Révisions d'algorithmique

Informatique #01

Partie A – Listes

Exercice 1 — échange d'éléments
Écrire une fonction `echange(L, i, j)` qui échange les deux éléments d'indices `i` et `j` d'une liste `L`.

Exercice 2 — moyenne et variance d'une liste de valeurs
Écrire deux fonctions `moyenne(L)` et `variance(L)` qui retournent respectivement la moyenne et la variance d'une liste de flottants `L` sans utiliser la fonction `sum`.

Exercice 3 — liste croissante
Écrire une fonction `croissante(L)` qui retourne `True` si la liste d'entiers `L` est croissante, `False` sinon.

Exercice 4 — présence de doublons
Écrire une fonction `doublon(L)` qui détecte la présence d'un doublon dans une liste `L` en utilisant puis sans utiliser la méthode `count`. Déterminer la complexité temporelle des deux fonctions.

Exercice 5 — maximum d'une liste, valeur et occurence
Écrire une fonction `recherchemax(L)` qui retoune la valeur maximum d'une liste d'entiers `L`, la position de la première valeur correspondante et son nombre d'occurence.

Exercice 6 — distance maximale dans une liste
Écrire une fonction `distancemax(L)` qui retourne le plus grand écart (en valeur absolue) entre deux valeurs d'une liste d'entiers `L`.

Partie B – Arithmétique

Exercice 7 — tables de multiplications
Écrire une fonction `tablemult(n)` qui retourne la table de multiplications associée à l'entier `n`.

Exercice 8 — années bissextiles
Dans le calendrier grégorien, on appelle année bissextile une année dont le numéro est soit divisible par 4 et non divisible par 100, soit divisible par 400.

1. Écrire une fonction `estbissextile(n)` qui retourne `True` si l'année `n` est bissextile, `False` sinon.
2. Écrire une fonction `suivbissextile(n)` qui retourne la première année bissextile après l'année `n`.

Exercice 9 — paiement en euros
On souhaite payer une certaine somme en euros à l'aide (uniquement) de pièces de 2 euros, de 1 euro et de billets de 5 euros. Écrire une fonction `euros` prenant en entrée un entier `n` et renvoyant le nombre de pièces et de billets correspondant.

Exercice 10 — diviseurs et nombres parfaits

1. Écrire une fonction `diviseurs(n)` qui retourne la liste des diviseurs entiers positifs d'un entier naturel `n`.
2. Un entier naturel `n` est dit *parfait* si la somme de ses diviseurs vaut 2`n`. Écrire une fonction `parfait(n)` qui détecte si l'entier `n` est parfait ou non.
3. Écrire une fonction `listeparfait(n)` qui retourne la liste des entiers parfaits inférieurs ou égaux à `n`.
4. Retourner la liste des entiers $2^{p-1}(2^p-1)$ inférieurs à 10000 pour 2^p-1 premier. Comparer avec `listeparfait(10000)`.

Exercice 11 — nombres premiers et crible d'Eratosthène

1. Écrire une fonction `estpremier(n)` qui détecte la primalité d'un entier `n` donné.
2. En déduire une fonction `listepremier(n)` qui retourne une liste des nombres premiers compris entre 2 et `n`.
3. La méthode du crible d'Ératosthène consiste à supprimer d'une liste d'entiers les nombres composés (i.e. non premiers) en supprimant successivement les multiples de 2, de 3, etc. Écrire, sans utiliser les fonctions précédentes, une fonction `eratosthene(n)` qui retourne une liste de tous les nombres premiers compris entre 1 et `n`.

Exercice 12 — algorithme d'Euclide
Écrire une fonction `euclide(a, b)` qui retoune le pgcd de deux entiers naturels `a` et `b` à l'aide de l'algorithme d'Euclide.

Partie C – Chaînes de caractères

Exercice 13 — palindromes
Écrire une fonction `palindrome(mot)` qui détecte la présence d'un palindrome.

Exercice 14 — somme des carrés des chiffres d'un nombre

1. Écrire une fonction `decompose(n)` qui prend en entrée un entier `n` et retourne une liste constituée de ses chiffres. *On pourra passer par des chaînes de caractères et la fonction* `list`.
2. En déduire une fonction `carres(n)` qui détermine la somme des carrés des chiffres d'un entier `n` donné.

Exercice 15 — recherche de chaînes de caractères

1. Écrire une fonction `rech1(mot, phrase)` qui retourne `True` si le mot `mot` est présent dans la phrase `phrase` et précise la position de celui-ci ; `False` sinon.
   On pourra utiliser les méthodes relatives aux chaînes de caractères.
2. Écrire une fonction `position(mot, phrase, i)` qui retourne `True` si le mot est présent dans la phrase, en position `i`, SANS utiliser de méthode.
3. En déduire une fonction `rech2(mot, phrase)` renvoyant le même résultat que la fonction `rech1` SANS utiliser de méthode.
4. Pour finir, écrire enfin une fonction `rech3(mot, phrase)` qui renvoie une liste vide si le mot n'est pas contenu dans la phrase ou une liste précisant la/les position(s) du mot dans la phrase.

Exercice 16 — séquence nucléotidique
Une séquence d'ADN est une suite de 4 nucléotides désignés par les lettres : A pour l'adénine, G pour la guanine, T pour la thymine et C pour la cytosine.

1. Écrire une fonction `sequence(n)` qui retourne une séquence aléatoire d'ADN de `n` caractères.
2. Écrire une fonction `recherche(mot,phrase)` qui détecte la présence du mot `mot` dans la séquence d'ADN `phrase`.

Exercice 17 — anagrammes
Écrire une fonction `anagramme(mot1, mot2)` qui renvoie `True` si `mot1` et `mot2` sont des anagrammes, `False` sinon.
On pourra utiliser la méthode `sort` *sur des listes bien choisies.*

Partie D – Miscellaneous

Exercice 18 — calcul de $\sum_{k=a}^{b} f(k)$
Écrire une fonction `somme(f, a, b)` qui retourne la valeur de $\sum_{k=a}^{b} f(k)$.

Exercice 19 — série harmonique
Trouver n tel que $\sum_{k=1}^{n} \frac{1}{k} > 10$.

Exercice 20 — évaluation d'un polynôme et méthode de Horner (☕)
On représente un polynôme $P = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ par sa liste de coefficients :

a_0	a_1	a_2	$\cdots$	$\cdots$	a_n

1. Écrire une fonction `eval1(P, a)` qui permet pour un polynôme `P` et un réel `a` donnés, de calculer le réel `P(a)`.
2. Écrire une fonction `eval2(P, a)` qui fait de même sans utiliser l'opérateur `**`. Combien de multiplications sont-elles effectuées ?
3. Écrire une fonction `horner(P, a)` qui calcule `P(a)` en se basant sur le principe suivant :

$$P(x) = a_0 + x \times (a_1 + x \times (a_2 + x \times \cdots (a_{n-1} + a_n \cdot x)))$$

Combien de multiplications sont cette fois-ci nécessaires ?