          resultat ← 0
6
7          pour i variant de 0 à lg-1 faire
8              a ← int(chaine[i])
9              resultat ← resultat + a
10         fin
11         return resultat
12     fin
```

A3. Exposé du problème :

On considère une carte bancaire dont le numéro comporte seize chiffres. Les quinze premiers chiffres donnent des renseignements sur le type de carte, le nom de la banque, le numéro du compte, etc. le chiffre des unités, c'est-à-dire celui de droite, est la clé de Luhn Il est calculé à partir des quinze autres chiffres en respectant l'algorithme décrit dans l'exemple suivant :

Supposons que les quinze premiers chiffres de la carte soient : **4970 1012 3456 789**

- **Étape n°1**

  On multiple le premier chiffre de gauche (d'indice 0) par 2. Si le résultat de la multiplication par 2 est strictement supérieur à 9, on retranche 9.
  Le deuxième chiffre (d'indice 1) en partant de la gauche reste inchangé.
  On répète ces deux instructions jusqu'au quinzième chiffre. Les chiffres d'indice pair sont traités comme le chiffre d'indice 0, les chiffres d'indice impair sont inchangé.

  Sur cette exemple :

  pour le premier chiffre, 4 devient 8.
  pour le deuxième chiffre, 9 reste 9.

pour le troisième chiffre, 7 devient 5 car 7×2 = 14 et 14-9 = 5.
pour le quatrième chiffre, 0 reste 0.
et ainsi de suite.

À la fin de la première étape, on obtient **8950 2022 6416 589**

- **Étape n°2**

  On calcule la somme des quinze chiffres obtenus à la fin de l'étape n°1.

  Sur l'exemple, 8+9+5+0+2+0+2+2+6+4+1+6+5+8+9 = **67**

- **Étape n°3**

  On calcule le reste de la division euclidienne par 10 du nombre obtenu à l'étape n °2.
  La clé de Luhn est 10 - reste précédent.

  Sur l'exemple, le reste de la division euclidienne de 67 par 10 est 7.
  La clé est 10 - 7 = **3**

Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier à 15 chiffres et qui calcule et affiche sa clé de Luhn. On pourra utiliser les question précédentes, notamment la fonction `Sommechiffre`.

Entrées : Nombre entier à 15 chiffres : *nombre*
Sortie : Clé de Luhn : *cle*
Variables :
Entiers : *lg, i ,a, resultat, sommeChiffres, cle*
Chaine de caractères : *chaine*

```
1  début
2      chaine ← str(nombre)
3      lg ← longueur(chaine)
4      resultat ← 0
5
6      sommeChiffres← 0
7
8      pour i variant de 0 à lg-1 faire
9          a ← int(chaine[i])
10
11         si i est pair alors
12             a ← 2 × a
13             si a > 9 alors
14                 a ← a − 9
15             fin
16         resultat ← resultat + a
17     fin
18     sommeChiffres ← Sommechiffre (resultat)
19     cle ← 10−(sommeChiffres  mod 10)
20     Afficher("La clé de Luhn est : ", cle)
21 fin
```

6.2 Partie B - Programmation

B1. Traduire en langage de programmation l'algorithme A1 :

```
1  nombre = int(input("Veuillez saisir un nombre entier à 7
       chiffres : "))
2  chaine = str(nombre)
3  lg = len(chaine)
4  chainebis = ""
5
6  for i in range(lg):
7      if i % 2 != 0:
8          chainebis += chaine[i]
9      else:
10         x = int(chaine[i])
11         x = 2 * x
12         if x > 9:
```

```
13              x = x - 9
14          chainebis += str(x)
15
16  nombrebis = int(chainebis)
17  print(nombrebis)
```

B2. Traduire en langage de programmation la fonction écrite à la question A2 :

```
1  def Sommechiffre(nombre):
2      chaine = str(nombre)
3      lg = len(chaine)
4      resultat = 0  # Initialisation du résultat à 0
5
6      for i in range(lg):
7          a = int(chaine[i])
8          resultat += a  # Ajout du chiffre à la somme
9
10     return resultat
```

B3. À la suite du programme B2, écrire le programme traduisant l'algorithme de la question A3 :

```
1  def Sommechiffre(nombre):
2      chaine = str(nombre)
3      lg = len(chaine)
4      resultat = 0
5
6      for i in range(lg):
7          a = int(chaine[i])
8          resultat += a
9
10     return resultat
11
12 def CalculCleLuhn(nombre):
13     chaine = str(nombre)
14     lg = len(chaine)
15     resultat = 0
16     sommeChiffres = 0
17
18     for i in range(lg):
19         a = int(chaine[i])
20         if i % 2 == 0:
21             a *= 2
22             if a > 9:
```

```
23                 a -= 9
24         resultat += a
25
26     sommeChiffres = Sommechiffre(resultat)
27     cle = 10 - (sommeChiffres % 10)
28
29     print("La clé de Luhn est :", cle)
30
31 # Demande à l'utilisateur de saisir un nombre entier à
       15 chiffres
32 nombre = int(input("Veuillez saisir un nombre entier à
       15 chiffres : "))
33 CalculCleLuhn(nombre)
```

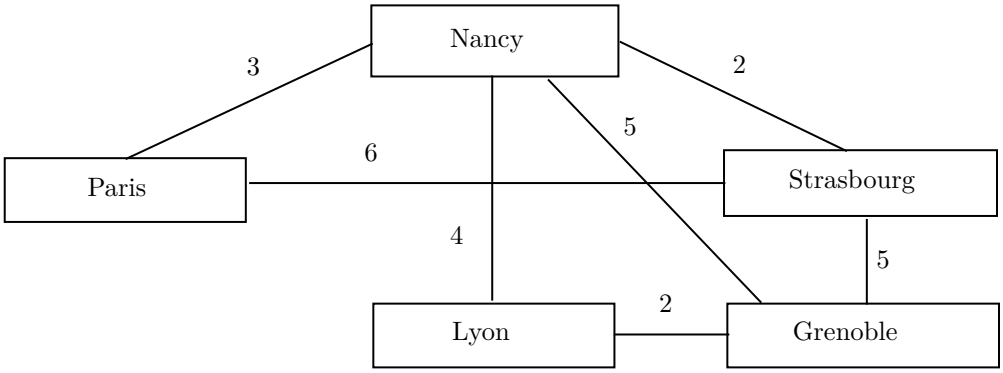
7 Calcul de Tournée

7.1 Préambule

Un représentant de commerce est susceptible de se rendre dans un certain nombre de villes. Lorsqu'il effectue une tournée, il part d'une ville, visite d'autres villes puis revient à son point de départ.

On suppose qu'il peut se déplacer dans 5 villes et que l'on connaît le temps de déplacement en heures, entre deux villes.

Les liaisons de ville à ville possibles sont représentées par un graphe, avec les temps de trajet :



On numérote les villes de 0 à 4 :

- 0 : Nancy
- 1 : Paris
- 2 : Lyon
- 3 : Grenoble
- 4 : Strasbourg

On définit la matrice des temps de trajet associée au graphe. La valeur -1 indique qu'il n'y a pas d'étape possible entre deux villes.

$$G = \begin{pmatrix} -1 & 3 & 4 & 5 & 2 \\ 3 & -1 & -1 & -1 & 6 \\ 4 & -1 & -1 & 2 & -1 \\ 5 & -1 & 2 & -1 & 5 \\ 2 & 6 & -1 & 5 & -1 \end{pmatrix}$$

On représente une tournée par un tableau contenant la ville de départ, puis les villes suivantes dans l'ordre. Après la dernier ville, le représentant de commerce doit revenir à la ville initiale.

Exemple : La tournée **Nancy - Grenoble - Strasbourg - Paris - Nancy** est représentée par le tableau :

0	3	4	1

7.2 Partie A - Algorithmique

On suppose que :

- Le premier indice d'un tableau est 0
- Le graphe est représenté par le tableau G de 5 lignes et 5 colonnes et qu'il est déjà initialisé.
- Le tableau `NomVilles` contient les noms des 5 villes :

Nancy	Paris	Lyon	Grenoble	Strasbourg

7.2.1 Questions et réponses

A1. Vérifier si la tournée suivante est possible :

2	0	1	4

Tournée :

2	0	1	4

- 0 : Nancy
- 1 : Paris
- 2 : Lyon
- 3 : Grenoble
- 4 : Strasbourg

On peut bien se rendre de Lyon à Nancy, de Nancy à Paris, de Paris à Strasbourg mais de Strasbourg on ne peut pas revenir à Lyon. Cette tournée n'est donc pas possible.

A2. Expliquer le rôle de la procédure suivante :

```
1 Procédure Mystère(j : entier) :
  |  Variables : i : entier
2 |  début
3 |  |  pour i allant de 0 à 4 faire
4 |  |  |  si G[i][j] ≠ −1 alors
5 |  |  |  |  Afficher NomVilles[i]
6 |  |  |  fin
7 |  |  fin
8 |  fin
```

Cette procédure affiche les prédécesseurs d'un sommet du graphe, c'est-à-dire les villes d'où l'on peut provenir lorsqu'on est dans la ville de n° i.

A3. Écrire un algorithme qui, à partir d'une ville donnée, détermine la ville la plus éloignée que le représentant peut rejoindre en une seule étape. Les villes pourront être désignée par leur numéro.

```
   /* Algorithme VilleEloignee                                          */
   Variables : V, DureeMax, Ville, i : entiers
 1 début
 |   /* On suppose que le n° de ville est bien compris entre 0
 |      et 4                                                            */
 2 |   V ← Saisir("Veuillez saisir le numéro de la ville : ")
 3 |   DureeMax ← −1
 4 |   Ville ← −1
 5 |   pour i allant de 0 à 4 faire
 6 |   |   si G[V][i] ≠ −1 et G[V][i] > DureeMax alors
 7 |   |   |   DureeMax ← G[V][i]
 8 |   |   |   Ville ← i
 9 |   |   fin
10 |   fin
11 |   si Ville¬ − 1 alors
12 |   |   Afficher("La ville la plus éloignée de ", NomVilles[V], "est : ",
   |   |    NomVilles[Ville], "(", DuréeMax, ")")
13 |   sinon
14 |   |   Afficher("Aucune ville n'est accessible à partir de la ville n°", V)
15 |   fin
16 fin
```

A4. Écrire une fonction qui détermine si une tournée est possible ou non.

```
1  Fonction TestTournee(T : tableau d'entiers) :
       Variables :
       Depart, i ,N : entiers
       Possible : booléen
2      début
3          N ← Nbéléments(T) /* N contient le nombre d'éléments
               dans le tableau T                                  */
4
5          Depart ← Tournee(0) /* On suppose qu'il y a au moins
               une ville dans le tableau de la tournée            */
6
7          Possible ←Vrai
8          i ← 0
9          tant que Possible et i < N faire
10             i ← i + 1
11             Arrivee ← T[i]
12         fin
13         si G[Depart][Arrivee] ≠ −1 alors
14             si G[i][j] ≠ −1 alors
15                 Depart ← Arrivee
16             sinon
17                 Possible ←Faux
18             fin
19         fin
           /* Il faut ensuite vérifier que l'on peut revenir au
              point de départ, c'est-à-dire qu'il y a une étape
              possible entre la dernière ville de la tournée et
              la première                                         */
20         Possible ← Possible et G[T[N − 1]][T[0]] ≠ −1
21
22         return Possible
23     fin
```

A5. (BONUS) Comment modifier la fonction de la question A4 pour que la fonction détermine en plus la durée du trajet lorsque la tournée est possible.

```
1  Fonction TestTournee(T : tableau d'entiers) :
       Variables :
       Depart, Arrivee, DureeTotale, N, i : entiers
       Possible : booléen
2      début
3          N ← Nbéléments(T)
4          Depart ← T[0]
5          Possible ← Vrai
6          DureeTotale ← 0
7          i ← 0
8          tant que Possible et i < N faire
9              i ← i + 1
10             Arrivee ← T[i]
11             si G[Depart][Arrivee] = −1 alors
12                 Possible ← Faux
13             fin
14             DureeTotale ← DureeTotale + G[Depart][Arrivee]
15             Depart ← Arrivee
16         fin
17         si Possible et G[T[N − 1]][T[0]] ≠ −1 alors
18             DureeTotale ← DureeTotale + G[T[N − 1]][T[0]]
19         sinon
20             Possible ← Faux
21         fin
22         retourner Possible, DureeTotale
23     fin
```

7.3 Partie B - Programmation

B1. Écrire dans le langage de votre choix, l'algorithme de la question A3 :

```
1 G = [[-1,3,4,5,2], [3,-1,-1,-1,6], [4,-1,-1,2,-1],
      [5,-1,2,-1,5], [2,6,-1,5,-1]]
2 NomVilles = ["Nancy", "Paris", "Lyon", "Grenoble", "
      Strasbourg"]
3
4 V = int(input("Saisir un numéro de ville : "))
5 DureeMax = -1
6 Ville = -1
7 for i in range(0, 5):
8     if G[V][i] != -1 and G[V][i] > DureeMax:
9         DureeMax = G[V][i]
```

```
        Ville = i
if Ville != -1:
    print("La ville la plus éloignée de", NomVilles[V],
        "est :", NomVilles[Ville], "(", DureeMax, "h)")
else:
    print("Aucune ville n'est accessible à partir de la
        ville numéro", V)
```

B2. Écrire dans le langage de votre choix, l'algorithme de la question A4 :

```
G = [[-1,3,4,5,2], [3,-1,-1,-1,6], [4,-1,-1,2,-1],
    [5,-1,2,-1,5], [2,6,-1,5,-1]]
NomVilles = ["Nancy", "Paris", "Lyon", "Grenoble", "
    Strasbourg"]

def TestTournee(T):
    Depart = T[0]
    Possible = True
    i = 0
    Duree = 0
    while Possible and i < len(T) - 1:
        i = i + 1
        Arrivee = T[i]

        if G[Depart][Arrivee] != -1:
            Duree = Duree + G[Depart][Arrivee]
            Depart = Arrivee
        else:
            Possible = False

    # Il faut ensuite vérifier que l'on peut revenir au
        point de départ
    Possible = Possible and G[T[len(T)-1]][T[0]] != -1
    if Possible:
        Duree = Duree + G[T[len(T)-1]][T[0]]  # Ajoutez
            le dernier trajet au temps total

    return Possible, Duree

Etape = int(input("Entrer le numéro de la ville de dé
    part : "))
Tournee = []

while Etape != -1:
    Tournee.append(Etape)
    Etape = int(input("Entrer le numéro de ville étape
        ou -1 pour arrêter : "))
```

```
32
33 P, DureeTournee = TestTournee(Tournee)
34 if P:
35     print("La tournée", Tournee, "est possible et dure",
           DureeTournee)
36 else:
37     print("La tournée", Tournee, "n'est pas possible")
```