# Algorithmes et Pensée Computationnelle

*Consolidation 1*

Les exercices de cette série sont une compilation d'exercices semblables à ceux vus 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 **Code**.

## 1 Introduction et architecture des ordinateurs

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.

### 1.1 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.

### 1.2 Conversion et arithmétique

**Question 2:** (*5 minutes*) **Conversion, addition et soustraction :**
Effectuez 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.

### 1.3 Modèle de Von Neumann

Dans cette section, nous allons simuler une opération d'addition dans le **modèle de Von 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).

**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.

**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**.

## 2 Logiciels système

**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.

> **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

**Question 7:** (🕒 *10 minutes*)

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 $01100100_{(2)}$ par $00110100_{(2)}$.
4. Déterminer au complément à deux l'opposé ( multiplication par -1 en base 10) de $0110000_{(2)}$.

> **Conseil**
>
> — Se référer aux techniques apprises dans la série 1 et la série 3
> — Faire un tableau des puissances de 2 sur 8 bits.

## 4 Itération et récursivité

**Question 8:** (🕒 *15 minutes*) 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 chaîne de caractère et incrémentez un compteur lorsque vous avez une voyelle.
> Pour la version récursive, diminuez systématiquement la taille de votre chaîne 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**

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

def nb_voyelles_récursive(T,S) :
    #TODO

voyelles = ['a','e','i','o','u','y']
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."

print(nb_voyelles_itérative(texte,voyelles))
print(nb_voyelles_récursive(texte,voyelles))
```

### Java

```java
import java.util.List;

public class Main {

    public static int nb_voyelles_itérative(String S, List L){
        //TODO
    }

    public static int nb_voyelles_récursive(String S, List L){
        //TODO
    }
    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 minutes*) 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 :

```python
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)
```

# 5 Fonctions, mémoire et exceptions

**Question 10:** (🕓 *10 minutes*) Fonctions et Exceptions (Python)

Implémenter une fonction **check_name()** qui prend en argument le paramètre **name** et qui affiche sur la console Bonjour suivi du nom passé en paramètre, faite en sorte que l'appel échoue avec une exception de type **ValueError** si le paramètre name n'est pas une chaine de caractère ou si c'est une chaine vide.

> **Conseil**
>
> Pour trouver le type d'une variable utilisez la methode **type(var)**.

**Question 11:** (🕓 *5 minutes*) Gestion d'exceptions (Java)

Quel mot clé est utilisé pour déclencher explicitement une exception ?

1. catch
2. throw
3. raise
4. try

**Question 12:** (🕓 *5 minutes*) Gestion d'exceptions (Java)

Qu'affiche le programme suivant ?

```
public class Question12{
  public static void main(String args[]) {
    try {
      int i  = 7 / 0;
      System.out.print("OUTPUT 1");
    }catch(ArithmeticException e)
    {
      System.out.print("OUTPUT 2");
    }
  }
}
```

# 6 Programmation orientée objet - classes et héritage

**Question 13:** (🕓 *15 minutes*) **Poker Game**
Cet exercice vous demande de construire un jeu de poker en implémentant un programme Java. Celui-ci comprend trois parties importantes : un paquet de cartes (un "**deck**"), un joueur, et une carte, et elles devraient être représentées par trois classes respectives.
Suivez les instructions ci-dessous pour écrire le programme.
**La classe Carte**

- Attributs : **valeur** (**int**), **couleur** (**int**). Attention, la couleur d'une carte est représentée ici par une valeur de 0 à 3 au lieu d'une chaîne de caractères.
- Implémentez un constructeur qui prend en argument une valeur et une couleur.
- Implémentez un getter **getInfo()** qui affiche dans la console la valeur et la couleur de la carte, vous pouvez vous aider de fonctions privées **getCouleur()** et **getValeur()** pour afficher les cas particuliers.

```
public class Carte {

    private int valeur;
    private int couleur;

    public static final int HEART = 0;
    public static final int DIAMOND = 1;
    public static final int SPADE = 2;
    public static final int CLUB = 3;


    public Carte(int valeur, int couleur){
        ...
        ...
    }

    public void getInfo(){
        ...
    }

    public String getValeur() {
        ...
    }

    public String getCouleur() {
        ...
    }
}
```

### La classe Joueur

- Attributs : **mainDeCartes** (**Carte[]**) (un tableau de carte de taille maximal 2), **balance** (**int**) (le solde total du joueur), **mise** (**int**) (la mise du joueur), **monTour** (**boolean**) (valant true si c'est au tour du joueur).
- Implémentez le constructeur qui prend en argument **deux cartes** (puis les ajoute à sa main), une balance initiale, une mise initiale de 0. Vous pouvez initialiser **monTour** comme **false** au début.
- Implémentez une fonction **miser()** qui propose une mise sur la table mais ne retourne rien (utiliser le mot **void**). Définir la nouvelle mise du joueur.
- Implémentez la méthode **montrerMain()** qui montre les cartes sur la table.
- Implémentez une méthode **gagner()** qui prend en argument la somme des gains sur la table que le joueur vient de gagner, ajoutez là à son solde.
- **Attention :** **miser()** et **gagner()** s'appliquent seulement si c'est au tour du joueur.

```
import java.util.ArrayList;

public class Joueur {

    private ...
    private ...
    private ...
    private ...


    public Joueur(...){
        ...
    }


    public void montrerMain(){
        ...
    }

    public void miser(int valeur){
        ...
    }

    public void gagner(int valeur){
```

```
    ...
  }

}
```

**La classe Paquet** :

- Attributs : **paquet (ArrayList<Carte>)** (un jeu de cartes complet), **NBR_CARTES (int)** (un attribut final et static (constante) ayant pour valeur **52**), **NBR_MELANGEs (int)** (une constante qui correspond au nombre de mélanges effectués de valeur **100**).
- Dans le constructeur ne prenant aucun argument, générer le deck en créant au fur et à mesure des cartes.
- Implémenter une fonction public **melanger()** qui mélange le jeu de carte et appeler là dans le constructeur après avoir construit le jeu de carte.
- Implémenter une fonction getter **getCarte()** qui retourne la carte placée en haut de la pile et la retire du jeu.

**Conseil**

Utilisez **Random r = new Random();** sa fonction **nextInt()** et utilisez des index et une variable **Carte** temporaire pour pouvoir échanger les positions des cartes.

```
import java.util.ArrayList;
import java.util.Random;

public class Paquet {

  private ...
  ...
  ...

  public Paquet(){
    ...
  }


  public void melanger(){
    ...

  }


  public Carte getCarte(){
    ...
  }
}
```

**Question 14:** (*15 minutes*) **Un jeu de rôle avec des personnages**

Dans cet exercice, vous allez mettre en place un simple jeu de rôle. Le concept d'héritage sera très utile dans notre jeu de rôle étant donné que les différentes classes de personnages possèdent certains attributs ou actions similaires. Les personnages de notre jeu sont le Guerrier, le Paladin, le Magicien et le Chasseur.
Ainsi, il semble intéressant de construire une première classe **Personnage**. Un personnage est un objet qui possède plusieurs arguments :

- **nom (String)** : le nom du personnage
- **niveau (int)** : le niveau du personnage
- **pv (int)** : les points de vie du personnage
- **vitalite (int)** : la vitalité (ou santé) du personnage
- **force (int)** : la force du personnage
- **dexterite (int)** : la dextérité du personnage
- **endurance (int)** : l'endurance du personnage
- **intelligence (int)** : l'intelligence du personnage

Suivez les instructions ci-dessous pour compléter le programme.

— Implémentez le constructeur de la classe **Personnage** qui prend tous les attributs cités ci-dessus en argument.
— Il peut être intéressant d'afficher les caractéristiques de votre personnage. Après avoir implémenté les getters utiles, implémentez une méthode **getInfo()** dans la classe **Character** qui affiche des informations du personnage dans la console.
— Chaque personnage de ce jeu a un compteur de leur vie restante. Celui-ci va être manipulé par un **setter**. Définissez le **setter** dans la classe **Personnage**.
— Implémentez les classes **Guerrier, Paladin, Magicien et Chasseur** qui héritent de **Personnage** en écrivant tout d'abord leur constructeur respectif.

```
1  import java.util.Random;
2
3  public abstract class Personnage {
4
5     private ...
6     ...
7
8     public Personnage(...){
9        this.nom = ...
10       ...
11
12    }
13
14    public void getInfo() {
15       ...
16    }
17    ...
18    ...
19 }
```

— Pour chacun des personnages, implémentez une méthode **attaqueBasique()** qui prend un autre personnage en argument et ne retourne rien. Celle-ci crée une attaque de votre choix en fonction des caractéristiques des personnages (ex : l'attaque du guerrier dépendra de sa force, l'attaque du chasseur de son endurance etc..), et détermine les points de vie restants en soustrayant la gravité de l'attaque à la vitalité du personnage. Affichez le nom de celui que vous avez attaqué et ses points de vie restants.
— Les méthodes communes à toutes les sous-classes, doivent être déclarées abstraites dans la classe parente **Personnage**. Changer cette classe pour qu'elle soit maintenant abstraite avec la/les méthode(s) abstraite(s) correspondante(s).
— **Attention :** Utiliser un "setter" pour réduire les **pv** de l'autre personnage.

```
1
2  public class Guerrier extends Personnage {
3
4  }
5
6  public class Magicien extends Personnage {
7
8  }
9
10 public class Paladin extends Personnage {
11
12 }
13
14 public class Chasseur extends Personnage {
15
16 }
```

# 7 Quiz général

## 7.1 Python

**Question 15:** (🕒 *2 minutes*)
En Python, 'Hello' équivaut à "Hello".

1. - Vrai

2. - Faux

**Question 16:** (*2 minutes*)
Dans une fonction, nous pouvons utiliser les instructions `print()` ou `return`, elles ont le même rôle.

1. - Vrai
2. - Faux

**Question 17:** (*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 :

```
def recherche_lineaire(Liste, x):
    for i in Liste:
        if i == x:
            return x in Liste
    return -1

Liste = [1,3,5,7,9]
x = 3

recherche_lineaire(Liste,x)
```

1. - Vrai
2. - Faux

**Question 18:** (*2 minutes*)
Si le programme Python contient une erreur, celle-ci sera détectée avant l'exécution du programme.

1. - Vrai
2. - Faux

**Question 19:** (*2 minutes*)
Il est possible de faire appel à une fonction définie "plus bas" dans le code sans que cela ne pose problème.

```
import math

nombre_decimal_pi(4)

def nombre_decimal_pi(int):
    return round(math.pi,int)
```

1. - Vrai
2. - Faux

**Question 20:** (*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.

```
import math

def nombre_decimal_pi(int):
    if int > 15:
        int = 15
    elif int < 0:
        int = 1
    resultat = round(math.pi,int)
    return resultat

print(nombre_decimal_pi(-2))
print(nombre_decimal_pi(4))
print(nombre_decimal_pi(20))
```

```
import math

def nombre_decimal_pi(int):
    if int > 15:
        resultat = round(math.pi,15)
    elif int < 0:
        resultat = round(math.pi,1)
    else:
        resultat = round(math.pi,int)
    return resultat

print(nombre_decimal_pi(-2))
```

```
print(nombre_decimal_pi(4))
print(nombre_decimal_pi(20))
```

```
import math

def nombre_decimal_pi(int):
    if int > 15:
        return round(math.pi,15)
    elif int < 0:
        return round(math.pi,1)
    else:
        return round(math.pi,int)

print(nombre_decimal_pi(-2))
print(nombre_decimal_pi(4))
print(nombre_decimal_pi(20))
```

1. - Vrai
2. - Faux

## 7.2 Java

**Question 21:** (*3 minutes*)
Observez les deux programmes suivants en Java. Lequel a-t-il une bonne structure et peut être compilé sans erreur ?

```
//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!");
        }
    }
}
```

```
//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!");
    }
}
```

1. Programme A
2. Programme B

**Question 22:** (*2 minutes*) Exercice 1
L'indentation des lignes de code en Java est aussi importante qu'en Python.

1. - Vrai
2. - Faux