|  |  |  |
| --- | --- | --- |
|  | **Révisions** | Touchard.png |

# Numération

Les ordinateurs qui utilisent les propriétés de l’électricité, ne connaissent que les 1 ou les 0. C’est pour cela que pour une bonne compréhension du monde des systèmes numériques, il est nécessaire de connaître certaines bases de numération (binaire, décimale et hexadécimale) et qu’il faut être capable d’effectuer des conversions entre-elles.

Le tableau suivant montre le principe de cette conversion directe.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Poids hexadécimal | 23    8 | 22    4 | 21    2 | 20    1 | 23    8 | | 22    4 | 21    2 | 20    1 | 23    8 | | 22    4 | 21    2 | 20    1 |
| Binaire | 0 | 0 | 1 | 1 | 1 | | 0 | 1 | 1 | 0 | | 1 | 1 | 0 |
| Hexadécimal | 3 | | | | | B | | | | | 6 | | | |

950(10) = 001110110110(2) = 3B6(16)

Compléter le tableau en réalisant les conversions dans les différentes bases.

|  |  |  |
| --- | --- | --- |
| Valeur décimale | Valeur binaire | Valeur hexadécimale |
| **23** | 0001 0111 | 17 |
| 55 | **0011 0111** | 37 |
| 973 | 0011 1100 1101 | **3CD** |
| **128** | 1000 0000 | 80 |
| 426 | **000110101010** | 1AA |
| 2046 | 0111 1111 1110 | **7FE** |

# Les boucles

## Table de multiplication (exemple corrigé)

Ecrire un programme qui demande un nombre de départ (opérande), et qui ensuite écrit la table de multiplication de ce nombre

Exemple pour la table de 4 :

4 \* 1 = 4

4 \* 2 = 8

4 \* 3 = 12

4 \* 4 = 16

4 \* 5 = 20

4 \* 6 = 24

4 \* 7 = 28

4 \* 8 = 32

4 \* 9 = 36

4 \* 10 = 40

|  |  |
| --- | --- |
| **Algorithme pseudo-code** | **Python** |
| afficher("votre table de multiplication") lire operande  pour n de 1 à 10 par pas de 1 faire afficher(operande,"\*",n,"=",n\*operande) finpour | operande=int(input("votre table de multiplication")) for n in range(1,11): print(operande,"\*",n,"=",n\*operande) |

## Exercices

* + 1. Écrire un programme qui affiche les 20 premiers termes de la table de multiplication par 7.
    2. Écrire un programme qui affiche les 20 premiers termes de la table de multiplication par 7, en signalant au passage (à l’aide d’une astérisque) ceux qui sont des multiples de 3.

Exemple : 7 14 21\* 28 35 42\* 49 ...

* + 1. Écrire un programme qui affiche une table de conversion de sommes d’argent exprimées en euros, en dollars US. La progression des sommes de la table sera « géométrique », comme dans l’exemple ci-dessous :

1 euro(s) = 1.12 dollar(s)

2 euro(s) = 2.24 dollar(s)

4 euro(s) = 4.49 dollar(s)

8 euro(s) = 8.97 dollar(s)

etc. (S’arrêter à 1024 euros.)

* + 1. Écrire un programme qui affiche une suite de 12 nombres dont chaque terme soit égal au triple du terme précédent.

# Les Fonctions

## Calcul de surface (exemple corrigé)

Ecrire une fonction calculSurface qui retourne le calcul de la surface d’un rectangle (largeur et longueur (chiffres décimaux) seront passés en paramètres.

|  |  |
| --- | --- |
| **Algorithme Pseudo-code** | **Programme en python** |
| Fonction calculSurface(largeur en décimal, longueur en décimal) retour d’une valeur en décimale  Surface ← largeur x longueur retourner surface  afficher(calculSurface(10.5,2)) | def calculSurface(largeur, longueur) :  surface = largeur \* longueur return surface  print(calculSurface(10.5,2)) |

## Exercices

* + 1. Écrire une fonction « conversion(tf) » qui convertisse en degrés Celsius une température exprimée au départ en degrés Fahrenheit. La formule de conversion est : TF=TC ×1,8+32
    2. Écrire une fonction « nomMois(n) » qui renvoie le nom du énième mois de l’année. Par exemple, l’exécution de l’instruction : print(nomMois(4)) doit donner le résultat : Avril.
    3. Convertir une note scolaire N quelconque, entrée par l’utilisateur sous forme de points (par exemple 27 sur 85), en une note standardisée suivant le code ci-après :

Note Appréciation N >= 80 A

80 > N >= 60 B

60 > N >= 50 C

50 > N >= 40 D N < 40 E

Écrire une fonction qui prends en paramètre le nombre N et qui retourne la lettre correspondante

* + 1. Demander à l’utilisateur d’entrer trois longueurs a, b, c. À l’aide de ces trois longueurs,

déterminer s’il est possible de construire un triangle. Déterminer ensuite si ce triangle est isocèle, équilatéral ou quelconque.

Écrire une fonction qui prends en paramètre les 3 longueurs a, b et c et qui retourne le nom du triangle correspondant.

## Affichage d’un triangle (exemple corrigé)

Ecrire une fonction triangle qui affiche un triangle avec des caractères étoiles (hauteur, chiffre entier est passé en paramètre.

Exemple :

|  |  |  |
| --- | --- | --- |
| Hauteur←5 | Hauteur←7 | Hauteur←9 |
|  |  |  |

|  |  |
| --- | --- |
| **Algorithme Pseudo-code** | **Programme en python** |
| fonction triangle(hauteur en entier): pour h de 1 à hauteur par pas de 1  pour c de 1 à h par pas de 1 affiche('\*' sans retour à la ligne)  fin pour  affiche un retour à la ligne fin pour  triangle(5) | def triangle(hauteur):  for h in range(1,hauteur+1): for c in range(1,h+1):  print('\*',end='') print()  triangle(5) |

Reprendre l’exercice précédent en inversant le triangle Exemple :

|  |  |  |
| --- | --- | --- |
| Hauteur←5 | Hauteur←7 | Hauteur←9 |
|  |  |  |

|  |  |
| --- | --- |
| **Algorithme Pseudo-code** | **Programme en python** |
|  |  |

# Les Listes

## Recherche d’une valeur minimale (exemple corrigé)

Ecrire une fonction qui recherche la valeur minimum dans une liste. La liste sera passée en paramètre.

|  |  |
| --- | --- |
| **Algorithme pseudo-code** | **Python** |
| Fonction rechercheMin(liste) retourne une valeur décimale min ← Liste[0]  Pour indice de 1 à longueur(Liste) -1 si liste[indice] < min alors  min ← Liste[indice]  Fin si Fin pour Retourner min  lst←[20,8,9,2,35,49]  afficher(rechercheMin(lst)) | def rechercheMin(liste): min=liste[0]  for n in range(1,len(liste)): if liste[n]<min:  min=liste[n] return min  lst=[20,8,9,2,35,49]  print(rechercheMin(lst)) |

## Exercices

* + 1. Ecrire une fonction qui calcule la moyenne des nombres contenus dans la liste. La liste sera passée en paramètre.

|  |  |
| --- | --- |
| **Algorithme pseudo-code** | **Python** |
| Fonction moyenneVersion1(liste) retourne une valeur décimale total ← 0  Pour n de 1 à longueur(Liste) -1 total ← total+Liste[indice]  Fin pour  moy←total/ longueur(Liste)  Retourner moy lst←[20,8,9,2,35,49]  afficher(moyenneVersion1(lst)) |  |

* + 1. Vous disposez d’une liste de nombres entiers quelconques, certains d’entre eux étant présents en plusieurs exemplaires. Écrivez un script qui recopie cette liste dans une autre, en omettant les doublons. La liste finale devra être triée.

# Les chaines de caractères

## Affichage d’une liste de chaines de caractères (exemple corrigé)

Écrivez une boucle for pour afficher à l'écran les éléments de la liste semaine, sur des lignes différentes, avec la première méthode (parcours des indices), puis avec la deuxième méthode (parcours direct de la liste)

semaine=[**"lundi"**,**"mardi"**,**"mercredi"**,**"jeudi"**,**"vendredi"**,**"samedi"**,**"dimanche"**]

**for** n **in** range(len(semaine)): print(semaine[n])

**for** jour **in** semaine: print(jour)

## Exercices

* + 1. Soit la liste suivante : [’Jean-Michel’, ’Marc’, ’Vanessa’, ’Anne’, ’Maximilien’, ’Alexandre-Benoît’, ’Louise’]

Écrivez un programme qui affiche chacun de ces noms avec le nombre de caractères correspondant.

* + 1. Écrivez une fonction « recherche(chaine,car) »qui détermine si une chaîne contient ou non le caractère « e ».
    2. Écrivez une fonction « compte (chaine) » qui compte le nombre d’occurrences du caractère

« e » dans une chaîne.

* + 1. Écrivez une fonction « recopie(chaine) » qui recopie une chaîne (dans une nouvelle variable), en insérant des astérisques entre les caractères.

Ainsi par exemple, « toto » devra devenir « t\*o\*t\*o »

* + 1. Écrivez une fonction « lePlusLong(chaine) » qui recherche le mot le plus long dans une phrase donnée (l’utilisateur du programme doit pouvoir entrer une phrase de son choix).

## Les dictionnaires

Les dictionnaires sont des collections similaires aux listes, mais au lieu d’utiliser des index (0, 1, 2, etc…), on utilise des clés alphanumériques ("Poires", "Pommes", "Fraises", etc…).

Les clés sont non ordonnées (c’est-à-dire qu’elles ; ne sont pas forcément rangées dans le même ordre), ainsi lors de l’affichage d’un dictionnaire l’ordre des éléments peut différer d’une exécution à l’autre.

Il faut penser à ne jamais écrire un programme qui prend en compte l’ordre des éléments d’un dictionnaire.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clés (keys) | Poires | Pommes | Fraises | Eléments (Items) |
| Valeurs (values) | 5 | 10 | 35 |

## Entrainement à la manipulation des dictionnaires (exemple corrigé)

Soit le dictionnaire suivant :

d = { 'nom':'Dupuis' , 'prenom':'Jacque', 'age':30}

|  |
| --- |
| **Corriger l'erreur dans le prénom, la bonne valeur est 'Jacques'.** |
| d['prenom'] = 'Jacques' |
| **Afficher la liste des clés du dictionnaire.** |
| print(d.keys()) |
| **Afficher la liste des valeurs du dictionnaire.** |
| print(d.values()) |
| **Afficher la liste des paires clé/valeur du dictionnaire.** |
| print(d.items()) |
| **Ecrire la phrase "Jacques Dupuis a 30 ans.".** |
| print(d['prenom'], d['nom'], 'a', d['age'], 'ans.') |

## Comptage du nombre d’occurrences

On veut réaliser une fonction ***occurrences(chaine)*** qui compte le nombre d’occurrences de chaque caractère d’une chaîne donnée, c’est à dire qu’elle compte le nombre de fois qu’apparait chaque lettre, chiffre, espace, etc… de la chaîne.

Le résultat sera sous la forme d’un dictionnaire où chaque clé sera un caractère de la chaîne et la valeur associée sera le nombre d’occurrences de ce caractère.

Exemple : occurrences('tortue') renvoie {'t':2, 'o':1, 'r':1, 'u':1, 'e':1}

A partir de l’algorithme, donner et tester le script python correspondant

|  |  |
| --- | --- |
| **Algorithme Pseudo-code** | **Python** |
| fonction occurrences(chaine) : D est un dictionnaire  Pour chaque caractère dans la chaine : Si caractère est dans les clés de D :  Alors D[caractere] ← D[caractere] +1 Sinon :  D[caractère] ← 1 FinSi  FinPour Renvoie D  Fin  Affiche(occurences('Tortue')) |  |

### Définition :

Le terme occurrence indique le nombre de répétitions d'un mot, d’une lettre ou d'une expression dans un texte.

|  |
| --- |
| Le résultat dans la console doit être : |
| {'t': 2, 'o': 1, 'r': 1, 'u': 1, 'e': 1} |

## Rendu de monnaie

On veut réaliser une fonction ***renduMonnaie(somme,pieces)*** qui détermine les pièces à rendre dans un monnayeur.

|  |  |
| --- | --- |
| Rendre la somme de 8€ | Solution |
|  | 1 billet de 5€ 1 pièce de 2€ 1 pièce de 1€ |

* 1. Traduire l’algorithme en code python

|  |  |
| --- | --- |
| **Algorithme pseudo code** | **Python** |
| Fonction renduMonnaie (somme en entier, pièces : liste des pièces du monnayeur dans l’ordre décroissant) : dictionnaire des pièces choisies  initialiser à zéro le dictionnaire choisies Pour p dans pieces  choisies[p]← 0  Tant que somme>= p  somme←somme-p choisies[p]← choisies[p]+1  fin tant que  fin pour  retourner choisies | **def** renduMonnaie(somme,pieces): choisies={}  **for** p **in** pieces: choisies[p]=0 **while** somme>=p:  somme=somme-p choisies[p]+=1  **return** choisies |

|  |  |
| --- | --- |
| **Résultat dans la console** | |
| *#pieces en centimes d'euros*  pieces=[500,200,100,50,20,10,5,2,1]  somme=780  print(**'Les pièces choisies sont'**) print(renduMonnaie(somme,pieces)) | Les pièces choisies sont  {500: 1, 200: 1, 100: 0, 50: 1, 20: 1, 10: 1, 5: 0, 2: 0, 1: 0} |

* 1. Refaire le programme de rendu de monnaie en utilisant une seule boucle pour.

|  |  |
| --- | --- |
| **Algorithme pseudo code** | **Python** |
| Fonction renduMonnaie (somme en entier, pièces : liste des pièces du monnayeur dans l’ordre décroissant) : dictionnaire des pièces choisies  initialiser à zéro le dictionnaire choisie Pour p dans pieces  nb← somme division entière par p choisies[p]← nb  somme←somme – nb \*p  fin pour  retourner choisies |  |

## Calcul d’une dimension

|  |  |
| --- | --- |
| **Énoncé** | Calculer la surface d’un panneau de bois connaissant sa longueur et sa largeur. |
| **Exemple** | Longueur : 3 mètres ; Largeur : 1.5 mètres ; Surface : 4.5 mètres carrés ; |
| **Solution** | Appeler une fonction qui effectue le produit entre la longueur et la largeur passées en paramètres. Retourner le résultat du calcul. Afficher le résultat. |
| **Entrées** | Largeur et longueur en mètre. |
| **Sortie** | Surface en mètre carré. |
| **Pré-condition** | La largeur et la longueur sont des valeurs entières ou décimales supérieures à zéro. |
| **Post-condition** | Surface = longueur x largeur. La surface est une valeur entière ou décimale positive. |

|  |  |
| --- | --- |
| **Algorithme Pseudo-code** | **Programme en python** |
| Fonction calculSurface(largeur en décimal, longueur en décimal) retour d’une  valeur en  décimale  Surface ← largeur x longueur Retourner surface  Afficher(calculSurface(10.5,2)) | def calculSurface(largeur, longueur) :  surface = largeur \* longueur return surface  print(calculSurface(10.5,2)) |

**Complexité de l’algorithme :** O(1) car il n’y a qu’un calcul à faire.

**Résultats du test :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Largeur (m)** | 10.5 | 3 | 2 |
| **Longueur (m)** | 2 | 0.2 | 8 |
| **Surface (m²)** | **21** | **0.6** | **16** |

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (surface en mètre carré).

## Rechercher une valeur dans un tableau

* 1. **Rechercher une valeur dans un tableau (version 1)**

|  |  |
| --- | --- |
| **Énoncé** | A partir d’une liste de n notes de la classe, rechercher la présence de la note 12/20. |
| **Exemple** | Liste de notes n°1 : 5,14,18,11,10,12,10,9,16 : oui la note est présente  Liste de notes n°2 : 5,14,18,11,10,11,10,9,16 : non, la note n’est pas présente |
| **Solution** | Rechercher une valeur dans un tableau de dimension n. Retourner vrai si la note est présente, faux dans le cas contraire. |
| **Entrées** | Tableau de valeurs et note à rechercher. |
| **Sortie** | Valeur booléenne (vrai ou faux) représentant la présence de la valeur à chercher. |
| **Pré-condition** | Le tableau de valeurs est compris entre 0 et 20 inclus. Le tableau possède au  moins 2 valeurs. |
| **Post-condition** | Un booléen vrai est retourné si la note est présente dans le tableau. |

|  |  |
| --- | --- |
| **Algorithme en pseudo-code** | **Programme en python** |
| Fonction rechercheNote(liste, note en décimal) valeur booléenne trouve ← faux  Pour indice de 0 longueur(Liste) -1 Si Liste[indice] = note alors  trouve ← vrai  Fin si Fin pour Retourner trouve  Afficher(rechercheNote([5,14,18,11,10,12,10,9,16],12)) | **def** rechercheNote(Liste,note): trouve=**False**  **for** indice **in** range(0,len(Liste)):  **if Liste[indice]==note: trouve=True**  **return** trouve print(rechercheNote([5,14,18,11,10,12,10,9,16],12)) |

**Complexité de l’algorithme :** O(n) car il faut parcourir le tableau de dimension n.

**Résultats du test :** Test effectué avec la liste : Liste=[5,18,10,12,10,14,18] et la note à rechercher 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Étape** | **Note à**  **chercher** | **Variable**  **indice** | **Liste[indice]** | **Variable**  **trouve** |
| **Avant de rentrer dans la boucle** | 12 |  |  | Faux |
| **Dans la boucle (1ère itération)** | 12 | 0 | 5 | Faux |
| **Dans la boucle (2ème itération)** | **12** | **1** | **18** | **Faux** |
| **Dans la boucle (3ème itération)** | **12** | **2** | **10** | **Faux** |
| **Dans la boucle (4ème itération)** | **12** | **3** | **12** | **Vrai** |
| **Dans la boucle (5ème itération)** | **12** | **4** | **10** | **Vrai** |
| **Dans la boucle (6ème itération)** | **12** | **5** | **14** | **Vrai** |
| **Dans la boucle (7ème itération)** | **12** | **6** | **18** | **Vrai** |
| **En sortie de boucle** | **12** | **6** | **18** | **Vrai** |

## Rechercher une valeur dans un tableau (version 2)

|  |  |
| --- | --- |
| **Énoncé** | A partir d’une liste de n notes de la classe, rechercher la présence de la note 12/20. |
| **Exemple** | Liste de notes n°1 : 5,14,18,11,10,12,10,9,16 : oui la note est présente  Liste de notes n°2 : 5,14,18,11,10,11,10,9,16 : non, la note n’est pas présente |
| **Solution** | Rechercher une valeur dans un tableau de dimension n. Retourner vrai si la note est présente, faux dans le cas contraire. |
| **Entrées** | Tableau de valeurs et note à rechercher. |
| **Sortie** | Valeur booléenne (vrai ou faux) représentant la présence de la valeur à chercher. |
| **Pré-condition** | Le tableau de valeurs est compris entre 0 et 20 inclus. Le tableau possède au  moins 2 valeurs. |
| **Post-condition** | Un booléen vrai est retourné si la note est présente dans le tableau. |

|  |  |
| --- | --- |
| **Algorithme en pseudo-code** | **Programme en python** |
| Fonction rechercheNote(liste, note en décimal) valeur  booléenne  trouve ← faux indice ← 0  Tant que trouve  vrai && indice < longueur(Liste)  Si Liste[indice] = note alors  trouve ← vrai  Fin si  indice ← indice+1 Fin tant que  Retourner trouve  Afficher(rechercheNote([5,14,18,11,10,12,10,9,16],12)) | **def** rechercheNote(Liste,note):  trouve=**False** indice=0  **while** trouve!=**True and** indice<len(Liste):  **if Liste[indice]==note: trouve=True**  **indice=indice+1 return** trouve  print(rechercheNote([5,14,18,**12**,10,**12**,10,9,16],12)) |

**Complexité de l’algorithme :** O(n) car il faut parcourir le tableau de dimension n.

**Résultats du test :** Test effectué avec la liste : Liste=[5,14,18,12,10,12,10,9,16] et la note à rechercher 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Étape** | **Note à**  **chercher** | **Variable**  **indice** | **Liste[indice]** | **Variable**  **trouve** |
| **Avant de rentrer dans la boucle** | 12 | 0 |  | Faux |
| **Dans la boucle (1ère itération)** | **12** | **0** | **5** | **Faux** |
| **Dans la boucle (2ème itération)** | **12** | **1** | **14** | **Faux** |
| **Dans la boucle (3ème itération)** | **12** | **2** | **18** | **Faux** |
| **Dans la boucle (4ème itération)** | **12** | **3** | **12** | **Vrai** |
| **En sortie de boucle** | **12** | **4** |  | **Vrai** |

**Intérêt de la boucle « while » par rapport à la boucle « for » :** Le parcours du tableau pour trouver la valeur peut se terminer plus rapidement en fonction de la position de la valeur dans le tableau.

## Rechercher une valeur maximale dans un tableau

|  |  |
| --- | --- |
| **Énoncé** | À partir d’une liste de n notes de la classe, rechercher la note maximale. |
| **Exemple** | Liste de notes : 5,14,18,11,10,12,10,9,16 note max : 18 |
| **Solution** | Rechercher la valeur maximale dans un tableau de dimension n, puis retourner cette valeur. |
| **Entrée** | Tableau de valeurs. |
| **Sortie** | Valeur de la note maximale. |
| **Pré-condition** | Le tableau de valeurs est compris entre 0 et 20 inclus. Le tableau possède au moins 2 valeurs. |
| **Post-condition** | Une valeur (entière ou décimale) maximale du tableau est retournée. |

|  |  |
| --- | --- |
| **Algorithme en pseudo-code** | **Programme en python** |
| Fonction maxNote(liste) valeur décimale valeur ← Liste[0]  Pour indice de 1 à longueur(Liste) -1 Si Liste[indice] > valeur alors  valeur ← Liste[indice]  Fin si Fin pour Retourner valeur  afficher(maxNote([5,14,18,12,10,12,10,9,16])) | **def** maxNote(Liste):  **valeur=Liste[0]**  **for indice in range(1,len(Liste)): if Liste[indice]>valeur:**  **valeur=Liste[indice] return valeur**  print(maxNote([5,14,18,12,10,12,10,9,16])) |

**Complexité de l’algorithme :** O(n) car il faut parcourir le tableau de dimension n.

**Résultats du test :** Test effectué avec la liste : Liste=[5,14,18,12,10,12,10,9,16]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Étape** | **Variable**  **indice** | **Liste[indice]** | **Variable**  **valeur** |
| **Avant de rentrer dans la boucle** |  | 5 | 5 |
| **Dans la boucle (1ère itération)** | 1 | 14 | 14 |
| **Dans la boucle (2ème itération)** | **2** | **18** | **18** |
| **Dans la boucle (3ème itération)** | **3** | **12** | **18** |
| **Dans la boucle (4ème itération)** | **4** | **10** | **18** |
| **Dans la boucle (5ème itération)** | **5** | **12** | **18** |
| **Dans la boucle (6ème itération)** | **6** | **10** | **18** |
| **Dans la boucle (7ème itération)** | **7** | **9** | **18** |
| **Dans la boucle (8ème itération)** | **8** | **16** | **18** |
| **En sortie de boucle** | **8** | **16** | **18** |

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (valeur maximale du tableau trouvée).

## Rechercher une valeur minimale dans un tableau

|  |  |
| --- | --- |
| **Énoncé** | À partir d’une liste de n notes de la classe, rechercher la note minimale. |
| **Exemple** | Liste de notes : 14,18,11,10,**5**,10,9,16 note min : **5** |
| **Solution** | **Rechercher la valeur minimale dans un tableau de dimension n, puis retourner**  **cette valeur.** |
| **Entrée** | **Tableau de valeurs.** |
| **Sortie** | **Valeur de la note minimale.** |
| **Pré-condition** | **Le tableau de valeurs est compris entre 0 et 20 inclus. Le tableau possède au**  **moins 2 valeurs.** |
| **Post-condition** | **Une valeur (entière ou décimale) minimale du tableau est retournée.** |

|  |  |
| --- | --- |
| **Algorithme en pseudo-code** | **Programme en python** |
| Fonction minNote(liste) valeur décimale  **valeur ← Liste[0]**  **Pour indice de 1 à longueur(Liste) -1 Si Liste[indice] < valeur alors**  **valeur ← Liste[indice]**  **Fin si Fin pour** Retourner valeur  Afficher(minNote([14,18,12,10,5,10,9,16])) | **def** minNote(Liste):  **valeur=Liste[0]**  **for indice in range(1,len(Liste)): if Liste[indice]<valeur:**  **valeur=Liste[indice] return valeur**  print(minNote([14,18,12,10,5,10,9,16])) |

**Complexité de l’algorithme :** O(n) car il faut parcourir le tableau de dimension n.

**Résultats du test** : Test effectué avec la liste : Liste=[14,18,12,10,5,10,9,16]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Étape** | **Variable**  **indice** | **Liste[indice]** | **Variable**  **valeur** |
| **Avant de rentrer dans la boucle** |  | 14 | 14 |
| **Dans la boucle (1ère itération)** | 1 | 18 | 14 |
| **Dans la boucle (2ème itération)** | **2** | **12** | **12** |
| **Dans la boucle (3ème itération)** | **3** | **10** | **10** |
| **Dans la boucle (4ème itération)** | **4** | **5** | **5** |
| **Dans la boucle (5ème itération)** | **5** | **10** | **5** |
| **Dans la boucle (6ème itération)** | **6** | **9** | **5** |
| **Dans la boucle (7ème itération)** | **7** | **16** | **5** |
| **En sortie de boucle** | **7** | **16** | **5** |

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (valeur minimale du tableau trouvée).

## Calculer la valeur moyenne d’un tableau

|  |  |
| --- | --- |
| **Énoncé** | À partir d’une liste de n notes de la classe, calculer la moyenne des notes. |
| **Exemple** | Liste de notes : 14,18,11,10,5,10,9,16 moyenne : **11.75** |
| **Solution** | **Calculer la somme des notes du tableau de dimension n, puis calculer et**  **retourner la moyenne.** |
| **Entrée** | **Tableau de valeurs.** |
| **Sortie** | **Valeur de la moyenne.** |
| **Pré-condition** | **Le tableau de valeurs est compris entre 0 et 20 inclus. Le tableau possède au**  **moins 1 valeur.** |
| **Post-condition** | **Une valeur (entière ou décimale) moyenne du tableau est retournée.** |

|  |  |
| --- | --- |
| **Algorithme** | **Programme en python** |
| Fonction moyenneNote(liste) valeur décimale  **somme ← 0**  **Pour indice de 0 à longueur(Liste) -1 somme ← somme + Liste[indice]**  **Fin pour**  **moyenne ←somme / longueur(Liste)**  Retourner moyenne Afficher(moyenneNote([14,18,12,10,5,10,9,16])) | **def** moyenneNote(Liste):  **somme=0**  **for indice in range(len(Liste)): somme=somme+Liste[indice]**  **moyenne=somme/len(Liste) return moyenne**  print(moyenneNote([14,18,12,10,5,10,9,16])) |

**Complexité de l’algorithme :** O(n) car il faut parcourir le tableau de dimension n.

**Résultats du test :** Test effectué avec la liste : Liste=[14,18,12,10,5,10,9,16]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Étape en pseudo-code** | **Variable**  **indice** | **Liste[indice]** | **Variable**  **somme** | **Variable**  **moyenne** |
| **Avant de rentrer dans la boucle** |  |  | 0 | X |
| **Dans la boucle (1ère itération)** | 0 | 14 | 14 | X |
| **Dans la boucle (2ème itération)** | **1** | **18** | **32** | X |
| **Dans la boucle (3ème itération)** | **2** | **12** | **44** | X |
| **Dans la boucle (4ème itération)** | **3** | **10** | **54** | X |
| **Dans la boucle (5ème itération)** | **4** | **5** | **59** | X |
| **Dans la boucle (6ème itération)** | **5** | **10** | **69** | X |
| **Dans la boucle (7ème itération)** | **6** | **9** | **78** | X |
| **Dans la boucle (8ème itération)** | **7** | **16** | **94** | X |
| **En sortie de boucle** | **7** | **16** | **94** | **11.75** |

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (calcul de la valeur moyenne).

## Tri par sélection

|  |  |
| --- | --- |
| **Énoncé** | Trier un tableau de valeurs dans l’ordre croissant. |
| **Exemple** | tableau non trié [47, 8, 38, 19, 20]  tableau trié [8, 19, 20, 38, 47] |
| **Solution** | **Utiliser la méthode de tri par sélection.** |
| **Entrée** | **Tableau de valeurs non triées.** |
| **Sortie** | **Tableau de valeurs triées.** |
| **Pré-condition** | **Le tableau de valeurs est de taille N. Les valeurs sont entières et positives.**  **Le tableau possède au moins 2 valeurs.** |
| **Post-condition** | **tableau[i] ≤ tableau[i+1]. Avec 0 ≤ i < N-1** |

|  |  |
| --- | --- |
| **Algorithme** | **Programme en python** |
| fonction tri\_Selection (tableau) pour i de 0 à N-2  indmini←i  pour j de i+1 à N‐1 faire  si tableau[j] < tableau[indmini]  alors  indmini←j  fin Si  fin Pour  si i≠indmini alors  temp=tableau[i] tableau[i]=tableau[indmini] tableau[indmini]=temp  finsi  finpour | *#N est la longueur du tableau*  **def** tri\_Selection(tableau)  **for** i **in** range(0, N - 1): indmini = i  **for** j **in** range(i + 1, N):  **if** (tableau[j] < tableau[indmini]): indmini = j  **if** (i != indmini): temp = tableau[i]  tableau[i] = tableau[indmini] tableau[indmini] = temp |

**Complexité de l’algorithme :** O(n²) car pour chaque indice de référence, il faut parcourir tout le reste du tableau

**Résultats du test :**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Indice dans le tableau** | | | | **0** | | **1** | | **2** | | **3** | | **4** |
| **Étape en pseudo-code** | **i** | **indmin** | **47** | | **8** | | **38** | | **19** | | **20** | |
| **Avant de rentrer dans la**  **boucle** | X | X | **47** | | **8** | | **38** | | **19** | | **20** | |
| **Dans la boucle (1ère itération)** | **0** | **1** | **8** | | **47** | | **38** | | **19** | | **20** | |
| **Dans la boucle (2ème itération)** | **1** | **3** | **8** | | **19** | | **38** | | **47** | | **20** | |
| **Dans la boucle (3ème itération)** | **2** | **4** | **8** | | **19** | | **20** | | **47** | | **38** | |
| **Dans la boucle (4ème itération)** | **3** | **4** | **8** | | **19** | | **20** | | **38** | | **47** | |

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (Tri des valeurs dans l’ordre croissant).

## 12. Recherche dichotomique

|  |  |
| --- | --- |
| **Énoncé** | Rechercher une valeur dans un tableau trié. |
| **Exemple** | Valeur 10 12 15 21 25 31 35 37 42 44 49 53 61 72 75 85  Indice 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  La valeur 35 se trouve à l'indice 6 dans le tableau |
| **Solution** | **Utiliser la méthode de recherche dichotomique.** |
| **Entrée** | **Valeur de l’élément à rechercher, tableau de valeurs triées** |
| **Sortie** | **Indice de l’élément à rechercher** |
| **Pré-condition** | **Le tableau de valeurs est de taille variable. Les valeurs sont entières, positives et triées dans l’ordre croissant. Le tableau possède au moins 1**  **valeur.** |
| **Post-condition** | **L’indice i existe tel que tableau[i] = valeur de l’élément à rechercher. Indice=-**  **1 dans le cas contraire.** |

|  |  |
| --- | --- |
| **Algorithme** | **Programme en python** |
| fonction recherche\_dichotomique( element, tableau ) valeur entière  bas ← 0  haut ← longueur(tableau)-1 trouve←faux  tant que bas <= haut et trouve=FAUX milieu ← (bas + haut) // 2  si element=tableau[milieu] alors  trouve←True  sinon  si element>tableau[milieu] alors bas ← milieu+1  sinon  haut ← milieu-1  fin si  fin si fin tant que  si trouve=vrai alors  indice ← milieu  sinon  indice ← -1  fin si  retourner indice | **def** recherche\_dichotomique( element, tableau  ):  bas = 0  haut = len(tableau)-1 trouve=**False**  **while** bas <= haut **and** trouve==**False**: milieu = (bas + haut) // 2  **if** element==tableau[milieu]: trouve=**True**  **else**:  **if** element>tableau[milieu]: bas = milieu+1  **else** :  haut = milieu-1  **if** trouve==**True**: indice = milieu  **else**:  indice = -1  **return** indice |

**Complexité de l’algorithme :** O(log n) A chaque boucle, on réduit la taille de recherche du tableau par 2

**Résultats du test :**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10  ***0*** | 12  ***1*** | 15  ***2*** | 21  ***3*** | 25  ***4*** | 31  ***5*** | 35  ***6*** | 37  ***7*** | 42  ***8*** | 44  ***9*** | 49  ***10*** | 53  ***11*** | 61  ***12*** | 72  **13** | 75  **14** | 85  **15** |

Chiffre à trouver 72

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Indice bas | Indice haut | Indice milieu | tableau[milieu] | Sens | Trouvé |
| 0 | 15 | 7 | 37 | A droite | Faux |
| 8 | 15 | 11 | 53 | A droite | Faux |
| 12 | 15 | 13 | 72 |  | Vrai |

La valeur 72 se trouve à l’indice 13 (milieu) dans le tableau

**Conclusion :** L’algorithme se termine en un temps fini et produit la sortie désirée (retourne l’indice de la valeur recherchée).