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**. Le temps indiqué (🕓) est à titre strictement indicatif.

> **Attention**
>
> Il n'est pas nécessaire de faire tous les exercices pendant la séance de TP. Privilégiez les exercices qui vous paraissent difficiles et posez des questions aux assistants.

1 Introduction et architecture des ordinateurs

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 $11101_{(2)}$ 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 1)
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: (🕓 *10 minutes*) En utilisant l'invite de commandes (Terminal), créer un programme Python demandant à l'utilisateur d'entrer son nom et son prénom. Lorsque le programme est exécuté, la phrase suivante doit s'afficher : *"Bonjour, je m'appelle *prénom nom*"*.

Question 7: (🕓 *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.

> ⚠ **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 8: (🕓 *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 pendant les semaines 1 et 3.
> — Faire un tableau des puissances de 2 sur 8 bits.

Question 9: (🕓 *5 minutes*) Qu'affiche le programme suivant ?

```
a = 2.0
b = 5
c = "hello world"

if type(a) == int:
    print("output 1")
if (type(a) == int or type(c) == str) and b!=2.0:
    print("output 2")
if (a*b >= 10 or type(c) == bool) and type(a) == float:
    print("output 3")
```

Question 10: (🕓 *5 minutes*) À partir du dictionnaire suivant :

```
jours_semaine = {1:"lundi", 2:"mardi", 3:"mercredi", 4:"jeudi", 5:"jeudi", 6:"samedi"}
```

Remplacez la valeur associée au nombre 5 par "vendredi". Ajoutez le couple clé-valeur **7 :** "dimanche". Affichez la valeur associée à la clé 3. À l'aide d'une écriture en compréhension, créez une liste contenant les valeurs associées aux clés qui sont des nombres pairs.

4 Fonctions, mémoire et exceptions

Question 11: (🕒 *5 minutes*) **Portée des variables** (Python)
Qu'affiche le programme suivant ?

```
x = 2

def fonction():
    x = 3

    def fonction2():
        global x
        print("x=" + str(x))

    fonction2()
    print("x=" + str(x))

print(fonction())
```

Question 12: (🕒 *10 minutes*) **Fonctions et Exceptions** (Python)
Écrivez une fonction **check_name()** qui prend en argument le paramètre **name** et qui affiche dans la console "*Bonjour suivi du nom passé en paramètre*". Levez une exception de type **ValueError** si le paramètre n'est pas de type **str** ou si c'est une chaîne de caractères vide.

> **Conseil**
>
> En Python, pour trouver le type d'une variable, on utilise la méthode **type(variable)**.

Question 13: (🕒 *5 minutes*) **Gestion d'exceptions** (Java)

En Java, quel mot-clé est utilisé pour lever une exception ?

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

Question 14: (🕒 *5 minutes*) **Gestion d'exceptions**(Java)

Qu'affiche le programme suivant ?

```
public class Question14{
  public static void main(String args[]) {
    try {
      int i  = 7 / 0;
      System.out.print("OUTPUT 1");
    }catch(ArithmeticException e)
    {
      System.out.print("OUTPUT 2");
    }
  }
}
```

5 Itération et récursivité

Question 15: (🕒 *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 caractères et incrémentez un compteur lorsque vous avez une voyelle.
Pour la version récursive, diminuez systématiquement la taille de votre chaîne de caractères. Si l'élément actuel est une voyelle, ajoutez 1, sinon, ajoutez 0.
Aidez vous d'une liste contenant toutes les voyelles et de l'instruction `in` en **Python** (**List.contains()** en **Java**).

Voici les templates que vous pouvez réutiliser :

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 16: (*5 minutes*) Lecture de code (Récursivité)

Qu'affichent 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)
```

Question 17: (*15 minutes*) **Itération et Récursivité** :
Créez un tableau en deux dimensions en utilisant les propriétés des boucles. Ce tableau doit être généré par l'utilisateur qui rentrera un nombre de lignes et de colonnes. Ensuite, faites en sorte de pouvoir sélectionner une case du tableau avec ses coordonnées en utilisant une condition.

Conseil

Utilisez l'argument **end** de la fonction **print** de Python pour ne pas faire de passage à la ligne pour chaque caractère et utilisez les caractères "-" et "|" pour générer les colonnes et les lignes.

6 Programmation orientée objet - classes et héritage

Question 18: (*15 minutes*) **Jeu de rôle**
Dans cet exercice, vous allez mettre en place un simple jeu de rôle. Le concept d'héritage sera très utile dans ce 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 serait judicieux de créer 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.

- Définir 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, redéfinissez la méthode **toString()** dans la classe **Personnage** qui retourne une chaîne de caractères avec toutes les informations du personnage. Cette méthode nous permettra d'afficher dans la console les informations du personnage lorsqu'on fait un **print** sur une instance de **Personnage**.
- Chaque personnage de ce jeu a un compteur de vies. Celui-ci va être modifié par un **setter**. Définissez le **setter** dans la classe **Personnage**.
- Créer les classes **Guerrier, Paladin, Magicien et Chasseur** qui héritent de **Personnage** en écrivant tout d'abord leurs constructeurs respectifs.

```
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.
- **Attention :** Utiliser un "setter" pour réduire les points de vie (**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 19: (*2 minutes*)
En Python, 'Hello' équivaut à "Hello".

1. - Vrai
2. - Faux

Question 20: (*2 minutes*)
Dans une fonction, nous pouvons utiliser les instructions **print()** ou **return**, elles ont le même rôle.

1. - Vrai
2. - Faux

Question 21: (🕒 *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 22: (🕒 *2 minutes*)
En Python, 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 23: (🕒 *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 24: (🕒 *3 minutes*)
Observez les deux programmes suivants en Java. Lequel a-t-il une bonne structure et peut être compilé sans erreur ?

```
1
2  //Programme A
3  public class Main {
4     public static void main(String[] args) {
5        ma_function();
6        autre_fonction();
7
8        static void ma_function(){
9           System.out.println("Voici ma fonction!");
10       }
11
12       static void autre_fonction(){
13          System.out.println("Une autre fonction!");
14       }
15    }
16 }
```

```
1  //Programme B
2  public class Main {
3
4     public static void main(String[] args) {
5     ma_function();
6     autre_fonction();
7     }
8
9     static void ma_function(){
10       System.out.println("Voici ma fonction!");
11    }
12
13    static void autre_fonction(){
14       System.out.println("Une autre fonction!");
15    }
16 }
```

1. Programme A
2. Programme B

Question 25: (*2 minutes*) Exercice 1
L'indentation des lignes de code en Java est aussi importante qu'en Python.

1. - Vrai
2. - Faux