# Algorithmes et Pensée Computationnelle

*Consolidation 1*

Les exercices de cette série sont une compilation d'exercices semblables à ceux vu lors des semaines précédentes. Le but de cette séance est de consolider les connaissances acquises lors des travaux pratiques des semaines 1 à 6.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

## 1 Introduction et architecture des ordinateurs

### 1.1 Introduction/Résumé

Le but de cette section est de comprendre le fonctionnement d'un ordinateur. La série d'exercices sera axée autour de conversions en base binaire, décimale, hexadécimale, base 5 et de calcul de base en suivant le modèle Von Neumann.

### 1.2 Conversion

**Question 1:** (*10 minutes*) **Conversion**

1. Convertir le nombre $FFFFFF_{(16)}$ en base 10.
2. Convertir le nombre $4321_{(5)}$ en base 10.
3. Convertir le nombre $ABC_{(16)}$ en base 2.
4. Convertir le nombre $254_{(10)}$ en base 15.
5. Convertir le nombre $11101_{(2)}$ en base 10.

**Conseil**

N'oubliez pas qu'en Hexadécimal, A vaut 10, B vaut 11, C vaut 12, D vaut 13, E vaut 14 et F vaut 15.

**Solution**

1. $FFFFFF_{(16)} = 16777215_{(10)}$
2. $4321_{(5)} = 586_{(10)}$
3. $ABC_{(16)} = 101010111100_{(2)}$
4. $254_{(10)} = 11E_{(15)}$
5. $11101_{(2)} = 29_{(10)}$

### 1.3 Conversion et arithmétique

**Question 2:** (*5 minutes*) **Conversion, addition et soustraction :**
Effectuer les opérations suivantes :

1. $10110101_{(2)} + 00101010_{(2)} = ..._{(10)}$
2. $70_{(10)} - 10101010_{(2)} = ..._{(10)}$

**Conseil**

Convertissez dans une base commune avant d'effectuer les opérations.

**Solution**

1. $10110101_{(2)} + 00101010_{(2)} = 202_{(10)}$
3. $70_{(10)} - 10101010_{(2)} = 240_{(10)}$

## 1.4 Modèle de Von Neuman

Dans cette section, nous allons simuler une opération d'addition dans le **modèle de Van Neumann**, il va vous être demandé à chaque étape (FDES) de donner la valeur des registres.

**État d'origine :**
A l'origine, notre **Process Counter (PC)** vaut **00100001**.

Dans la mémoire, les instructions sont les suivantes :

| Adresse | Valeur |
|---|---|
| 00100001 | 00110100 |
| 00101100 | 10100110 |
| 01110001 | 111111101 |

Les registres sont les suivants :

| Registre | Valeur |
|---|---|
| 00 | 01111111 |
| 01 | 00100000 |
| 10 | 00101101 |
| 11 | 00001100 |

Les opérations disponibles pour l'unité de contrôle sont les suivantes :

| Numéro | Valeur |
|---|---|
| 00 | ADD |
| 01 | XOR |
| 10 | MOV |
| 11 | SUB |

**Question 3:** (*5 minutes*) **Fetch**
À la fin de l'opération **FETCH**, quelles sont les valeurs du **Process Counter** et de l'**Instruction Register** ?

**Conseil**

Pour rappel, l'unité de contrôle (Control Unit) commande et contrôle le fonctionnement du système. Elle est chargée du séquençage des opérations. Après chaque opération FETCH, la valeur du Program Counter est incrémentée (valeur initiale + 1).

**Solution**

PC = $00100001_{(2)} + 1 = 00100010_{(2)}$
IR = $00110100_{(2)}$

**Question 4:** (*5 minutes*) **Decode**

1. Quelle est la valeur de l'opération à exécuter ?
2. Quelle est l'adresse du registre dans lequel le résultat doit être enregistré ?
3. Quelle est la valeur du premier nombre de l'opération ?

4. Quelle est la valeur du deuxième nombre de l'opération ?

**Conseil**

Pensez à décomposer la valeur de l'**Instruction Register** pour obtenir toutes les informations demandées.
Utilisez la même convention que celle présentée dans les diapositives du cours (Architecture des ordinateurs (Semaine 2) - Diapositive 15)
Les données issues de la décomposition de l'**Instruction Register** ne sont pas des valeurs brutes, mais des références. Trouvez les tables concordantes pour y récupérer les valeurs.

**Solution**

00 : **ADD** (valeur de l'opération à exécuter)
11 : Adresse du registre dans lequel le résultat doit être enregistré
01 : $00100000_{(2)}$ (premier nombre)
00 : $01111111_{(2)}$ (deuxième nombre)

**Question 5:** (*5 minutes*) **Execute**
Quel est résultat de l'opération ?

**Conseil**

Toutes les informations permettant d'effectuer l'opération se trouvent dans les données de l'**Instruction Register**.

**Solution**

$00100000_{(2)} + 01111111_{(2)} = 10011111_{(2)}$

# 2 Logiciels système

## 2.1 Introduction/Résumé

## 2.2 Operating system

**Question 6:** (*5 minutes*) Sous Linux et MacOS, laquelle de ces commandes modifie le **filesystem** ?

1. **ls −la**
2. **sudo rm −rf ~/nano**
3. **sudo kill −9 3531**
4. **more nano.txt**
5. Aucune réponse n'est correcte.

**Solution**

La commande **sudo rm −rf ~/nano** permet de supprimer le répertoire **nano** situé dans le dossier **/Users/<Utilisateur_courant>** en mode super-utilisateur (utilisateur ayant des droits étendus sur le système).

**Conseil**

**Attention !**
Certaines commandes listées ci-dessus peuvent avoir des conséquences irréversibles.
Pour avoir une description détaillée d'une commande, vous pouvez ajouter **man** devant la commande sous Linux/MacOS ou ajouter **−h**, **−−help** ou **/?** après la commande sous Windows.

# 3 Programmation de base

## 3.1 Introduction/Résumé

## 3.2 Exercices

**Question 7:** (*Durée 5*)

1. Convertir $52_{(10)}$ en base 2 sur 8 bits.
2. Convertir $100_{(10)}$ en base 2 sur 8 bits.
3. Calculer en base 2 la soustraction de la question de l'exercice 1.2 par 1.1.
4. Déterminer au complément à deux l'opposé ( multiplication par -1 en base 10) de l'exercice 3.3.

**Conseil**

— Se référer aux techniques apprises dans la série 1 et la série 3

— Faire un tableau des puissances de 2 sur 8 bits.

**Solution**

— $52_{(10)} = 32_{(10)} + 16_{(10)} + 4_{(10)} = 00110100_{(2)}$

— $100_{(10)} = 64_{(10)} + 32_{(10)} + 4_{(10)} = 01100100_{(2)}$

— $01100100_{(2)}$ - $00110100_{(2)} = 0110000_{(2)}$

— **Complément à 1 :** not($0110000_{(2)}$) = $1001111_{(2)}$

— **Complément à 2 :** Complément à 1 + 1 = $1001111_{(2)} + 1_{(2)} = 1010000_{(2)}$

# 4 Itération et récursivité

## 4.1 Introduction/Résumé

## 4.2 Exercices

**Question 8:** (*15 min*) Itération et Récursivité

Créez une fonction itérative, puis une fonction récursive qui calculent le nombre de voyelles présentes dans un texte donné.

**Conseil**

Pour la version itérative, parcourez toute la chaine de caractère et incrémentez un compteur lorsque vous avez une voyelle.
Pour la version récursive, diminuez systématiquement la taille de votre chaine de caractère. Si l'élément actuel est une voyelle, ajoutez 1, sinon, ajoutez 0.
Aidez vous d'une liste de toutes les voyelles et de la fonction **in** en **Python** (**List.contains()** en **Java**).

Voici les templates :

**Python**

```
def nb_voyelles_itérative(T,S) :
    #TODO

def nb_voyelles_récursive(T,S) :
    #TODO
```

```
6
7  voyelles = ['a','e','i','o','u','y']
8  texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna aliquet\
9   eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit \
10  turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa. \
11  Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida in\
12   eros ac, luctus ullamcorper nisi."
13
14 print(nb_voyelles_itérative(texte,voyelles))
15 print(nb_voyelles_récursive(texte,voyelles))
```

### Java

```
1  import java.util.List;
2
3  public class Main {
4
5      public static int nb_voyelles_itérative(String S, List L){
6          //TODO
7      }
8
9      public static int nb_voyelles_récursive(String S, List L){
10         //TODO
11     }
12     public static void main(String[] args) {
13         List voyelles = List.of('a','e','i','o','u','y');
14         String texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna
             aliquet" +
15                 " eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit" +
16                 " turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa." +
17                 " Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida in"
            +
18                 "  eros ac, luctus ullamcorper nisi.";
19
20         System.out.println(nb_voyelles_itérative(texte, voyelles));
21         System.out.println(nb_voyelles_récursive(texte, voyelles));
22         }
23     }
```

## Solution

### Python :

```
1  def nb_voyelles_itérative(T,S) :
2      c = 0
3      for i in T :
4          if i in S :
5              c += 1
6          else :
7              pass
8      return c
9
10 def nb_voyelles_récursive(T,S) :
11     if len(T) == 1 :
12         if T[0] in S :
13             return 1
14         else :
15             return 0
16     else :
17         if T[0] in S :
18             return 1 + nb_voyelles_récursive(T[1:],S)
19         else :
20             return 0 + nb_voyelles_récursive(T[1:],S)
21
22 voyelles = ['a','e','i','o','u','y']
23 texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna
       aliquet\
24  eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit \
25  turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa. \
26  Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida
       in\
27  eros ac, luctus ullamcorper nisi."
28
29 print(nb_voyelles_itérative(texte,voyelles))
30 print(nb_voyelles_récursive(texte,voyelles))
```

**Solution**

**Java :**

```java
import java.util.List;

public class Main {

    public static int nb_voyelles_itérative(String S, List L){
        int c = 0;
        for(int i = 0; i<S.length(); i++){
            if(L.contains(S.charAt(i))){
                c ++;
            }
        }
        return c;
    }

    public static int nb_voyelles_récursive(String S, List L){
        if (S.length() == 1){
            if (L.contains(S.charAt(0))) {
                return 1;
            }
            else{
                return 0;
            }
        }
        else{
            if (L.contains(S.charAt(0))) {
                return 1 + nb_voyelles_récursive(S.substring(1,S.length()),L);
            }
            else{
                return 0 + nb_voyelles_récursive(S.substring(1,S.length()),L);
            }
        }
    }
    public static void main(String[] args) {
        List voyelles = List.of('a','e','i','o','u','y');
        String texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna aliquet" +
                " eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit" +
                " turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa." +
                " Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida in" +
                "  eros ac, luctus ullamcorper nisi.";

        System.out.println(nb_voyelles_itérative(texte, voyelles));
        System.out.println(nb_voyelles_récursive(texte, voyelles));
        }
    }
```

**Question 9:** (*5 min*) Lecture de code (Récursivité)

Qu'afficheront les programmes suivants ?

**Conseil**

Ces deux programmes comportent des fonctions itératives, lisez bien le code de haut en bas et lorsque la fonction fait appel à elle-même, revenez au début de la fonction et effectuez de nouveau les instructions avec les nouveaux paramètres.
Une feuille de papier pourrait vous être utile !

**Programme 1 :**

```
def recursion_1(S) :
    if len(S) == 1 :
        return S[0]
    else :
        return S[0] + recursion_1(S[1:]) + S[0]

print(recursion_1("Python"))
```

**Programme 2 :**

```
def recursion_2(L) :
    if len(L) == 1 :
        print(L[0])
    else :
        recursion_2(L[1:])
        print(L[0])
        recursion_2(L[1:])

Liste = ["J","adore","Python"]
recursion_2(Liste)
```

**Solution**

**Programme 1 :**

Python
adore
Python
J
Python
adore
Python

**Solution**

**Programme 2 :**

PythonohtyP

# 5 Algorithmes et compléxité

## 5.1 Introduction/Résumé

## 5.2 Exercices

**Question 10:** (*5 min*) Compléxité - Partie 1
Que fait le programme suivant ?
Estimez le nombre d'opérations qu'effectuera l'algorithme de la fonction **algo1** à chaque étape en fonction des paramètres qui lui seront assignés :

```
import math

# L est une liste d'entiers et x est un nombre entier
def algo1(L, x):
    n = len(L)

    for i in range(n):
        L[i] = L[i] + math.pow(x, i)
```

**Conseil**

Les opérations dans une boucle **for** sont répétées autant de fois que le nombre d'éléments sur lesquels nous itérons.

**Solution**

L'algorithme effectue 1 opération pour initialiser $n$ à **len(L)**. Ensuite, l'algorithme effectue une boucle **for** qui itère sur $n$ éléments. Dans la boucle for, l'algorithmes éxécute deux additions. Ainsi, nous avons au total : $1 + 2 \times n$ opérations.
En notation $O()$, nous avons une compléxité de : $O(2 * n + 1) = O(n)$ car les constantes sont ignorées.

**Question 11:** (*5 min*) Complexité - Partie 2
Que fait le programme suivant ?
Estimez le nombre d'opérations qu'effectuera l'algorithme de la fonction **algo2** à chaque étape en fonction des paramètres qui lui seront assignés :

```
1  # L liste de nombres entiers
2  def algo2(L):
3      n = len(L)
4
5      for i in range(n):
6          for j in range(n):
7              if i != j and L[i] == L[j]:
8                  return True
9
10     return False
```

**Conseil**

Les opérations dans une boucle **for** sont répétées autant de fois que le nombre d'éléments sur lesquels nous itérons.

**Solution**

L'algorithme vérifie si la liste contient des doublons.

1. L'algorithme effectue 1 opération pour initialiser $n$ à **len(L)**.
2. Ensuite, l'algorithme effectue 2 boucles for imbriquées qui itère chacune sur $n$ éléments.
3. Dans la deuxième boucle **for**, l'algorithme exécute deux comparaisons. Ainsi, nous avons au total : $1 + 2 \times n^2$ opérations.

En notation $O()$, nous avons une complexité de : $O(2n^2 + 1) = O(n^2)$ car les constantes sont ignorées.

**Question 12:** (*15 minutes*) **Tri fusion (Merge Sort)**

1. Ecrire une fonction "merge" qui prend deux listes triées comme argument et retourne une liste fusionnée triée.
2. Quelle est le nombre d'opérations effectuées ? Déterminer ensuite la complexité de la fonction, en posant **n = longueur de la liste fusionnée**.

Pour les tests utilisez les listes suivantes :
**l1=[3,10,12]** et **l2 = [5,7,14,15].**

**Conseil**

— Cette fonction est une des deux parties de l'algorithme de tri fusion.

— Inspirez vous de la solution de l'exercice 11 de la série 5.

— N'hésitez pas à revoir le processus montré en cours (visualisation de l'algorithme dans les diapositives 83 à 111) pour comprendre comment marche concrètement le tri fusion.

**Solution**

**Python :**

```python
def merge(partie_gauche, partie_droite):
    # créer la liste qui sera retournée à la fin
    liste_fusionnee = []

    # définir un compteur pour l'index de la liste de gauche
    compteur_gauche = 0
    # pareil pour la liste de droite
    compteur_droite = 0

    longueur_gauche = len(partie_gauche)
    longueur_droite = len(partie_droite)

    # continuer jusqu'à ce que l'un des index (ou les deux) atteigne l'une des longueurs (ou les deux)
    while compteur_gauche < longueur_gauche and compteur_droite < longueur_droite:
        # comparer les éléments actuels, ajouter le plus petit à la liste fusionnée
        # et augmenter le compteur de cette liste
        if partie_gauche[compteur_gauche] < partie_droite[compteur_droite]:
            liste_fusionnee.append(partie_gauche[compteur_gauche])
            compteur_gauche += 1
        else:
            liste_fusionnee.append(partie_droite[compteur_droite])
            compteur_droite += 1

    # s'il y a encore des éléments dans les listes, il faut les ajouter à la liste fusionnée
    liste_fusionnee += partie_gauche[compteur_gauche:longueur_gauche]
    liste_fusionnee += partie_droite[compteur_droite:longueur_droite]

    return liste_fusionnee  # retourner la liste fusionnée



if __name__ == "__main__":
    l1 = [3, 10, 12]
    l2 = [5, 7, 14, 15]
    print(merge(l1,l2))
```

## Solution

### Java :

```java
public class question_conso1_mergesort {
    // Fusionne 2 sous-listes de arr[].
    // Première sous-liste est arr[l..m]
    // Deuxième sous-liste est arr[m+1..r]
    public static int[] merge(int[] arr1, int[] arr2) {
        // Trouver la taille des deux sous-listes à fusionner
        int n1 = arr1.length;
        int n2 = arr2.length;
        /* Créer des listes temporaires */
        int L[] = new int[n1];
        int R[] = new int[n2];
        int F[] = new int[n1 + n2];
        /*Copier les données dans les sous-listes temporaires */
        for (int i = 0; i < n1; ++i) {
            L[i] = arr1[i];
        }
        for (int j = 0; j < n2; ++j) {
            R[j] = arr2[j];
        }
        /* Fusionner les sous-listes temporaires */
        // Indexes initiaux de la première et seconde sous-liste
        int i = 0, j = 0;
        // Index initial de la sous-liste fusionnée
        int k = 0;
        // Boucle qui fusionne L et R de manière ordonnée
        while (i < n1 && j < n2) {

            if (L[i] <= R[j]) {
                F[k] = L[i];
                i++;
                ++k;
            } else {
                F[k] = R[j];
                j++;
                ++k;
            }
        }
        // On rajoute les valeurs qui ne sont pas encore ajoutées dans F
        while (i < n1 || j < n2) {
            if (i < n1) {
                F[k] = L[i];
                ++i;
                ++k;
            } else {
                F[k] = R[j];
                ++j;
                ++k;
            }
        }

        return F;
    }
    public static void affiche_liste(int l[]) {
        int n = l.length;
        for (int i = 0; i < n; ++i)
            System.out.print(l[i] + " ");
    }
    public static void main(String[] args) {
        int[] l1 = {3, 10, 12};
        int[] l2 = {5, 7, 14, 15};
        int[] l = merge(l1, l2);
        affiche_liste(l);

    }
}
```

**Solution**

— **En python il y a :** n = 3+ 4 = 7 itérations. Les opérations avant et après la boucle sont des opérations simples, donc la complexité "pire cas" est calculé grâce à la boucle **while** et donc en fonction de la taille de la liste finale.

— **En java il y a :** n = 3 + 2 itérations effectuées dans la première boucle while.
Dans la deuxième boucle while il y a uniquement m = 2 itérations effectuées. Les boucles n'étant pas imbriquées les complexités s'additionnent.

— Donc la complexité est $O(n+m)$ car cela va dépendre de la grandeur des listes rentrées en argument.

# 6 Algorithmes de recherche

## 6.1 Introduction/Résumé

## 6.2 Exercices

**Question 13:** (*20min*) Recherche binaire

Dans cet exercice, vous devrez retrouver l'élément d'une liste d'entiers triés qui est le plus proche d'un élément **e** donné. Pour ce faire, vous devrez utiliser une version récursive de l'algorithme de recherche binaire.

Vous pouvez faire cet exercice aussi bien en Java qu'en Python.
Exemple (python) :

```
L = [1, 11, 22, 33, 44, 55, 66, 77, 88, 99]
e = 73

print(plus proche(L, e, recherche binaire recursive(L, 0, len(L)−1, e)))
```

Index retourné par la première fonction : 6
Résultat final : 77

**Conseil**

Dans cet exercice, vous devrez déclarer deux fonctions, et les combiner afin de retrouver l'élément de la liste qui est le plus proche de **e**.

La première fonction sera la fonction de recherche binaire récursive qui prendra en paramètre la liste, l'index du premier élément de la liste, l'index du dernier élément de la liste et **e**. Cette fonction retournera l'index de l'un des éléments le plus proche de **e**. La deuxième fonction effectuera les comparaisons de différences entre e et les éléments se situant autour de l'élément correspondant à l'index retourné par la première fonction. Elle pourra ainsi déterminer lequel est le plus proche de e. Elle prendra en paramètre notre liste, **e**, et la valeur retournée par la première fonction.

Voici les templates :

### Python

```
1  def recherche_binaire_recursive(L, s, r, e):
2      #TODO
3
4  def plus_proche(L,e,v):
```

```
    #TODO

L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
s = 0
r = len(L)-1
e = 68
print(plus_proche(L, e, recherche_binaire_recursive(L, s, r, e)))
```

## Java

```
import java.util.List;

public class Main {

    public static int recherche_binaire_recursive(List L,int s,int r,int e){
        //TODO
    }

    public static int plus_proche(List L,int e,int v){
        //TODO
    }

    public static void main(String[] args) {
        List L = List.of(1,2,5,8,12,16,24,56,58,63);
        int s = 0;
        int r = L.size()-1;
        int e = 64;
        System.out.println(plus_proche(L,e,recherche_binaire_recursive(L,s,r,e)));
        }
    }
```

## Solution

### Python :

```
1  def recherche_binaire_recursive(L, s, r, e): #fonction 1, recherche binaire
2      if s<=r :
3          mid = int((s + r) / 2)
4          if L[mid] == e:
5              return mid
6          elif L[mid] > e:
7              return recherche_binaire_recursive(L, s, mid − 1, e)
8          else:
9              return recherche_binaire_recursive(L, mid + 1, r, e)
10     else :
11         mid = int((s + r) / 2)
12         return mid
13
14 def plus_proche(L,e,v): #fonction 2, comparaison pour trouver l'élément le plus proche
15
16     if v == len(L)−1 : # si l'élément retourné par la première fonction est le dernier élément de la liste,
17         if abs(L[v − 1] − e) < abs(L[v] − e):  # il n'a pas besoin de comparer avec l'élément suivant
18             return L[v − 1]
19         else :
20             return L[v]
21
22     elif v == 0 : # si l'élément retourné par la première fonction est le premier élément de la liste,
23         if abs(L[v + 1] − e) < abs(L[v] − e):  # il n'a pas besoin de comparer avec l'élément d'avant
24             return L[v + 1]
25         else :
26             return L[v]
27
28     else : # cas normal
29         if abs(L[v − 1] − e) < abs(L[v] − e):
30             return L[v − 1]
31         elif abs(L[v + 1] − e) < abs(L[v] − e):
32             return L[v + 1]
33         else:
34             return L[v]
35
36 L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
37 s = 0
38 r = len(L)−1
39 e = 68
40 print(recherche_binaire_recursive(L, s, r, e))
41 print(plus_proche(L, e, recherche_binaire_recursive(L, s, r, e)))
```

## Solution

### Java :

```java
import java.util.List;

public class Main {

    public static int recherche_binaire_recursive(List L,int s,int r,int e){  // fonction 1, recherche binaire
        if (s<=r){
            int mid = (s+r)/2;
            int element = (int) L.get(mid);
            if (element == e){
                return mid;
            }
            if(element > e){
                return recherche_binaire_recursive(L,s,mid-1,e);
            }
            else {
                return recherche_binaire_recursive(L,mid+1,r,e);
            }
        }
        else{
            int mid = (s+r)/2;
            return mid;
        }
    }

    public static int plus_proche(List L,int e,int v){  //fonction 2, comparaison pour trouver l élément le plus proche

        if (v==L.size()-1){  // si l élément retourné par la première fonction est le dernier élément de la liste,
            if (Math.abs((int) L.get(v-1)-e) < Math.abs((int) L.get(v)-e)){
                // Pas besoin de comparer avec l élément suivant
                return (int) L.get(v-1);
            }
            else {
                return (int) L.get(v);
            }
        }

        else if (v==0){ // si l élément retourné par la première fonction est le premier élément de la liste,
            if (Math.abs((int) L.get(v+1)-e) < Math.abs((int) L.get(v)-e)){
                // Pas besoin de comparer avec l élément d avant
                return (int) L.get(v+1);
            }
            else {
                return (int) L.get(v);
            }
        }

        else{ // cas normal
            if (Math.abs((int) L.get(v-1)-e) < Math.abs((int) L.get(v)-e)){
                return (int) L.get(v-1);
            }
            else if (Math.abs((int) L.get(v+1)-e) < Math.abs((int) L.get(v)-e)){
                return (int) L.get(v+1);
            }
            else {
                return (int) L.get(v);
            }
        }
    }
    public static void main(String[] args) {
        List L = List.of(1,2,5,8,12,16,24,56,58,63);
        int s = 0;
        int r = L.size()-1;
        int e = 64;
        System.out.println(plus_proche(L,e,recherche_binaire_recursive(L,s,r,e)));
        }
    }
```

# 7 Quiz général

## 7.1 Python

**Question 14:** (*2 minutes*)
En python, 'Hello' équivaut à "Hello".

A - Vrai

B - Faux

**Solution**

**Vrai** : En python, les doubles guillemets et les guillemets sont équivalents.

**Question 15:** (*2 minutes*)
Dans une fonction, nous pouvons utiliser les instructions `print()` ou `return`, elles ont le même rôle.

A - Vrai

B - Faux

**Solution**

**Faux**
`print()` permet uniquement d'afficher un message dans la console. Autrement dit, `print()` sert à communiquer un message à l'utilisateur final du programme, celui-ci n'ayant pas accès au code.

`return` est une déclaration qui s'utilise à l'intérieur d'une fonction pour renvoyer le résultat de la fonction lorsqu'elle a été appellée. Exemple : la fonction `len(L)` renvoie la longeur de la liste L.

Admettons que nous ayons une fonction qui renvoie le double d'un nombre. Dans notre programme, nous souhaitons ajouter le résultat de cette fonction à un nombre quelconque. En Python, notre programme ressemblera à ça :

```
1  def double_nombre(nombre):
2      return nombre ** 2
3
4
5  nombre1 = 2
6  resultat = nombre1 + double_nombre(2)
7  print(resultat)
```

Notre programme renverra une erreur de type `TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'` si on remplace la ligne 2 par `print(nombre**2)` car la fonction `double_nombre` renverra `None`.

**Question 16:** (*2 minutes*)
Lorsqu'on fait appel à une fonction, les arguments doivent nécessairement avoir le(s) même(s) noms tel(s) que définit dans la fonction. Exemple :

```
1  def recherche_lineaire(Liste, x):
2      for i in Liste:
3          if i == x:
4              return x in Liste
5      return -1
6
7  Liste = [1,3,5,7,9]
8  x = 3
9
10 recherche_lineaire(Liste,x)
```

A - Vrai

B - Faux

**Solution**

**Faux**
Le nom des variables données en argument n'a aucune importance tant que le type de variable est respecté. Dans notre exemple, la fonction s'attend à recevoir une **liste** comme premier argument et un **entier** comme deuxième argument. Ici, nous aurions pu nommer la liste **nbr_impair** et x **valeur** et ainsi appelé la fonction **recherche_lineaire(nbr_impair, valeur)**

**Question 17:** (*2 minutes*)
Si le programme Python contient une erreur, celle-ci sera detectée avant l'exécution du programme.

A - Vrai

B - Faux

**Solution**

**Faux** : En Python, les erreurs sont détectées pendant l'exécution du programme.

**Question 18:** (*2 minutes*)
Il est possible de faire appel à une fonction définie "plus bas" dans le code sans que cela ne pose problème.

```
1  import math
2
3  nombre_decimal_pi(4)
4
5  def nombre_decimal_pi(int):
6      return round(math.pi,int)
```

A - Vrai

B - Faux

**Solution**

**Faux** : À l'exception des fonctions intégrées (il s'agit des fonctions déjà intégrées au langage Python telles que **print(), len(), abs(), etc...**). Dans les langages suivant une logique de programmation impérative (c'est le cas de Java et Python), une fonction doit nécessairement être définie **avant** d'être appelée. https://fr.wikipedia.org/wiki/Programmation_impérative

**Question 19:** (*5 minutes*)
Les trois fonctions suivantes renvoient-elles systématiquement des résultats identiques ?
Les fonctions sont censées retourner le nombre **pi** avec le nombre de décimales (au moins une et au maximum 15) indiqué en paramètre.

```
1  import math
2
3  def nombre_decimal_pi(int):
4      if int > 15:
5          int = 15
6      elif int < 0:
7          int = 1
8      resultat = round(math.pi,int)
9      return resultat
10
11 print(nombre_decimal_pi(-2))
12 print(nombre_decimal_pi(4))
13 print(nombre_decimal_pi(20))
```

```
1  import math
2
3  def nombre_decimal_pi(int):
4      if int > 15:
5          resultat = round(math.pi,15)
6      elif int < 0:
7          resultat = round(math.pi,1)
8      else:
9          resultat = round(math.pi,int)
10     return resultat
11
12 print(nombre_decimal_pi(-2))
13 print(nombre_decimal_pi(4))
14 print(nombre_decimal_pi(20))
```

```
1  import math
2
3  def nombre_decimal_pi(int):
4      if int > 15:
5          return round(math.pi,15)
6      elif int < 0:
7          return round(math.pi,1)
8      else:
9          return round(math.pi,int)
10
11
12 print(nombre_decimal_pi(-2))
13 print(nombre_decimal_pi(4))
14 print(nombre_decimal_pi(20))
```

A - Vrai

B - Faux

**Solution**

**Vrai** : Les trois fonctions produisent des résultats identiques. Si besoin, exécutez le code dans IntelliJ.

## 7.2 Java

**Question 20:** (*3 minutes*)
Observez les deux programmes suivants en Java. Lequel a-t-il une bonne structure et peut être compilé sans erreur ?

```java
//Programme A
public class Main {

    public static void main(String[] args) {
        ma_function();
        autre_fonction();


        static void ma_function(){
            System.out.println("Voici ma fonction!");
        }

        static void autre_fonction(){
            System.out.println("Une autre fonction!");
        }
    }
}
```

```java
//Programme B
public class Main {

    public static void main(String[] args) {
    ma_function();
    autre_fonction();
    }

    static void ma_function(){
        System.out.println("Voici ma fonction!");
    }

    static void autre_fonction(){
        System.out.println("Une autre fonction!");
    }
}
```

A (à gauche)

B (à droite)

**Solution**

**Le programme B**
Le fichier dans son ensemble représente une classe Java, ici la classe s'appelle **Main** (Vous aurez plus d'informations sur le fonctionnement des classes lorsque nous aborderons la programmation orientée objet). À l'intérieur de cette classe se trouve la méthode **public static void main()**, il s'agit de la porte d'entrée de notre programme, celle que l'on exécute et celle dans laquelle nous rédigeons notre code.
Les autres fonctions, qui peuvent être appelées, se définissent au sein de la classe au même échelon que la fonction **public static void main()** comme dans le programme B ci-dessus.

**Question 21:** (*2 minutes*) Exercice 1
L'indentation des lignes de code en Java est aussi importante qu'en python.

A - Vrai

B - Faux

**Solution**

**Faux**
En Java, le compilateur ne prend pas en compte l'indentation pour interpréter le programme, il comprend la structure à l'aide des parenthèses, des accolades et encore des point-virgules qui indiquent la fin d'une instruction. Toutefois, l'indentation est un aspect important de la programmation car elle sert à bien structurer visuellement votre code.

En Python, l'indentation définit la structure de votre code. Elle est donc indispensable pour la bonne interprétation et la bonne exécution de votre programme.