

NSI Terminale 2022-2023

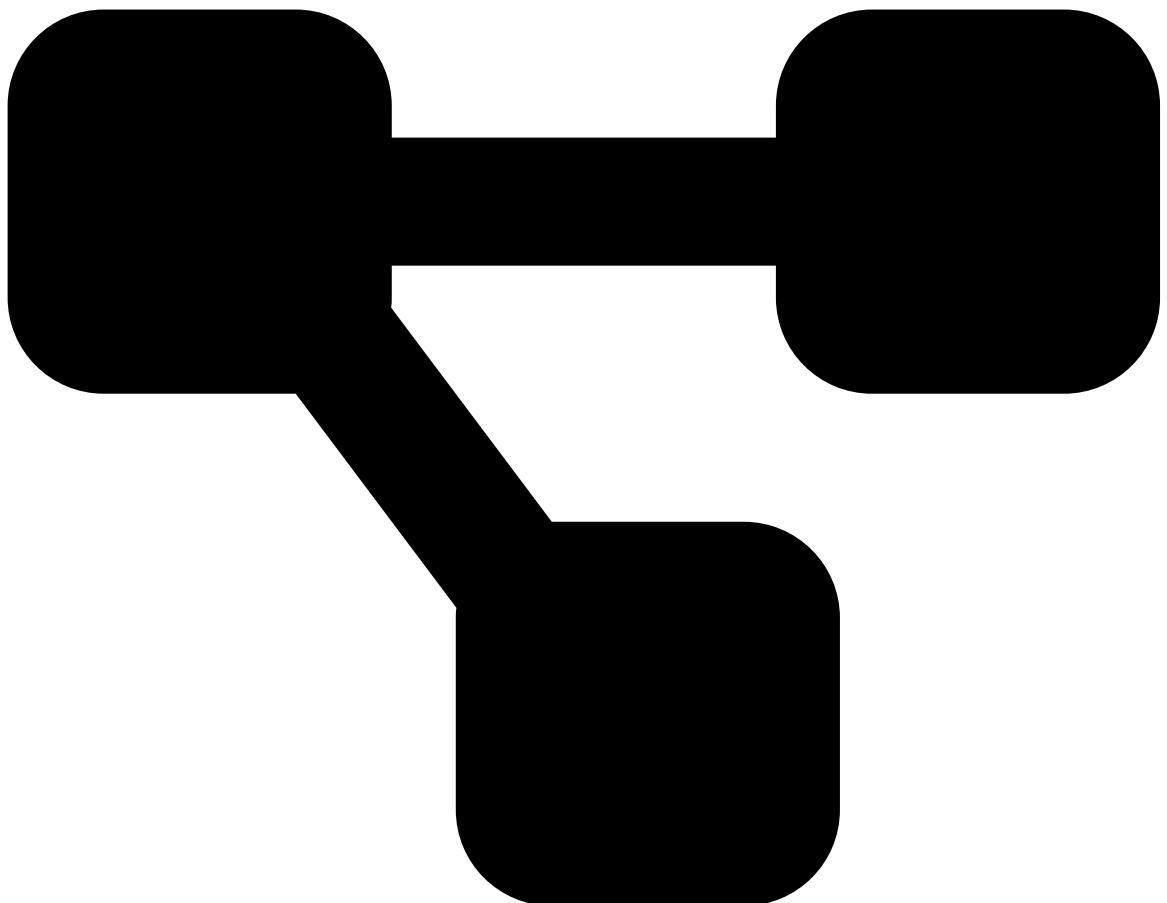
1. None

Michael Meyroneinc-condy

2023 sous licence CC BY-NC-SA 4.0

Table of contents

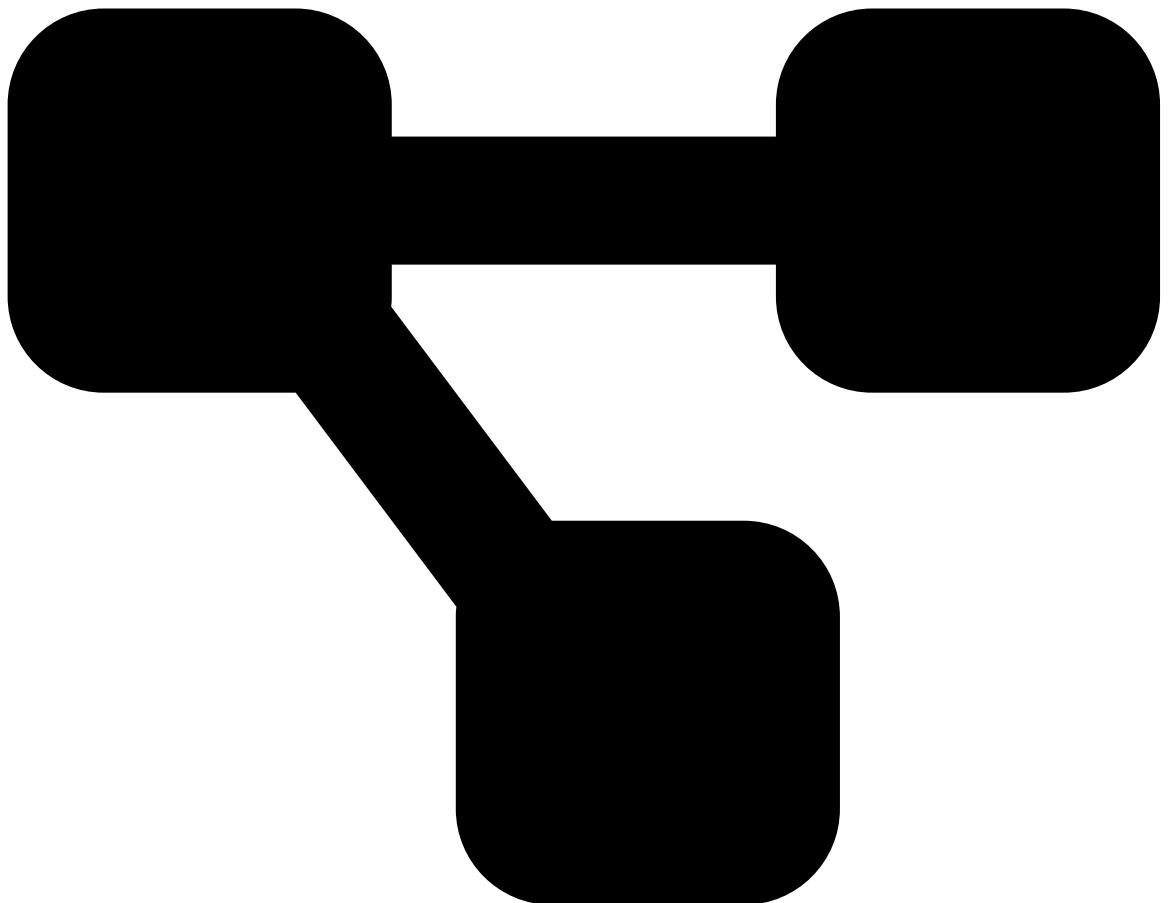
1. Accueil	10
1.1 Progression sur l'année	10
1.2 Notions à l'épreuve écrite de Mars 2023 (mis à jours 30/09/2022):	11
1.3 Comment calculer sa note au BAC 2023 et Répartition des notes :	11
2. T1 - Struct. de données	13
2.1 C4 Programmation Orientée Objet	
13	



2.2 POO : TD	20
2.3 POO : TD	28
2.4 POO : Exercices BAC - Correction	38

2.5 **C5** Listes et Piles

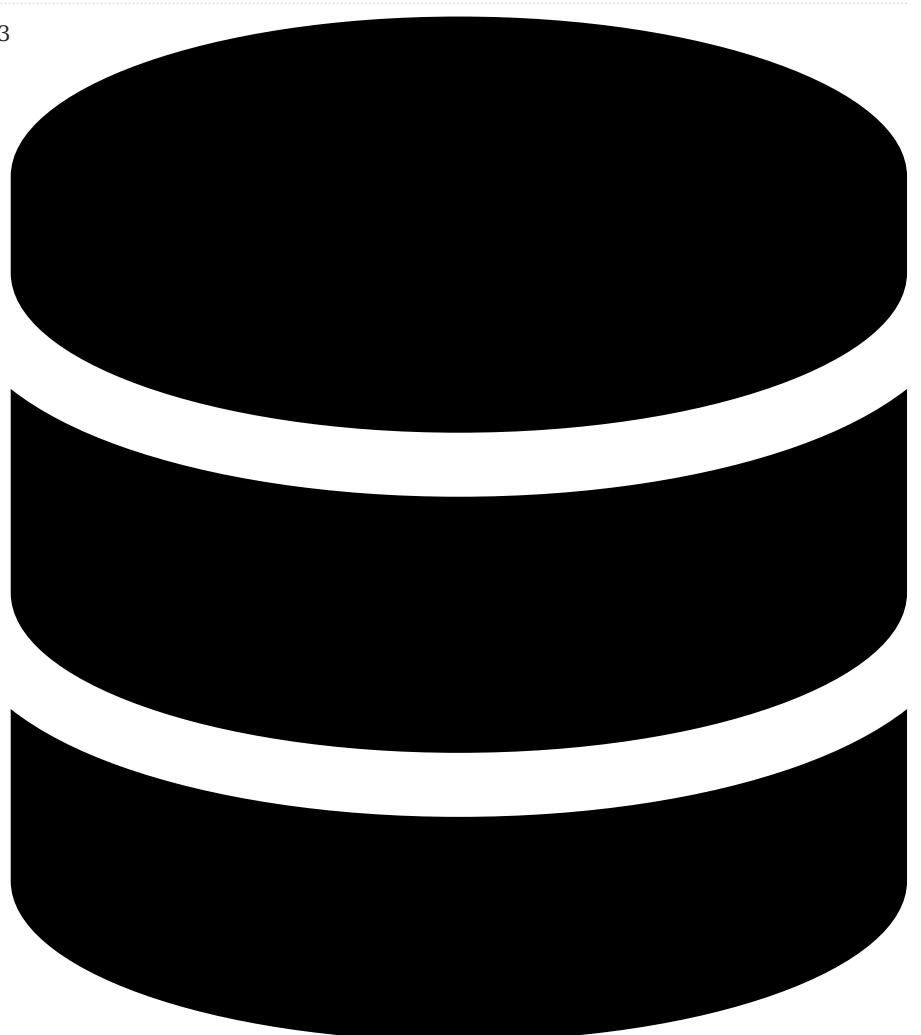
44



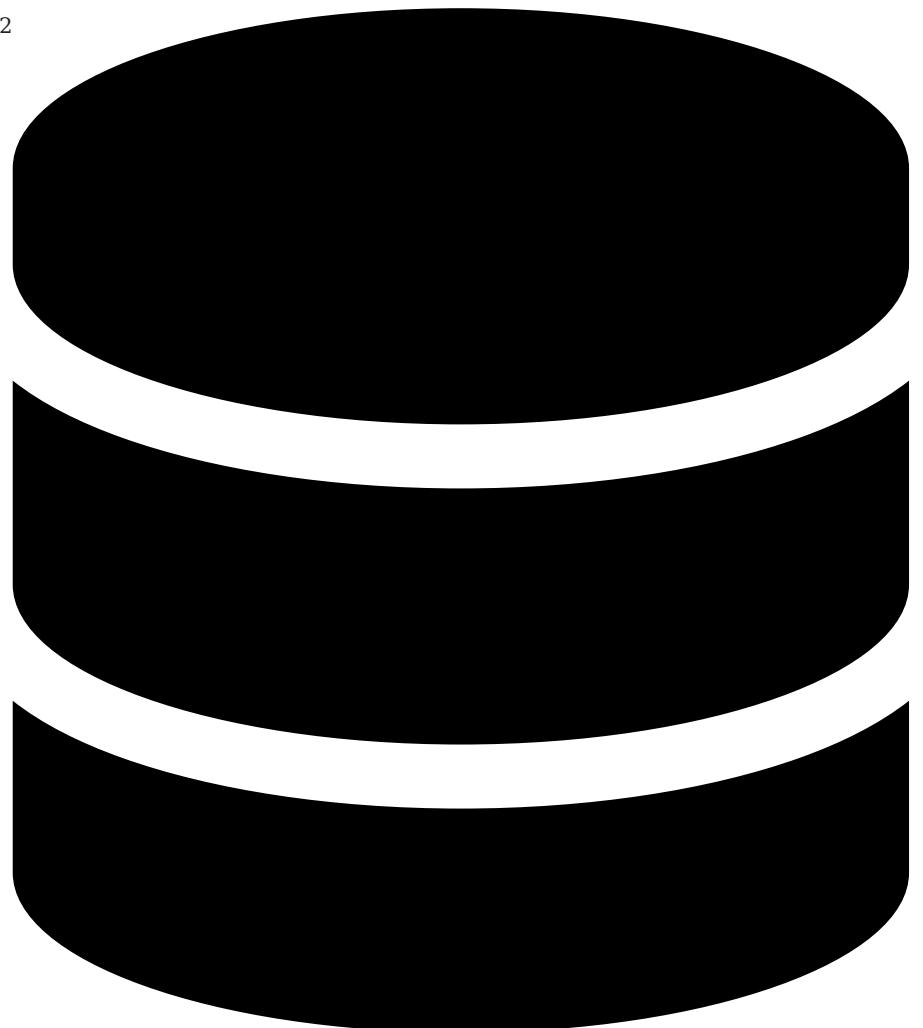
2.6 Thème 1 - Structure de données

65

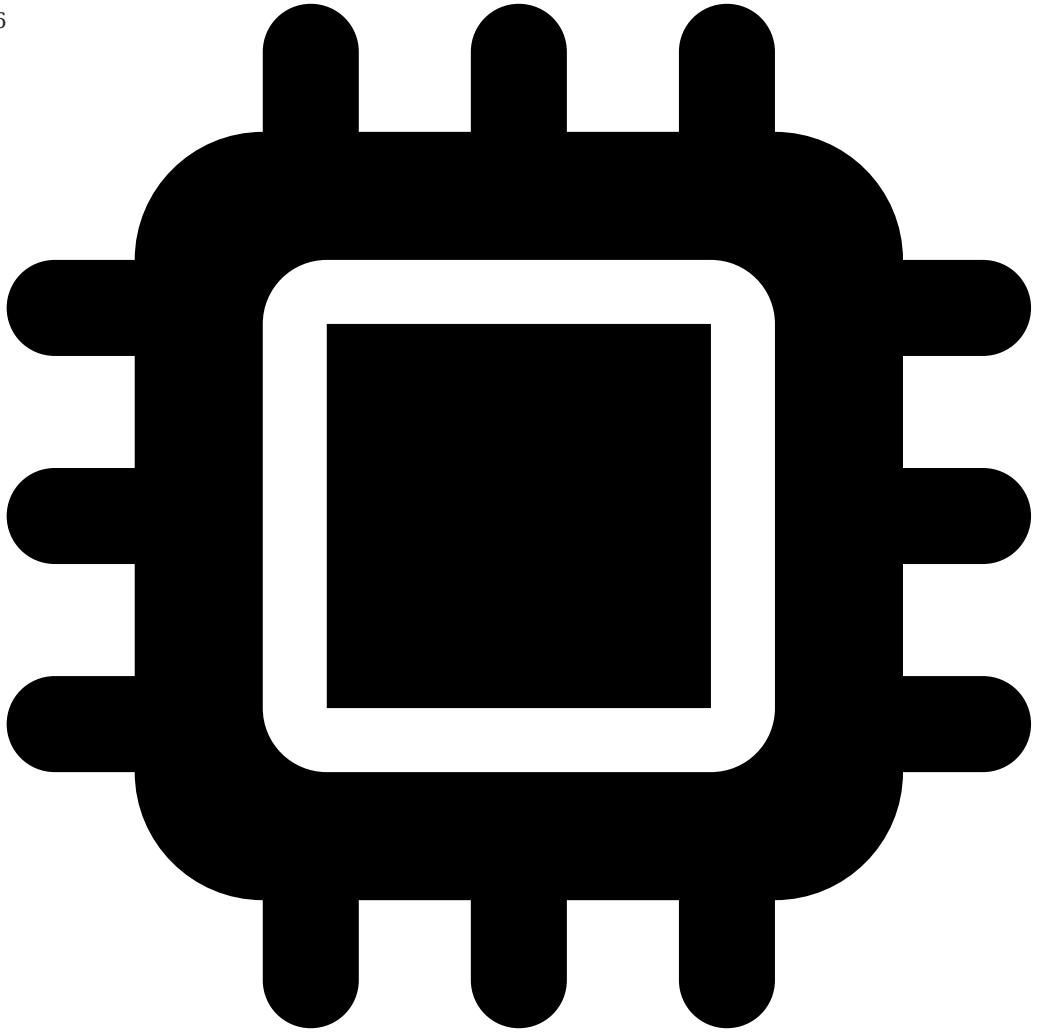
3. T2 - Bases de données	93
3.1 C1 Le Modèle relationnel	93



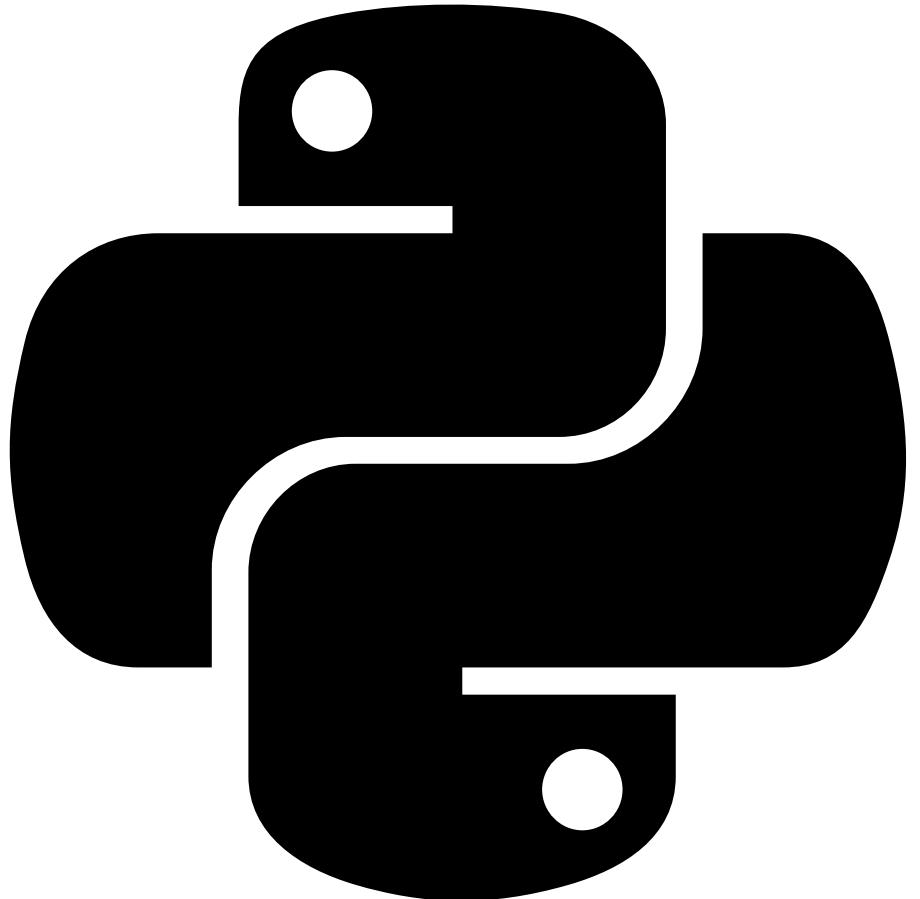
3.2 **C2** Langage SQL 102



3.3 SQL : Exercices BAC	111
3.4 SQL : Exercices BAC - Correction	116

4. T3 - Archi. matérielle	126
4.1 C6 Protocole de routage	
126	
	
4.2 Thème 3 - Architecture matérielle	139
4.3 Exercices BAC	139
4.4 Thème 3 - Architecture matérielle	155

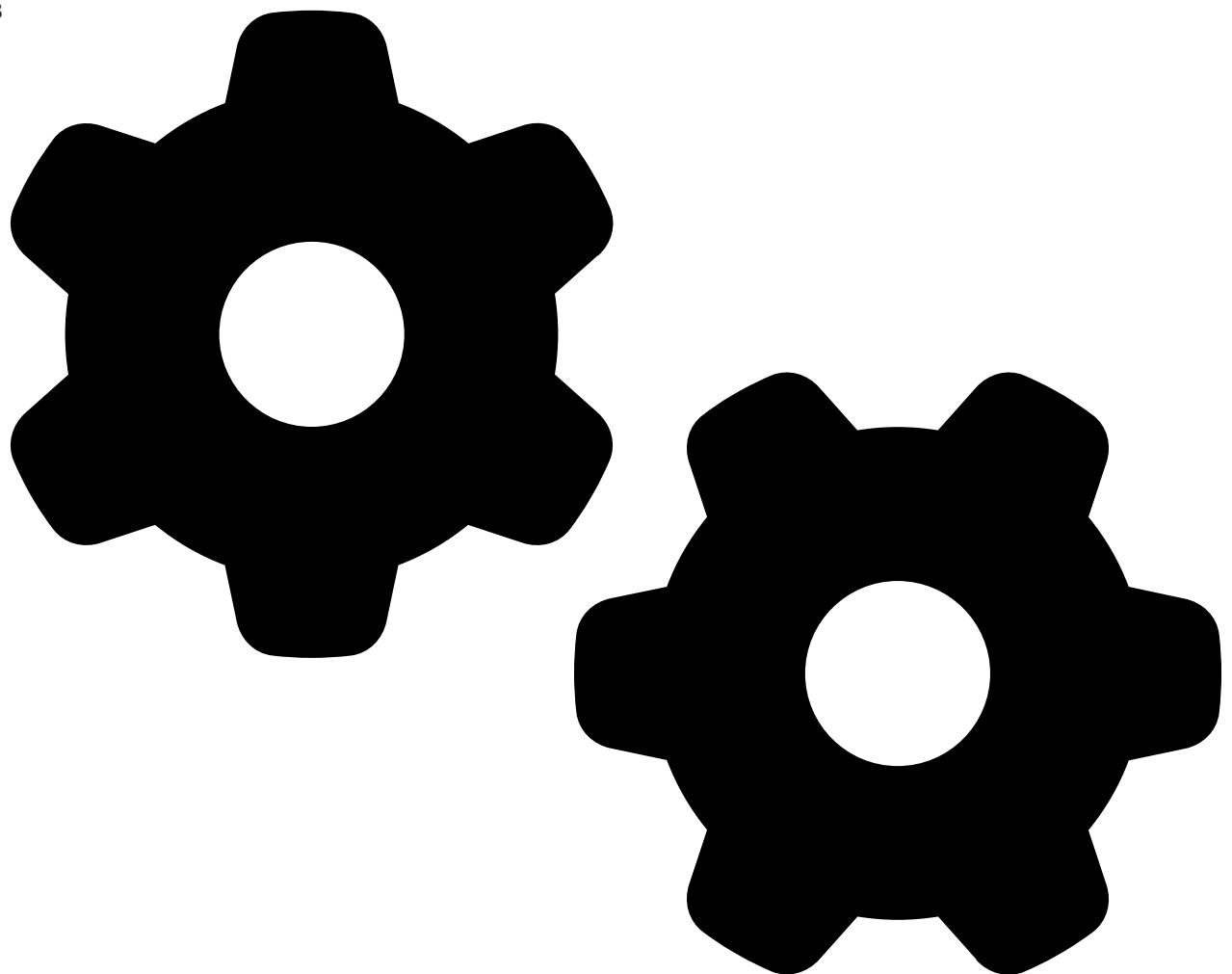
5. T4 - Prog.	180
5.1 C3 Récursivité	180



6. T5 Algo. 193

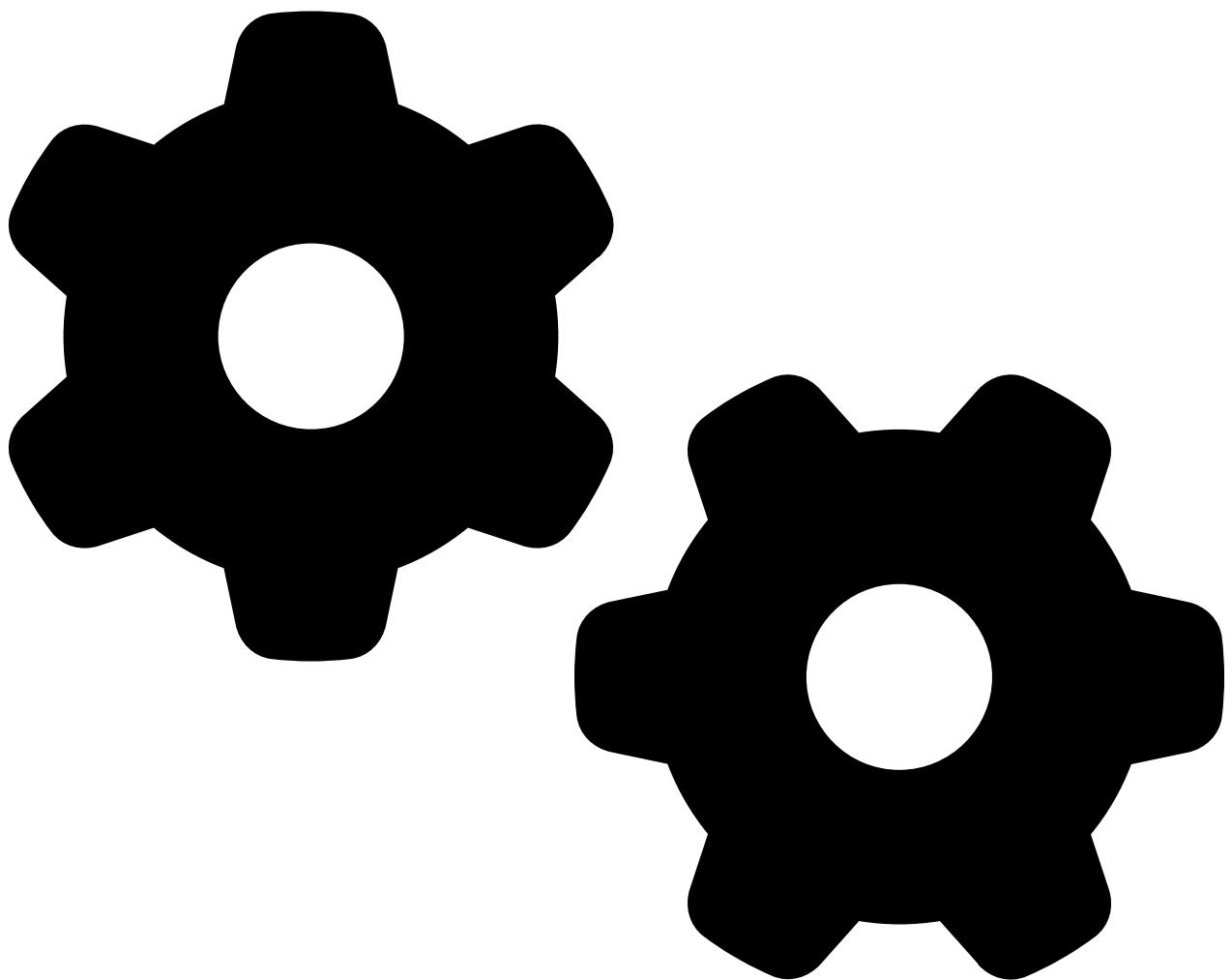
6.1 [C7] Algorithmes de tri

193



6.2 **C8** Diviser pour régner

211



7. BAC		232	
7.1	2021	232	
7.2	2022	236	
8.	Annales du Bac E.P.		241
8.1	Épreuve pratique	241	
8.2	Epreuve Pratique - Consigne	242	
8.3	2022	244	
9.	Evaluations	299	
9.1	SQL : Devoir n°1	299	
9.2	DS 0011	301	
10.	Extras	308	
10.1	Projet	308	
11.	Conseils de travail	336	
11.1	Conditions matérielles	336	
11.2	Dossiers, fichiers et versionning	336	
11.3	Usage du clavier	337	

1. Accueil

Ce site est dédié aux élèves de Terminales NSI du Lycée Murat à Issoire afin de retrouver les divers TP & TD travaillés en classe.

Les notebooks sont corrigés et les cellules peuvent être copiées pour être testées dans vos notebooks Capytale.

Actualités

- **Thème en cours :** T5 - Les tris
- Correction des exercices BAC Routage disponibles
- Correction des exercices BAC Piles et Files disponibles
- **[NEW] : Bientôt disponible** un onglet annales de BAC épreuves écrites.

1. 1.1 Progression sur l'année

1.1. 1.1.1 Liste des chapitres

	Titre	Durée
	C1- Le Modèle relationnel	1
	C2- Langage SQL	2
	C3- Récursivité	1
	C4- Programmation Orientée Objet	2
	C5- Listes et Piles	2
	C6- Protocole de routage	1
	C7- Algorithmes de tri	1
	C8- Diviser pour régner	1

Epreuve BAC 2023

- Les épreuves de spécialité se dérouleront les 20 et 21 mars 2023 pour la N.S.I.
- Les épreuves pratiques seront organisées je Jeudi 30 mars 2023.

Le sujet de l'épreuve écrite de la spécialité numérique et sciences informatiques, comporte trois exercices indépendants les uns des autres, qui permettent d'évaluer les connaissances et compétences des candidats. Le sujet comprend obligatoirement au moins un exercice relatif à chacune des trois rubriques suivantes :

- traitement de données en tables et bases de données;
- architectures matérielles, systèmes d'exploitation et réseaux;
- algorithmique, langages et programmation.

2. 1.2 Notions à l'épreuve écrite de Mars 2023 (mis à jours 30/09/2022):

2.1. 1.2.1 ➔ Thème : Structure de données

- Structure de données abstraites(file) : interface et implémentation
- Vocabulaire de la programmation objet : classes, attributs, méthodes, objets
- Listes, piles, files : structures linéaires.
- Dictionnaires, index et clé
- Arbres : structures hiérarchiques. Arbres binaires : nœuds, racines, feuilles, sous-arbres gauches, sous-arbres droits

2.2. 1.2.2 ➔ Thème : Base de données

- Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel
- Base de données relationnelle et SGBD.
- Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données

2.3. 1.2.3 ➔ Thème : Architectures matérielles, système d'exploitation et réseaux(ARSE)

- Gestion des processus et des ressources par un système d'exploitation
- Protocoles de routage

2.4. 1.2.4 ➔ Thème : Langage de programmation(LP)

- Récursivité
- Modularité
- Mise au point des programmes. Gestion des bugs.

2.5. 1.2.5 ➔ Thème : Algorithmique(A)

- Algorithmes sur les arbres binaires et sur les arbres binaires de recherche.
- Méthode « diviser pour régner »

3. 1.3 Comment calculer sa note au BAC 2023 et Répartition des notes :

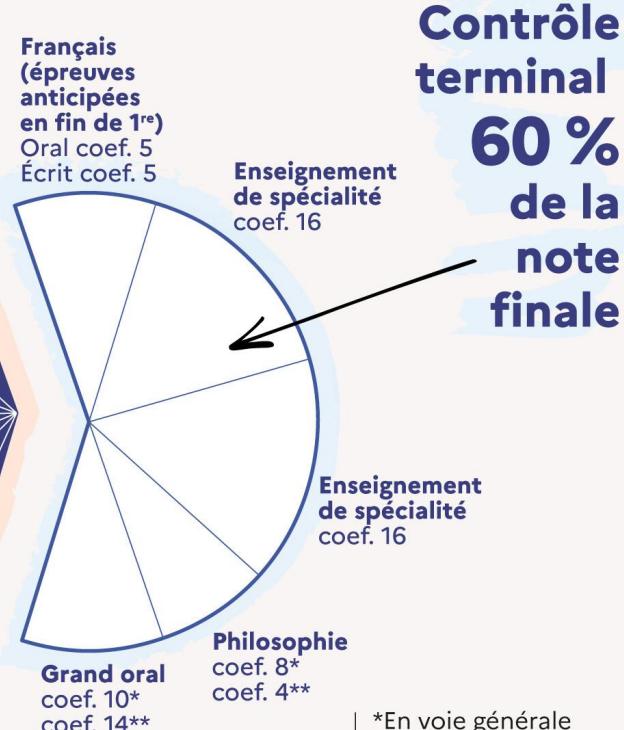
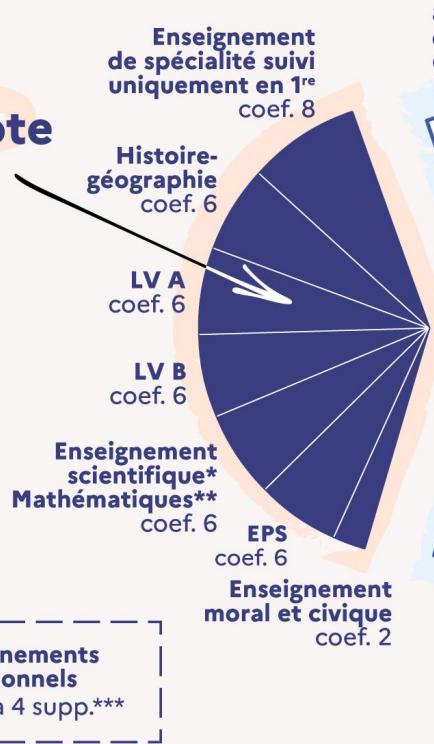
Fiche pour calculer sa note au BAC 2023

BACCALAURÉAT GÉNÉRAL ET TECHNOLOGIQUE

Répartition de la note finale



**Contrôle continu
40 %
de la note finale**



MENJS - Octobre 2021

*En voie générale
**En voie technologique
*** 2 si suivi uniquement une année, 4 si suivi en 1^{re} et terminale



2. T1 - Struct. de données

1. 2.1 C4 Programmation Orientée Objet



Programme Terminale

Contenus	Capacités attendues	Commentaires
Vocabulaire de la programmation objet : classes attributs, méthodes, objets	Ecrire la définition d'une classe. Accéder aux attributs et méthodes d'une classe	On n'aborde pas ici tous les aspects de la programmation objet comme le polymorphisme et l'héritage.



Le paradigme objet

La **POO** est un paradigme de programmation, au même titre que la programmation impérative (que nous pratiquons déjà) ou la programmation fonctionnelle , ou encore d'autres paradigmes (la liste est longue).

Un paradigme de programmation pourrait se définir comme une philosophie dans la manière de programmer : c'est un parti-pris revendiqué dans la manière d'aborder le problème à résoudre. Une fois cette décision prise, des outils spécifiques au paradigme choisi sont utilisés.

Nous nous limiterons cette année, comme le programme l'indique, à une brève introduction de la programmation objet.

1.1. 2.1.1 Vocabulaire de la programmation objet

La programmation objet consiste à regrouper données et traitements dans une même structure appelée **objet**. Elle possède l'avantage de localiser en un même endroit toute l'implémentation d'une structure de données abstraite.



Objets, attributs, méthodes

Concrètement, un objet est une structure de données abstraite regroupant :

- des **données** associées à l'objet que l'on appelle des **attributs**.
- des **fonctions** (ou procédures) s'appliquant sur l'objet que l'on appelle **méthodes**.

1.2. 2.1.2 Classes et objets en Python

En Python, tout est objet !

Vous ne le saviez sans doute pas, mais les objets vous connaissez déjà.

Vous avez manipulé des objets depuis que vous programmez en Python, tout simplement car dans ce langage tout est objet. On peut le voir facilement.

Script Python

```
m=[4,5,8,2]
type(m)

<class 'list'>
```

`m` est une liste, ou plus précisément un **objet** de type `list`. Et en tant qu'objet de type `list`, il est possible de lui appliquer certaines fonctions prédéfinies (qu'on appellera **méthodes**) :

🐍 Script Python

```
m.reverse()
m
[2, 8, 5, 4]
```

La syntaxe utilisée (le `.` après le nom de l'objet) est spécifique à la POO. Chaque fois que vous voyez cela, c'est que vous êtes en train de manipuler des objets.

Nous ne sommes pas surpris par ce résultat car la personne qui a programmé la méthode `reverse()` lui a donné un nom explicite. Comment a-t-elle programmé cette inversion des valeurs de la liste ? Nous n'en savons rien et cela ne nous intéresse pas. Nous sommes juste utilisateurs de cette méthode. L'objet de type `list` nous a été livré avec sa méthode `reverse()` (et bien d'autres choses) et nous n'avons pas à démonter la boîte pour en observer les engrenages : on parle de principe d'**encapsulation**.

On peut obtenir la liste de toutes les fonctions disponibles pour un objet de type `list`, par la fonction `dir` :

🐍 Script Python

```
dir(m)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Les méthodes encadrées par un double underscore `__` sont des méthodes **privées**, a priori non destinées à l'utilisateur. Les méthodes **publiques**, utilisables pour chaque objet de type `list`, sont donc `append`, `clear`, ...

Comment savoir ce que font les méthodes ? Si elles ont été correctement codées (et elles l'ont été), elles possèdent une *docstring*, accessible par :

🐍 Script Python

```
m.append.__doc__
'Append object to the end of the list.'
```

🐍 Script Python

```
m.index.__doc__
'Return first index of value.\n\nRaises ValueError if the value is not present.'
```

🐍 Script Python

```
help(m.index)
Help on built-in function index:

index(value, start=0, stop=2147483647, /) method of builtins.list instance
    Return first index of value.

    Raises ValueError if the value is not present.
```

1.3. 2.1.3 Mise en pratique

1.3.1. Une première classe en Python

Nous allons voir comment implémenter une classe en Python.

Vocabulaire :

Par convention, les noms de classes en Python sont écrits en capitales (première lettre en majuscule). On a documenté notre classe avec une docstring qui sera accessible à quiconque souhaite utiliser notre classe.

Script Python

```
class Voiture :
    """une classe sur les voitures ...."""
    def __init__(self, annee, coul, vmax):
        """Initialisation avec les caractéristiques d'une voiture
        annee, couleur, vitesse max, age """
        self.annee = annee
        self.couleur = coul
        self.vitesse_max = vmax
        self.age = 2022 - self.annee
```

CRÉATION ET INITIALISATION D'UN OBJET EN PYTHON

Retenir : la méthode constructeur

La **méthode constructeur**, toujours appelée `__init__()`, est une méthode (une «def») qui sera automatiquement appelée à la création de l'objet. Elle va donc le doter de tous les attributs de sa classe

Script Python

```
class Voiture :
    """une classe sur les voitures ...."""
    def __init__(self, annee, marque, coul, vmax):
        """Initialisation avec les caractéristiques d'une voiture
        annee, couleur, vitesse max, age """
        self.annee = annee
        self.marque=marque
        self.couleur = coul
        self.vitesse_max = vmax
        self.age = 2022 - self.annee
```

- le mot-clé `self`, omniprésent en POO (d'autres langages utilisent `this`), fait référence à l'objet lui-même, qui est en train d'être construit.
- pour construire l'objet, 4 paramètres seront nécessaires : `annee`, `marque`, `coul` et `vmax`. Ils donneront respectivement leur valeur aux attributs `annee`, `couleur` et `vitesse_max`.
- dans cet exemple, les noms `coul` et `vmax` ont été utilisés pour abréger `couleur` et `vitesse_max`, mais il est recommandé de garder les mêmes noms, même si ce n'est pas du tout obligatoire.

Construisons donc notre première voiture !

Script Python

```
ma_voiture = Voiture(2012, "Citroën", "Grise", 180)

```mon bolide``` possède 5 attributs :
- ````annee```` , ````marque```` , ````couleur```` et ````vitesse_max```` ont été donnés par l'utilisateur lors de la création.
- ````age```` s'est créé «tout seul» par l'instruction ````self.age = 2022 - self.annee````.
```

### Script Python

```
type(ma_voiture)
<class '__main__.Voiture'>
```

### Script Python

```
print(ma_voiture.annee)
print(ma_voiture.marque)
print(ma_voiture.couleur)
print(ma_voiture.vitesse_max)
print(ma_voiture.age)
```

```
2012
Citroën
Grise
180
10
```

Bien sûr, on peut créer autant de voitures que l'on veut en suivant le même principe :

#### Script Python

```
batmobile = Voiture(2036, 'Audi', "noire", 525)
batmobile.marque

'Audi'
```



### Exercice 1 : Class Rationnel

Nous allons créer une classe `Rationnel` permettant de créer un objet `R` dont on récupérera le numérateur par la variable `R.num` et le dénominateur par la variable `R.den`.

On déclare une classe en Python à l'aide du mot clé `class` :

#### Script Python

```
class Rationnel:
 """Manipulation de rationnels définis par leurs numérateur et dénominateur"""


```

Par convention, les noms de classes en Python sont écrits en capitales (première lettre en majuscule). On a documenté notre classe avec une docstring qui sera accessible à quiconque souhaite utiliser notre classe.



Compléter le code suivant pour créer cette classe

#### Script Python

```
class Rationnel:
 """Manipulation de rationnels définis par leurs numérateur et dénominateur"""

 def __init__(self, numerateur, denominateur):
 """Initialise le rationnel avec les valeurs indiquées"""
 self.num = numerateur
 self.den = denominateur
```

On peut désormais créer un objet `R` par appel du constructeur en fournissant les valeurs des paramètres prévus dans la méthode spéciale d'initialisation. On peut accéder aux attributs de l'objet en utilisant la notation pointée.

#### Script Python

```
R=Rationnel(2,3)
print(R.num)
print(R.den)

2
3
```

On peut modifier les attributs d'un objet en les redéfinissant.

#### Script Python

```
R.num = 8 # modification de la valeur du numérateur
R.num, R.den
```

(8, 3)

Les attributs `num` et `den` sont propres à chaque objet. Dans la terminologie des langages à objet, on parle d'*attributs d'instance*.

Notre objet est bien du type abstrait de données `Rationnel` que l'on vient de créer en définissant notre classe.

### Script Python

```
type(R)
<class '__main__.Rationnel'>
```

ÉCRITURE DES MÉTHODES DÉDIÉES

Si on veut pouvoir manipuler nos objets, il faut ajouter à notre classe les méthodes souhaitées.

Par exemple, on ajoute les méthodes `ajouter` et `egal` en définissant deux fonctions dans notre classe.

### Script Python

```
class Rationnel:
 """Manipulation de rationnels définis par leurs numérateur et dénominateur"""

 def __init__(self, numerateur, denominateur):
 """Initialise le rationnel avec les valeurs indiquées"""
 self.num = numerateur
 self.den = denominateur

 def ajouter(self, other):
 """Renvoie un nouveau rationnel égal à la somme"""
 import math
 num = self.num * other.den + other.num * self.den # calcul du numérateur
 den = self.den * other.den # calcul du dénominateur
 d = math.gcd(num, den) # calcul du pgcd pour simplifier le rationnel
 return Rationnel(num // d, den // d) # on renvoie un nouvel objet 'Rationnel'

 def egal(self, other):
 """Renvoie Vrai si les deux rationnels sont égaux, Faux sinon."""
 return self.num == other.num and self.den == other.den
```

On peut alors accéder à ces méthodes en utilisant également la notation pointée sur l'objet auquel s'applique la méthode.

### Script Python

```
r1 = Rationnel(1, 4)
r2 = Rationnel(1, 2)
r3 = r1.ajouter(r2) # on ajoute r2 à r1
r3.num, r3.den

(3, 4)
```

### Script Python

```
r4 = Rationnel(3, 4)
r3.egal(r4) # pour vérifier si r3 = r4

True
```

**Remarque :** Vous noterez que l'on fournit toujours un paramètre de moins lors de l'appel à une méthode que dans la définition de la méthode. En effet, le paramètre `self` n'est pas utilisé car il désigne la référence à l'objet auquel s'applique la méthode.

On peut utiliser la fonction `dir` pour lister tous les attributs et méthodes d'un objet.

### Script Python

```
r= Rationnel(5, 3)
dir(r)

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'ajouter', 'den', 'egal', 'num']
```

On constate qu'il y a de nombreuses méthodes spéciales repérables par leur nom encadré de `__`. Ces méthodes sont appelées dans des contextes particuliers et peuvent être redéfinies par le programmeur pour une classe particulière.

L'usage de ces méthodes spéciales n'est pas un attendu du programme mais cela peut se révéler très utile. C'est pourquoi nous en présenterons quelques-unes.

#### 1.4. 2.1.4 Méthodes spéciales en Python : Hors programme

Nous nous contenterons ici de présenter trois méthodes spéciales (quelques autres seront évoquées dans les activités) :

- la méthode `__repr__(self)` est appelée pour calculer la représentation *officielle* en chaîne de caractères d'un objet (c'est cette méthode qui est appelée lorsque l'on veut évaluer un objet) ;
- la méthode `__str__(self)` est appelée pour calculer une chaîne de caractères *informelle* ou joliment mise en forme de représentation de l'objet (c'est cette méthode qui est appelée par la fonction `print()`) ;
- la méthode `__eq__(self, other)` est appelée pour tester l'égalité entre deux objets.

On pourrait être tenté d'afficher une instance ou de tester l'égalité entre deux instances d'une même classe. Par exemple, avec notre classe `Rationnel` on aimeraient écrire.

##### Script Python

```
r1 = Rationnel(1, 2)
r2 = Rationnel(1, 2)
```

##### Script Python

```
r1 # évaluation
<__main__.Rationnel object at 0xc72088>
```

##### Script Python

```
print(r1) # affichage
<__main__.Rationnel object at 0xc72088>
```

##### Script Python

```
r1 == r2 # test d'égalité
False
```

Nous ne pouvons nous satisfaire des résultats. L'évaluation d'un objet, son affichage et le test d'égalité (avec les notations habituelles) font appel respectivement aux méthodes `__repr__`, `__str__` et `__eq__` que nous avons besoin de redéfinir pour obtenir des résultats cohérents.

##### Script Python

```
class Rationnel:
 """Manipulation de rationnels définis par leurs numérateur et dénominateur"""

 def __init__(self, numerateur, denominateur):
 """Initialise le rationnel avec les valeurs indiquées"""
 self.num = numerateur
 self.den = denominateur

 def ajouter(self, other):
 """Renvoie un nouveau rationnel égal à la somme"""
 import math
 num = self.num * other.den + other.num * self.den # calcul du numérateur
 den = self.den * other.den # calcul du dénominateur
 d = math.gcd(num, den) # calcul du pgcd pour simplifier le rationnel
 return Rationnel(num // d, den // d) # on renvoie un nouvel objet 'Rationnel'

 def egal(self, other):
 """Renvoie Vrai si les deux rationnels sont égaux, Faux sinon."""
 return self.num == other.num and self.den == other.den

 def __repr__(self):
 return "Rationnel(" + str(self.num) + ", " + str(self.den) + ")" # ou f'Rationnel({self.num}, {self.den})'

 def __str__(self):
 return str(self.num) + " / " + str(self.den) # ou f'{self.num} / {self.den}'

 def __eq__(self, other): # on pourrait aussi écrire simplement __eq__ = egal
 return self.num * other.den == other.num * self.den
```

On peut désormais utiliser les instructions classiques d'évaluation (ou d'affichage) et de test d'égalité avec les objets de notre classe.

#### Script Python

```
r1 = Rationnel(1, 2)
r2 = Rationnel(1, 2)
r1

Rationnel(1, 2)
```

#### Script Python

```
print(r1)

1 / 2
```

#### Script Python

```
r3 = Rationnel(1, 4)
r4 = r3.ajouter(r1)
r4

Rationnel(3, 4)
```

#### Script Python

```
r1 == r2

True
```

#### Que se passe-t-il pour la dernière instruction ?

Python reconnaît qu'il doit tester l'égalité entre deux instances de la classe `Rationnel`.

Ce test (`=`) invoque la méthode spéciale `__eq__` de la classe `Rationnel`. Plus précisément, `r1 == r2` appelle `r1.__eq__(r2)` et comme nous venons de définir cette méthode, le résultat est cohérent. On peut désormais tester l'égalité de deux rationnels sans utiliser l'instruction un peu plus lourde `r1.egal(r2)`.

#### Bilan :

La *paradigme objet* est une autre façon de voir la programmation qui consiste à utiliser un structure de donnée appelée *objet* qui réunit des données et des fonctionnalités. Les données sont appelées **attributs** et les fonctionnalités sont appelées **méthodes**.

Une **classe** permet de définir un modèle d'objet en spécifiant des attributs et des méthodes. On peut ensuite utiliser cette classe pour fabriquer des objets selon ce modèle.

En Python, on utilise le mot clé `class` pour définir une classe qui devient alors un nouveau type abstrait de données. On peut alors créer de nouveaux objets en appelant le constructeur qui porte le nom de la classe. Les objets ainsi créés s'appellent des *instances* de la classe.

En Python, la méthode spéciale `__init__` est appelée à la construction d'un nouvel objet. C'est dans cette méthode que l'on définit les attributs de nos objets.

Les attributs et méthodes d'une instance de classe sont accessibles en utilisant la notation pointée : `objet.attribut` et `objet.methode(arguments)`.

## 2. 2.2 POO : TD

## Thème 1 - Structure de données

05

TD : Programmation  
Orientée Objet (POO)

## 2.1. 2.2.1 Exercice 1 : Class Eleve

 Exercice 1

1. Écrire une classe `Eleve` qui contiendra les attributs `nom`, `classe` et `note`.
2. Instancier trois élèves de cette classe.
3. Écrire une fonction `compare(eleve1, eleve2)` qui renvoie le nom de l'élève ayant la meilleure note.

**Exemple d'utilisation de la classe** Script Python

```
>>> riri = Eleve("Henri", "TG2", 12)
>>> fifi = Eleve("Philippe", "TG6", 15)
>>> loulou = Eleve("Louis", "TG1", 8)
>>> compare(riri, fifi)
'Philippe'
```

 Script Python

```
class Eleve:
 def __init__(self, nom, classe, note):
 self.nom = nom
 self.classe = classe
 self.note = note

 def __repr__(self):
 return self.nom + ' ' + self.classe + ' ' + str(self.note)

 def compare(self, other):
 if self.note > other.note:
 return self.nom
 else:
 return other.nom
```

 Script Python

```
riri = Eleve("Henri", "TG2", 12)
fifi = Eleve("Philippe", "TG6", 15)
loulou = Eleve("Louis", "TG1", 8)
riri.compare(fifi)
```

 Texte

'Philippe'

 Script Python

riri

 Texte

Henri TG2 12

## 2.2. 2.2.2 Exercice 2 : Class Player

 Exercice 2

Écrire une classe `Player` qui :

- ne prendra aucun argument lors de son instantiation.
- affectera à chaque objet créé un attribut `energie` valant 3 par défaut.
- affectera à chaque objet créé un attribut `alive` valant `True` par défaut.
- fournira à chaque objet une méthode `blessure()` qui diminue l'attribut `energie` de 1.
- fournira à chaque objet une méthode `soin()` qui augmente l'attribut `energie` de 1.
- si l'attribut `energie` passe à 0, l'attribut `alive` doit passer à `False` et ne doit plus pouvoir évoluer.

## Exemple d'utilisation de la classe

 Script Python

```
>>> mario = Player()
>>> mario.energie
3
>>> mario.soin()
>>> mario.energie
4
>>> mario.blessure()
>>> mario.blessure()
>>> mario.blessure()
>>> mario.alive
True
>>> mario.blessure()
>>> mario.alive
False
>>> mario.soin()
>>> mario.alive
False
>>> mario.energie
0
```

 Script Python

```
class Player():
 def __init__(self, energie = 3, alive = True):
 self.hp = energie
 self.alive = alive

 def soin(self, h = 1):
 if self.alive: self.hp += h

 def blessure(self, damage = 1):
 self.hp -= damage
 if self.hp <= 0:
 self.alive = False
 self.hp = 0
```

 Script Python

```
mario = Player()
mario.hp
mario.blessure()
mario.hp
mario.blessure(3)
mario.hp
mario.soin(5)
mario.hp
```

 Texte

0

### Exercice 3

Créer une classe `CompteBancaire` dont la méthode constructeur recevra en paramètres :

- un attribut `titulaire` stockant le nom du propriétaire.
- un attribut `solde` contenant le solde disponible sur le compte.

Cette classe contiendra deux méthodes `retrait()` et `depot()` qui permettront de retirer ou de déposer de l'argent sur le compte.

#### Exemple d'utilisation de la classe

##### Script Python

```
>>> compteGL = CompteBancaire("G.Lassus", 1000)
>>> compteGL.retrait(50)
Vous avez retiré 50 euros
Solde actuel du compte : 950 euros
>>> compteGL.retrait(40000)
Retrait impossible
>>> compteGL.depot(1000000)
Vous avez déposé 1000000 euros
Solde actuel du compte : 10000950 euros
```

##### Script Python

### 2.3. 2.2.3 Exercice 3 : Class Pokemon

### Exercice 4 Pokemon et POO

#### 2.3.1. Création d'un Pokemon

- On considère une classe `Pokemon` qui permet de créer des pokemons.
- Chaque Pokemon a les caractéristiques suivantes :
  - un nom
  - un type
  - une vitesse
  - une attaque
  - un nombre de points de vie maximal

**Question n° A - 1 :** >En étudiant le code ci-dessous, modifier la ligne de code appropriée pour créer un Pokemon ayant les caractéristiques suivantes : \* nom : Aqua \* type : eau \* vitesse : 35 \* attaque : 75 \* defense : 20 \* points de vie maximal : 141

##### Script Python

```
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
```

```

 self.pvActuels = pvMax

aqua = Pokemon('Aqua', 'eau', 35, 75, 20, 141)
assert aqua.nom == "Aqua"
assert aqua.type == "eau"
assert aqua.vitesse == 35
assert aqua.attaque == 75
assert aqua.defense == 20
assert aqua.pvActuels == 141

```

**Question n° A - 2 :** > Modifier l'attribut `defense` du Pokemon `aqua` pour qu'il prenne la valeur 15.

### 🐍 Script Python

```
#Ecrire le code ici !!!
assert aqua.defense == 15
```

### 2.3.2. Méthode spéciale `__str__`

**Question n° B - 1 :**

Surcharger la méthode `__str__(self)` pour qu'elle affiche l'ensemble des attributs du Pokemon

### 🐍 Script Python

```

#Reprendre le code de la classe "Pokemon" et compléter !!!
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
 self.pvActuels = pvMax

 def __str__(self):
 return self.nom + ', ' + self.type + ', ' + str(self.vitesse) + ', ' + str(self.attaque) + ', ' + str(self.defense) + ', ' + str(self.pvMax) + ', ' + str(self.pvActuels)

aqua = Pokemon('Aqua', 'eau', 35, 75, 20, 141)
print(aqua)

```

### 📋 Texte

Aqua, eau, 35, 75, 20, 141, 141

**Question n° B - 2 :** > Créer une méthode `etre_ko(self)` qui renvoie `True` si le Pokemon a 0 point de vie ou moins et `False` sinon.

### 🐍 Script Python

```

#Reprendre le code de la classe "Pokemon" et compléter !!!
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
 self.pvActuels = pvMax

 def etre_ko(self):
 if self.pvActuels <= 0:
 return True
 else:
 return False

```

```
aqua = Pokemon('Aqua', 'eau', 35, 75, 20, 141)

assert aqua.etre_ko() == False
aqua.pvActuels = 0
assert aqua.etre_ko() == True
aqua.pvActuels = -7
assert aqua.etre_ko() == True
```

**Question n° B - 3 :** > Crée une méthode `se_reposer(self)` qui redonne tous ses points de vie au Pokemon.

### 🐍 Script Python

```
#Reprendre le code la classe "Pokemon" et compléter !!!
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
 self.pvActuels = pvMax

 def __str__(self):
 return self.nom + ', ' + self.type + ', ' + str(self.vitesse) + ', ' + str(self.attaque) + ', ' + str(self.defense) + ', ' + str(self.pvMax) + ', ' + str(self.pvActuels)

 def etre_ko(self):
 if self.pvActuels <= 0:
 return True
 else:
 return False

 def se_reposer(self):
 self.pvActuels = self.pvMax
```

### 2.3.3. Interaction avec un autre Pokemon

Un pokémon peut être de 4 types : \* Eau \* Feu \* Air \* Terre

On sait que : \* l'eau domine le feu \* le feu domine l'air \* l'air domine la terre \* la terre domine l'eau

**Question n° C - 1 :** > Crée une méthode `dominer(self, adversaire)` prenant en paramètre une autre instance de la classe `Pokemon`, qui renvoie `True` si le Pokémon a un type qui domine celui du Pokémon donné en paramètre et `False` sinon.

### 🐍 Script Python

```
#Reprendre le code la classe "Pokemon" et compléter !!!
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
 self.pvActuels = pvMax

 def __str__(self):
 return self.nom + ', ' + self.type + ', ' + str(self.vitesse) + ', ' + str(self.attaque) + ', ' + str(self.defense) + ', ' + str(self.pvMax) + ', ' + str(self.pvActuels)

 def etre_ko(self):
 if self.pvActuels <= 0:
 return True
 else:
 return False

 def se_reposer(self):
 self.pvActuels = self.pvMax
```

```

def dominer(self, other):
 if self.type == 'eau':
 if other.type == 'feu': return True
 else: return False
 if self.type == 'feu':
 if other.type == 'air': return True
 else: return False
 if self.type == 'air':
 if other.type == 'terre': return True
 else: return False
 if self.type == 'terre':
 if other.type == 'eau': return True
 else: return False

aqua = Pokemon('Aqua', 'eau', 35, 75, 20, 141)

flamichou = Pokemon("Flamichou", "feu")
assert aqua.dominer(flamichou) == True
rocamon = Pokemon("Rocamon", "terre")
assert aqua.dominer(rocamon) == False

```

Lorsque qu'un Pokemon attaque un autre Pokemon, il inflige des dégâts comptabilisés de la façon suivante : \* la valeur des dégâts est égale à la différence entre l'attaque du Pokemon attaquant et la défense de l'autre Pokemon (si la différence est nulle ou négative, alors les dégâts seront égaux à 1) \* si le type du Pokemon attaquant domine celui de l'autre Pokemon, alors les dégâts sont multipliés par 2

**Question n° C - 2 :** > Créer une méthode `attaquer(self, adversaire)` prenant en paramètre une autre instance de la classe `Pokemon` et qui retire des points de vie à ce dernier.

### 🐍 Script Python

```

#Reprendre le code la classe "Pokemon" et compléter !!!
from random import *

class Pokemon:
 def __init__(self, nom : str = "Anonyme",
 typ = choice(["eau", "air", "terre", "feu"]),
 vitesse = randint(1,51),
 attaque = randint(51,100),
 defense = randint(1,50),
 pvMax = randint(101,150)):
 self.nom = nom
 self.type = typ
 self.vitesse = vitesse
 self.attaque = attaque
 self.defense = defense
 self.pvMax = pvMax
 self.pvActuels = pvMax

 def __str__(self):
 return self.nom + ', ' + self.type + ', ' + str(self.vitesse) + ', ' + str(self.attaque) + ', ' + str(self.defense) + ', ' + str(self.pvMax) + ', ' + str(self.pvActuels)

 def etre_ko(self):
 if self.pvActuels <= 0:
 return True
 else:
 return False

 def se_reposer(self):
 self.pvActuels = self.pvMax

 def dominer(self, other):
 if self.type == 'eau':
 if other.type == 'feu': return True
 else: return False
 if self.type == 'feu':
 if other.type == 'air': return True
 else: return False
 if self.type == 'air':
 if other.type == 'terre': return True
 else: return False
 if self.type == 'terre':
 if other.type == 'eau': return True
 else: return False

 def attaquer(self, other):
 damage = self.attaque - other.defense
 if damage <= 1: damage = 1
 if self.dominator(other):
 damage *= 2
 other.pvActuels -= damage
 if other.pvActuels <= 0: other.pvActuels = 0

aqua = Pokemon('Aqua', 'eau', 35, 75, 20, 141)

```

```

flamichou = Pokemon("Flamichou", "feu", 15, 63, 32, 118)
rocamon = Pokemon("Rocamon", "terre", 44, 95, 48, 101)
aqua.atttaquer(flamichou)
assert flamichou.pvActuels == 32
aqua.atttaquer(rocamon)
assert rocamon.pvActuels == 74

```

### 2.3.4. Combat de Pokemons

Lorsque deux Pokemons combattent l'un contre l'autre : \* le premier à attaquer est celui ayant la vitesse la plus élevée \* s'est ensuite à l'autre d'attaquer et ainsi de suite ... \* le combat s'arrête dès qu'un des Pokemons est ko \* le vainqueur est le Pokemon n'étant pas ko

**Question n° D - 1 :** > Créez une fonction `combattre(pokemon1, pokemon2)` prenant en paramètres deux instances de la classe `Pokemon` et renvoyant le pokemon vainqueur du combat

#### Script Python

```

def combattre(pokemon1, pokemon2):
 if pokemon1.vitesse > pokemon2.vitesse:
 p1 = pokemon1
 p2 = pokemon2
 else:
 p1 = pokemon2
 p2 = pokemon1
 while pokemon1.etre_ko() == False and pokemon2.etre_ko() == False:
 p1.atttaquer(p2)
 print(pokemon1.nom + ':' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ':' + str(pokemon2.pvActuels) + 'hp')
 p2.atttaquer(p1)
 print(pokemon1.nom + ':' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ':' + str(pokemon2.pvActuels) + 'hp')
 if pokemon1.etre_ko() == True:
 return pokemon1.nom
 else:
 return pokemon2.nom

vainqueur = combattre(aqua, flamichou)
print(vainqueur)

```

#### Texte

Flamichou

**Question n° D - 2 :** > Complétez la fonction `combattre` pour que les Pokemons se reposent avant d'engager le combat

#### Script Python

```

#Reprendre le code de la fonction "Combattre" et le compléter

def combattre(pokemon1, pokemon2):
 pokemon1.se_reposer()
 pokemon2.se_reposer()
 if pokemon1.vitesse > pokemon2.vitesse:
 p1 = pokemon1
 p2 = pokemon2
 else:
 p1 = pokemon2
 p2 = pokemon1
 while pokemon1.etre_ko() == False and pokemon2.etre_ko() == False:
 print(pokemon1.nom + ':' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ':' + str(pokemon2.pvActuels) + 'hp')
 p1.atttaquer(p2)
 print('\t' + pokemon1.nom + ':' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ':' + str(pokemon2.pvActuels) + 'hp')
 p2.atttaquer(p1)
 print('\t' + pokemon1.nom + ':' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ':' + str(pokemon2.pvActuels) + 'hp')
 if pokemon1.etre_ko() == True:
 return pokemon2.nom
 else:
 return pokemon1.nom

aqua = Pokemon('Aqua', 'eau', 35, 65, 20, 141)
flamichou = Pokemon("Flamichou", "feu", 15, 63, 32, 118)
rocamon = Pokemon("Rocamon", "terre", 20, 35, 48, 101)

vainqueur = combattre(aqua, rocamon)
print(vainqueur)

```

#### Texte

Aqua: 141hp | Rocamon: 101hp  
Aqua: 141hp | Rocamon: 84hp  
Aqua: 111hp | Rocamon: 84hp

```
Aqua: 111hp | Rocamon: 84hp
Aqua: 111hp | Rocamon: 67hp
Aqua: 81hp | Rocamon: 67hp
Aqua: 81hp | Rocamon: 67hp
Aqua: 51hp | Rocamon: 50hp
Aqua: 51hp | Rocamon: 50hp
Aqua: 51hp | Rocamon: 33hp
Aqua: 21hp | Rocamon: 33hp
Aqua: 21hp | Rocamon: 33hp
Aqua: 21hp | Rocamon: 16hp
Aqua: 0hp | Rocamon: 16hp
Rocamon
```

 **Script Python**

## 3. 2.3 POO : TD

## Thème 1 - Structure de données

**BAC**

# TD : Sujet BAC

## Programmation Orientée

## Objet (POO)

## 3.1. 2.3.1 Exercice 1 : (d'après Bac) - Locations de chambres.

**Principaux thèmes abordés :**

Structure de données (programmation objet) et langages et programmation (spécification).

La société LOCAVACANCES doit gérer la réservation de l'ensemble des chambres de ses gîtes. Chaque chambre d'un même complexe sera différenciée par son nom.

Pour cela, d'un point de vue informatique, on a créé deux classes : `Chambre` et `Gite` dont le code ci-dessous.

 **Script Python**

```
class Chambre:
 def __init__(self, nom: str):
 self._nom = nom
 self._occupation = [False for i in range(365)]

 def get_nom(self):
 return self._nom

 def get_occupation(self):
 return self._occupation

 def reserver(self, date: int):
 self._occupation[date - 1] = True

class Gite:
 def __init__(self, nom: str):
 self._nom = nom
 self._chambres = []

 def __str__(self):
 n = len(self._chambres)
 if n == 0:
 return "L'hôtel " + self._nom + " n'a aucune chambre."
 else:
 return "L'hôtel " + self._nom + " a " + str(n) + " chambre(s)"

 def get_chambres(self):
 return self._chambres

 def get_nchambres(self):
 return [ch.get_nom() for ch in self._chambres]

 def ajouter_chambres(self, nom_ch : str):
 self._chambres.append(Chambre(nom_ch))

 def mystere(self, date):
```

```

l_ch = []
for ch in self._chambres :
 if ch.get_occupation()[date - 1] == False :
 l_ch.append(ch.get_nom())
return(l_ch)

```

### 3.1.1. Partie A - Étude de la classe Chambre :

1. Lister les attributs en donnant leur type. Préciser s'ils sont modifiables dans la classe, en explicitant la méthode associée.

#### Reponse :

Les attributs sont en principe définis et initialisés dans la méthode `__init__`. On les désigne en faisant précéder leur nom du mot réservé `self`. Ici on a donc les attributs appelés `_nom` et `_occupation`.

On constate que lors de l'initialisation la valeur attribuée à `_nom` est le paramètre `nom` de la méthode `__init__`, dont le type est précisé comme étant `str`, c'est à dire chaîne de caractères.

`_occupation` est initialisé sous la forme d'une liste de valeurs booléennes (définie en compréhension)

La méthode `reserver` permet de modifier la valeur de l'attribut `_occupation`.

1. Écrire un `assert` dans la méthode `reserver` pour vérifier si le nombre `date` passé en paramètre est bien compris entre 1 et 365 (on ne gère pas les années bissextiles).

#### Reponse :

Modification de la méthode `reserver`:

##### 🐍 Script Python

```

def reserver(self, date: int):
 assert date >=1 and date <=365
 self._occupation[date - 1] = True

```

ou

##### 🐍 Script Python

```

def reserver(self, date: int):
 assert 1<= date <=365
 self._occupation[date - 1] = True

```

1. Écrire la méthode `annuler_reserver(self, date : int)` qui annule la réservation pour le jour `date`.

#### Reponse :

##### 🐍 Script Python

```

def annuler_reserver(self, date:int):
 self._occupation[date - 1] = False

```

### 3.1.2. Partie B - Étude de la classe Gite :

Le gîte « BonneNuit » a 5 chambres dénommées :

'Ch1', 'Ch2', 'Ch3', 'Ch4', 'Ch5'

On définit l'objet `giteBN` par l'instruction : `giteBN = Gite("BonneNuit")`.

#### 1. Méthode `ajouter_chambres()`

Écrire l'instruction Python pour ajouter '`Ch1`' à l'objet `giteBN`.

#### Reponse :

Ajout d'une chambre à `giteBN` en appelant sa méthode `ajouter_chambre` avec en paramètre le nom de la chambre :

#### Script Python

```
giteBN.ajouter_chambres('Ch1')
```

Dans les questions suivantes 2, 3 et 4, on considère que l'objet `giteBN` contient toutes les chambres du gite « BonneNuit ».

- La méthode `ajouter_chambres` permet d'enregistrer une nouvelle chambre, mais elle ne teste pas si le nom de cette chambre existe déjà.

Modifier la méthode pour éviter cet éventuel doublon.

#### Reponse :

Les chambres sont enregistrées dans l'attribut `_chambres` de l'objet `gite`. Il faut inspecter tous les noms de chambre dans cette liste pour savoir si le nouveau nom proposé existe déjà.

La méthode `get_nchambres` renvoie la liste des noms de toutes les chambres du gite. On peut donc faire :

#### Script Python

```
def ajouter_chambres(self, nom_ch : str):
 if nom_ch not in self.get_nchambres():
 self._chambres.append(Chambre(nom_ch))
```

ou bien, sans utiliser `get_nchambres`, une démarche classique où l'on parcourt la liste de chambres en positionnant un indicateur booléen (`nom_nouveau`) pour signaler éventuellement que le nom choisi existait déjà :

#### Script Python

```
def ajouter_chambres(self, nom_ch : str):
 nom_nouveau = True
 for chambre in self._chambres:
 if chambre.get_nom() == nom_ch:
 nom_nouveau = False
 if nom_nouveau:
 self._chambres.append(Chambre(nom_ch))
```

#### 1. Étude des méthodes : `get_chambres()` et `get_nchambres()`.

- Parmi les 4 propositions ci-dessous, quel est le type renvoyé par l'instruction Python : `giteBN.get_chambres()` .

- String
- Objet Chambre
- Tableau de String
- Tableau d'objets Chambre

**Réponse :**

Cette méthode renvoie la valeur de l'attribut `_chambres`, qui est une liste. La méthode `ajouter_chambres` de la classe `Gite` modifie cette liste en lui ajoutant un nouvel objet `Chambre` (créé par l'appel de `Chambre(nom_ch)`).

La réponse est donc "tableau d'objets Chambre" (une liste Python est un tableau dynamique)

b. Qu'affiche la suite d'instructions suivante ?

**Script Python**

```
ch = giteBN.get_chambres()[1]
print(ch.get_nom())
```

**Réponse :**

La première ligne récupère le second élément de la liste des chambres et la seconde affiche le nom de cette chambre.

c. Quelle différence existe-t-il entre les deux méthodes `get_nchambres()` et `get_chambres()` ?

**Réponse :**

La méthode `get_nchambres` renvoie une liste de chaînes de caractères qui sont les noms des chambres, et `get_chambres` renvoie une liste d'objets `Chambre`.

1. Les chambres '`Ch1`', '`Ch3`', '`Ch5`' sont réservées pour tout le mois de Janvier 2021.

a. Que va renvoyer l'instruction `giteBN.mystere(3)` ?

**Réponse :**

La méthode `mystere` renvoie une liste à laquelle ont été ajoutés les noms des chambres du gîte à condition qu'elles ne soient pas réservées à la date passée en paramètre.

Donc `giteBN.mystere(3)` va renvoyer la liste des noms des chambres du gîte "BonneNuit" disponibles le 4 janvier. Cette liste ne contiendra pas les noms '`Ch1`', '`Ch3`', '`Ch5`' puisque ces chambres sont réservées en Janvier. Elle contiendra peut-être '`Ch2`' et '`Ch4`' dont on ne sait pas si elles sont réservées le 4 janvier.

b. Dans la méthode `mystere` de la classe `Gite`, quel est le type des variables en paramètre et en sortie ? Quelles sont les méthodes ou attributs dont cette méthode a besoin ?

**Réponse :**

La méthode `mystere` prend comme paramètre `date`, qui est un entier (`int`).

Elle retourne une liste Python.

Elle a besoin de l'attribut `_chambres` de la classe `Gite` (ainsi que de la méthode `get_occupation` et `get_nom` de la classe `Chambre`, et la méthode `append` de la classe `list`). La question n'est pas très précise quand à ce qui est attendu)

### 3.2. 2.3.2 Exercice 2 : (d'après Bac) - Pots de yaourts "

#### Principaux thèmes abordés :

structure de données (programmation objet) et langages et programmation (spécification).

Une entreprise fabrique des yaourts qui peuvent être soit nature (sans arôme), soit aromatisés (fraise, abricot ou vanille).

Pour pouvoir traiter informatiquement les spécificités de ce produit, on va donc créer une classe Yaourt qui possèdera un certain nombre d'attributs :

- Son genre : nature ou aromatisé
- Son arôme : fraise, abricot, vanille ou aucun
- Sa date de durabilité minimale (DDM) exprimée par un entier compris entre 1 et 365 (on ne gère pas les années bissextiles). Par exemple, si la DDM est égale à 15, la date de durabilité minimale est le 15 janvier.

On va créer également des méthodes permettant d'interagir avec l'objet Yaourt pour attribuer un arôme ou récupérer un genre par exemple. On peut représenter cette classe par le tableau de spécifications ci-dessous :

Yaourt	
ATTRIBUTS	METHODES
genre	construire(arome,duree)
arome	obtenir_arome()
duree	obtenir_genre()
	obtenir_duree()
	attribuer_arome(arome)
	attribuer_duree(duree)
	attribuer_genre(arome)

1. Code partiel de la classe Yaourt, à compléter aux endroits indiqués en suivant les consignes des questions suivantes :

#### Script Python

```
class Yaourt:
 """ Classe définissant un yaourt caractérisé par :
 - son arôme
 - son genre
 - sa durée de durabilité minimale"""

 def __init__(self,arome,duree):
 # **** Assertions : à compléter suivant les indications de la question 1.a.
 self._arome = arome
 self._duree = duree
 if arome == 'aucun':
 self._genre = 'nature'
 else:
 self._genre = 'aromatise'

 # **** Méthode get_arome(self) à compléter suivant les indications de la question 1.c.

 def get_duree(self):
 return self._duree

 def get_genre(self):
 return self._genre

 def set_duree(self,duree):
 # **** Mutateur de durée
 if duree > 0 :
 self._duree = duree
```

```
**** Mutateur d'arôme set_arome(self,arome) - à compléter suivant les indications de la question 2.

def __set_genre(self,arome):
 if arome == 'aucun':
 self.__genre = 'nature'
 else:
 self.__genre = 'aromatise'
```

1.a. Quelles sont les assertions à prévoir pour vérifier que l'arôme et la durée correspondent bien à des valeurs acceptables? Il faudra aussi expliciter les commentaires qui seront renvoyés.

Pour rappel :

- L'arôme doit prendre comme valeur 'fraise', 'abricot', 'vanille' ou 'aucun'.
- Sa date de durabilité minimale (DDM) est une valeur positive.

### Reponse :

Modification de la méthode `__init__` de la classe Yaourt :

#### Script Python

```
def __init__(self,arome,duree):
 assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
 assert duree >= 0, "La DDM doit être une valeur positive"
 self.__arome = arome
 self.__duree = duree
 if arome == 'aucun':
 self.__genre = 'nature'
 else:
 self.__genre = 'aromatise'
```

1.b. Pour créer un yaourt, on exécutera la commande suivante :

#### Script Python

```
mon_yaourt = Yaourt('fraise',24)
```

Quelle valeur sera affectée à l'attribut `genre` associé à `mon_yaourt` ?

### Reponse :

L'attribut `genre` aura comme valeur 'aromatise' puisque l'arôme du yaourt n'est pas égal à 'aucun'.

1.c. Écrire en Python une fonction `get_arome(self)`, renvoyant l'arôme du yaourt créé.

### Reponse :

Méthode `get_arome` à insérer dans le code de la classe Yaourt :

#### Script Python

```
def get_arome(self):
 return self.__arome
```

1. On appelle mutateur une méthode permettant de modifier un ou plusieurs attributs d'un objet.

Ecrire en Python le mutateur `set_arome(self,arome)` permettant de modifier l'arôme du yaourt.

*On veillera à garder une cohérence entre l'arôme et le genre.*

### Reponse :

Méthode `set_arome` à insérer dans le code de la classe Yaourt :

#### Script Python

```
def set_arome(self, arôme):
 assert arôme in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
 self.__arôme=arôme
 if arôme == 'aucun':
 self.__genre = 'nature'
 else:
 self.__genre = 'aromatise'
```

On peut aussi utiliser la méthode `set_genre` déjà définie pour garder une cohérence entre l'arôme et le genre.:

#### Script Python

```
def set_arome(self, arôme):
 assert arôme in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
 self.__arôme=arôme
 self.__set_genre(arôme)
```

1. On veut créer une pile contenant le stock de yaourts. Pour cela il faut tout d'abord créer une pile vide :

#### Script Python

```
def creer_pile():
 pile = []
 return pile
```

3.a. Créer une fonction `empiler(p, yaourt:Yaourt)` qui renvoie la pile `p` après avoir ajouté un objet de type Yaourt au sommet de la pile.

### Reponse :

Fonction `empiler` (on suppose que le sommet de la pile est la fin de la liste):

#### Script Python

```
def empiler(p, yaourt):
 p.append(yaourt)
```

3.b. Créer une fonction `dépiler(p)` qui renvoie l'objet à dépiler.

### Reponse :

Fonction `dépiler` :

#### Script Python

```
def dépiler(p):
 return p.pop(-1)
```

3.c. Créer une fonction `est_vide(p)` qui renvoie `True` si la pile est vide et `False` sinon.

### Réponse :

Fonction `est_vide` :

#### Script Python

```
def est_vide(p):
 return len(p) == 0
```

ou bien, en plus explicite :

#### Script Python

```
def est_vide(p):
 if len(p) == 0:
 return True
 else:
 return False
```

3.d. Qu'affiche le bloc de commandes ci-dessous ?

#### Script Python

```
mon_yaourt1 = Yaourt('aucun',18)
mon_yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_yaourt1)
empiler(ma_pile, mon_yaourt2)
print(depiler(ma_pile).get_duree())
print(est_vide(ma_pile))
```

### Reponse :

Le sujet d'origine contient un espace entre `mon_yaourt` et `2` à la ligne 5, ce qui fait que l'exécution du code tel quel générera une erreur. Il s'agit cependant sans doute d'une coquille.

Pour le code corrigé donné ci-dessus :

Les deux premières lignes créent deux objets Yaourt. La troisième crée une pile, où les deux Yaourts sont empilés aux deux lignes suivantes. A la ligne 6, la commande dépile le dernier Yaourt empilé, `mon_yaourt2`, et affiche sa DDN, c'est à dire 24. Enfin la dernière ligne affiche `False` puisqu'à ce stade la pile contient encore `mon_yaourt1`, donc n'est pas vide.

#### Script Python

```
class Yaourt:
 """ Classe définissant un yaourt caractérisé par :
 - son arôme
 - son genre
 - sa durée de durabilité minimale"""

 def __init__(self,arome,duree):
 assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
 assert duree >= 0, "La DDM doit être une valeur positive"
 self.__arome = arome
 self.__duree = duree
 if arome == 'aucun':
 self.__genre = 'nature'
 else:
 self.__genre = 'aromatise'

 def get_arome(self):
 return self.__arome

 def get_duree(self):
 return self.__duree

 def get_genre(self):
 return self.__genre

 def set_duree(self,duree):
 # **** Mutateur de durée
 if duree > 0 :
 self.__duree = duree

 def set_arome(self, arome):
 assert arome in ('fraise', 'abricot', 'vanille', 'aucun') , "valeur invalide pour l'arôme"
 self.__arome=arome
 self.__set_genre(arome)

 def __set_genre(self,arome):
 if arome == 'aucun':
 self.__genre = 'nature'
 else:
 self.__genre = 'aromatise'

 def creer_pile():
 pile = []
 return pile

 def empiler(p, yaourt):
 p.append(yaourt)

 def depiler(p):
 return p.pop(-1)

 def est_vide(p):
 return len(p) == 0

mon_yaourt1 = Yaourt('aucun',18)
mon_yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_yaourt1)
empiler(ma_pile, mon_yaourt2)
print(depiler(ma_pile).get_duree())
print(est_vide(ma_pile))
```

24  
False

## 4. 2.4 POO : Exercices BAC - Correction

### 4.1. 2.4.1 Exercice n°1 : Métropole J1 : Ex.5 - 2022



- Laser Game

Les participants à un jeu de LaserGame sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges.

Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe `Joueur` est définie comme suit :

```

1 class Joueur:
2 def __init__(self, pseudo, identifiant, equipe):
3 " Appelle le constructeur et initialise "
4 self.pseudo = pseudo
5 self.equipe = equipe
6 self.id = identifiant
7 self.nb_de_tirs_emis = 0
8 self.liste_id_tirs_recus = []
9 self.est_actif = True
10
11 def tire(self):
12 " Méthode déclenchée par l'appui sur la gâchette "
13 if self.est_actif:
14 self.nb_de_tirs_emis += 1
15
16 def est_determine(self):
17 " Le joueur réalise-t-il un grand nombre de tirs ? "
18 return self.nb_de_tirs_emis > 500 # Un booléen est renvoyé.
19
20 def subit_un_tir(self, id_recu):
21 " Méthode déclenchée par les capteurs de la veste "
22 if self.est_actif:
23 self.est_actif = False
24 self.liste_id_tirs_recus.append(id_recu)

```

**1.** Parmi les instructions suivantes, recopier celle qui permet de déclarer un objet `joueur_1`, instance de la classe `Joueur`, correspondant à un joueur dont le pseudo est `"Sniper"`, dont l'identifiant est `319` et qui est intégré à l'équipe `"A"` :

- **Instruction 1** : `joueur_1 = ["Sniper", 319, "A"]`
- **Instruction 2** : `joueur_1 = new Joueur["Sniper", 319, "A"]`
- **Instruction 3** : `joueur_1 = Joueur("Sniper", 319, "A")`
- **Instruction 4** : `joueur_1 = Joueur{"pseudo":"Sniper", "id":319, "equipe":"A"}`



C'est l'instruction **3** : `joueur_1 = Joueur("Sniper", 319, "A")`

**2.** La méthode `subit_un_tir` réalise les actions suivantes :

Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut `liste_id_tirs_recus` et l'attribut `est_actif` prend la valeur `False` (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.

**2.a.** Écrire la méthode `redevient_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.

## Reponse

### Script Python

```
def redevenir_actif(self):
 if not self.est_actif:
 self.est_actif = True
```

**2.b.** Écrire la méthode `nb_de_tirs_recus` qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut `liste_id_tirs_recus`.

## Reponse

### Script Python

```
def nb_tirs_recus(self):
 return len(self.liste_id_tirs_recus)
```

**3.** Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe `Base`, récupère les données.

La classe `Base` est définie par :

- ses attributs :
  - `equipe` : nom de l'équipe ( str ), par exemple, "A",
  - `liste_des_id_de_l_equipe` qui correspond à la liste ( list ) des identifiants connus des joueurs de l'équipe,
  - `score` : score ( int ) de l'équipe, dont la valeur initiale est 1000 ;
- ses méthodes :
  - `est_un_id_allie` qui détermine si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, en renvoyant un booléen,
  - `decremente_score` qui diminue l'attribut `score` du nombre passé en paramètre,
  - `collecte_information` qui récupère les statistiques d'un participant passé en paramètre (instance de la classe `Joueur`) pour calculer le score de l'équipe.

### Script Python

```
def collecte_information(self, participant):
 if participant.equipe == self.equipe : # test 1
 for id in participant.liste_id_tirs_recus:
 if self.est_un_id_allie(id): # test 2
 self.decremente_score(20)
 else:
 self.decremente_score(10)
```

**3.a.** Indiquer le numéro du test (test 1 ou test 2) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.

## Reponse

C'est le **test 1** qui permet de savoir si le joueur ne s'est pas égaré en terrain adverse.

**3.b.** Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.

**Réponse**

Un *tir ami* fait perdre **20 points** au score.

On souhaite accorder à la base un bonus de 40 points pour chaque joueur particulièrement déterminé (qui réalise un grand nombre de tirs).

**4.** Recopier et compléter, en utilisant les méthodes des classes `Joueur` et `Base`, les 2 lignes de codes suivantes qu'il faut ajouter à la fin de la méthode `collecte_information` :

**Script Python**

```
... # si le participant réalise un grand nombre de tirs
... # le score de la Base augmente de 40
```

**Réponse**

Pour augmenter de  $(40)$ , on peut soustraire  $(-40)$ .

**Script Python**

```
if participant.est_determine(): # si le participant réalise un grand nombre de tirs
 self.decremente_score(-40) # le score de la Base augmente de 40
```

**4.2. 2.4.2 Exercice 2 : D'après 2022, Centres étrangers, J2, Ex. 4****1.0.0**

- La Bataille

Simon souhaite créer en Python le jeu de cartes « la bataille » pour deux joueurs. Les questions qui suivent demandent de reprogrammer quelques fonctions du jeu.

## Règles du jeu de la bataille

### Préparation

- Distribuer toutes les cartes aux deux joueurs.
- Les joueurs ne prennent pas connaissance de leurs cartes et les laissent en tas face cachée devant eux.

### Déroulement

- À chaque tour, chaque joueur dévoile la carte du haut de son tas.
- Le joueur qui présente la carte ayant la plus haute valeur emporte les deux cartes qu'il place sous son tas.
- Les **valeurs des cartes** sont : dans l'ordre de la plus forte à la plus faible : As, Roi, Dame, Valet, 10, 9, 8, 7, 6, 5, 4, 3 et 2 (la plus faible)

Si deux cartes sont de même valeur, il y a « bataille ».

- Chaque joueur pose alors une carte face cachée, suivie d'une carte face visible sur la carte dévoilée précédemment.
- On recommence l'opération s'il y a de nouveau une bataille sinon, le joueur ayant la valeur la plus forte emporte tout le tas.

Lorsque l'un des joueurs possède toutes les cartes du jeu, la partie s'arrête et ce dernier gagne.

Pour cela Simon crée une classe Python `Carte`. Chaque instance de la classe a deux attributs : un pour sa `valeur` et un pour sa `couleur`. Il donne au valet la valeur \11\, à la dame la valeur \12\, au roi la valeur \13\ et à l'as la valeur \14\. La couleur est une chaîne de caractères : "trefle", "carreau", "coeur" ou "pique".

**1.** Simon a écrit la classe Python `Carte` suivante, ayant deux attributs `valeur` et `couleur`, et dont le constructeur prend deux arguments : `val` et `coul`.

**1.a.** Recopier et compléter les ... des lignes 3 et 4 ci-dessous.

```
1 class Carte:
2 def __init__(self, val, coul):
3 valeur = ...
4 ... = coul
```

## Réponse

```
1 class Carte:
2 def __init__(self, val, coul):
3 self.valeur = val
4 self.couleur = coul
```

**1.b.** Parmi les propositions ci-dessous quelle instruction permet de créer l'objet « 7 de cœur » sous le nom `c7` ?

- `c7.__init__(self, 7, "coeur")`
- `c7 = Carte(self, 7, "coeur")`
- `c7 = Carte(7, "coeur")`
- `from Carte import 7, "coeur"`

### Reponse

`c7 = Carte(7, "coeur")` crée une instance de la classe `Carte` de valeur `\(7\)` et de couleur "coeur", puis l'affecte à la variable `c7`.

2. On souhaite créer le jeu de cartes. Pour cela, on écrit une fonction `initialiser` :

- sans paramètre
- qui renvoie une liste de 52 objets de la classe `Carte` représentant les 52 cartes du jeu.

Voici une proposition de code. Recopier et compléter les lignes suivantes pour que la fonction réponde à la demande :

#### Script Python

```
def initialiser():
 jeu = []
 for coul in ["coeur", "carreau", "trefle", "pique"]:
 for val in range(...):
 carte_cree = ...
 jeu.append(carte_cree)
 return jeu
```

### Reponse

#### Script Python

```
def initialiser():
 jeu = []
 for coul in ["coeur", "carreau", "trefle", "pique"]:
 for val in range(2, 15):
 carte_cree = Carte(val, coul)
 jeu.append(carte_cree)
 return jeu
```

3. On rappelle que dans une partie de bataille, les deux joueurs tirent chacun une carte du dessus de leur tas, et celui qui tire la carte la plus forte remporte les deux cartes et les place en dessous de son tas.

Parmi les structures linéaires de données suivantes : Tableau, File, Pile, quelle est celle qui modélise le mieux un tas de cartes dans ce jeu de la bataille ? Justifier votre choix.

### Reponse

On a besoin d'une structure linéaire pour

- extraire une carte à une seule extrémité ;
- ajouter une carte à l'autre extrémité (elle sera donc loin d'être piochée).

FILO : *First In Last Out* ; premier entré, dernier sorti, pour la File.

C'est la **File** qui répond le mieux à la modélisation souhaitée.

4. Écrire une fonction `comparer` qui prend en paramètres deux objets de la classe `Carte` : `carte_1`, `carte_2`. Cette fonction renvoie :

- `\(0\)` si la valeur des deux cartes est identique ;
- `\(1\)` si la carte `carte_1` a une valeur strictement plus forte que celle de `carte_2` ;
- `\(-1\)` si la carte `carte_2` a une valeur strictement plus forte que celle de `carte_1`.

## Reponse

### 🐍 Script Python

```
def comparer(carte_1, carte_2):
 if carte_1.valeur > carte_2.valeur:
 return 1
 elif carte_1.valeur < carte_2.valeur:
 return -1
 else:
 return 0
```

## 5. 2.5 C5 Listes et Piles



### Programme Terminale

Contenus	Capacités attendues	Commentaires
Structures de données, interface et implémentation	Spécifier une structure de données par son interface. Distinguer interface et implémentation. Écrire plusieurs implémentations d'une même structure de données.	L'abstraction des structures de données est introduite après plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deux piles).
Listes, piles, files : structures linéaires. Dictionnaires, index et clé.	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	On distingue les modes FIFO (First In First Out) et LIFO (Last In First Out) des piles et des files.

### 5.1. 2.5.1 Préambule : interface ≠ implémentation ★★★

#### A savoir

Les structures que nous allons voir peuvent s'envisager sous deux aspects :

- le côté utilisateur, qui utilisera une **interface** pour manipuler les données.
- **interface** : Vue "logique" de la structure de données. Elle spécifie la nature des données ainsi que l'ensemble des opérations permises sur la structure.
- le côté concepteur, qui aura choisi une **implémentation** pour construire la structure de données.
- **Implémentation** : Vue "physique" de la structure de données. Il s'agit de la programmation effective des opérations définies dans l'interface, en utilisant des types de données déjà existants.

Nous avons déjà abordé ces deux aspects lors de la découverte de la Programmation Orientée Objet. Le principe d'encapsulation fait que l'utilisateur n'a qu'à connaître l'existence des méthodes disponibles, et non pas le contenu technique de celle-ci. Cela permet notamment de modifier le contenu technique (l'implémentation) sans que les habitudes de l'utilisateur (l'interface) ne soient changées.

### 5.2. 2.5.2 Structures de données linéaires

#### 5.2.1. À chaque donnée sa structure

En informatique comme dans la vie courante, il est conseillé d'adapter sa manière de stocker et de traiter des données en fonction de la nature de celles-ci.

En informatique, pour chaque type de données, pour chaque utilisation prévue, une structure particulière de données se revèlera (peut-être) plus adaptée qu'une autre.

#### DONNÉES LINÉAIRES

Intéressons nous par exemple aux **données linéaires**. Ce sont des données qui ne comportent pas de *hiérarchie* : toutes les données sont de la même nature et ont le même rôle. Par exemple, un relevé mensuel de températures, la liste des élèves d'une classe, un historique d'opérations bancaires...

Ces données sont «plates», n'ont pas de sous-domaines : la structure de **liste** paraît parfaitement adaptée.

Lorsque les données de cette liste sont en fait des couples (comme dans le cas d'une liste de noms/numéros de téléphone), alors la structure la plus adaptée est sans doute celle du **dictionnaire**.

Les listes et les dictionnaires sont donc des exemples de structures de **données linéaires**.

#### DONNÉES NON-LINÉAIRES

Même si ce n'est pas l'objet de ce cours, donnons des exemples de structures adaptées aux données non-linéaires :

Si une liste de courses est subdivisée en "rayon frais / bricolage / papeterie" et que le rayon frais est lui-même séparé en "laitages / viandes / fruits & légumes", alors une structure d'**arbre** sera plus adaptée pour la représenter. Les structures arborescentes seront vues plus tard en Terminale.

Enfin, si nos données à étudier sont les relations sur les réseaux sociaux des élèves d'une classe, alors la structure de **graphe** s'imposera d'elle-même. Cette structure sera elle-aussi étudiée plus tard cette année.

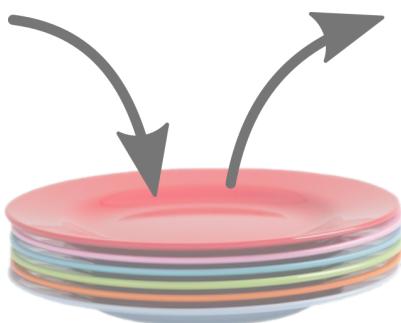
#### 5.2.2. Comment seront traitées ces données linéaires ? Introduction des listes, des piles et des files

La nature des données ne fait pas tout. Il faut aussi s'intéresser à la manière dont on voudra les traiter :

- À quelle position les faire entrer dans notre structure ?
- À quel moment devront-elles en éventuellement en sortir ?
- Veut-on pouvoir accéder rapidement à n'importe quel élément de la structure, ou simplement au premier ? ou au dernier ?

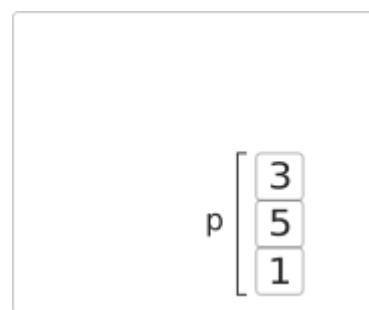
Lorsque ces problématiques d'entrée/sortie n'interviennent pas, la structure «classique» de liste est adaptée. Mais lorsque celle-ci est importante, il convient de différencier la structure de **pile** de celle de **file**.

#### LES PILES (STACK)



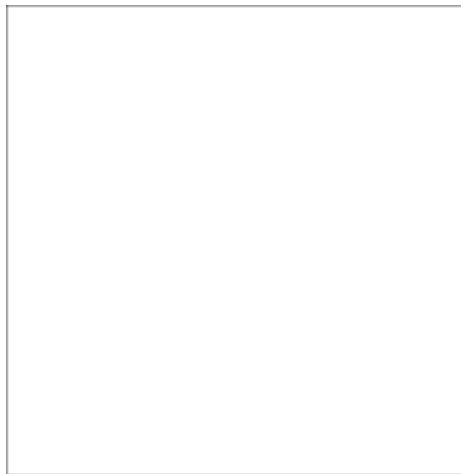
Une structure de **pile** (penser à une pile d'assiette) est associée à la méthode **LIFO** (Last In, First Out) :

- les éléments sont empilés les uns au-dessus des autres,
- et on ne peut toujours dépiler que l'élément du haut de la pile.
- Le dernier élément à être arrivé est donc le premier à être sorti.

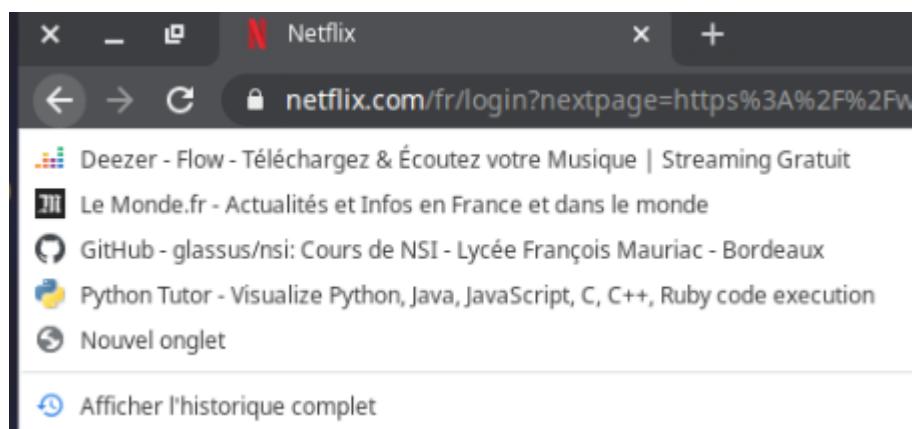


#### Exemples de données stockées sous forme de pile

- lors de l'exécution d'une fonction récursive, le processeur empile successivement les appels à traiter : seule l'instruction du haut de la pile peut être traitée.



- historiques de navigation sur le Web, la liste des pages parcourues est stockée sous forme de pile : la fonction «Back» permet de «dépiler» peu à peu les pages précédemment parcourues :



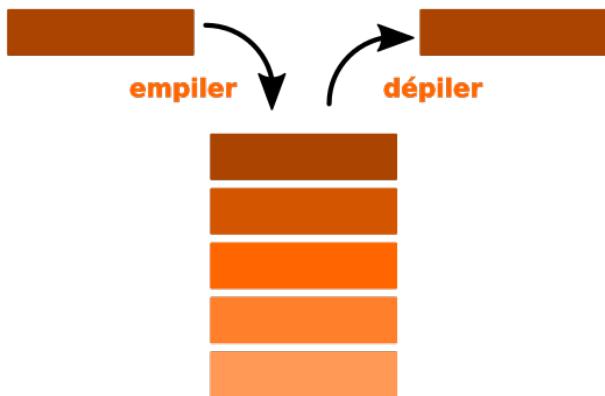
- historiques d'annulation d'instructions (Ctrl+Z)

### Interface

On dispose (ou souhaite disposer) sur une pile des méthodes/primitives suivantes:

- déterminer si la pile est vide (`est_vide`, `is_empty`)
- empiler** un nouvel élément au sommet de la pile (`empiler`, `push`)
- dépiler** l'élément du sommet de la pile (`dépiler`, `pop`) et le renvoyer

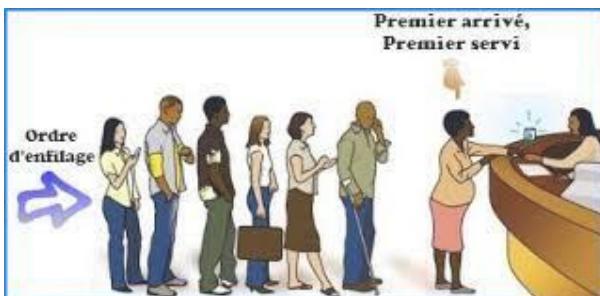
Ces opérations doivent être réalisées **en temps constant**, soit en  $\mathcal{O}(1)$ .



### 5.2.3. Les files

#### Reconnaître

Une **file** (queue) est une structure de données *linéaire* contenant des éléments généralement *homogènes* fondée sur le principe «premier arrivé, premier sorti» (en anglais **FIFO** : Fast In, First Out).



#### Exemples de situations utilisant une file:

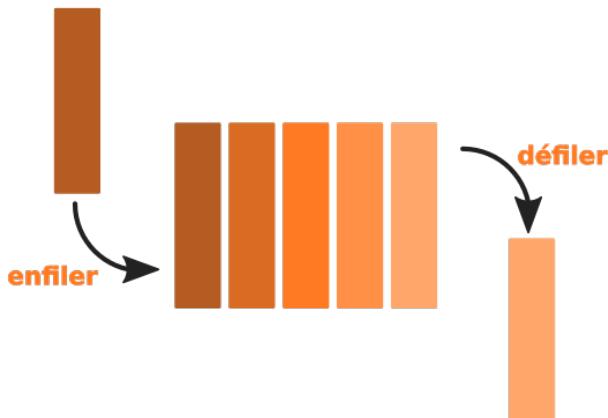
- file d'attente : documents soumis à impression, élèves à la cantine...
- gestion des processus (plus tard...)
- parcours en largeur d'un arbre/graphe (plus tard...)

### Interface :

On dispose (ou souhaite disposer) sur une file des méthodes/primitives suivantes:

- déterminer si la file est vide (`is_empty`)
- **enfiler** (ajouter) un nouvel élément dans la file (`enqueue`)
- **défiler** l'élément de tête de la file (`dequeue`) et le renvoyer

Ces opérations doivent être réalisées **en temps constant**, soit en  $\mathcal{O}(1)$ .

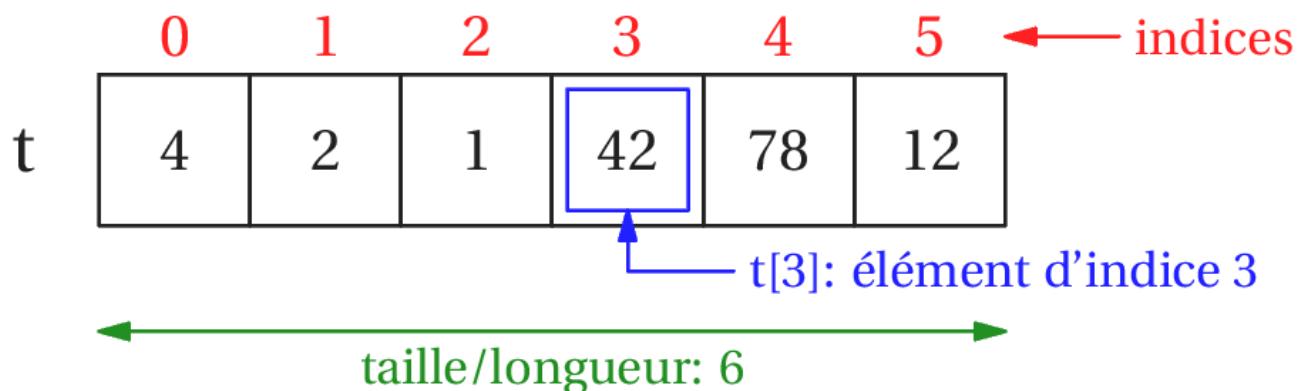


### 5.3. 2.5.3 Listes chaînées

#### 5.3.1. Retour sur les tableaux

Dans une structure de **tableau** (**array** en anglais), les données (ou une référence vers les données) sont organisées de manière séquentielle en mémoire, où chaque élément (ou référence) est de même type. On peut donc calculer la position de l'élément (ou de la référence) en mémoire en fonction de son numéro d'ordre dans la séquence.

En règle générale, la taille du tableau est connue à la déclaration. Dans ce cas, on ne peut pas ajouter d'élément au delà de la dernière case prévue (ce qui n'est pas le cas en python).



Quelques propriétés des tableaux :

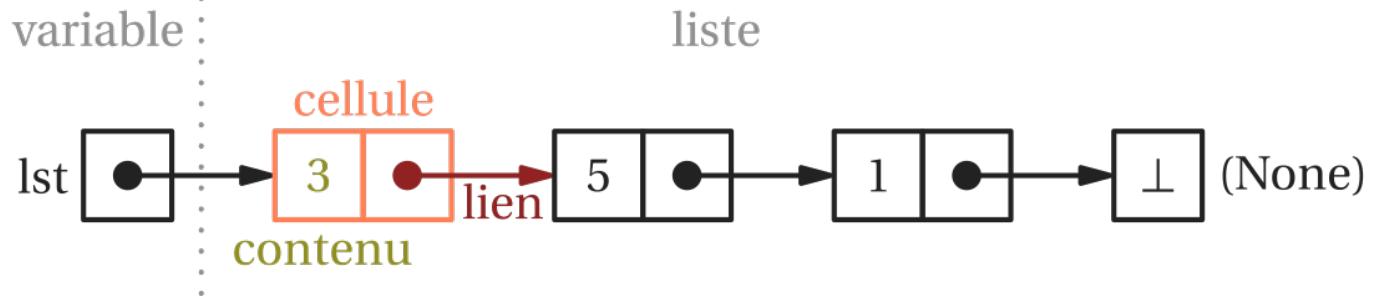
- création d'un tableau de taille donnée
- accès à un élément à partir de son indice en temps constant ( $\mathcal{O}(1)$ )
- modification d'un élément à partir de son indice en temps constant ( $\mathcal{O}(1)$ )

En revanche, l'insertion d'un élément dans le tableau impose de décaler tous les éléments d'un indice, elle se fait donc en temps linéaire, soit  $\mathcal{O}(n)\dots$

### 5.3.2. Liste chaînée

Avec une structure de **liste (chaînée)**, on représente à nouveau une séquence d'éléments (à nouveau le plus souvent homogènes), mais les données ne sont pas nécessairement séquentielles en mémoire. On dispose en revanche d'un moyen permettant de passer d'un élément au suivant, d'où le terme *chaîné*.

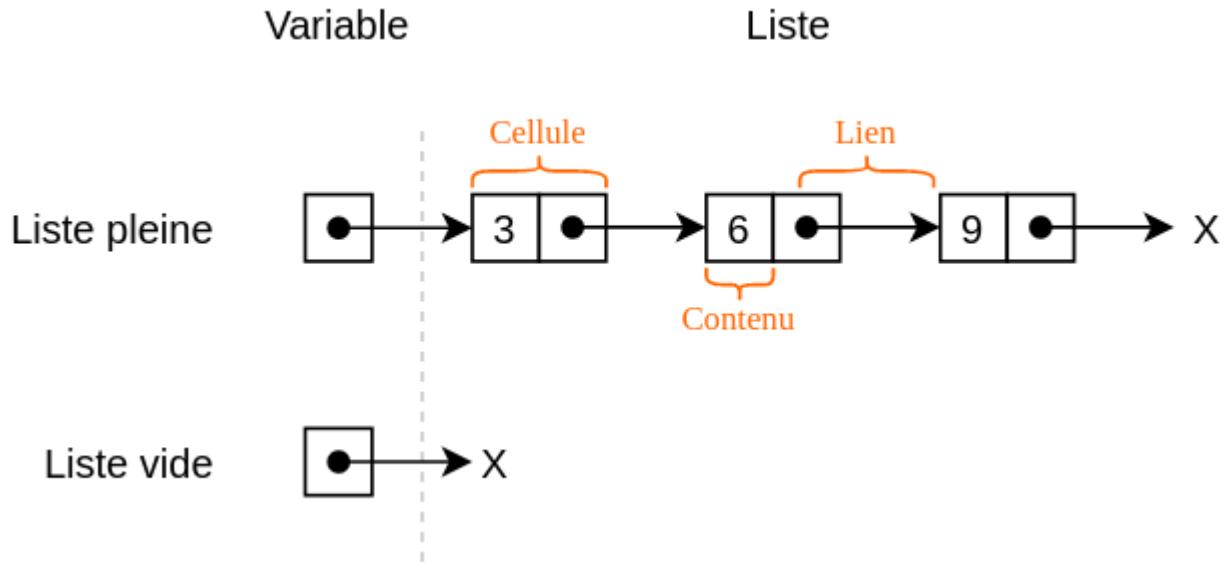
Chaque élément est donc stocké dans un bloc mémoire **avec** une deuxième information: l'adresse de l'élément suivant. On parle de **maillon ou cellule** (ou encore **node**) pour désigner ces blocs.



On peut généralement ajouter de nouveaux éléments pour augmenter la taille de la structure dynamiquement.

Et finalement, une liste chaînée est soit vide, soit n'est qu'un lien vers une cellule, qui contient une valeur et un lien vers une cellule, qui est soit vide, soit n'est qu'un lien vers une cellule, qui contient une valeur et un lien vers une cellule, qui...

On parle donc de **définition récursive** d'une liste chaînée.

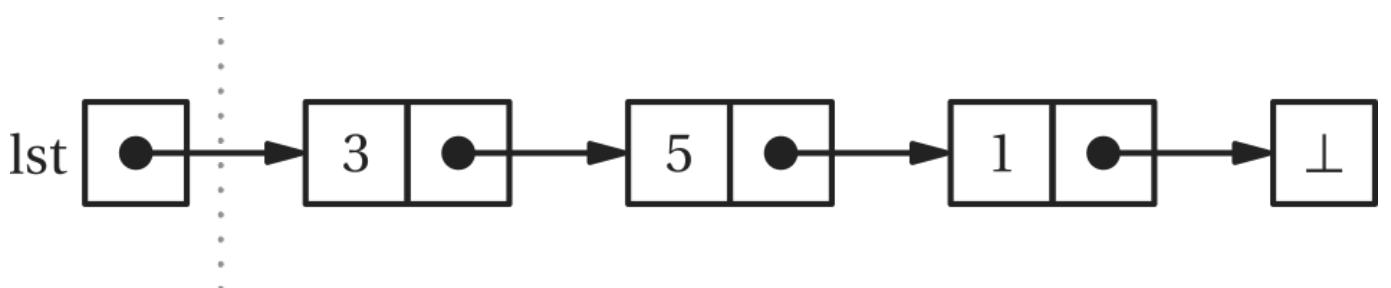


### Interface :

On dispose (ou souhaite disposer) sur une liste chaînée des méthodes/primitives suivantes:

- construire une liste vide, souvent nommée `nil`
- déterminer si la liste est vide (`est_vide`, `is_empty`)
- insérer un élément en tête de liste (`insert`)
- récupérer l'élément en tête de liste (`tete`, `head`)
- récupérer la liste privée de son premier élément, appelée la **queue** (`queue`, `tail`)

Ces opérations doivent être réalisées **en temps constant**, soit en  $\backslash(O(1))$ .



**Remarque :** Accès à un élément

Pour accéder à un élément quelconque, il faut parcourir toute la liste jusqu'à trouver l'élément: le temps d'accès est linéaire, c'est-à-dire proportionnel à la taille de la liste ( $\backslash(O(n))$ ) et donc non constant.

### IMPLÉMENTATION CHOISIE :

- Une liste est caractérisée par un ensemble de cellules.
- Le lien (on dira souvent le «pointeur») de la variable est un lien vers la première cellule, qui renverra elle-même sur la deuxième, etc.
- Chaque cellule contient donc une valeur et un lien vers la cellule suivante.
- Une liste peut être vide (la liste vide est notée `x` ou bien `None` sur les schémas)

Une conséquence de cette implémentation sous forme de liste chaînée est la non-constance du temps d'accès à un élément de liste : pour accéder au 3ème élément, il faut obligatoirement passer par les deux précédents.

#### EXEMPLE D'IMPLÉMENTATION MINIMALE D'UNE LISTE CHAÎNÉE

**Exemple : implémentation d'une liste chaînée en POO ❤**

##### 🐍 Script Python

```
class Cellule :
 def __init__(self, contenu, suivante):
 self.contenu = contenu
 self.suivante = suivante
```

Cette implémentation rudimentaire permet bien la création d'une liste :

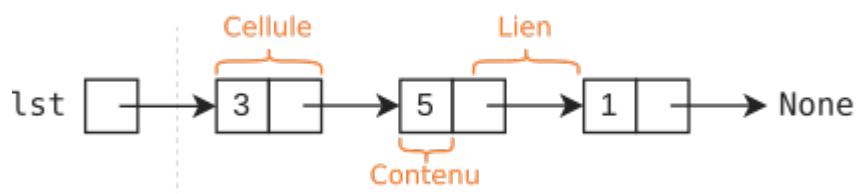
##### 🐍 Script Python

```
lst = Cellule(3, Cellule(5, Cellule(1,None)))
```

La liste créée est donc :



Mais plus précisément, on a :



##### 📝 Exercice

Retrouvez comment accéder aux éléments 3, 5 et 1.

On pourra remarquer que l'interface proposée à l'utilisateur n'est pas des plus pratiques...

#### UN EXEMPLE D'INTERFACE POUR LES LISTES

Imaginons que nous possédons une interface offrant les fonctionnalités suivantes :

- `ListeC()` : crée une liste vide.
- `est_vide()` : indique si la liste est vide.
- `ajoute_tete()` : insère un élément en tête de liste.
- `renvoie_tete()` : renvoie la valeur de l'élément en tête de liste ET le supprime de la liste.

## Exercice

On considère l'enchaînement d'opérations ci-dessous. Écrire à chaque étape l'état de la liste `lst` et la valeur éventuellement renvoyée.

### Script Python

```
lst = ListeC()
lst.ajoute_tete(3)
lst.ajoute_tete(5)
lst.ajoute_tete(1)
lst.revoie_tete()
lst.est_vide()
lst.ajoute_tete(2)
lst.revoie_tete()
lst.revoie_tete()
lst.revoie_tete()
lst.est_vide()
```

## Implémentation

On ne définit la méthode spéciale `__str__` uniquement pour vérifier et afficher de façon pratique la liste.

### Script Python

```
class Cellule:
 def __init__(self, contenu=None, suivante=None):
 self.contenu = contenu
 self.suivante = suivante

 def __str__(self):
 if self.suivante == None :
 return f"{self.contenu}"
 else :
 return f"{self.contenu} -> {str(self.suivante)}"

class ListeC:
 def __init__(self):
 self.tete = None # Liste vide

 def ajoute_tete(self,valeur):
 nouvelle_cellule=Cellule(valeur,self.tete)
 self.tete=nouvelle_cellule

 def est_vide(self):
 return self.tete == None

 def revvoie_tete(self):
 t=self.tete.contenu
 self.tete=self.tete.queue()
 return t

 def queue(self):
 return self.tete.suivante

 def __str__(self):
 c=self
 l = []
 while not c.est_vide():
 l.append(str(c.revoie_tete()))
 l.reverse()
 for v in l:
 self.ajoute_tete(v)
 l.reverse()
 return ' -> '.join(l)
```

## Vérification de la question 2

### Script Python

```
lst = ListeC()
print(lst)
lst.ajoute_tete(3)
print(lst)
lst.ajoute_tete(5)
print(lst)
lst.ajoute_tete(1)
```

```

print(lst)
print("valeur renvoyé : ",lst.revoie_tete())

print("Est vide :",lst.est_vide())

lst.ajoute_tete(2)

print(lst)
print("valeur renvoyé : ",lst.revoie_tete())
print(lst)
print("valeur renvoyé : ",lst.revoie_tete())
print(lst)
print("valeur renvoyé : ",lst.revoie_tete())
print(lst)
print("Est vide : ",lst.est_vide())

```

### Texte

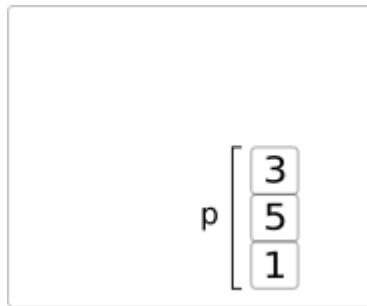
```

3
5 -> 3
1 -> 5 -> 3
valeur renvoyé : 1
Est vide : False
2 -> 5 -> 3
valeur renvoyé : 2
5 -> 3
valeur renvoyé : 5
3
valeur renvoyé : 3

Est vide : True

```

## 5.4. 2.5.4 Les Piles



Comme expliqué précédemment, une pile travaille en mode LIFO (Last In First Out). Pour être utilisée, l'interface d'une pile doit permettre au moins :

- la création d'une pile vide
- l'ajout d'un élément dans la pile (qui sera forcément au dessus). On dira qu'on **empile**.
- le retrait d'un élément de la pile (qui sera forcément celui du dessus) et le renvoi de sa valeur. On dira qu'on **dépile**.

#### 5.4.1. Utilisation d'une interface de pile

##### Exercice

On considère l'enchaînement d'opérations ci-dessous.

Écrire à chaque étape l'état de la pile `p` et la valeur éventuellement renvoyée.

Bien comprendre que la classe `Pile()` et ses méthodes n'existent pas vraiment. Nous *jouons* avec son interface.

##### Script Python

```
p = Pile() #
p.empile(3) # p=
p.empile(5) # p=
p.est_vide() #
p.empile(1) # p=
p.depile() # p= valeur renvoyée :
p.depile() # p= valeur renvoyée :
p.empile(9) # p=
p.depile() # p= valeur renvoyée :
p.depile() # p= valeur renvoyée :
p.est_vide() #
```

#### 5.4.2. Implémentation(s) d'une pile

##### Interface :

L'objectif est de créer une classe `Pile`. L'instruction `Pile()` créera une pile vide. Chaque objet `Pile` disposera des méthodes suivantes :

- `est_vide()` : indique si la pile est vide.
- `empile()` : insère un élément en haut de la pile.
- `depile()` : renvoie la valeur de l'élément en haut de la pile ET le supprime de la pile.
- `__str__()` : permet d'afficher la pile sous forme agréable (par ex : `|3|6|2|5|`) par `print()`

À L'AIDE DU TYPE `LIST` DE PYTHON

##### Exercice

Créer la classe ci-dessus. Le type `List` de Python est parfaitement adapté. Des renseignements intéressants à son sujet peuvent être trouvés [ici](#).

Dans les deux exercices qui suivent, quelle que soit l'implémentation de la pile, le code suivant :

##### Script Python

```
p = Pile()
p.empiler(1)
p.empiler(2)
print(p.depiler())
p.empiler(3)
while not p.est_vide():
 print(p.depiler())
```

doit afficher :

##### Script Python

2  
3  
1

### Script Python

```
class Pile:
 def __init__(self):
 self.data = ...

 def est_vide(self):
 pass

 def empile(self,x):
 pass

 def depile(self):
 if self.est_vide() == True :
 raise IndexError("Vous avez essayé de dépiler une pile vide !")
 else :
 pass

 def __str__(self): # Hors-Programme : pour afficher
 s = "|"
 # convenablement la pile avec print(p)
 for k in self.data :
 s = s + str(k) + "|"
 return s

 def __repr__(self): # Hors-Programme : pour afficher
 s = "|"
 # convenablement la pile avec p
 for k in self.data :
 s = s + str(k) + "|"
 return s
```

### Texte

#A tester

## Correction et commentaire(s)

### Script Python

```
class Pile:
 def __init__(self):
 self.data = []

 def est_vide(self):
 return len(self.data) == 0

 def empile(self,x):
 self.data.append(x)

 def depile(self):
 if self.est_vide() == True :
 raise IndexError('Vous avez essayé de dépiler une pile vide !')
 else :
 return self.data.pop()

 def __str__(self): # Hors-Programme : pour afficher
 s = '|'
 # convenablement la pile avec print(p)
 for k in self.data :
 s = s + str(k) + '|'
 return s

 def __repr__(self): # Hors-Programme : pour afficher
 s = '|'
 # convenablement la pile avec p
 for k in self.data :
 s = s + str(k) + '|'
 return s
```

## A connaître (possibilité d'exercice à compléter lors de l'épreuve pratique)

À L'AIDE D'UNE LISTE CHAÎNÉE ET DE LA CLASSE `CELLULE` CRÉÉE AU 2.3

Au **III.** nous avons créé la classe `Cellule` :

#### Script Python

```
class Cellule :
 def __init__(self, contenu, suivante):
 self.contenu = contenu
 self.suivante = suivante
```

#### Exercice

A l'aide cette classe, re-créer une classe `Pile` disposant exactement de la même interface que dans l'exercice précédent.

```
'''python class Pile: def __init__(self): pass
```

#### Texte

```
def est_vide(self):
 pass

def empile(self, x):
 self.data = Cellule(..., ...)

def depile(self):
 v = ... #on récupère la valeur à renvoyer
 self.data = ... # on supprime la 1ère cellule
 return ...

def __str__(self): # Hors-Programme : pour afficher
 s = "|"
 c = self.data
 while c != None :
 s += str(c.contenu)+"|"
 c = c.suivante
 return s
```

'''

#### Texte

```
#A tester
```

## Correction et commentaire(s)

### Script Python

```
class Pile:
 def __init__(self):
 self.data = None

 def est_vide(self):
 return self.data == None

 def empile(self, x):
 self.data = Cellule(x, self.data)

 def depile(self):
 v = self.data.contenu # on récupère la valeur à renvoyer
 self.data = self.data.suivante # on supprime la 1ère cellule
 return v

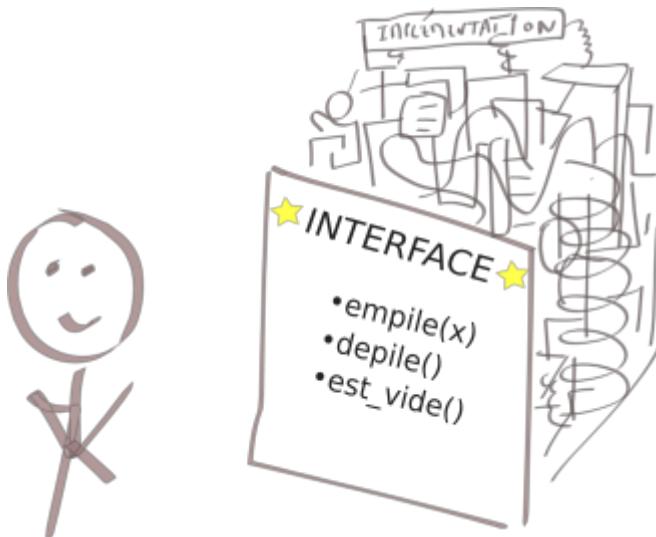
 def __str__(self):
 s = '|'
 c = self.data
 while c != None:
 s += str(c.contenu) + '|'
 c = c.suivante
 return s
```

### Commentaires

A connaître (possibilité d'exercice à compléter lors de l'épreuve pratique)

### A retenir :

Pour l'utilisateur, les interfaces du 3.2.1 et 3.2.2 sont strictement identiques. Il ne peut pas savoir, en les utilisant, l'implémentation qui est derrière.



### 5.4.3. Application des piles

À l'aide de deux variables `adresses` et `adresse_courante`, et de la classe `Pile` créée plus haut, simulez une gestion de l'historique de navigation internet.

Seules deux fonctions `go_to(nouvelle_adresse)` et `back()` sont à créer.

 Exercice

Compléter le code suivant

 Script Python

```
adresses = Pile()
adresse_courante = ""

def go_to(nouvelle_adresse) :
 global adresse_courante
 adresses.empile(...)
 adresse_courante = ...

def back():
 global adresse_courante
 ...
 ...
```

**Exemple d'utilisation :**

 Script Python

```
go_to("google.fr")
go_to("lemonde.fr")
go_to("blabla.fr")
```

 Script Python

```
back()
adresse_courante
```

 Correction et commentaires

 Script Python

```
adresses = Pile()
adresse_courante = ""

def go_to(nouvelle_adresse) :
 global adresse_courante
 adresses.empile(nouvelle_adresse)
 adresse_courante = nouvelle_adresse

def back():
 global adresse_courante
 adresses.depile()
 adresse_courante = adresses.depile()
```

## 5.5. 2.5.5 Les Files



Comme expliqué précédemment, une file travaille en mode FIFO (First In First Out). Pour être utilisée, une file doit permettre à minima :

- la création d'une file vide
- l'ajout d'un élément dans la file (qui sera forcément **au dessous**). On dira qu'on **enfile**.
- le retrait d'un élément de la file (qui sera forcément celui **du dessus**) et le renvoi de sa valeur. On dira qu'on **défile**.

### 5.5.1. Utilisation d'une interface de file

#### Exercice

On considère l'enchaînement d'opérations ci-dessous. Écrire à chaque étape l'état de la file `f` et la valeur éventuellement renvoyée.

#### Script Python

```
f = File()
f.enfile(3) # f =
f.enfile(5) # f =
f.est_vide() #
f.enfile(1) # f =
f.defile() # val renvoyée : , f =
f.defile() # val renvoyée : , f =
f.enfile(9) # f =
f.defile() # val renvoyée : , f =
f.defile()# val renvoyée : , f =
f.est_vide() # True
```

### 5.5.2. Implémentation d'une file

#### Interface :

L'objectif est de créer une classe `File`, disposant des méthodes suivantes :

- `File()` : crée une file vide.
- `est_vide()` : indique si la file est vide.
- `enfile()` : insère un élément en bas de la file.
- `defile()` : renvoie la valeur de l'élément en haut de la file ET le supprime de la file.
- `__str__()` : permet d'afficher la file sous forme agréable (par ex : `|3|6|2|5|`) par `print()`

### Test de l'implémentation

Dans les trois exercices qui suivent, quelle que soit l'implémentation de la file, le code suivant:

#### Script Python

```
f = File()
f.enfiler(1)
f.enfiler(2)
print(f.defiler())
f.enfiler(3)
while not f.est_vide():
 print(f.defiler())
```

doit afficher :

#### Script Python

```
1
2
3
```

### Exercice

Créer la classe ci-dessus. Là encore, le type «list» de Python est peut être utilisé, voir [ici](#). Néanmoins quelques remarques seront à apporter.

#### Script Python

```
class File:
 def __init__(self):
 pass

 def est_vide(self):
 pass

 def enfile(self,x):
 pass

 def defile(self):
 if self.est_vide() == True :
 raise IndexError("Vous avez essayé de défiler une file vide !")
 else :

 def __str__(self): # Hors-Programme : pour afficher
 s = "|"
 # convenablement la file avec print(p)
 for k in self.data :
 s = s + str(k) + "|"
 return s
```

## Correction et commentaire(s)

### Script Python

```
class File:
 def __init__(self):
 self.data = []

 def est_vide(self):
 return len(self.data) == 0

 def enfile(self,x):
 self.data.append(x)

 def defile(self):
 if self.est_vide() == True :
 raise IndexError("Vous avez essayé de défiler une file vide !")
 else :
 return self.data.pop(0)

 def __str__(self): # Hors-Programme : pour afficher
 s = "|"
 for k in self.data :
 s = s + str(k) + "|"
 return s
```

#### Remarque :

Notre implémentation répond parfaitement à l'interface qui était demandée. Mais si le «cahier des charges» obligeait à ce que les opérations `enfile()` et `defile()` aient lieu en temps constant (en  $O(1)$ ), notre implémentation ne conviendrait pas.

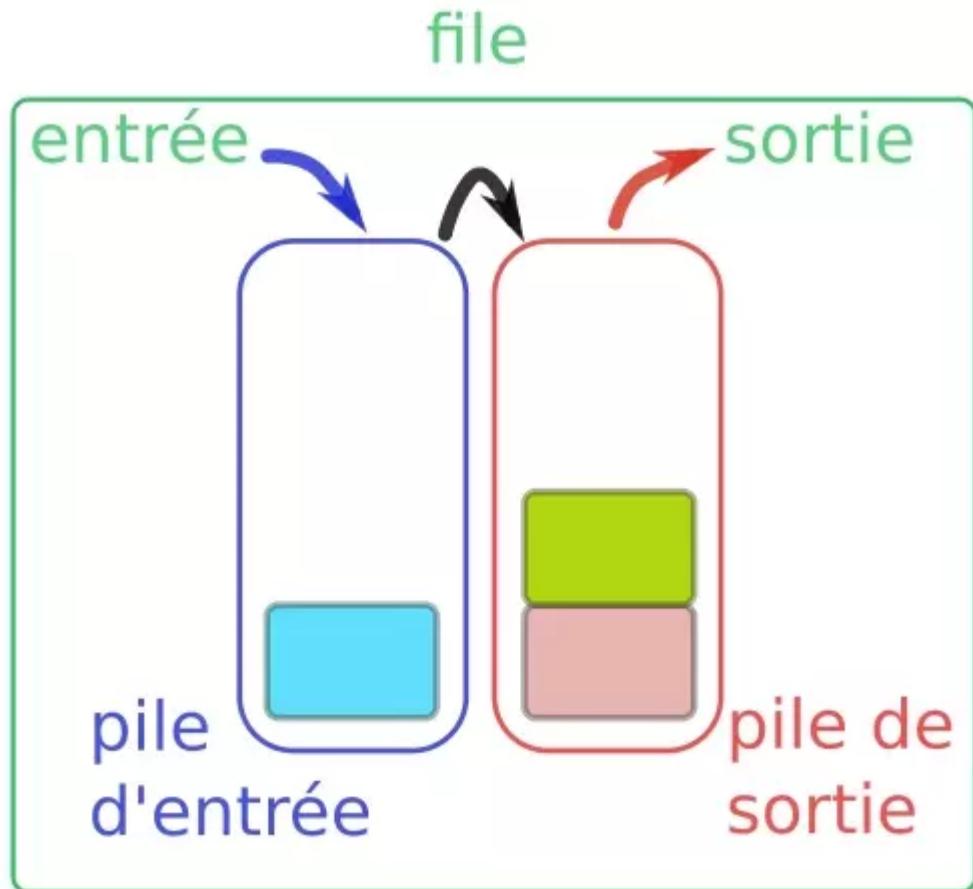
En cause : notre méthode `defile()` agit en temps linéaire ( $O(n)$ ) et non pas en temps constant. L'utilisation de la structure de «liste» de Python (les *tableaux dynamiques*) provoque, lors de l'instruction `self.data.pop(0)` un redimensionnement de la liste, qui voit disparaître son premier élément. Chaque élément doit être recopié dans la case qui précède, avant de supprimer la dernière case. Ceci nous coûte un temps linéaire.

#### 5.5.3. Implémentation d'une file avec deux piles

Comment créer une file avec 2 piles ?

L'idée est la suivante : on crée une pile d'entrée et une pile de sortie.

- quand on veut enfiler, on empile sur la pile d'entrée.
- quand on veut défiler, on dépile sur la pile de sortie.
- si celle-ci est vide, on dépile entièrement la pile d'entrée dans la pile de sortie.



### Script Python

```
il est impératif de comprendre qu'on peut choisir l'implémentation
de la classe Pile qu'on préfère parmi les deux traitées plus haut.
Comme elles ont la MÊME INTERFACE et qu'on ne va se servir que
de cette interface, leur mécanisme interne n'a aucune influence
sur le code de la classe File que nous ferons ensuite.

au hasard, on choisit celle avec la liste chaînée :

class Pile:
 def __init__(self):
 self.data = None

 def est_vide(self):
 return self.data == None

 def empile(self, x):
 self.data = Cellule(x, self.data)

 def depile(self):
 v = self.data.contenu #on récupère la valeur à renvoyer
 self.data = self.data.suivante # on supprime la première cellule
 return v

 def __str__(self):
 s = "|"
 c = self.data
 while c != None :
 s += str(c.contenu)+"|"
 c = c.suivante
 return s

il ne faut pas oublier de remettre la classe Cellule qui intervient
dans notre classe Pile :

class Cellule :
 def __init__(self, contenu, suivante):
```

```
self.contenu = contenu
self.suivante = suivante
```

### Script Python

```
class File:
 def __init__(self):
 self.entree = Pile()
 self.sortie = Pile()

 def est_vide(self):
 return self.entree.est_vide() and self.sortie.est_vide()

 def enfile(self,x):
 self.entree.empile(x)

 def defile(self):
 if self.est_vide():
 raise IndexError("File vide !")

 if self.sortie.est_vide() == True :
 while self.entree.est_vide() == False :
 self.sortie.empile(self.entree.depile())

 return self.sortie.depile()
```

### Script Python

```
f = File()
f.enfile(5)
f.enfile(8)

f.defile()
```

## 5.6. 2.5.6 Sujet Epreuve Pratique

### Sujet 13 :

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

#### Script Python

```
class Maillon :
 def __init__(self,v) :
 self.valeur = v
 self.suivant = None
```

Compléter la classe `File` suivante où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

#### Script Python

```
class File:
 def __init__(self):
 self.tete = None
 self.queue = None

 def est_vide(self):
 return self.queue is None

 def affiche(self):
 maillon = self.tete
 print('--- sens de la file ---')
 while maillon is not None:
 print(maillon.valeur, end=' ')
 maillon = maillon.suivant
 print()

 def enfile(self, element):
 nouveau_maillon = ...
 if self.est_vide():
 self.tete = ...
 else:
 self.queue.suivant = ...
 self... = nouveau_maillon

 def defile(self):
 if ...:
 resultat = ...
 self.tete = ...
 return ...
```

## 6. 2.6 Thème 1 - Structure de données

BAC

# Listes, Piles et Files

### 6.1. 2.6.1 D'après 2022, Métropole, J1, Ex. 1 - Vérification syntaxique de parenthèses ou de balises



#### D'après 2022, Métropole, J1, Ex. 1

##### 6.1.1. Partie A : Expression correctement parenthésée

On veut déterminer si une expression arithmétique est correctement parenthésée. À chaque parenthèse fermante ")" correspond une parenthèse précédemment ouverte "(".

#### Exemples

- L'expression arithmétique "(2 + 3) × (18/(4 + 2))" est correctement parenthésée.
- L'expression arithmétique "(2 + 3) × (18/(4 + 2" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.

#### Expression arithmétique

"(2 + 3) × (18/(4 + 2))"

#### Structure de données

()())()

1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

#### Réponse

*Le premier à être retiré était le premier à être ajouté, donc cela correspond à une file.*

Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")".

### Exemple

On considère l'expression simplifiée A : "())()"

Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0.

Le parenthésage est correct.

**2.** Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.

- Expression simplifiée B : "((00)"
- Expression simplifiée C : "((0))("

### Réponse

- Expression simplifiée B : **1, 2, 3, 2, 3, 2**
- Expression simplifiée C : **1, 2, 1, 0, -1, 0**

**3.** L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse.

L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse.

Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```

1 def parenthesage_correct(expression):
2 """ fonction renvoyant True si l'expression arithmétique
3 simplifiée (str) est correctement parenthésée, False sinon.
4 Condition: expression ne contient que
5 des parenthèses ouvrantes et fermantes
6 """
7 controleur = 0
8 for parenthese in expression: # pour chaque parenthèse
9 if parenthese == '(':
10 controleur = controleur + 1
11 else: # parenthese == ')'
12 controleur = controleur - 1
13 if controleur ... : # test 1 (à recopier et compléter)
14 # parenthèse fermante sans parenthèse ouvrante
15 return False
16 return controleur ... # test 2 (à recopier et compléter)
17 # test 2 est un booléen renvoyé
18 # True : le parenthésage est correct
19 # False : parenthèse(s) fermante(s) manquante(s)

```

### Réponse

- ligne 13: `(controleur < 0)`
- ligne 16: `(controleur == 0)`

Les parenthèses sont inutiles.

### 6.1.2. Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes `<nom>` et fermantes `</nom>`) et les expressions parenthésées : par exemple, l'expression HTML simplifiée : "`<p><strong><em></em></strong></p>`" est correctement balisée.

On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme `<br>` ou `<img ...>`.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

- On parcourt successivement chaque balise de l'expression :
- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilerée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

## Exemple

État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée "`<p><em></p></em>`" qui n'est pas correctement balisée.

État de la pile lors du déroulement de l'algorithme

Départ      Étape 1      Étape 2      Étape 3

### Parcours de l'expression

```
"<p></p>"
↑
```

```
flowchart TD
A["

=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise <p> ouvrante, on empile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise ouvrante, on empile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise </p> fermante, on dépile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

`<em>` et `</p>` ne correspondent pas !

Le balisage est incorrect.

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

4.a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression "`<p><em></em></p>`" (balisage correct).

## Réponse

Départ      Étape 1      Étape 2      Étape 3      Étape 4      Fin

### Parcours de l'expression

```
"<p></p>"
↑
```

```
flowchart TD
A["

=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise <p> ouvrante, on empile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise ouvrante, on empile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

### Parcours de l'expression

```
"<p></p>"
↑ Balise fermante, on dépile
```

```
flowchart TD
A["

<p>
=====
Pile"]
```

<em> et </em> se correspondent.

### Parcours de l'expression

```
"<p></p>"
↑ Balise </p> fermante, on dépile
```

```
flowchart TD
A["

=====
Pile"]
```

<p> et </p> se correspondent.

### Parcours de l'expression

```
"<p></p>"
↑
```

```
flowchart TD
A["

=====
Pile"]
```

La pile est vide. Le balisage est correct.

- 4.b.** Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

**Réponse**

Il suffirait de vérifier que la pile est **vide**.

**5.** Une expression HTML correctement balisée contient 12 balises.

Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

**Réponse**

**6 éléments** au maximum seront empilés, dans le cas où 12 balises HTML sont imbriquées. 6 ouvrantes qui seront empilées, puis les 6 fermantes.

## 6.2. 2.6.2 D'après 2022, Métropole, J2, Ex. 2 - Jeu de la poussette

**D'après 2022, Métropole, J2, Ex. 2**

title: Jeu de la poussette author: Nicolas Revéret

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

- Si la pile contient du haut vers le bas le triplet 1 ; 0 ; 3, on supprime le 0, car 1 et 3 sont tous les deux impairs.
- Si la pile contient du haut vers le bas le triplet 1 ; 0 ; 8, la pile reste inchangée, car 1 et 8 n'ont pas la même parité.

On parcourt la pile ainsi de haut en bas et on procède aux réductions.

Arrivé en bas de la pile, **on recommence la réduction** en repartant du sommet de la pile jusqu'à ce que la pile ne soit plus réductible.

Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

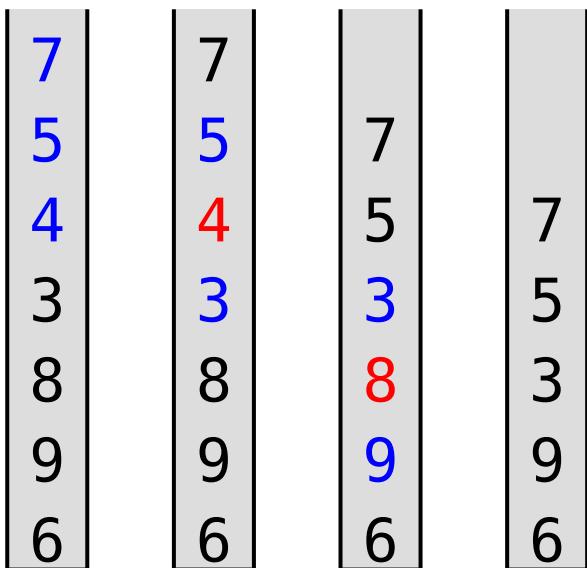
Voici un exemple détaillé de déroulement d'une partie.

Premier parcours de la pile

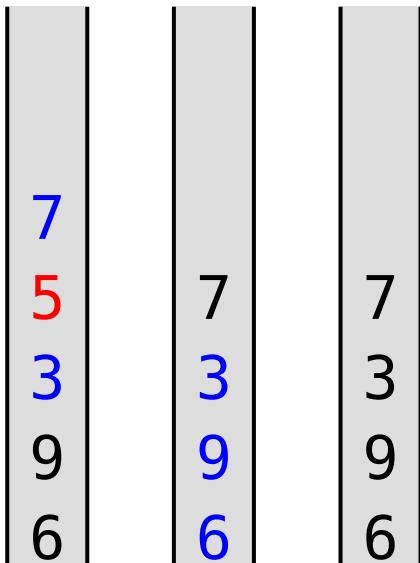
Deuxième parcours

Troisième parcours

Quatrième parcours



- La première comparaison (7, 5 et 4) laisse la pile inchangée.
- On retire le 4 lors de la deuxième itération.
- On retire le 8 lors de la troisième.
- Il ne reste plus que deux valeurs en bas de la pile (9 et 6) : on a fini le premier parcours.



- On recommence à partir du haut de la pile : on retire le 5.
- Le triplet suivant (3, 9 et 6) n'entraîne pas de suppression.
- Il ne reste plus que deux valeurs à étudier (9 et 6) : on a terminé le deuxième parcours.



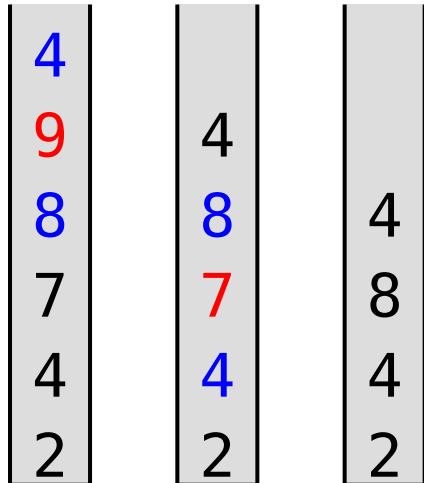
**1.a.** Donner les différentes étapes de réduction de la pile suivante :

4
9
8
7
4
2

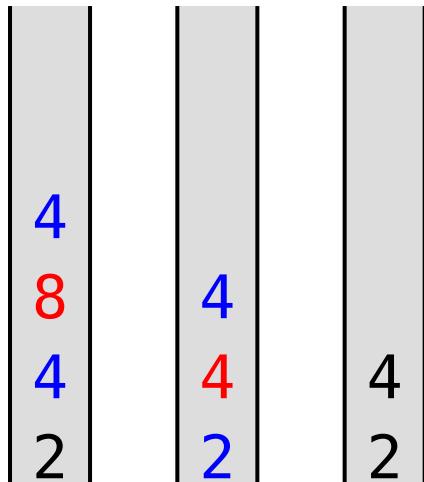
**Réponse**

Il s'agit d'une pile gagnante :

Premier parcours      Second parcours



- Lors de la première comparaison, on retire le 9.
- Lors de la seconde comparaison, on retire le 7.
- Il ne reste plus que deux valeurs en bas de la pile (4 et 2) : on a fini le premier parcours.



- Lors de la première comparaison, on retire le 8.
- Lors de la seconde comparaison, on retire le 4.
- Il ne reste plus que deux valeurs en bas de la pile (4 et 2) : la pile est gagnante.

**1.b.** Parmi les piles proposées ci-dessous, donner celle qui est gagnante.

Pile A      Pile B      Pile C

5
4
5
4
2
1

4
5
4
9
2
0

3
4
8
7
6
1

**Réponse**

Seule la pile B est gagnante. On fournit ci dessous les piles en début et fin de partie :



Pile A      Pile B      Pile C

5		
4		
5	5	
4	4	
2	2	
1	1	

4		
5		
4		
9		
2	4	
0	0	

3		
4		
8	3	
7	4	
6	6	
1	1	

L'interface d'une pile est proposée ci-dessous :

- `creer_pile_vide()` renvoie une pile vide,
- `est_vide(p)` renvoie `True` si `p` est vide, `False` sinon,
- `empiler(p, element)` ajoute `element` au sommet de `p`,
- `depiler(p)` retire l'élément au sommet de `p` et le renvoie,
- `sommet(p)` renvoie l'élément au sommet de `p` sans le retirer de `p`,
- `taille(p)` renvoie le nombre d'éléments de `p`.

Dans la suite de l'exercice on utilisera uniquement ces fonctions.

**2.** La fonction `reduire_triplet_au_sommet` permet de supprimer l'élément central des trois premiers éléments en partant du haut de la pile, si l'élément du bas et du haut sont de même parité. Les éléments dépliés et non supprimés sont replacés dans le bon ordre dans la pile.

Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et la modifiant en place. Cette fonction renvoie le booléen `est_reduit` indiquant si le triplet du sommet a été réduit ou non.

```

1 def reduire_triplet_au_sommet(p):
2 haut = depiler(p)
3 milieu = depiler(p)
4 bas = sommet(p)
5 est_reduit = ...
6 if haut % 2 != ...:
7 empiler(p, ...)
8 ...
9 empiler(p, ...)
10 return ...

```

## Reponse

```

1 def reduire_triplet_au_sommet(p):
2 haut = depiler(p)
3 milieu = depiler(p)
4 bas = sommet(p)
5 est_reduit = True
6 if haut % 2 != bas % 2:
7 empiler(p, milieu)
8 est_reduit = False
9 empiler(p, haut)
10 return est_reduit

```

**3.** On se propose maintenant d'écrire une fonction `parcourir_pile_en_reduisant` qui parcourt la pile du haut vers le bas en procédant aux réductions pour chaque triplet rencontré quand cela est possible.

La pile est toujours modifiée en place.

La fonction `parcourir_pile_en_reduisant` renvoie un booléen indiquant si la pile a été réduite à au moins une reprise lors du parcours.

**3.a.** Donner la taille minimale que doit avoir une pile pour être réductible.

## Résultat

Si une pile a une taille de 2 ou moins, elle n'est pas réductible.

Si une pile est réductible, alors sa taille est supérieure ou égale à 3.

**3.b.** Recopier et compléter sur la copie :

```

1 def parcourir_pile_en_reduisant(p):
2 q = creer_pile_vide()
3 reduction_pendant_parcours = False
4 while taille(p) >= 3:
5 if ...:
6 reduction_pendant_parcours = ...
7 e = depiler(p)
8 empiler(q, e)
9 while not est_vide(q):
10 ...
11 ...
12 return ...

```

**Résultat**

```

1 def parcourir_pile_en_reduisant(p):
2 q = creer_pile_vide()
3 reduction_pendant_parcours = False
4 while taille(p) >= 3:
5 if reduire_triplet_au_sommet(p):
6 reduction_pendant_parcours = True
7 e = depiler(p)
8 empiler(q, e)
9 while not est_vide(q):
10 e = depiler(q)
11 empiler(p, e)
12 return reduction_pendant_parcours

```

**4.** Partant d'une pile d'entiers `p`, on propose ici d'implémenter une fonction récursive `jouer` jouant une partie complète sur la pile `p`.

On effectue donc autant de parcours que nécessaire.

Une fois la pile parcourue de haut en bas, on effectue un nouveau parcours à condition que le parcours précédent ait modifié la pile. Si à l'inverse, la pile n'a pas été modifiée, on ne fait rien, car la partie est terminée.

```

1 def jouer(p):
2 if parcourir_pile_en_reduisant(...):
3 ...

```

**Résultat**

```

1 def jouer(p):
2 if parcourir_pile_en_reduisant(p):
3 jouer(p)

```

**6.3. 2.6.3 D'après 2022, Polynésie, J1, Ex. 4 - Tri d'une pile****D'après 2022, Polynésie, J1, Ex. 4**

La classe `Pile` utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparaît à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

## Exemples :

### Console Python

```
>>> ma_pile = Pile()
>>> ma_pile.empiler(2)
>>> ma_pile
[2]
>>> ma_pile.empiler(3)
>>> ma_pile.empiler(50)
>>> ma_pile
[2, 3, 50]
>>> ma_pile.depiler()
50
>>> ma_pile
[2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en défilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

```
1 def est_triee(self):
2 if not self.est_vide():
3 e1 = self.depiler()
4 while not self.est_vide():
5 e2 = self.depiler()
6 if e1 > e2 :
7 return False
8 e1 = ...
9 return True
```

**1.** Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

## Réponse

```
1 def est_triee(self):
2 if not self.est_vide():
3 e1 = self.depiler()
4 while not self.est_vide():
5 e2 = self.depiler()
6 if e1 > e2 :
7 return False
8 e1 = e2
9 return True
```

On crée dans la console la pile `A` représentée par `[1, 2, 3, 4]`.

**2.a.** Donner la valeur renvoyée par l'appel `A.est_triee()`.

## Réponse

La valeur `(4)` est d'abord défilée, puis `(3)`. L'ordre n'est pas croissant, ainsi `A.est_triee()` renvoie `False`.

**2.b.** Donner le contenu de la pile `A` après l'exécution de cette instruction.

## Réponse

`A` sera représenté par `[1, 2]`.

On souhaite maintenant écrire le code d'une méthode `depile_max` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.

Après l'exécution de `p.depile_max()`, le nombre d'éléments de la pile `p` diminue donc de 1.

```

1 def depile_max(self):
2 assert not self.est_vide(), "Pile vide"
3 q = Pile()
4 maxi = self.depiler()
5 while not self.est_vide():
6 elt = self.depiler()
7 if maxi < elt:
8 q.empiler(maxi)
9 maxi = ...
10 else :
11 ...
12 while not q.est_vide():
13 self.empiler(q.depiler())
14 return maxi

```

**3.** Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

### Réponse

```

1 def depile_max(self):
2 assert not self.est_vide(), "Pile vide"
3 q = Pile()
4 maxi = self.depiler()
5 while not self.est_vide():
6 elt = self.depiler()
7 if maxi < elt:
8 q.empiler(maxi)
9 maxi = elt
10 else :
11 q.empiler(elt)
12 while not q.est_vide():
13 self.empiler(q.depiler())
14 return maxi

```

On crée la pile `B` représentée par `[9, -7, 8, 12, 4]` et on effectue l'appel `B.depile_max()`.

**4.a.** Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle `while` de la ligne 5.

**Réponse**

Initialisation	Fin du tour 1	Fin du tour 2	Fin du tour 3	Fin du tour 4
----------------	---------------	---------------	---------------	---------------

- `B` contient [9, -7, 8, 12] ;
- `q` est vide ;
- `maxi` est égal à 4.

Juste avant le premier tour de boucle

- `B` contient [9, -7, 8] ;
- `q` contient [4] ;
- `maxi` est égal à 12 .
- `B` contient [9, -7] ;
- `q` contient [4, 8] ;
- `maxi` est égal à 12 .
- `B` contient [9] ;
- `q` contient [4, 8, -7] ;
- `maxi` est égal à 12 .
- `B` est vide ;
- `q` contient [4, 8, -7, 9] ;
- `maxi` est égal à 12 .

**4.b.** Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.

**Réponse**

La dernière boucle renverse la pile `q` dans la pile `B`, ainsi, à la ligne 14 :

- `q` est vide ;
- `B` contient [9, -7, 8, 4] .

**4.c.** Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depile_max`.

## Réponse

Avec une pile `B` qui contient [3, 1, 2]

Initialisation      Fin du tour 1      Fin du tour 2

- `B` contient [3, 1] ;
- `q` est vide ;
- `maxi` est égal à 2.

Juste avant le premier tour de boucle

- `B` contient [3] ;
- `q` contient [1] ;
- `maxi` est égal à 2.
- `B` est vide ;
- `q` contient [1, 2] ;
- `maxi` est égal à 3.

La dernière boucle renverse la pile `q` dans la pile `B`, ainsi, à la ligne 14 :

- `q` est vide ;
- `B` contient [2, 1].

Sans 3 dans la pile `B` initiale, on a dans l'ordre [1, 2] ce qui est différent de [2, 1] obtenu ici avec `depile_max`.

On a ainsi un exemple où l'ordre des éléments restants n'est pas préservé après l'exécution de `depile_max`.

On donne le code de la fonction `traiter` :

```

1 def traiter(self):
2 q = Pile()
3 while not self.est_vide():
4 q.empiler(self.depile_max())
5 while not q.est_vide():
6 self.empiler(q.depiler())

```

**5.a.** Donner les contenus successifs des piles `B` et `q`

- avant la ligne 3,
- avant la ligne 5,
- à la fin de l'exécution de la fonction `traiter` lorsque la fonction `traiter` est appelée avec la pile `B` contenant [1, 6, 4, 3, 7, 2].

**Réponse**

Avec `B = [1, 6, 4, 3, 7, 2]`, un appel `B.traiter()` conduit successivement à :

- Avant la ligne 3,
- `B` contient `[1, 6, 4, 3, 7, 2]` ;
- `q` est vide.
- Avant la ligne 5,
- `B` est vide ;
- `q` contient `[7, 6, 4, 3, 2, 1]`
- À la fin,
- `B` contient `[1, 2, 3, 4, 6, 7]`
- `q` est vide.

**5.b.** Expliquer le traitement effectué par cette méthode.

**Réponse**

Ce traitement est un tri de la pile. On construit d'abord `q` comme la pile des éléments de `self` dans l'ordre décroissant. On renverse ensuite la pile, qui se retrouve comme si on avait empilé les éléments de `self` dans l'ordre croissant.

Attention, il s'agit de l'ordre inverse de celui proposé par la fonction `est_triee` vu à la question 1. ici, si on dépile les éléments, ils sont désormais dans l'ordre décroissant.

**D'après 2022, Centres étrangers, J1, Ex. 3**

Afin d'organiser les répertoires et les fichiers sur un disque dur, une structure arborescente est utilisée. Les fichiers sont dans des répertoires qui sont eux-mêmes dans d'autres répertoires, etc.

Dans une arborescence, chaque répertoire peut contenir des fichiers et des répertoires, qui sont identifiés par leur nom. Le contenu d'un répertoire est modélisé par la structure de données dictionnaire. Les clés de ce dictionnaire sont des chaînes de caractères donnant le nom des fichiers et des répertoires contenus.

## Exemple illustré

Le répertoire appelé Téléchargements contient deux fichiers rapport.pdf et jingle.mp3 ainsi qu'un répertoire Images contenant simplement le fichier logo.png .

Il est représenté ci-dessous.

```
%%init: {'themeVariables': {'fontFamily': 'monospace'}}}%%
flowchart TB
 n0[[Téléchargement]] --> n1[Images]
 n1 --> n4(logo.png)
 n0 --> n2(rapport.pdf)
 n0 --> n3(jingle.mp3)
```

Ce répertoire Téléchargements est modélisé en Python par le dictionnaire suivant :

```
{"Images": {"Logo.png": 36}, "rapport.pdf": 450, "jingle.mp3": 4800}
```

Les valeurs numériques sont exprimées en ko (kilo-octets).

"logo.png": 36 signifie que le fichier logo.png occupe un espace mémoire de 36 ko sur le disque dur.

On rappelle, ci-dessous, quelques commandes sur l'utilisation d'un dictionnaire :

- `dico = dict()` crée un dictionnaire vide appelé `dico`,
- `dico[cle] = contenu` met la valeur `contenu` pour la clé `cle` dans le dictionnaire `dico`,
- `dico[cle]` renvoie la valeur associée à la clé `cle` dans le dictionnaire `dico`,
- `cle in dico` renvoie un booléen indiquant si la clé `cle` est présente dans le dictionnaire `dico`.
- `for cle in dico:` permet d'itérer sur les clés d'un dictionnaire.
- `len(dico)` renvoie le nombre de clés d'un dictionnaire.

L'**adresse** d'un fichier ou d'un répertoire correspond au nom de tous les répertoires à parcourir depuis la racine afin d'accéder au fichier ou au répertoire. Cette adresse est modélisée en Python par la liste des noms de répertoire à parcourir pour y accéder.

Exemple : L'adresse du répertoire : /home/pierre/Documents/ est modélisée par la liste ["home", "pierre", "Documents"] .

1. Dessiner l'arbre donné par le dictionnaire suivant, qui correspond au répertoire Documents .

### Script Python

```
Documents = {
 "Administratif": {
 "certificat_JDC.pdf": 1500,
 "attestation_recensement.pdf": 850
 },
 "Cours": {
 "NSI": {
 "TP.html": 60,
 "dm.odt": 345
 },
 "Philo": {
 "Tractatus_Logico-Philosophicus.epub": 2600
 }
 },
 "liste_de_courses.txt": 24
}
```

## Reponse

```
%{init: {'themeVariables': {'fontFamily': 'monospace'}}}%
flowchart TB
 doc[Documents] --> adm[Administratif]
 adm --> certif(certificat_JDC.pdf)
 adm --> attest(attestation_recensement.pdf)

 doc --> cours[[Cours]]
 cours --> nsi[[NSI]]
 nsi --> tp(TP.html)
 nsi --> dm(dm.odt)

 cours --> philo[[Philo]]
 philo --> tlp(Tractatus_logico-philosophicus.epub)

 doc --> lst(liste_de_courses.txt)
```

- 2.** On donne la fonction `parcourt` suivante qui prend en paramètres un répertoire racine et une liste représentant une adresse, et qui renvoie le contenu du répertoire cible correspondant à l'adresse.

Exemple : Si la variable `docs` contient le dictionnaire de l'exemple de la question 1 alors `parcourt(docs, ["Cours", "Philo"])` renvoie le dictionnaire `{"Tractatus_logico-philosophicus.epub": 2600}`.

- 2.a.** Recopier et compléter la ligne 4

### Script Python

```
def parcourt(racine, adr):
 repertoire = racine
 for nom_repertoire in adr:
 repertoire = ...
 return repertoire
```

## Reponse

### Script Python

```
def parcourt(racine, adr):
 repertoire = racine
 for nom_repertoire in adr:
 repertoire = repertoire[nom_repertoire]
 return repertoire
```

- 2.b.** Soit la fonction suivante :

### Script Python

```
def affiche(racine, adr, nom_fichier):
 repertoire = parcourt(racine, adr)
 print(repertoire[nom_fichier])
```

Qu'affiche l'instruction `affiche(docs, ["Cours", "NSI"], "TP.html")` sachant que la variable `docs` contient le dictionnaire de la question 1 ?

## Reponse

- La première instruction fait que `repertoire` correspond au dictionnaire "NSI" qui vaut `{"TP.html": 60, "dm.odt": 345}`.
- La seconde affiche la valeur associée à la clé "TP.html" de ce dictionnaire, c'est-à-dire le poids en ko de ce fichier.

L'affichage est donc

### Console Python

```
>>> affiche(docs, ["Cours", "NSI"], "TP.html")
60
```

- 3.a.** La fonction `ajoute_fichier` suivante, de paramètres `racine`, `adr`, `nom_fichier` et `taille`, ajoute au dictionnaire `racine`, à l'adresse `adr`, la clé `nom_fichier` associé à la valeur `taille`.

Une ligne de la fonction donnée ci-dessous contient une erreur. Laquelle ? Proposer une correction.

### Script Python

```
def ajoute_fichier(racine, adr, nom_fichier, taille):
 repertoire = parcourt(racine, adr)
 taille = repertoire[nom_fichier]
```

## Reponse

### Script Python

```
def ajoute_fichier(racine, adr, nom_fichier, taille):
 repertoire = parcourt(racine, adr)
 repertoire[nom_fichier] = taille
```

- 3.b.** Écrire une fonction `ajoute_reperoire` de paramètres `racine`, `adr` et `nom_reperoire` qui crée un dictionnaire représentant un répertoire vide appelé `nom_reperoire` dans le dictionnaire `racine` à l'adresse `adr`.

## Reponse

### Script Python

```
def ajoute_reperoire(racine, adr, nom_reperoire):
 repertoire = parcourt(racine, adr)
 repertoire[nom_reperoire] = dict()
```

**4.a.**

### instance pour vérifier le type d'une variable

`isinstance(variable, A)` renvoie `True` si `variable` est de type `A` et `False` sinon.

`A` peut être le type `int`, `dict` ou tout autre type Python.

Écrire une fonction `est_fichier` de paramètre `racine`, un dictionnaire non vide, qui détermine si `racine` est un répertoire ou un fichier. **On supposera que l'arborescence est bien formée :**

- les répertoires et les fichiers sont des dictionnaires ;
- les répertoires ne contiennent, comme clés, que des répertoires et des fichiers ;
- un répertoire peut être vide ;
- la valeur associée à un fichier associée est toujours un entier.

On pourra compléter le code suivant ou en proposer un autre

#### Script Python

```
def est_fichier(racine):
 for cle in racine:
 if isinstance(racine[cle], ...):
 return ...
 return ...
```

#### Réponse

##### Script Python

```
def est_fichier(racine):
 for cle in racine:
 if isinstance(racine[cle], int):
 return True
 return False
```

**4.b** Écrire une fonction `taille` de paramètre `racine` qui prend en paramètre un dictionnaire `racine` modélisant un répertoire et qui renvoie le total d'espace mémoire occupé par les fichiers contenus dans ce répertoire.

#### Réponse

##### Script Python

```
def taille(racine):
 if est_fichier(racine):
 for cle in racine:
 return racine[cle]
 # il n'y a qu'un tour de boucle
 # racine[cle] sera la taille du fichier visé
 else:
 cumul = 0
 for cle in racine:
 cumul += taille(racine[cle])
 # racine[cle] sera soit un fichier, soit un répertoire
 # la fonction est donc récursive
 return cumul
```

### 3. T2 - Bases de données

#### 1. 3.1 C1 Le Modèle relationnel

##### Programme Terminale

Contenus	Capacités attendues	Commentaires
Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel	Identifier les concepts définissant le modèle relationnel	Ces concept permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence)

L'année dernière nous avons eu l'occasion de travailler sur des données structurées en les stockant dans des fichiers au format CSV. Même si cette méthode de stockage de l'information peut s'avérer pratique dans certains cas précis, il est souvent souhaitable d'utiliser une base de données pour stocker des données.

En effet si le nombre de données à stocker devient très grand, est-ce que ma solution choisie pourra les gérer ? (on peut par exemple méditer sur le cas du Royaume-Uni dont le comptage des patients positifs au Covid est devenu faux car il a dépassé les limites de leur [feuille Excel](#))

- Est-ce que d'autres personnes que moi sont susceptibles de consulter ou modifier ces données, éventuellement en même temps que moi ?
- Si une donnée se retrouve à plusieurs endroits dans mes données, devrais-je aller modifier cette donnée partout où elle se trouve ou bien une seule fois ?

L'étude des Bases de Données tente d'apporter des réponses à toutes ces questions.

Dans une base de données, l'information est stockée dans des fichiers, mais à la différence des fichiers au format CSV, il n'est pas possible de travailler sur ces données avec un simple éditeur de texte. Pour manipuler les données présentes dans une base de données (écrire, lire ou encore modifier), il est nécessaire d'utiliser un type de logiciel appelé "**système de gestion de base de données**" très souvent abrégé **en SGBD**.

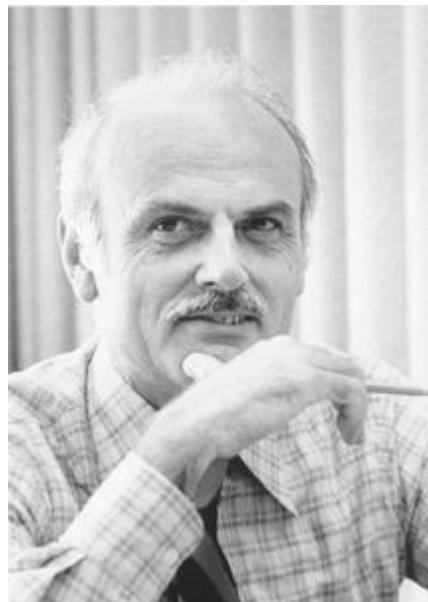
Il existe une multitude de SGBD : des gratuites, des payantes, des libres ou bien encore des propriétaires. Les SGBD permettent de grandement simplifier la gestion des bases de données :

- les SGBD permettent de gérer la lecture, l'écriture ou la modification des informations contenues dans une base de données
- les SGBD permettent de gérer les autorisations d'accès à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur A de lire et d'écrire dans la base de données alors que l'utilisateur B aura uniquement la possibilité de lire les informations contenues dans cette même base de données.
- les fichiers des bases de données sont stockés sur des disques durs dans des ordinateurs, ces ordinateurs peuvent subir des pannes. Il est souvent nécessaire que l'accès aux informations contenues dans une base de données soit maintenu, même en cas de panne matérielle. Les bases de données sont donc dupliquées sur plusieurs ordinateurs afin qu'en cas de panne d'un ordinateur A, un ordinateur B contenant une copie de la base de données présente dans A, puisse prendre le relais. Tout cela est très complexe à gérer, en effet toute modification de la base de données présente sur l'ordinateur A doit entraîner la même modification de la base de données présente sur l'ordinateur B. Cette synchronisation entre A et B doit se faire le plus rapidement possible, il est fondamental d'avoir des copies parfaitement identiques en permanence. C'est aussi les SGBD qui assurent la maintenance des différentes copies de la base de données.
- plusieurs personnes peuvent avoir besoin d'accéder aux informations contenues dans une base données en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent). Ces problèmes d'accès concurrent sont aussi gérés par les SGBD.

Comme nous venons de la voir, les SGBD jouent un rôle fondamental. L'utilisation des SGBD explique en partie la supériorité de l'utilisation des bases de données sur des solutions plus simples à mettre en oeuvre, mais aussi beaucoup plus limitées comme les fichiers au format CSV.

## 1.1. 3.1.1 Le modèle relationnel

Le programme de Terminale NSI prévoit uniquement l'étude du **modèle relationnel**.



Théorisé en 1970 par le Britannique Edgard J. Codd, le modèle relationnel est à ce jour le modèle de base de données le plus utilisé, même si l'ère actuelle du Big Data tend à mettre en avant d'autres modèles non relationnels (nous en reparlerons).

### Les principes de base du modèle relationnel

- Les données sont regroupées dans différentes **tables** (qu'on appellera plutôt **relations** et qui donnent son nom au modèle). Chaque relation contient des éléments directement en lien avec le sujet général de la table.
- Autant que possible, des données identiques ne doivent pas se trouver dans des tables différentes : on évite la **redondance** des données.
- Les données ne doivent pas contenir elles-mêmes d'autres données : on parle d'**atomicité** des données.

Un modèle relationnel est donc basé sur des... **relations**.

## 1.2. 3.1.2 Première relation

Prenons l'exemple d'une bibliothèque dont la base de données possède une relation «livres» :

### Relation «Livres»

code	Titre	Auteur	ann_publi	note	En-tête
1	1984	Orwell	1949	10	
2	Dune	Herbert	1965	8	
3	Fondation	Asimov ann_publi	1951	9	
4	Le meilleur des mondes	Huxley	1931	7	
5	Fahrenheit 451	Bradbury	1953	7	Enregistrement, tuple, n-uplet
6	Ubik	K.Dick	1969	9	
7	Chroniques martiennes	Bradbury	1950	8	
8	La nuit des temps	Barjavel	1968	7	
9	Blade Runner	K.Dick	1968	8	
10	Les Robots	Van Vogt	1950	9	
11	La planète des singes	Boulle	1963	8	
12	Ravage	Barjavel	1943	8	
13	Le maître du Haut Château	K.Dick	1962	8	
14	Les mondes des A	Van Vogt	1945	7	
15	La Fin de l'éternité	Asimov	1955	8	
16	De la Terre à la Lune	Verne	1865	10	
					attribut

## Vocabulaire

- **relation**, ou **table** : c'est l'endroit où sont rangées les données. L'ordre des lignes (que l'on appellera des enregistrements) n'a pas d'importance.
- **enregistrement**, ou **tuple**, ou **n-uplet**, ou **t-uplet**, ou **vecteur** : cela correspond à une ligne du tableau, et donc un ensemble de valeurs liées entre elles : l'auteur «Bradbury» a bien écrit le livre «Fahrenheit 451». Il est **interdit** que deux enregistrements soient totalement identiques. Le nombre d'enregistrements d'une relation s'appelle son **cardinal**.
- **attribut** : c'est l'équivalent d'une colonne. Il y a dans notre relation un attribut «code», un attribut «Titre», etc.
- **domaine** : le domaine désigne «le type» (au sens type `Int`, `Float`, `String`). L'attribut «Auteur» est une chaîne de caractères, par contre l'attribut «code» est un nombre.
- **schéma** : le schéma d'une relation est le regroupement de tous les attributs et de leur domaine respectif. Ici notre schéma serait `((Code, Entier), (Titre, Chaîne de caractères), (Auteur, Chaîne de caractères), (ann_publi, date), (note, Entier))`

### 1.3. 3.1.3 Contrainte d'intégrité : Contrainte de domaine

Pour chaque attribut d'une relation, il est nécessaire de définir un domaine : Le domaine d'un attribut donné correspond à un ensemble fini ou infini de valeurs admissibles.

- Par exemple, le domaine de l'attribut "code" correspond à l'ensemble des entiers (noté INT) : la colonne "code" devra obligatoirement contenir des entiers.
- Autre exemple, le domaine de l'attribut "titre" correspond à l'ensemble des chaînes de caractères (noté TEXT ou CHAR).
- Dernier exemple, le domaine de l'attribut "note" correspond à l'ensemble des entiers positifs.

Au moment de la création d'une relation, il est nécessaire de renseigner le domaine de chaque attribut.

Le SGBD s'assure qu'un élément ajouté à une relation respecte bien le domaine de l'attribut correspondant : si par exemple vous essayez d'ajouter une note non entière (par exemple 8.5), le SGBD signalera cette erreur et n'autorisera pas l'écriture de cette nouvelle donnée.

#### 1.4. 3.1.4 Contrainte d'intégrité : Clé Primaire

##### Clé primaire ❤

Une clé primaire est un attribut (ou une réunion d'attributs) **dont la connaissance suffit à identifier avec certitude un unique enregistrement.**

Par exemple, la clé primaire de la relation des personnes nées en France pourrait être leur [numéro de Sécurité Sociale](#).

Observons, dans notre relation précédente, ce qui peut être une clé primaire et ce qui ne peut pas l'être.

- Titre : cet attribut pourrait jouer le rôle de clé primaire. En effet, notre table ne contient pas deux livres ayant le même titre. Mais en réalité, à éviter car des livres peuvent avoir le même nom parfois.
- Auteur : cet attribut ne pourrait pas jouer le rôle de clé primaire. En effet, notre table contient des livres ayant le même auteur.
- ann\_publi : cet attribut ne peut **pas** jouer le rôle de clé primaire. En effet, la donnée de l'attribut «1951» renvoie vers plusieurs livres différents.
- note : cet attribut ne peut pas jouer le rôle de clé primaire.
- Code : cet attribut peut jouer le rôle de clé primaire. En effet, notre table ne contient pas deux livre ayant le même code.

Alors, quelle clé primaire choisir ?

Il faut pour cela réfléchir à ce que deviendrait notre relation si elle contenait 1000 livres au lieu de 10. Il est fort probable que deux livres aient alors le même auteur : l'attribut «Auteur» ne serait donc plus une clé primaire.

Il peut arriver aussi que deux livres aient le même titre : l'attribut «Titre» n'est donc pas une bonne clé primaire.

L'attribut «Code», qui correspond à une nomenclature «maison», c'est donc une clé primaire qu'on qualifiera d'**«artificielle»**.

Attention, il ne peut pas y avoir deux clés primaires dans une table. La clé primaire choisie ici serait sans aucun doute l'attribut «Code».

On note :

([Code](#) : [Entier](#), [Titre](#) : [Chaîne de caractères](#), [Auteur](#) : [Chaîne de caractères](#), [ann\\_publi](#) : [date](#), [note](#) : [Entier](#))

#### 1.5. 3.1.5 Contrainte d'intégrité : clé étrangère

Ajoutons maintenant les relations ci-dessous :

##### Relation «Emprunts»

<b>id_emprunteur</b>	<b>date</b>	<b>Nom</b>	<b>Prénom</b>	<b>titre</b>	<b>auteur</b>	<b>code</b>
845	12/10/2020	DURAND	Michel	Fondation	Asimov	3
125	13/10/2020	MARTIN	Jean	Blade Runner	K.Dick	9
125	13/10/2020	MARTIN	Jean	De la Terre à la Lune	Verne	16

### Relation «Emprunteurs»

<b>id_emprunteur</b>	<b>Nom</b>	<b>Prénom</b>
129	DUPOND	Marcel
845	DURAND	Michel
125	MARTIN	Jean

L'attribut «id\_emprunteur» est une clé primaire de la relation «Emprunteurs».

### Notion de clé étrangère

Y-a-t-il une clé primaire dans la relation «Emprunts» ?

«id\_emprunteur» est bien une clé primaire (d'«Emprunteurs») mais ne peut pas être une clé primaire d'«Emprunts», car une personne peut prendre plusieurs livres à la fois : on dit que c'est une **clé étrangère**.

#### Cle étrangère ❤

Une clé étrangère est une clé primaire d'une autre relation.

«code» est aussi une clé étrangère : c'est une clé primaire (de la relation «livres») mais elle ne peut pas jouer le rôle de clé primaire pour la relation emprunt, car un même livre pourra être pris à différentes dates.

### 1.6. 3.1.6 Redondance des données

La relation «Emprunts» contient des informations qui sont déjà disponibles dans d'autres relations : on dit qu'elle est **redondante**, et c'est quelque chose qu'il faut éviter. À la fois pour des raisons d'espace de stockage mais aussi de cohérence : si une modification doit être faite (un emprunteur change de prénom), cette modification ne doit être faite qu'à un seul endroit de notre base de données.

Une version non-redondante de la relation «Emprunteurs» serait donc celle-ci :

### Relation «Emprunts»

<b>id_emprunteur</b>	<b>date</b>	<b>code</b>
845	12/10/2020	3
125	13/10/2020	9
125	13/10/2020	16

### 1.7. 3.1.7 Résumé : Contraintes d'intégrité

#### 1.7.1. Contrainte de domaine

#### Contrainte de domaine ❤

Tout attribut d'un enregistrement doit respecter le domaine indiqué dans le schéma relationnel.

Attention, certains domaines sont subtils. Par exemple, si une relation possède un attribut "Code Postal", le domaine de cet attribut devra être `String` plutôt que `Entier`. Dans le cas contraire, un enregistrement possédant le code postal `03150` serait converti en `3150` (car pour les entiers, `03150 = 3150`). Or le code postal `3150` n'existe pas.

### 1.7.2. Contrainte de relation

#### Cle primaire ❤

La contrainte de relation impose que tout enregistrement soit unique : cette contrainte est réalisée par **l'existence obligatoire d'une clé primaire**.

Cette clé primaire est souvent créée de manière artificielle (voir `id_emprunteurs` dans la table ci-dessus par exemple).

### 1.7.3. Contrainte de référence

#### Cle primaire ❤

La **cohérence entre les différentes tables** d'une base de données **est assurée par les clés étrangères** : dans une table, la valeur d'un attribut qui est clé étrangère doit obligatoirement pouvoir être retrouvée dans la table dont cet attribut est clé primaire.

Par exemple, la relation «Emprunts\_v2» ci-dessous n'est pas valable :

#### Relation «Emprunts\_v2»

<code>id_emprunteur</code>	<code>date</code>	<code>code</code>
845	12/10/2020	3
125	13/10/2020	9
125	13/10/2020	<b>27</b>

En effet, le code 27 (clé étrangère de la table «Emprunts\_v2») ne correspond à aucun enregistrement dans la table dont il est clé primaire (la table «Livres») :

Donc ma relation «Emprunts\_v2» ne respecte pas la contrainte de référence, et provoquerait une erreur du SGBD.

## 1.8. 3.1.8 Représentation usuelles des bases de données en modèle relationnel

Considérons la base de données Tour de France 2020, contenant les relations suivantes : (d'après une idée de [Didier Boule](#))

**relation Équipes**

codeEquipe	nomEquipe
ALM	AG2R La Mondiale
AST	Astana Pro Team
TBM	Bahrain - McLaren
BOH	BORA - hansgrohe
CCC	CCC Team
COF	Cofidis, Solutions Crédits
DQT	Deceuninck - Quick Step
EF1	EF Pro Cycling
GFC	Groupama - FDJ
LTS	Lotto Soudal
...	...

Le schéma relationnel de cette table s'écritra souvent : Equipes ( codeEquipe String , nomEquipe String )

Notez le soulignement sous le mot «codeEquipe», qui signifie que cet attribut est une clé primaire. Les clés étrangères, lorsqu'elles existent, peuvent être signalées par une astérisque \* ou #.

**relation Coureurs**

dossard	nomCoureur	prénomCoureur	codeEquipe
141	LÓPEZ	Miguel Ángel	AST
142	FRAILE	Omar	AST
143	HOULE	Hugo	AST
11	ROGLIČ	Primož	TJV
12	BENNETT	George	TJV
41	ALAPHILIPPE	Julian	DQT
44	CAVAGNA	Rémi	DQT
45	DECLERCQ	Tim	DQT
121	MARTIN	Guillaume	COF
122	CONSONNI	Simone	COF
123	EDET	Nicolas	COF
...	...	...	...

Schéma : Equipes ( dossard Int , nomCoureur String , prénomCoureur String , #codeEquipe String )

**relation Étapes**

numéroEtape	villeDépart	villeArrivée	km
1	Nice	Nice	156
2	Nice	Nice	185
3	Nice	Sisteron	198
4	Sisteron	Orcières-Merlette	160
5	Gap	Privas	198
...	...	...	...

Schéma : Étapes ( numéroEtape Int , villeDépart String , villeArrivée String , km Int )

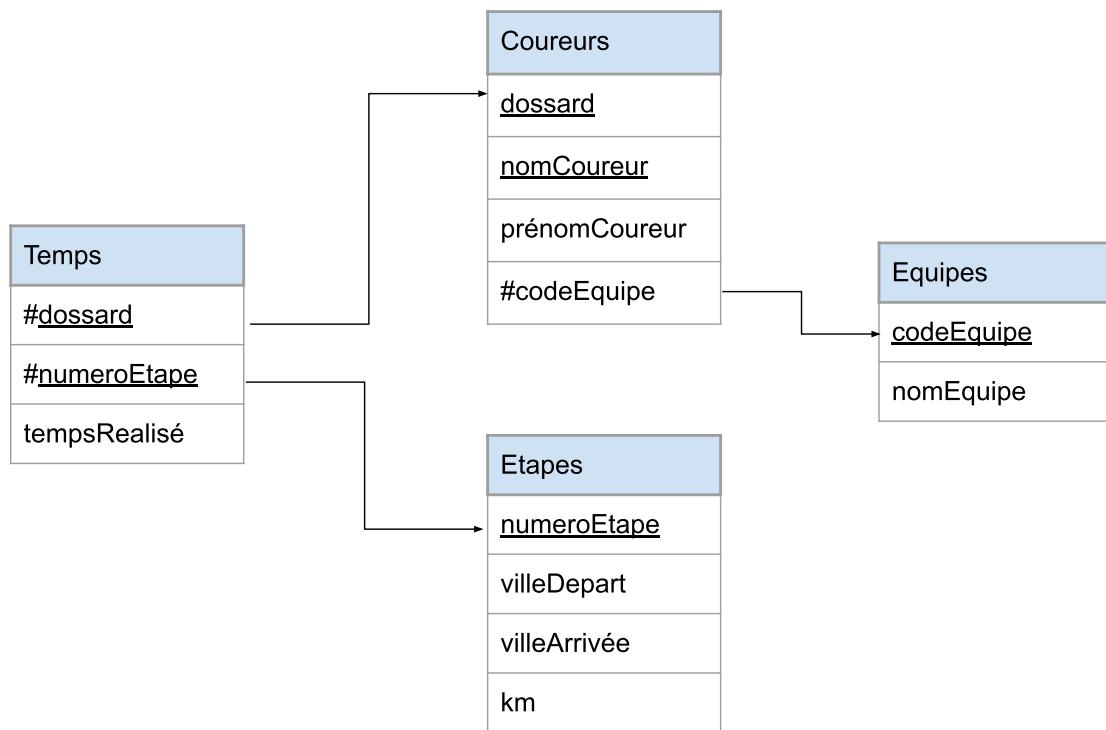
**relation Temps**

dossard	numéroEtape	tempsRéalisé
41	2	04:55:27
121	4	04:07:47
11	5	04:21:22
122	5	04:21:22
41	4	04:08:24
...	...	...

Schéma : Temps ( #dossard Int , #numéroEtape Int , tempsRéalisé String )

Remarquez que la clé primaire de cette relation est le couple dossard-numéroEtape.

**Représentation graphique**



## 2. 3.2 C2 Langage SQL



### Programme Terminale

Contenus	Capacités attendues	Commentaires
Langage SQL : requête d'interrogation et de mise à jour d'une base de données	Identifier les composants d'une requête. Construire des requêtes 'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser GROUP BY et HAVING

**Voir TP sur Capytal**

### 2.1. 3.2.1 Du modèle relationnel au SGBD

Nous allons maintenant d'aborder la partie logicielle : les SGBD (Systèmes de Gestion de Bases de Données).

Les SGBD jouent le rôle d'interface entre l'être humain et la base de données.

Par l'intermédiaire de **requêtes**, l'utilisateur va consulter ou modifier la base de données. Le SGBD est garant de l'intégrité de cette base, et prévient notamment que les modifications ne soient pas préjudiciables à la base de données.

Le langage utilisé pour communiquer avec le SGBD est le langage **SQL**, pour Structured Query Langage (pour *langage de requêtes structurées*).

Les SGBD les plus utilisés sont basés sur le modèle relationnel. Parmi eux, citons Oracle, MySQL, Microsoft SQL Server, PostgreSQL, Microsoft Access, SQLite, MariaDB...

Mais de plus en plus de SGBD **non-relationnels** sont utilisés, spécialement adaptés à des données plus diverses et moins structurées. On les retrouve sous l'appellation **NoSQL** (pour *Not only SQL*). Citons parmi eux MongoDB, Cassandra (Facebook), BigTable (Google)...

La quasi-totalité de ces SGBD fonctionnent avec un modèle client-serveur.

Nous allons travailler principalement avec le langage SQLite peut lui s'utiliser directement sans démarrer un serveur : la base de données est entièrement représentée dans le logiciel utilisant SQLite.

### 2.2. 3.2.2 Crédation de tables

**Requête SQL**

```
DROP TABLE IF EXISTS LIVRES;
CREATE TABLE LIVRES
(code INT, titre TEXT, auteur TEXT, ann_publi INT, note INT, PRIMARY KEY (code));
```

#### 2.2.1. Crédation de la tables LIVRES

**Requête SQL**

```
INSERT INTO LIVRES
(code,titre,auteur,ann_publi,note)
VALUES
(1,'1984','Orwell',1949,10),
(2,'Dune','Herbert',1965,8),
(3,'Fondation','Asimov',1951,9),
(4,'Le meilleur des mondes','Huxley',1931,7),
(5,'Fahrenheit 451','Bradbury',1953,7),
(6,'Ubik','K.Dick',1969,9),
(7,'Chroniques martiennes','Bradbury',1950,8),
(8,'La nuit des temps','Barjavel',1968,7),
(9,'Blade Runner','K.Dick',1968,8),
(10,'Les Robots','Asimov',1950,9),
(11,'La Planète des singes','Boulle',1963,8),
```

```
(12,'Ravage','Barjavel',1943,8),
(13,'Le Maître du Haut Château','K.Dick',1962,8),
(14,'Le monde des Â','Van Vogt',1945,7),
(15,'La Fin de l'éternité','Asimov',1955,8),
(16,'De La Terre à la Lune','Verne',1865,10);
```

## 2.2.2. Création de la table AUTEURS

### Requête SQL

```
DROP TABLE IF EXISTS AUTEURS;
CREATE TABLE AUTEURS
(id INT, nom TEXT, prenom TEXT, ann_naissance INT, Langue_ecriture TEXT, PRIMARY KEY (id));

```

### Requête SQL

```
INSERT INTO AUTEURS
(id,nom,prenom,ann_naissance,Langue_ecriture)
VALUES
(1,'Orwell','George',1903,'anglais'),
(2,'Herbert','Frank',1920,'anglais'),
(3,'Asimov','Isaac',1920,'anglais'),
(4,'Huxley','Aldous',1894,'anglais'),
(5,'Bradbury','Ray',1920,'anglais'),
(6,'K.Dick','Philip',1928,'anglais'),
(7,'Barjavel','René',1911,'français'),
(8,'Boulle','Pierre',1912,'français'),
(9,'Van Vogt','Alfred Elton',1912,'anglais'),
(10,'Verne','Jules',1828,'français');
```

## 2.3. 3.2.3 Sélection de données

### 2.3.1. ↗ Requête basique : SELECT, FROM

### Requête SQL

```
SELECT *
FROM LIVRES
```

code		titre	id_auteur	ann_publi	note
1	1984		1	1949	10
2	Dune		2	1965	8
3	Fondation		3	1951	9
4	Le meilleur des mondes		4	1931	7
5	Fahrenheit 451		5	1953	7
6	Ubik		6	1969	9
7	Chroniques martiennes		5	1950	8
8	La nuit des temps		7	1968	7
9	Blade Runner		6	1968	8
10	Les Robots		3	1950	9
11	La Planète des singes		8	1963	8
12	Ravage		7	1943	8
13	Le Maître du Haut Château	6		1962	8
14	Le monde des Â		9	1945	7
15	La Fin de l'éternité		3	1955	8
16	De la Terre à la Lune		10	1865	10

### Requête SQL

```
SELECT titre, auteur, note
FROM LIVRES
```

	titre	auteur	note
1984		Orwell	10
Dune		Herbert	8
Fondation		Asimov	9
Le meilleur des mondes		Huxley	7
Fahrenheit 451		Bradbury	7
Ubik		K.Dick	9
Chroniques martiennes		Bradbury	8
La nuit des temps		Barjavel	7
Blade Runner		K.Dick	8
Les Robots		Asimov	9
La Planète des singes		Boulle	8
Ravage		Barjavel	8
Le Maître du Haut Château	K.Dick		8
Le monde des A		Van Vogt	7
La Fin de l'éternité		Asimov	8
De la Terre à la Lune		Verne	10

### 2.3.2. ↪ Requête basique : SELECT, FROM, WHERE

#### 📁 Requête SQL

```
SELECT titre, ann_publi
FROM LIVRES
WHERE auteur='Asimov'
```

	titre	ann_publi
Fondation		1951
Les Robots		1950
La Fin de l'éternité		1955

#### 📁 Requête SQL

```
SELECT *
FROM LIVRES
WHERE auteur='Asimov'
```

code	titre	auteur	ann_publi	note
3	Fondation	Asimov	1951	9
10	Les Robots	Asimov	1950	9
15	La Fin de l'éternité	Asimov	1955	8

#### 📁 Requête SQL

```
SELECT auteur,titre, ann_publi
FROM LIVRES
WHERE auteur='Asimov' AND note>=9
```

auteur	titre	ann_publi
Asimov	Fondation	1951
Asimov	Les Robots	1950

## 2.4. 3.2.4 ➔ Renommage : AS

Pour rendre l'affichage plus "lisible" on peut renommer les colonnes : **AS**

- **Commande :**

### Requête SQL

```
SELECT titre,auteur,ann_publi AS publication
FROM LIVRES
WHERE ann_publi >= 1945;
```

- **Traduction :**

Lors de l'affichage du résultats et dans la suite de la requête (important), la colonne "ann\_publi" est renommée "publication".

### Requête SQL

```
SELECT titre,auteur,ann_publi AS publication
FROM livres
WHERE ann_publi >= 1945;
```

	titre	auteur	publication
1984		Orwell	1949
Dune		Herbert	1965
Fondation		Asimov	1951
Fahrenheit 451		Bradbury	1953
Ubik		K.Dick	1969
Chroniques martiennes		Bradbury	1950
La nuit des temps		Barjavel	1968
Blade Runner		K.Dick	1968
Les Robots		Asimov	1950
La Planète des singes		Boulle	1963
Le Maître du Haut Château	K.Dick		1962
Le monde des Â		Van Vogt	1945
La Fin de l'éternité		Asimov	1955

### 2.4.1. ➔ Mettre dans l'ordre les réponses la clause ORDER BY

Il est aussi possible de rajouter la clause SQL ORDER BY afin d'obtenir les résultats classés dans un ordre précis.

### Requête SQL

```
SELECT titre
FROM LIVRES
WHERE auteur='K.Dick' ORDER BY ann_publi
```

titre
Le Maître du Haut Château
Blade Runner
Ubik

### Remarques :

- **Comportement par défaut :** Si le paramètre ASC ou DESC est omis, le classement se fait par ordre **croissant** (donc ASC est le paramètre par défaut).

## 2.5. 3.2.5 => La clause DISTINCT

Il est possible d'éviter les doublons grâce à la clause DISTINCT

### Requête SQL

```
SELECT auteur
FROM LIVRES
```

#### auteur

Orwell  
Herbert  
Asimov  
Huxley  
Bradbury  
K.Dick  
Bradbury  
Barjavel  
K.Dick  
Asimov  
Boulle  
Barjavel  
K.Dick  
Van Vogt  
Asimov  
Verne

### Requête SQL

```
SELECT DISTINCT auteur
FROM LIVRES
```

#### auteur

Orwell  
Herbert  
Asimov  
Huxley  
Bradbury  
K.Dick  
Barjavel  
Boulle  
Van Vogt  
Verne

## 2.5.1. => La clause LIKE

On veut les titres de la table «livre» dont le titre contient la chaîne de caractères "Astérix".

Le symbole % est un joker qui peut symboliser n'importe quelle chaîne de caractères.

### Requête SQL

```
SELECT titre
FROM livres
WHERE titre LIKE 'F%';
```

permet d'obtenir les titres de livres commençant par F

**Requête SQL**

```
SELECT titre
FROM livres
WHERE titre LIKE '%s';
```

permet d'obtenir les titres de livres finissant par s

**Requête SQL**

```
SELECT titre
FROM livres
WHERE titre LIKE 'F%';
```

titre

Fondation

Fahrenheit 451

**Requête SQL**

```
SELECT titre
FROM livres
WHERE titre LIKE '%s';
```

titre

Le meilleur des mondes

Chroniques martiennes

La nuit des temps

Les Robots

La Planète des singes

**2.6. 3.2.6 Opérations sur les données : sélection avec agrégation**

Les requêtes effectuées jusqu'ici ont juste sélectionné des données grâce à différents filtres : aucune action à partir de ces données n'a été effectuée.

Nous allons maintenant effectuer des opérations à partir des données sélectionnées.

On appelle ces opérations des **opérations d'agrégation**.

**2.7. 3.2.7 ↞ La clause COUNT**

On veut compter le nombre d'enregistrements de la tables livres publiés en 1968.

**Requête SQL**

```
SELECT COUNT(*) AS total
FROM livres
WHERE ann_publi=1968;
```

total

2

**2.8. 3.2.8 ↞ La clause : SUM - Additionner**

- **Commande :**

**Requête SQL**

```
SELECT SUM(ann_publi) AS somme
FROM livres
WHERE auteur LIKE "F%";
```

- **Traduction :**

On veut additionner les années des livres de la tables livres commençant par F.

Le résultat sera le seul élément d'une colonne nommée «somme». *Attention : dans notre cas précis, ce calcul n'a aucun sens...*

```
somme
3904
```

## 2.9. 3.2.9 => La clause : AVG - Moyenne"

On veut calculer la moyenne des notes des livres de la table livres de l'auteur "Bradbury". Le résultat sera le seul élément d'une colonne nommée «moyenne».

### Requête SQL

```
SELECT AVG(note) AS note moyenne
FROM livres
WHERE auteur="Bradbury";
```

```
moyenne
7.5
```

### 2.9.1. => La clause : MIN, MAX - Trouver les extremums:

- **Commande :**

### Requête SQL

```
SELECT MIN(note) AS minimum
FROM livres;
```

auteur	titre minimum
Huxley	Le meilleur des mondes

## 2.10. 3.2.10 Des recherches croisées sur les tables : les jointures

Nous avons 2 tables, grâce aux jointures nous allons pouvoir associer ces 2 tables dans une même requête.

Repartons sur la bases LIVRES légèrement modifiées.

### Requête SQL

```
DROP TABLE IF EXISTS LIVRES;
CREATE TABLE LIVRES
(code INT, titre TEXT, id_auteur TEXT, ann_publi INT, note INT, PRIMARY KEY (code,id_auteur));
```

### Requête SQL

```
INSERT INTO LIVRES
(code,titre,id_auteur,ann_publi,note)
VALUES
(1,'1984',1,1949,10),
(2,'Dune',2,1965,8),
(3,'Fondation',3,1951,9),
(4,'Le meilleur des mondes',4,1931,7),
(5,'Fahrenheit 451',5,1953,7),
(6,'Ubik',6,1969,9),
(7,'Chroniques martiennes',5,1950,8),
(8,'La nuit des temps',7,1968,7),
(9,'Blade Runner',6,1968,8),
(10,'Les Robots',3,1950,9),
(11,'La Planète des singes',8,1963,8),
(12,'Ravage',7,1943,8),
(13,'Le Maître du Haut Château',6,1962,8),
(14,'Le monde des Â',9,1945,7),
(15,'La Fin de l'éternité',3,1955,8),
(16,'De la Terre à la Lune',10,1865,10);
```

### 2.10.1. ↪ Jointures simples

En général, les jointures consistent à associer des lignes de 2 tables. Elles permettent d'établir un lien entre 2 tables.

- **Commande :**

 **Requête SQL**

```
SELECT *
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id
```

- **Traduction :** Comme plusieurs tables sont appelées, nous préfixons chaque colonne avec le nom de la table.

code	titre	id_auteur	ann_publi	note_id	nom	prenom	ann_naissance	langue_ecriture
1	1984	1	1949	10	1 Orwell	George	1903	anglais
2	Dune	2	1965	8	2 Herbert	Frank	1920	anglais
3	Fondation	3	1951	9	3 Asimov	Isaac	1920	anglais
4	Le meilleur des mondes	4	1931	7	4 Huxley	Aldous	1894	anglais
5	Fahrenheit 451	5	1953	7	5 Bradbury	Ray	1920	anglais
6	Ubik	6	1969	9	6 K.Dick	Philip	1928	anglais
7	Chroniques martiennes	5	1950	8	5 Bradbury	Ray	1920	anglais
8	La nuit des temps	7	1968	7	7 Barjavel	René	1911	français
9	Blade Runner	6	1968	8	6 K.Dick	Philip	1928	anglais
10	Les Robots	3	1950	9	3 Asimov	Isaac	1920	anglais
11	La Planète des singes	8	1963	8	8 Boulle	Pierre	1912	français
12	Ravage	7	1943	8	7 Barjavel	René	1911	français
13	Le Maître du Haut Château	6	1962	8	6 K.Dick	Philip	1928	anglais
14	Le monde des Â	9	1945	7	9 Van Vogt	Alfred Elton	1912	anglais
15	La Fin de l'éternité	3	1955	8	3 Asimov	Isaac	1920	anglais
16	De la Terre à la Lune	10	1865	10	10 Verne	Jules	1828	français

Des informations (id et id\_auteur) sont en double.

On peut être plus précis.

 **Requête SQL**

```
SELECT AUTEURS.nom,LIVRES.titre,LIVRES.note,AUTEURS.ann_naissance AS Naissance, LIVRES.ann_publi AS Publication, AUTEURS.langue_ecriture AS Langue
FROM LIVRES
```

```
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id
ORDER BY AUTEURS.nom
```

nom		titre	note	Naissance	Publication	Langue
Asimov	Fondation		9	1920	1951	anglais
Asimov	Les Robots		9	1920	1950	anglais
Asimov	La Fin de l'éternité		8	1920	1955	anglais
Barjavel	La nuit des temps		7	1911	1968	français
Barjavel	Ravage		8	1911	1943	français
Boulle	La Planète des singes		8	1912	1963	français
Bradbury	Fahrenheit 451		7	1920	1953	anglais
Bradbury	Chroniques martiennes		8	1920	1950	anglais
Herbert	Dune		8	1920	1965	anglais
Huxley	Le meilleur des mondes		7	1894	1931	anglais
K.Dick	Ubik		9	1928	1969	anglais
K.Dick	Blade Runner		8	1928	1968	anglais
K.Dick	Le Maître du Haut Château		8	1928	1962	anglais
Orwell	1984		10	1903	1949	anglais
Van Vogt	Le monde des Å		7	1912	1945	anglais
Verne	De la Terre à la Lune		10	1828	1865	français

## 2.11. 3.2.11 Modifications d'une base

### 2.11.1. ➔ INSERT

Insérer les données suivantes dans la base Auteurs:

#### 📁 Requête SQL

```
INSERT INTO LIVRES
(code,titre,id_auteur,ann_publi,note)
VALUES
(17,'Hypérion','Simmons',1989,8)
```

### 2.11.2. ➔ UPDATE

#### 📁 Requête SQL

```
UPDATE LIVRES
SET note=7
WHERE titre = 'Hypérion'
```

### 2.11.3. ➔ DELETE

#### 📁 Requête SQL

```
DELETE FROM LIVRES
WHERE titre='Hypérion'
```

### 3. 3.3 SQL : Exercices BAC

## Thème 2 : Base de données

# BAC

## Langage SQL

### 3.1. 3.3.1 Exercice n°1 : Métropole J1 : Base de données cinématographique



- 3 relations dans une base de données sur le cinéma
- 2 tables : `individu` et `realisation`

On pourra utiliser les mots clés SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT`, `INTO`, `VALUES`, `UPDATE`, `SET`, `AND`.

Nous allons étudier une base de données traitant du cinéma dont voici le schéma relationnel qui comporte 3 relations :

- la relation `individu` (`id_ind`, nom, prenom, naissance)
- la relation `realisation` (`id_rea`, titre, annee, type)
- la relation `emploi` (`id_emp`, description, #`id_ind`, #`id_rea`)

Les clés primaires sont soulignées et les clés étrangères sont précédées d'un #.

Ainsi `emploi.id_ind` est une clé étrangère faisant référence à `individu.id_ind`.

Voici un extrait des tables `individu` et `realisation` :

- Extrait de `individu`

<code>id_ind</code>	nom	prenom	naissance
105	'Hulka'	'Daniel'	'01-06-1968'
403	'Travis'	'Daniel'	'10-03-1968'
688	'Crog'	'Daniel'	'07-07-1968'
695	'Pollock'	'Daniel'	'24-08-1968'

-Extrait de `realisation`

<code>id_rea</code>	titre	annee	type
105	'Casino Imperial'	2006	'action'
325	'Ciel tombant'	2012	'action'
655	'Fantôme'	2015	'action'
950	'Mourir pour attendre'	2021	'action'

1. On s'intéresse ici à la récupération de données dans une relation.

**1.a.** Décrire ce que renvoie la requête ci-dessous : **Requête SQL**

```
SELECT nom, prenom, naissance
FROM individu
WHERE nom = 'Crog';
```

**1.b.** Fournir une requête SQL permettant de récupérer le titre et la clé primaire de chaque film dont la date de sortie est strictement supérieure à 2020.

**2.** Cette question traite de la modification de relations.

**2.a.** Dire s'il faut utiliser la requête 1 ou la requête 2 proposées ci-dessous pour modifier la date de naissance de Daniel Crog. Justifier votre réponse en expliquant pourquoi la requête refusée ne pourra pas fonctionner.

 **Requête SQL 1**

```
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

 **Requête SQL 2**

```
INSERT INTO individu
VALUES (688, 'Crog', 'Daniel', '02-03-1968');
```

**2.b.** Expliquer si la relation `individu` peut accepter (ou pas) deux individus portant le même nom, le même prénom et la même date de naissance.

**3.** Cette question porte sur la notion de clés étrangères.

**3.a.** Recopier sur votre copie les demandes ci-dessous, dans leur intégralité, et les compléter correctement pour qu'elles ajoutent dans la relation `emploi` les rôles de Daniel Crog en tant que James Bond dans le film nommé '`Casino Impérial`' puis dans le film '`Ciel tombant`'.

 **Requête SQL**

```
INSERT INTO emploi
VALUES (5400, 'Acteur(James Bond)', ... , ...);

INSERT INTO emploi
VALUES (5401, 'Acteur(James Bond)', ... , ...);
```

**3.b.** On désire rajouter un nouvel emploi de Daniel Crog en tant que James Bond dans le film '`Docteur Yes`'.

Expliquer si l'on doit d'abord créer l'enregistrement du film dans la relation `realisation` ou si l'on doit d'abord créer le rôle dans la relation `emploi`.

**4.** Cette question traite des jointures.

**4.a.** Recopier sur votre copie la requête SQL ci-dessous, dans son intégralité, et la compléter de façon à ce qu'elle renvoie le nom de l'acteur, le titre du film et l'année de sortie du film, à partir de tous les enregistrements de la relation `emploi` pour lesquels la description de l'emploi est '`Acteur(James Bond)`'.

 **Requête SQL**

```
SELECT ...
FROM emploi
JOIN individu ON ...
JOIN realisation ON ...
WHERE emploi.description = 'Acteur(James Bond)';
```

**4.b.** Fournir une requête SQL permettant de trouver toutes les descriptions des emplois de Denis Johnson (Denis est son prénom et Johnson est son nom).

On veillera à n'afficher que la description des emplois et non les films associés à ces emplois.

### 3.2. 3.3.2 Exercice n°2 : D'après 2022, Métropole, J2



- 2 relations dans une base de données sur la musique
- 2 tables : `morceaux` et `interpretes`

On pourra utiliser les mots clés SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT`, `INTO`, `VALUES`, `UPDATE`, `SET`, `AND`.

La clause `ORDER BY` suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut. L'instruction `COUNT(*)` renvoie le nombre de lignes d'une requête.

Un musicien souhaite créer une base de données relationnelle contenant ses morceaux et interprètes préférés. Pour cela il utilise le langage SQL.

Il crée une table `morceaux` qui contient entre autres attributs les titres des morceaux et leur année de sortie :

- Table `morceaux`

<code>id_morceau</code>	<code>titre</code>	<code>annee</code>	<code>id_interprete</code>
1	Like a Rolling Stone	1965	1
2	Respect	1967	2
3	Imagine	1970	3
4	Hey Jude	1968	4
5	Smells Like Teen Spirit	1991	5
6	I Want To hold Your Hand	1963	4

Il crée la table `interpretes` qui contient les interprètes et leur pays d'origine :

- Table `interpretes`

<code>id_interprete</code>	<code>nom</code>	<code>pays</code>
1	Bob Dylan	États-Unis
2	Aretha Franklin	États-Unis
3	John Lennon	Angleterre
4	The Beatles	Angleterre
5	Nirvana	États-Unis

`id_morceau` de la table `morceaux` et `id_interprete` de la table `interpretes` sont des clés primaires.

L'attribut `id_interprete` de la table `morceaux` fait directement référence à la clé primaire de la table `interpretes`.

**1.a.** Écrire le résultat de la requête suivante :

#### Requête SQL

```
SELECT titre
FROM morceaux
WHERE id_interprete = 4;
```

**1.b.** Écrire une requête permettant d'afficher les noms des interprètes originaires d'Angleterre.

**1.c.** Écrire le résultat de la requête suivante : Requête SQL

```
SELECT titre, annee
FROM morceaux
ORDER BY annee;
```

**1.d.** Écrire une requête permettant de calculer le nombre de morceaux dans la table `morceaux`.**1.e.** Écrire une requête affichant les titres des morceaux par ordre alphabétique.**2.a.** Citer, en justifiant, la clé étrangère de la table `morceaux`.**2.b.** Écrire un schéma relationnel des tables `interpretes` et `morceaux`.**2.c.** Expliquer pourquoi la requête suivante produit une erreur : Requête SQL

```
INSERT INTO interpretes
VALUES (1, 'Trust', 'France');
```

**3.a.** Une erreur de saisie a été faite. Écrire une requête SQL permettant de changer l'année du titre « Imagine » en 1971.**3.b.** Écrire une requête SQL permettant d'ajouter l'interprète « The Who » venant d'Angleterre à la table `interpretes`. On lui donnera un `id_interprete` égal à 6.**3.c.** Écrire une requête SQL permettant d'ajouter le titre « My Generation » de « The Who » à la table `morceaux`. Ce titre est sorti en 1965 et on lui donnera un `id_morceau` de 7 ainsi que l'`id_interprete` qui conviendra.**4.** Écrire une requête permettant de lister les titres des interprètes venant des États-Unis.**3.3. 3.3.3 Exercice n°3 : Métropole, Candidats libres, J2 2021** SQL

- 2 relations dans une base de données sur un CDI
- 3 tables : `Livres`, `Emprunts` et `Eleves`

L'énoncé de cet exercice utilise les mots du langage SQL suivants :

```
SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES, UPDATE, SET, DELETE,
COUNT, AND, OR.
```

On considère dans cet exercice une gestion simplifiée des emprunts des ouvrages d'un CDI. La base de données utilisée sera constituée de trois relations (ou tables) nommées `Eleves`, `Livres` et `Emprunts` selon le schéma relationnel suivant :

- `Livres` (`isbn (CHAR 13)`, `titre (CHAR)`, `auteur (CHAR)`)
- `Emprunts` (`#idEmprunt (INT)`, `#idEleve (INT)`, `#isbn (CHAR 13)`, `dateEmprunt (DATE)`, `dateRetour (Date)`)
- `Eleves` (`#idEleve (INT)`, `nom (CHAR)`, `prenom (CHAR)`, `classe (CHAR)`)

Dans ce schéma relationnel, un attribut souligné indique qu'il s'agit d'une clé primaire.

Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère. Ainsi, l'attribut `#idEleve` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `#idEleve` de la relation `Eleves`. De même l'attribut `#isbn` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `#isbn` de la relation `Livres`.**1.** Expliquer pourquoi le code SQL ci-dessous provoque une erreur. Requête SQL

```
INSERT INTO Eleves VALUES (128, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (200, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (128, 'Dubois', 'Jean', 'T2') ;
```

- 2.** Dans la définition de la relation `Emprunts`, qu'est-ce qui assure qu'on ne peut pas enregistrer un emprunt pour un élève qui n'a pas encore été inscrit dans la relation `Eleves` ?
- 3.** Écrire une requête SQL qui renvoie les titres des ouvrages de Molière détenus par le CDI.
- 4.** Décrire le résultat renvoyé par la requête ci-dessous.

#### Requête SQL

```
SELECT COUNT(*)
FROM Eleves
WHERE classe = 'T2' ;
```

- 5.** Camille a emprunté le livre « *Les misérables* ». Le code ci-dessous a permis d'enregistrer cet emprunt.

#### Requête SQL

```
INSERT INTO Emprunts
VALUES (640, 192, '9782070409228', '2020-09-15', NULL);
```

Camille a restitué le livre le 30 septembre 2020. Recopier et compléter la requête ci-dessous de manière à mettre à jour la date de retour dans la base de données.

#### Requête SQL

```
UPDATE Emprunts
SET ;
WHERE ;
```

- 6.** Décrire le résultat renvoyé par la requête ci-dessous.

#### Requête SQL

```
SELECT DISTINCT nom, prenom
FROM Eleves, Emprunts
WHERE Eleves.idEleve = Emprunts.idEleve
AND Eleves.classe = 'T2' ;
```

- 7.** Écrire une requête SQL qui permet de lister les noms et prénoms des élèves qui ont emprunté le livre « *Les misérables* ».

## 4. 3.4 SQL : Exercices BAC - Correction

**03**

### Thème 2 : Base de données

## BAC : langage SQL



### Exercice n°1 : Métropole J1 : Base de données cinématographique

- 3 relations dans une base de données sur le cinéma
- 2 tables : `individu` et `realisation`

On pourra utiliser les mots clés SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT`, `INTO`, `VALUES`, `UPDATE`, `SET`, `AND`.

Nous allons étudier une base de données traitant du cinéma dont voici le schéma relationnel qui comporte 3 relations :

- la relation `individu` (`id_ind`, nom, prenom, naissance)
- la relation `realisation` (`id_rea`, titre, annee, type)
- la relation `emploi` (`id_emp`, description, #`id_ind`, #`id_rea`)

Les clés primaires sont soulignées et les clés étrangères sont précédées d'un #.

Ainsi `emploi.id_ind` est une clé étrangère faisant référence à `individu.id_ind`.

Voici un extrait des tables `individu` et `realisation` :

#### Extrait de individu

<code>id_ind</code>	nom	prenom	naissance
105	'Hulka'	'Daniel'	'01-06-1968'
403	'Travis'	'Daniel'	'10-03-1968'
688	'Crog'	'Daniel'	'07-07-1968'
695	'Pollock'	'Daniel'	'24-08-1968'

#### Extrait de realisation

<code>id_rea</code>	titre	annee	type
105	'Casino Imperial'	2006	'action'
325	'Ciel tombant'	2012	'action'
655	'Fantôme'	2015	'action'
950	'Mourir pour attendre'	2021	'action'

1. On s'intéresse ici à la récupération de données dans une relation.

**1.a.** Décrire ce que renvoie la requête ci-dessous : **Requête SQL**

```
SELECT nom, prenom, naissance
FROM individu
WHERE nom = 'Crog';
```

 **Réponse**

La requête renvoie les nom, prénom et date de naissance de tous les individus qui portent Crog comme nom de famille. Dans la mesure où l'on ne fournit que des extraits des tables, on ne peut pas fournir le résultat de cette requête de façon certaine.

**1.b.** Fournir une requête SQL permettant de récupérer le titre et la clé primaire de chaque film dont la date de sortie est strictement supérieure à 2020. **Réponse** **Requête SQL**

```
SELECT titre, id_rea
FROM realisation
WHERE annee > 2020;
```

**2.** Cette question traite de la modification de relations.**2.a.** Dire s'il faut utiliser la requête 1 ou la requête 2 proposées ci-dessous pour modifier la date de naissance de Daniel Crog. Justifier votre réponse en expliquant pourquoi la requête refusée ne pourra pas fonctionner. **Requête SQL 1**

```
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

 **Requête SQL 2**

```
INSERT INTO individu
VALUES (688, 'Crog', 'Daniel', '02-03-1968');
```

 **Réponse**

Compte tenu de l'extrait fourni de la table `individu`, l'identifiant `688` est déjà utilisé pour un enregistrement et il ne peut pas y avoir de doublon pour les clés primaires, ainsi **la requête 2 provoquera une erreur**.

La requête 1 est correcte.

Bien que valide cette requête peut être simplifiée en n'utilisant que la clé primaire de la table :

 **Requête SQL 1**

```
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688;
```

**2.b.** Expliquer si la relation `individu` peut accepter (ou pas) deux individus portant le même nom, le même prénom et la même date de naissance.

### Reponse

Aucun des champs correspondant ne possède la contrainte `UNIQUE` (*hypothèse réaliste*). Les deux individus n'auront donc pas le même identifiant ! Ainsi, **oui**, la relation `individu` peut accepter deux tels individus.

**3.** Cette question porte sur la notion de clés étrangères.

**3.a.** Recopier sur votre copie les demandes ci-dessous, dans leur intégralité, et les compléter correctement pour qu'elles ajoutent dans la relation `emploi` les rôles de Daniel Crog en tant que James Bond dans le film nommé 'Casino Impérial' puis dans le film 'Ciel tombant' .

### Requête SQL

```
INSERT INTO emploi
VALUES (5400, 'Acteur(James Bond)', ... , ...);

INSERT INTO emploi
VALUES (5401, 'Acteur(James Bond)', ... , ...);
```

### Reponse

#### Requête SQL

```
INSERT INTO emploi
VALUES (5400, 'Acteur(James Bond)', 688, 105);

INSERT INTO emploi
VALUES (5401, 'Acteur(James Bond)', 688, 325);
```

**3.b.** On désire rajouter un nouvel emploi de Daniel Crog en tant que James Bond dans le film 'Docteur Yes' .

Expliquer si l'on doit d'abord créer l'enregistrement du film dans la relation `realisation` ou si l'on doit d'abord créer le rôle dans la relation `emploi` .

### Reponse

Il faut d'abord créer l'enregistrement du film dans la relation `realisation` , car l'identifiant du film doit être connu afin d'être utilisé comme clé étrangère dans la relation `emploi` .

**4.** Cette question traite des jointures.

**4.a.** Recopier sur votre copie la requête SQL ci-dessous, dans son intégralité, et la compléter de façon à ce qu'elle renvoie le nom de l'acteur, le titre du film et l'année de sortie du film, à partir de tous les enregistrements de la relation `emploi` pour lesquels la description de l'emploi est 'Acteur(James Bond)' .

### Requête SQL

```
SELECT ...
FROM emploi
JOIN individu ON ...
JOIN realisation ON ...
WHERE emploi.description = 'Acteur(James Bond)';
```

**Réponse****Requête SQL**

```
SELECT nom, titre, annee
FROM emploi
JOIN individu ON emploi.id_ind = individu.id_ind
JOIN realisation ON emploi.id_rea = realisation.id_rea
WHERE emploi.description = 'Acteur(James Bond)';
```

- 4.b.** Fournir une requête SQL permettant de trouver toutes les descriptions des emplois de Denis Johnson (Denis est son prénom et Johnson est son nom).

On veillera à n'afficher que la description des emplois et non les films associés à ces emplois.

**Réponse****Requête SQL**

```
SELECT description
FROM emploi
JOIN individu ON emploi.id_ind = individu.id_ind
WHERE prenom = 'Denis' AND nom = 'Johnson';
```

**Exercice n°2 : D'après 2022, Métropole, J2**

- 2 relations dans une base de données sur la musique
- 2 tables : `morceaux` et `interpretes`

On pourra utiliser les mots clés SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `INSERT`, `INTO`, `VALUES`, `UPDATE`, `SET`, `AND`.

La clause `ORDER BY` suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut. L'instruction `COUNT(*)` renvoie le nombre de lignes d'une requête.

Un musicien souhaite créer une base de données relationnelle contenant ses morceaux et interprètes préférés. Pour cela il utilise le langage SQL.

Il crée une table `morceaux` qui contient entre autres attributs les titres des morceaux et leur année de sortie :

**Table morceaux**

<code>id_morceau</code>	<code>titre</code>	<code>annee</code>	<code>id_interprete</code>
1	Like a Rolling Stone	1965	1
2	Respect	1967	2
3	Imagine	1970	3
4	Hey Jude	1968	4
5	Smells Like Teen Spirit	1991	5
6	I Want To hold Your Hand	1963	4

Il crée la table `interpretes` qui contient les interprètes et leur pays d'origine :

**Table** interpretes

<b>id_interprete</b>	<b>nom</b>	<b>pays</b>
1	Bob Dylan	États-Unis
2	Aretha Franklin	États-Unis
3	John Lennon	Angleterre
4	The Beatles	Angleterre
5	Nirvana	États-Unis

`id_morceau` de la table `morceaux` et `id_interprete` de la table `interpretes` sont des clés primaires.

L'attribut `id_interprete` de la table `morceaux` fait directement référence à la clé primaire de la table `interpretes`.

**1.a.** Écrire le résultat de la requête suivante :

#### Requête SQL

```
SELECT titre
FROM morceaux
WHERE id_interprete = 4;
```

#### Réponse

On obtient les titres 'Hey Jude' et 'I Want To hold Your Hand' .

**1.b.** Écrire une requête permettant d'afficher les noms des interprètes originaires d'Angleterre.

#### Réponse

#### Requête SQL

```
SELECT nom
FROM interpretes
WHERE pays = 'Angleterre';
```

**1.c.** Écrire le résultat de la requête suivante :

#### Requête SQL

```
SELECT titre, annee
FROM morceaux
ORDER BY annee;
```

**Réponse**

On obtient :

titre	annee
I Want To hold Your Hand	1963
Like a Rolling Stone	1965
Respect	1967
Hey Jude	1968
Imagine	1970
Smells Like Teen Spirit	1991

**1.d.** Écrire une requête permettant de calculer le nombre de morceaux dans la table `morceaux`.

**Réponse****Requête SQL**

```
SELECT COUNT(*)
FROM morceaux;
```

**1.e.** Écrire une requête affichant les titres des morceaux par ordre alphabétique.

**Réponse****Requête SQL**

```
SELECT titre
FROM morceaux
ORDER BY titre;
```

**2.a.** Citer, en justifiant, la clé étrangère de la table `morceaux`.

**Réponse**

La clé étrangère est `id_interprete` qui fait référence à un attribut de la table `interpretes`.

**2.b.** Écrire un schéma relationnel des tables `interpretes` et `morceaux`.

## Reponse

On propose :

- `morceaux (id_morceau, titre, annee, #id_interprete)`
- `interpretes (id_interprete, nom, pays)`

Les clés primaires sont soulignées (`id_morceau` et `id_interprete`). Dans la table `morceaux`, l'attribut `id_interprete` est précédé d'un `#` : c'est une clé étrangère faisant référence à l'attribut `id_interprete` de la table `interpretes`.

**2.c.** Expliquer pourquoi la requête suivante produit une erreur :

### Requête SQL

```
INSERT INTO interpretes
VALUES (1, 'Trust', 'France');
```

## Reponse

La table contient déjà une entrée dont l'attribut `id_interprete` vaut `1`. Comme il s'agit de la clé primaire cela provoque une erreur.

**3.a.** Une erreur de saisie a été faite. Écrire une requête SQL permettant de changer l'année du titre « Imagine » en 1971.

## Reponse

On utilise la clé primaire du morceau afin d'éviter toute méprise :

### Requête SQL

```
UPDATE morceaux
SET annee = 1971
WHERE id_morceau = 3;
```

Si l'on considère que les tables fournies représentent l'ensemble des données (le sujet est ambigu à ce titre), on peut aussi se contenter de :

### Requête SQL

```
UPDATE morceaux
SET annee = 1971
WHERE titre = 'Imagine';
```

**3.b.** Écrire une requête SQL permettant d'ajouter l'interprète « The Who » venant d'Angleterre à la table `interpretes`. On lui donnera un `id_interprete` égal à 6.

## Reponse

### Requête SQL

```
INSERT INTO interpretes
VALUES (6, 'The Who', 'Angleterre');
```

**3.c.** Écrire une requête SQL permettant d'ajouter le titre « My Generation » de « The Who » à la table `morceaux`. Ce titre est sorti en 1965 et on lui donnera un `id_morceau` de 7 ainsi que l'`id_interprete` qui conviendra.

### Reponse

#### Requête SQL

```
INSERT INTO morceaux
VALUES (7, 'My Generation', 1965, 6);
```

**4.** Écrire une requête permettant de lister les titres des interprètes venant des États-Unis.

### Reponse

On utilise une jointure :

#### Requête SQL

```
SELECT titre
FROM morceaux
JOIN interpretes ON interpretes.id_interprete = morceaux.id_interprete
WHERE interpretes.pays = 'États-Unis';
```

### Exercice n°3 : Métropole, Candidats libres, J2 2021

- 2 relations dans une base de données sur un CDI
- 3 tables : `Livres`, `Emprunts` et `Eleves`

L'énoncé de cet exercice utilise les mots du langage SQL suivants :

`SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES, UPDATE, SET, DELETE,`  
`COUNT, AND, OR.`

On considère dans cet exercice une gestion simplifiée des emprunts des ouvrages d'un CDI. La base de données utilisée sera constituée de trois relations (ou tables) nommées `Eleves`, `Livres` et `Emprunts` selon le schéma relationnel suivant :

- `Livres` (`isbn (CHAR 13)`, `titre (CHAR)`, `auteur (CHAR)`)
- `Emprunts` (`#idEmprunt (INT)`, `#idEleve (INT)`, `#isbn (CHAR 13)`, `dateEmprunt (DATE)`, `dateRetour (Date)`)
- `Eleves` (`idEleve (INT)`, `nom (CHAR)`, `prenom (CHAR)`, `classe (CHAR)`)

Dans ce schéma relationnel, un attribut souligné indique qu'il s'agit d'une clé primaire.

Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère. Ainsi, l'attribut `idEleve` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `idEleve` de la relation `Eleves`. De même l'attribut `isbn` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `isbn` de la relation `Livres`.

**1.** Expliquer pourquoi le code SQL ci-dessous provoque une erreur.

#### Requête SQL

```
INSERT INTO Eleves VALUES (128, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (200, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (128, 'Dubois', 'Jean', 'T2') ;
```

**Réponse**

On insère deux entrées dans lesquelles l'attribut `idEleve` est égal à `128`. Or cet attribut est la clé primaire de la table, il ne peut pas exister en doublon.

- 2.** Dans la définition de la relation `Emprunts`, qu'est-ce qui assure qu'on ne peut pas enregistrer un emprunt pour un élève qui n'a pas encore été inscrit dans la relation `Eleves` ?

**Réponse**

Il s'agit de la clé étrangère `idEleve` qui doit respecter la contrainte d'intégrité référentielle.

- 3.** Écrire une requête SQL qui renvoie les titres des ouvrages de Molière détenus par le CDI.

**Réponse****Requête SQL**

```
SELECT titre
FROM Livres
WHERE auteur = 'Molière'
```

- 4.** Décrire le résultat renvoyé par la requête ci-dessous.

**Requête SQL**

```
SELECT COUNT(*)
FROM Eleves
WHERE classe = 'T2' ;
```

**Réponse**

On compte les élèves de la table `Eleves` dont la classe est la `'T2'`.

- 5.** Camille a emprunté le livre « *Les misérables* ». Le code ci-dessous a permis d'enregistrer cet emprunt.

**Requête SQL**

```
INSERT INTO Emprunts
VALUES (640, 192, '9782070409228', '2020-09-15', NULL);
```

Camille a restitué le livre le 30 septembre 2020. Recopier et compléter la requête ci-dessous de manière à mettre à jour la date de retour dans la base de données.

**Requête SQL**

```
UPDATE Emprunts SET WHERE ;
```

**Réponse** **Requête SQL**

```
UPDATE Emprunts
SET dateRetour = '2020-09-30'
WHERE idEmprunt = 640
```

6. Décrire le résultat renvoyé par la requête ci-dessous.

 **Requête SQL**

```
SELECT DISTINCT nom, prenom
FROM Eleves, Emprunts
WHERE Eleves.idEleve = Emprunts.idEleve
AND Eleves.classe = 'T2' ;
```

**Réponse**

On récupère les noms et prénoms des élèves de la classe 'T2' qui ont déjà emprunté un livre.

7. Écrire une requête SQL qui permet de lister les noms et prénoms des élèves qui ont emprunté le livre « *Les misérables* ».

**Réponse**

On propose (en utilisant l'ISBN cité dans la question 5):

 **Requête SQL**

```
SELECT nom, prenom
FROM Eleves
JOIN Emprunts ON Eleves.idEleves = Emprunts.idEleves
WHERE Emprunts.isbn = 192
```

Sans l'ISBN :

 **Requête SQL**

```
SELECT nom, prenom
FROM Eleves
JOIN Emprunts ON Eleves.idEleves = Emprunts.idEleves
JOIN Livres ON Livres.isbn = Emprunts.isbn
WHERE Livres.titre = 'Les Misérables'
```

## 4. T3 - Archi. matérielle

### 1. 4.1 C6 Protocole de routage



#### Programme Terminale

Contenus	Capacités attendues	Commentaires
Protocole de routage	Identifier, suivant le protocole de routage utilisé, la route empruntée par un paquet.	<p>En mode débranché, les tables de routage étant données, on se réfère au nombre de sauts (protocole RIP) ou au coût des routes (protocole OSPF)</p> <p>Le lien avec les algorithmes de recherche de chemin sur un graphe est mis en évidence.</p>



#### 1.1. 4.1.1 Résumé des épisodes précédents

##### 1.1.1. ■ Activité 1 : Adresse IP, masque

#### Kappel

- Deux machines ne peuvent communiquer que si elles sont sur le même réseau, c'est à dire que leurs adresses ip démarre par une partie commune. La longueur de cette partie commune est définie par le **masque de sous réseau**.
- Pour la connaître, on écrit le masque de sous réseau en binaire. Le nombre de 1 en début de masque donne la longueur de la partie commune dans les adresses IP.
- Par exemple le masque 255.255.254.0 donne en écriture binaire 11111111.11111111.11111110.00000000 . La partie commune doit donc être de 23 bits car cette écriture débute par 23 fois le chiffre 1 . Un masque de 23 bits peut se noter de façon plus concise /23 (notation cidr). Pour savoir si deux machines de ce réseau peuvent communiquer on écrit leurs adresses IP en binaire et on regarde si les 23 premiers bits sont identiques ou non.

### Kappel

L'adresse du réseau s'obtient en réalisant un *et logique* entre l'adresse IP d'un ordinateur du réseau et le masque de sous réseau.

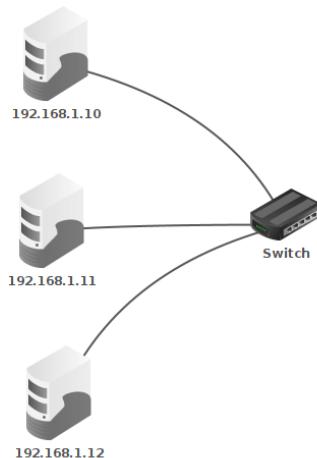
1. Lancer **Filius**, un outil de simulation de réseaux, et créer un simple réseau constitué de deux ordinateurs. Dans chacun des cas suivants, prévoir si les deux ordinateurs peuvent communiquer et le vérifier à l'aide d'une commande `ping`

IP ordinateur 1	Masque Ordinateur 1	IP ordinateur 2	Masque ordinateur 2
203.147.154.100	255.255.255.192	203.147.154.119	255.255.255.192
203.147.154.100	255.255.255.192	203.147.154.129	255.255.255.192
172.19.247.15	255.255.240.0	172.19.230.150	255.255.240.0
172.19.247.15	255.255.240.0	172.19.248.118	255.255.240.0

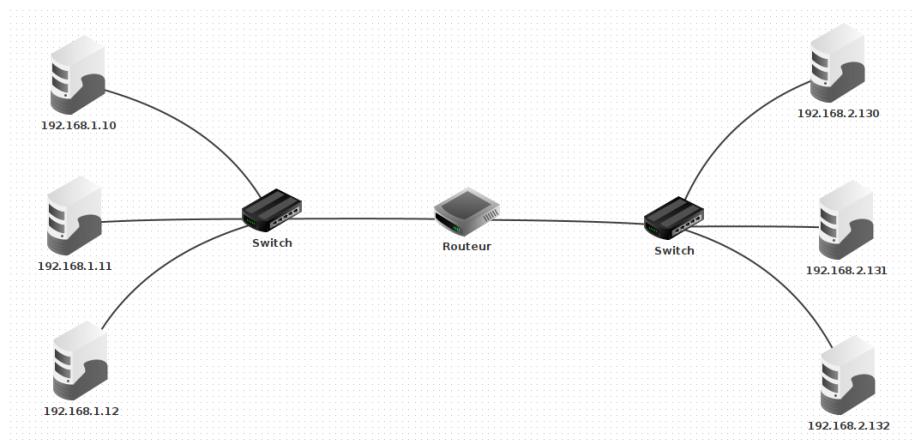
### Aide

En cas de difficultés pour utiliser Filius, faire la [première activité du cours de première](#).

1. Un *switch* sert à connecter plusieurs ordinateurs d'un même sous réseau, ainsi chez un particulier, une box internet joue le rôle de switch et permet de connecter les divers appareils de la maison (téléphone, ordinateur, imprimante, ...). Connecter trois ordinateurs au même sous réseau comme ci-dessous et vérifier à l'aide de la commande `ping` qu'ils peuvent communiquer.



1. Un *routeur* permet d'interconnecter plusieurs sous réseau. Réaliser dans filius le réseau ci-dessous (le routeur a deux interfaces). Attribuer les mêmes adresses ip que sur le schéma et remarquer bien que les ordinateurs de droite et de gauche ne font pas partie du même sous réseau. C'est le routeur qui permet leur interconnection, pour cela il faut indiquer pour chaque machine du réseau de gauche l'adresse du routeur dans le champ passerelle et faire de même, pour chaque machine du sous réseau de droite.



En cas de difficultés, se référer à la vidéo suivante :



## 1.2. 4.1.2 Protocoles de routage

### 1.2.1. ■ Activité 2 : Protocoles de routage

On peut assimiler un réseau à un graphe, les noeuds sont les routeurs et les arcs les liaisons entre ces routeurs. Par exemple :

```
graph TD
 A(("A"))
 B(("B"))
 C(("C"))
 D(("D"))
 E(("E"))
 A --- B
 A --- E
 B --- E
 B --- D
 C --- E
 C --- D
```

Dans cet exemple, plusieurs chemins permettent de relier A à C. Un *protocole de routage* est un mécanisme permettant de choisir l'un de ces chemins. Pour mettre en place un protocole de routage, chaque routeur doit être doté d'une *table de routage* qui indique le routeur suivant selon la destination du paquet.

## 1. Le protocole rip : Routing Information Protocol

- a. Sachant que dans ce protocole, on tente de minimiser le nombre de routeurs traversés, quel serait les chemins empruntés pour :
- relier A et E ?
  - relier A et C ?
  - relier E et D ?
- b. De plus dans ce protocole, à l'initialisation, la table de routage de chaque routeur ne contient que ses voisins immédiats associés à une distance de 1. Par exemple la table de routage de A est :

Destination	Distance
B	1
E	1

et celle de B est :

Destination	Distance
A	1
D	1
E	1

Donner à l'initialisation, les tables de routages des autres routeurs : C, D et E.

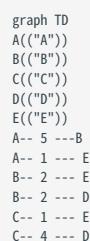
- c. Les tables de routages sont mises à jour à intervalles réguliers, en effet chaque routeur consulte la table de routage de ses voisins et met la sienne à jour en conséquence. Par exemple, A reçoit la table de routage de B, il y trouve D qui ne figure pas encore dans sa table et le rajoute donc en augmentant sa distance de 1. La table de A devient donc :

Destination	Distance
B	1
E	1
D (via B)	2

Le routeur E figure déjà dans la table avec une distance plus courte, il n'est pas mis à jour. Construire la table de routage de B après la première mise à jour.

## 2. Le protocole ospf

La qualité des liaisons entre différents routeurs dépend du type de connection, on peut donc représenter un réseau par un graphe pondéré. Dans l'exemple suivant, la qualité de la liaison entre A et E (poids 1) est bien meilleure que celle entre A et B (poids 5)



- a. Sachant que dans ce protocole, on tente de minimiser le poids du chemin parcouru, quel serait les chemins empruntés pour :
- relier A et E ?
  - relier A et C ?
  - relier E et D ?

### 1.3. 4.1.3 COURS - Protocoles de Routage

#### Quelques dates clés de l'historique d'internet

- Début des années 1960 : idée de la création d'un réseau informatique global permettant d'interconnecter de multiples sous-réseaux.
- Début des années 1970 : naissance d'*arpnet*, ancêtre d'internet.
- 1973 : définition des protocoles *tcp* (Transmission Control Protocol) et *ip* (Internet Protocol)
- 1983 : premier serveur de noms de domaines (*dns* pour domain name server)
- 1989 : Naissance du *web*
- 2010 : Plus de 5 milliards de machines connectées.

#### 1.3.1. Protocole TCP/IP

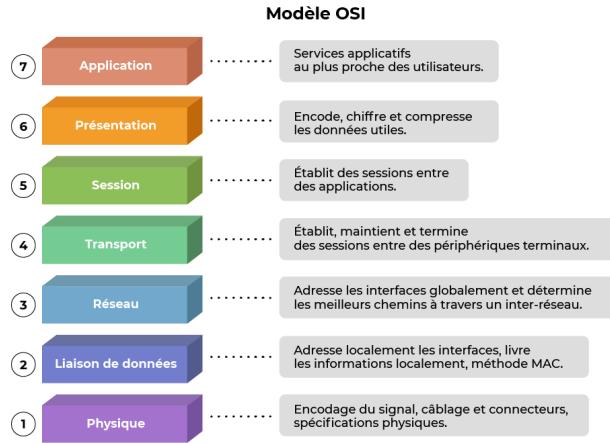
#### Protocoles TCP/IP

Internet fonctionne suivant une architecture réseau en 4 couches :

##### Modèle en 4 couches

Application	http, pop, ftp ...
Transport	tcp, udp
Réseau	ipv4, ipv6
Liaison	Ethernet, wifi

- Chaque couche ne communique qu'avec les couches voisines.
- Chaque couche ajoute les données dont il a besoin pour fonctionner à l'information transmise. C'est ce qu'on appelle **l'encapsulation** des données.
- Un autre modèle en 7 couches existe : le modèle **osi**.



### 1.3.2. Adresses MAC et IP

#### **Adresse MAC (Media Access Control)**

Une adresse matérielle, ou **adresse MAC**, parfois nommée adresse physique, est un identifiant physique stocké dans une carte réseau ou une interface réseau similaire (Wifi par exemple). À moins qu'elle n'ait été modifiée par l'utilisateur, elle est unique au monde.

Elle constitue la couche inférieure de la couche de liaison, c'est-à-dire la couche deux du modèle OSI. Elle est constituée de six octets, il existe donc potentiellement  $(2^{48})$  (environ 281 000 milliards) d'adresses MAC possibles.

- On considère un réseau local, de deux ordinateurs (reliés par un simple cable) ou de plusieurs ordinateurs (reliés par un **switch**)
- Les adresses **MAC** (pour *Media Access Control*) permettent d'identifier de façon unique un élément du réseau. Elles sont composées de six nombres en hexadécimal séparés par le caractère deux points :, par exemple 1A:B2:EC:AE:B0:DE.
- Le protocole **ARP** permet d'associer l'**adresse MAC** à l'**adresse IP**.
- Une commande **ping** entre deux de ces machines, commence donc par un appel au protocole **ARP** afin d'obtenir l'**adresse MAC** de la machine cible.

#### **Adresse IP**

Une **adresse IP (Internet Protocol)** est un numéro d'identification qui est attribué de façon permanente ou provisoire à chaque périphérique relié à un réseau informatique qui utilise l'Internet Protocol. L'adresse IP est à la base du système d'acheminement (le routage) des paquets de données sur Internet.

Deux machines ne peuvent communiquer que si elles sont sur le même réseau, c'est à dire que leur adresse IP débute par une partie commune. La longueur de cette partie commune dépend du masque de sous réseau.

- Par exemple si le masque de sous réseau est 255.255.255.0, cela signifie que les trois premiers octets de l'adresse IP doivent être communs.
- Un masque de sous réseau se note aussi en notation **CIDR** (pour Classless Inter-Domain Routing) en indiquant simplement le nombre de bits communs. Comme 3 octets = 24 bits, le masque précédent se note donc plus simplement /24.
- Si le masque de sous réseau est {tt 255.255.255.0}, le nombre maximal de machines sur le réseau est de 254 (parmi les 256 possibilités, deux sont réservées. L'une pour le **broadcast** (envoi à tout le réseau) et l'autre pour le sous-réseau lui-même).

### Exemple :

On considère trois machines : A ( 192.168.130.10 ), B ( 192.168.155.100 ) et C ( 192.168.144.203 ) et on suppose qu'on a défini pour ces machines le masque de sous réseau 255.255.240.0

- Donner ce masque de sous réseau en notation **CIDR**.

#### Réponse ▾

On convertit 240 en binaire :  $\backslash((240)\_{10} = (11110000)\_2\}$ , il y a donc 20 bits (8+8+4) en commun le masque est donc /20 . }

- Indiquer les machines qui peuvent communiquer entre elles.

#### Réponse ▾

Le problème porte sur le troisième octet, les adresses IP doivent avoir leur 4 premier octet communs pour pouvoir communiquer :

- A : 192.168.130.10 , or  $\backslash((130)\_{10} = (100000010)\_2\}$
- B : 192.168.155.100 , or  $\backslash((155)\_{10} = (10011011)\_2\}$
- C : 192.168.144.203 , or  $\backslash((144)\_{10} = (10010000)\_2\}$

B et C peuvent communiquer entre elles mais pas avec A.

- Confirmer éventuellement par un test dans Filius

### 1.3.3. Les Routeurs

#### Routeurs

Les routeurs permettent de faire communiquer des ordinateurs appartenant à des sous réseaux différents.

- A titre d'exemple une *box internet* dans une maison joue le rôle de switch (elle permet aux différents ordinateurs du foyer de communiquer entre eux) mais aussi de routeur (elle permet de communiquer avec des ordinateurs hors de la maison).
- Les *tables de routage* sont des informations stockées localement dans chaque routeur et lui permettant d'orienter les paquets qu'ils reçoit vers un autre routeur ou un sous réseau avec lequel il communique.

### 1.3.4. Protocoles de routage

COMMENT SONT CONSTRUITES LES TABLES DE ROUTAGE ?

#### Protocoles de routage

- Pour des réseaux de petites tailles, les tables de routage peuvent être écrites "à la main".
- Les inconvénients sont nombreux car les tables sont alors statiques et ne s'adaptent pas (par exemple lors d'une panne ou de l'ajout de nouveaux éléments au réseau).
- Des protocoles de routage permettant de générer de façon automatique les tables de routage ont donc été mis au point, nous allons en présenter deux :
- Le protocole de routage **RIP** (Routing Information Protocol) transmet les paquets de proche en proche en essayant de minimiser le nombre de routeurs traversé.
- Le protocole de routage **OSPF** (Open Shortest Path First) transmet les paquets en tenant compte de la vitesse de connection entre les routeurs

### 1.4. 4.1.4 Le protocole RIP

#### protocole RIP :

- A l'origine les routeurs ne connaissent que les réseaux auxquels ils sont directement connectés
- Puis régulièrement (toutes les 30 secondes), chaque routeur reçoit la table de routage de ses voisins et il met alors à jour sa propre table :
- S'il y découvre une route vers un réseau inconnu, il l'ajoute à sa propre table en augmentant de 1 la distance.
- S'il y découvre une route vers un réseau connu mais plus courte, il met à jour sa table.
- Si un routeur ne reçoit pas d'information d'un voisin figurant dans sa table (pendant 3 min), il considère celui ci comme en panne (et lui affecte la distance maximale de 16).

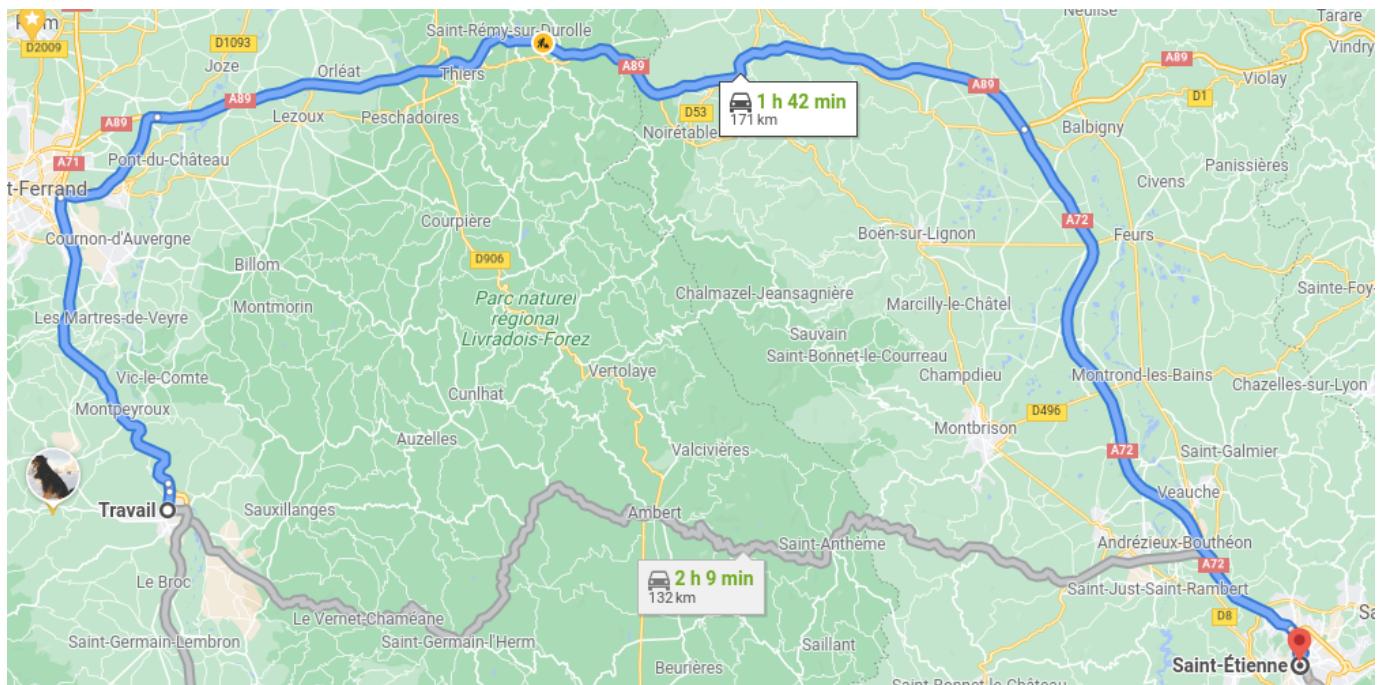
### Rémarques et inconvénients :

- L'échange régulier des tables de routage augmente la quantité d'information qui circule sur le réseau
- Le protocole se limite au réseau de petites tailles car une distance maximale de 15 sauts est imposée (ceci explique que 16 soit considéré comme la distance infinie).
- Chaque routeur n'a jamais connaissance de la topologie du réseau tout entier : il ne le connaît que par ce que les autres routeurs lui ont raconté. On dit que ce protocole de routage est du *routing by rumor*.
- Le protocole se base uniquement sur le nombre de sauts et pas sur la qualité de la liaison entre deux routeurs qui peut être très différentes :

Technologie	BP descendante	BP montante
Bluetooth	3 Mbit/s	3 Mbit/s
Ethernet	10 Mbit/s	10 Mbit/s
Wi-Fi	10 Mbit/s ~ 10 Gbit/s	10 Mbit/s ~ 10 Gbit/s
ADSL	13 Mbit/s	1 Mbit/s
4G	100 Mbit/s	50 Mbit/s
Satellite	50 Mbit/s	1 Mbit/s
Fast Ethernet	100 Mbit/s	100 Mbit/s
FFTH (fibre)	10 Gbit/s	10 Gbit/s
5G	20 Gbit/s	10 Gbit/s

- La *métrique* utilisée (le nombre de sauts) ne tient pas compte de la qualité de la liaison, contrairement au protocole OSPF.

En voiture, le chemin le plus rapide n'est pas forcément le plus court.



En gris, le chemin RIP. En bleu, l'OSPF.

## 1.5. 4.1.5 Le protocole OSPF

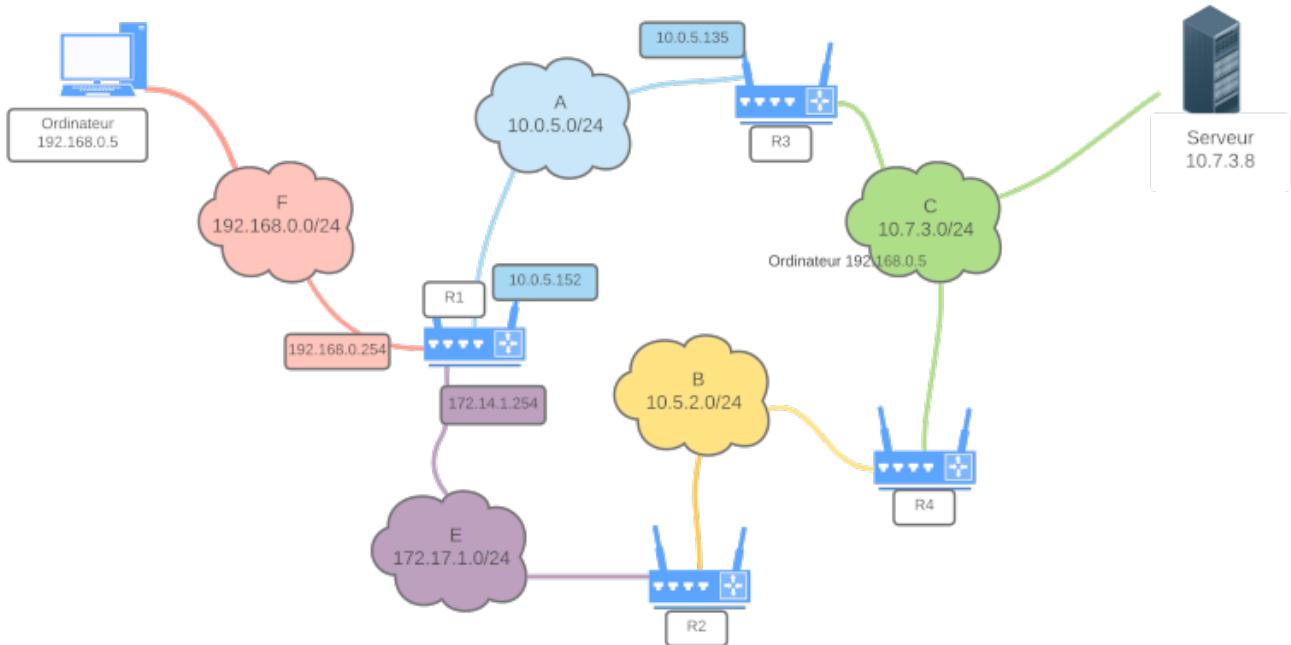
### OSPF : Open Shortest Path First

On commence par attribuer un coût **inversement proportionnel à la qualité de la connection** à chaque liaison.

- Cette formule qui calcule le coût  $(c)$  d'une liaison ayant un débit  $(d)$  (en bit/s) peut être par exemple :
- $(c = \frac{1}{d})$ . Mais elle peut-être différente et sera donnée à chaque exercice.
- On transforme donc le réseau en un **graphe pondéré** (plus tard) dans lequel on doit rechercher le plus court chemin entre deux sommets.
- Dans les cas simples de petits graphes, la réponse est intuitive.
- Sinon, on dispose d'un algorithme permettant de répondre à ce problème : l'algorithme de **Dijkstra** (informaticien néerlandais, 1930--2002).

#### 1.5.1. Exemple

Reprenons le réseau suivant :



et simplifions-le en ne gardant que les liens entre routeurs, en indiquant leur débit :

```
graph LR
O(("Ordinateur"))
R1(("R1"))
R3(("R3"))
R2(("R2"))
R4(("R4"))
S(("Serveur 10.7.3.8"))
0-- 25 Mbit/s ---R1
R1-- 20 Mbit/s ---R3
R1-- 50 Mbit/s ---R2
R2-- 50 Mbit/s ---R4
R4-- 10 Mbit/s ---S
R3-- 10 Mbit/s ---S
```

Notre réseau est devenu un **graph**. Nous allons pondérer ses arêtes avec la fonction coût introduite précédemment. L'unité étant le Mbit/s, l'arête entre R1 et R3 aura un poids de  $100/20=5$ .

Le graphe pondéré est donc :

```
graph LR
O(("Ordinateur"))
R1(("R1"))
R3(("R3"))
R2(("R2"))
R4(("R4"))
S(("Serveur 10.7.3.8"))
O--- 4 ---R1
R1--- 5 ---R3
R1--- 2 ---R2
R2--- 2 ---R4
R4--- 10 ---S
R3--- 10 ---S
```

Le chemin le plus rapide pour aller de l'ordinateur au serveur est donc R1-R2-R4, et non plus R1-R3 comme l'aurait indiqué le protocole RIP.

### 1.5.2. Trouver le plus court chemin dans un graphe pondéré

Utilisation de **l'algorithme de Dijkstra**, pour le comprendre, vous pouvez regarder la vidéo d'un célèbre YouTuber :

#### Dijkstra

Cet algorithme, ici exécuté de manière manuelle, est bien sûr programmable. Et c'est donc grâce à lui que chaque routeur calcule la route la plus rapide pour acheminer les données qu'il reçoit.

**Exemple** d'application de l'algorithme de Dijkstra : donner le plus court chemin de A jusqu'à F

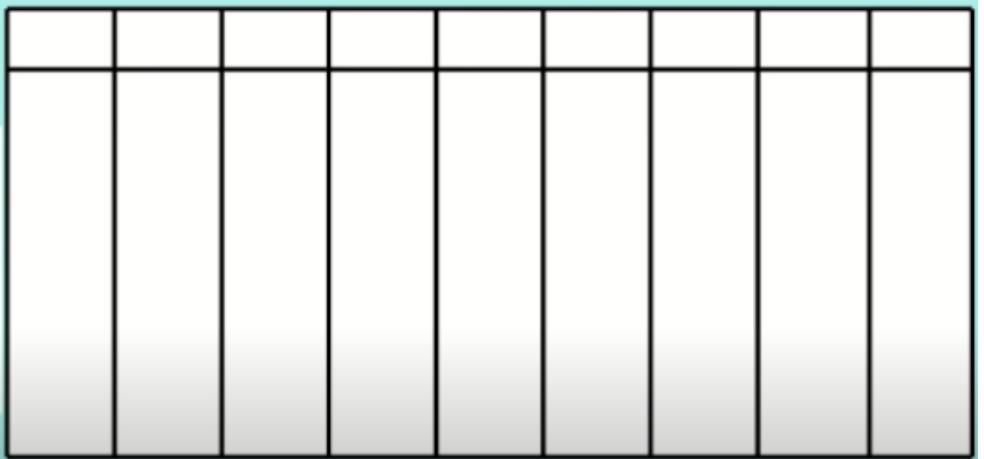
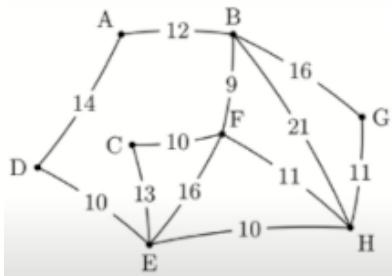
```
graph LR
A(("A"))
B(("B"))
C(("C"))
D(("D"))
E(("E"))
F(("F"))
A--- 1 ---C
A--- 5 ---B
B--- 3 ---C
B--- 4 ---D
C--- 7 ---F
B--- 2 ---E
C--- 5 ---E
D--- 3 ---F
E--- 1 ---F
```

Response

A	B	C	D	E	F	
0 (A)	5 (A)	1 (A)				A
✓	4 (C)	1 (A)		6 (C)	8 (C)	C
✓	4 (C)	✓	8 (B)	6 (B)	8 (C)	B
✓	✓	✓	8 (B)	6 (B)	7 (E)	E
✓	✓	✓	8 (B)	✓	7 (E)	F
✓	✓	✓	8 (B)	✓	✓	D

## algorithme de Dijkstra

recherche du plus court chemin de A à H



Donner le plus court chemin pour aller de A à H.

## 2. 4.2 Thème 3 - Architecture matérielle

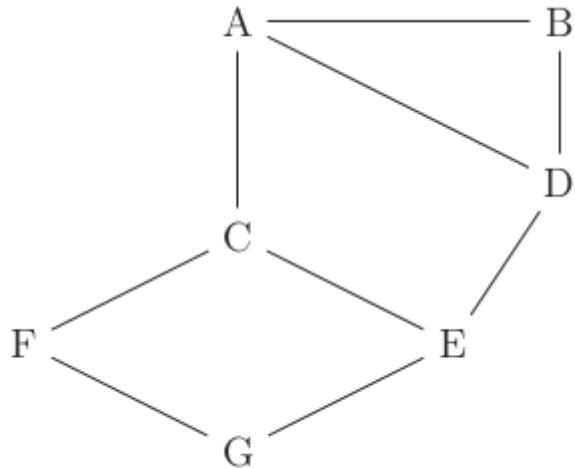
**BAC****Protocole de Routage**

## 3. 4.3 Exercices BAC

## 3.1. 4.3.1 Sujet n°1 : sujet zéro

**Sujet zéro**

On considère un réseau composé de plusieurs routeurs reliés de la façon suivante :



## 3.1.1. → Le protocole RIP

Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur la distance, en nombre de sauts, qui le sépare d'un autre routeur. Pour le réseau ci-dessus, on dispose des tables de routage suivantes :

Table de routage du routeur A		
Destination	Routeur suivant	Distance
B	B	1
C	C	1
D	D	1
E	C	2
F	C	2
G	C	3

Table de routage du routeur B		
Destination	Routeur suivant	Distance
A	A	1
C	A	2
D	D	1
E	D	2
F	A	3
G	D	3

Table de routage du routeur C		
Destination	Routeur suivant	Distance
A	A	1
B	A	2
D	E	2
E	E	1
F	F	1
G	F	2

Table de routage du routeur D		
Destination	Routeur suivant	Distance
A	A	1
B	B	1
C	E	2
E	E	1
F	A	3
G	E	2

Table de routage du routeur E		
Destination	Routeur suivant	Distance
A	C	2
B	D	2
C	C	1
D	D	1
F	G	2
G	G	1

Table de routage du routeur F		
Destination	Routeur suivant	Distance
A	C	2
B	C	3
C	C	1
D	C	3
E	G	2
G	G	1

### Question 1

- Le routeur A doit transmettre un message au routeur G, en effectuant un nombre minimal de sauts. Déterminer le trajet parcouru.
- Déterminer une table de routage possible pour le routeur G obtenu à l'aide du protocole RIP.

### Question 2

Le routeur C tombe en panne. Reconstruire la table de routage du routeur A en suivant le protocole RIP.

#### 3.1.2. → Le protocole OSPF

Contrairement au protocole RIP, l'objectif n'est plus de minimiser le nombre de routeurs traversés par un paquet. La notion de distance utilisée dans le protocole OSPF est uniquement liée aux coûts des liaisons.

L'objectif est alors de minimiser la somme des coûts des liaisons traversées.

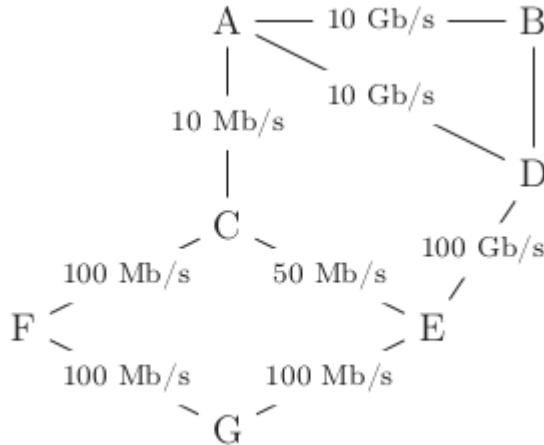
Le coût d'une liaison est donné par la formule suivante :

$$\text{coût} = \frac{1}{d}$$

où  $d$  est la bande passante en bits/s entre les deux routeurs.

On a rajouté sur le graphe représentant le réseau précédent les différents débits des liaisons.

On rappelle que  $1 \text{ Gb/s} = 1000 \text{ Mb/s} = 10^9 \text{ bits/s}$ .



### Question 3

1. Vérifier que le coût de la liaison entre les routeurs A et B est 0,01.
2. La liaison entre le routeur B et D a un coût de 5. Quel est le débit de cette liaison ?

### Question 4

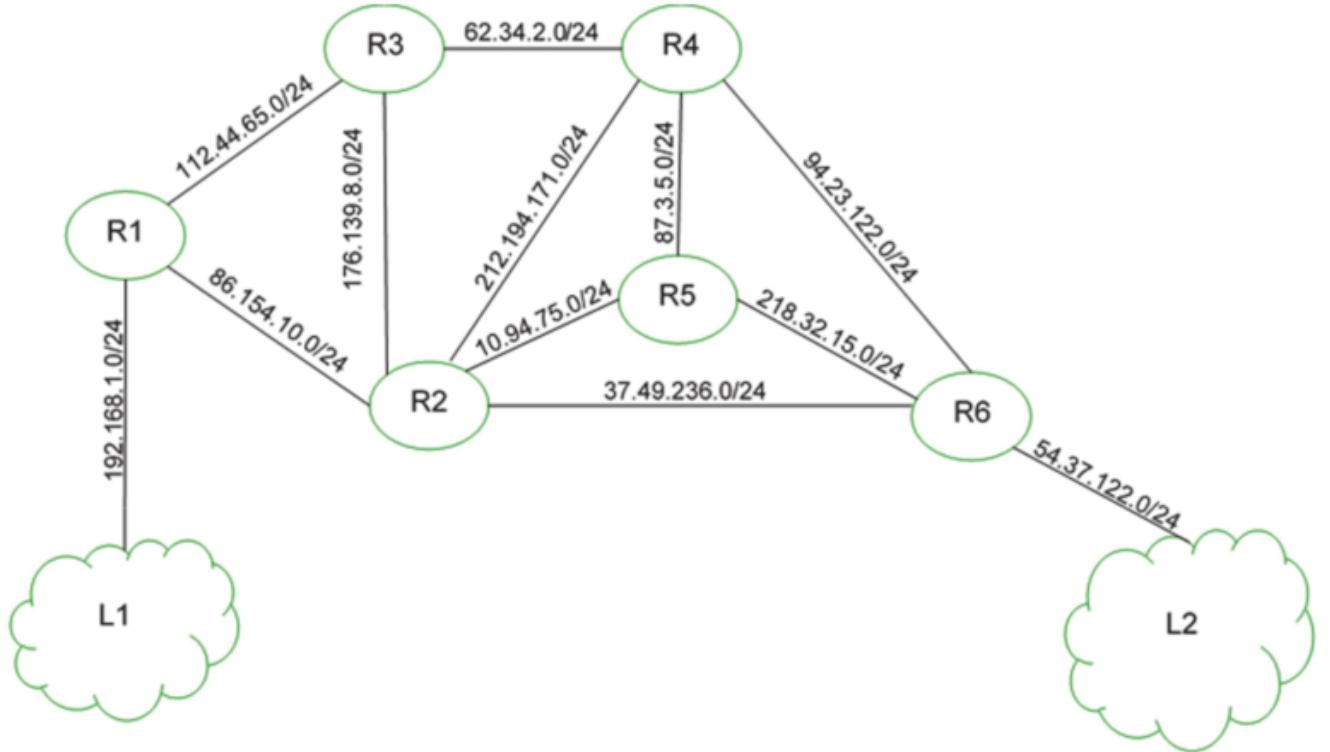
Le routeur A doit transmettre un message au routeur G, en empruntant le chemin dont la somme des coûts sera la plus petite possible. Déterminer le chemin parcouru. On indiquera le raisonnement utilisé.

#### 3.2. 4.3.2 Sujet n°2



Cet exercice porte sur les réseaux et les protocoles de routage.

On représente ci-dessous un réseau dans lequel R1, R2, R3, R4, R5 et R6 sont des routeurs. Le réseau local L1 est relié au routeur R1 et le réseau local L2 au routeur R6.



### Rappels et notations

Dans cet exercice, les adresses IP sont composées de 4 octets, soit 32 bits. Elles sont notées X1.X2.X3.X4, où X1, X2, X3 et X4 sont les valeurs des 4 octets, convertis en notation décimale.

La notation X1.X2.X3.X4/n signifie que les n premiers bits de poids forts de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

Toutes les adresses des hôtes connectés à un réseau local ont la même partie réseau et peuvent donc communiquer directement. L'adresse IP dont tous les bits de la partie « hôte » sont à 0 est appelée « adresse du réseau ».

On donne également des extraits de la table de routage des routeurs R1 à R5 dans le tableau suivant :

Routeur	Réseau destinataire	Passerelle	Interface
R1	54.37.122.0/24	86.154.10.1	86.154.10.56
R2	54.37.122.0/24	37.49.236.22	37.49.236.23
R3	54.37.122.0/24	62.34.2.8	62.34.2.9
R4	54.37.122.0/24	94.23.122.10	94.23.122.11
R5	54.37.122.0/24	218.32.15.1	218.32.15.2

### Question 1

Un paquet part du réseau local L1 à destination du réseau local L2.

- En utilisant l'extrait de la table de routage de R1, vers quel routeur R1 envoie-t-il ce paquet : R2 ou R3 ? Justifier.
- A l'aide des extraits de tables de routage ci-dessus, nommer les routeurs traversés par ce paquet, lorsqu'il va du réseau L1 au réseau L2.

## Question 2

La liaison entre R1 et R2 est rompue.

- Sachant que ce réseau utilise le protocole RIP (distance en nombre de sauts), donner l'un des deux chemins possibles que pourra suivre un paquet allant de L1 vers L2.
- Dans les extraits de tables de routage ci-dessus, pour le chemin de la question 2.a, quelle(s) ligne(s) sera (seront) modifiée(s) ?

## Question 3

On a rétabli la liaison entre R1 et R2.

Par ailleurs, pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût minimal des liaisons) pour effectuer le routage. Le coût des liaisons entre les routeurs est donné par le tableau suivant :

Liaison	R1-R2	R1-R3	R2-R3	R2-R4	R2-R5	R2-R6	R3-R4	R4-R5	R4-R6	R5-R6
Coût	100	100	?	1	10	10	10	1	10	1

- Le coût  $(C)$  d'une liaison est donné ici par la formule :

$$(C = \frac{10^9}{BP})$$

où  $(BP)$  est la bande passante de la connexion en bps (bit par seconde).

Sachant que la bande passante de la liaison R2-R3 est de 10 Mbps, calculer le coût correspondant.

- Déterminer le chemin parcouru par un paquet partant du réseau L1 et arrivant au réseau L2, en utilisant le protocole OSPF.

- Indiquer pour quel(s) routeur(s) l'extrait de la table de routage sera modifié pour un paquet à destination de L2, avec la métrique OSPF.

 Annexe

R1 :

IP réseau de destination	Passerelle suivante	Interface
192.168.1.0/24	192.168.1.1	Interface 2
10.1.1.0/24	10.1.1.2	Interface 1
0.0.0.0/0	10.1.1.1	Interface 1

R2 :

IP réseau de destination	Passerelle suivante	Interface
10.1.1.0/24	10.1.1.1	Interface 1
10.1.2.0/24	10.1.2.1	Interface 2
10.1.3.0/24	10.1.3.1	Interface 3
192.168.1.0/24	10.1.1.2	Interface 2
172.16.0.0/16	10.1.3.2	Interface 3

R5 :

IP réseau de destination	Passerelle suivante	Interface
10.1.3.0/24	10.1.3.2	Interface 1
10.1.4.0/24	10.1.4.2	Interface 2
10.1.6.0/24	10.1.6.2	Interface 3
10.1.7.0/24	10.1.7.1	Interface 4

R6 :

IP réseau de destination	Passerelle suivante	Interface
172.16.0.0/16	172.16.0.1	Interface 1

## 3.3. 4.3.3 Sujet n°3 : 2022, Métropole, J2



## Après 2022, Métropole, J2, Ex. 3

**Rappels :**

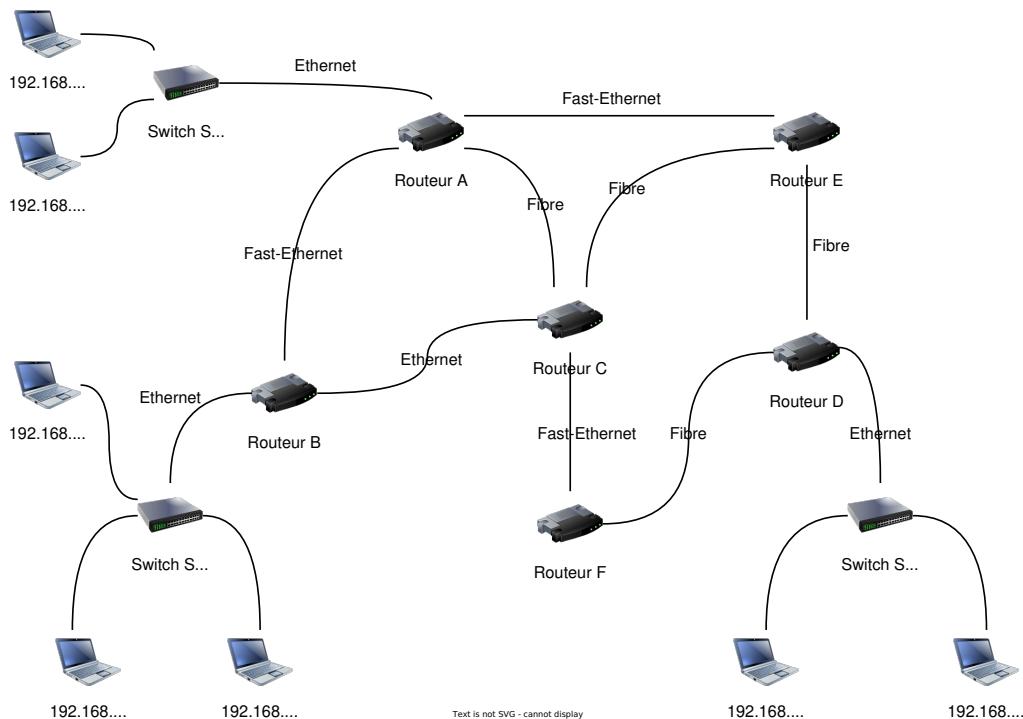
Une adresse IPv4 est composée de 4 octets, soit 32 bits. Elle est notée a.b.c.d, où a, b, c et d sont les valeurs des 4 octets.

La notation a.b.c.d/n signifie que les n premiers bits de l'adresse IP représentent la partie « réseau », les bits qui suivent représentent la partie « machine ».

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 0 est appelée « adresse du réseau ».

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 1 est appelée « adresse de diffusion ».

On considère le réseau représenté sur la ci-dessous :



### Question 1

On considère la machine d'adresse IPv4 192.168.1.1.

- Donner l'adresse du réseau sur lequel se trouve cette machine.
- Donner l'adresse de diffusion (broadcast) de ce réseau.
- Donner le nombre maximal de machines que l'on peut connecter sur ce réseau.
- On souhaite ajouter une machine sur ce réseau, proposer une adresse IPv4 possible pour cette machine.

### Question 2

La machine d'adresse IPv4 192.168.1.1 transmet un paquet IPv4 à la machine d'adresse IPv4 192.168.4.2.

- Donner toutes les routes pouvant être empruntées par ce paquet IPv4, chaque routeur ne pouvant être traversé qu'une seule fois.
- Expliquer l'utilité d'avoir plusieurs routes possibles reliant les réseaux 192.168.1.0/24 et 192.168.4.0/24.

### Question 3

Dans cette question, on suppose que le protocole de routage mis en place dans le réseau est RIP. Ce protocole consiste à minimiser le nombre de sauts.

Le schéma du réseau est celui de la figure ci-dessus.

Les tables de routage utilisées sont données ci-dessous :

Routeur A	Routeur B	Routeur C	Routeur D	Routeur E	Routeur F
-----------	-----------	-----------	-----------	-----------	-----------

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

B	...
---	-----

C	...
---	-----

D	E
---	---

E	...
---	-----

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

C	C
---	---

D	C
---	---

E	C
---	---

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

B	B
---	---

D	E
---	---

E	E
---	---

F	F
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	E
---	---

B	F
---	---

C	F
---	---

E	E
---	---

F	F
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

B	C
---	---

C	C
---	---

D	D
---	---

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	C
---	---

B	C
---	---

C	C
---	---

D	D
---	---

E	C
---	---

- 3.a.** Recopier et compléter sur la copie la table de routage du routeur A.
- 3.b.** Un paquet IP doit aller du routeur B au routeur D. En utilisant les tables de routage, donner le parcours emprunté par celui-ci.
- 3.c.** Les connexions entre les routeurs B-C et A-E étant coupées, sur la copie, réécrire les tables de routage des routeurs A, B et C.
- 3.d.** Déterminer le nouveau parcours emprunté par le paquet IP pour aller du routeur B au routeur D.

#### Question 4

Dans cette question, on suppose que le protocole de routage mis en place dans le réseau est OSPF. Ce protocole consiste à minimiser la somme des coûts des liaisons empruntées.

Le coût d'une liaison est défini par la relation  $\text{coût} = \frac{10^8}{d}$  où  $d$  représente le débit en  $\text{bit}/\text{s}$  et  $\text{coût}$  est sans unité. Le schéma du réseau reste celui du début de l'exercice.

- 4.a.** Déterminer le coût des liaisons Ethernet ( $d = 10^7 \text{ bit/s}$ ), Fast-Ethernet ( $d = 10^8 \text{ bit/s}$ ) et Fibre ( $d = 10^9 \text{ bit/s}$ ).

- 4.b.** On veut représenter schématiquement le réseau de routeurs à partir du schéma du réseau.

Recopier sur la copie le schéma ci-dessous et tracer les liaisons entre les routeurs en y indiquant leur coût.

```
graph LR
 A((A)) --- B((B))
 A --- C((C))
 A --- E((E))
 B --- C
 C --- E
 E --- D((D))
 D --- F((F))
 C --- F
 LinkStyle 0 stroke-width:0;
 LinkStyle 1 stroke-width:0;
 LinkStyle 2 stroke-width:0;
 LinkStyle 3 stroke-width:0;
 LinkStyle 4 stroke-width:0;
 LinkStyle 5 stroke-width:0;
 LinkStyle 6 stroke-width:0;
 LinkStyle 7 stroke-width:0;
```

- 4.c.** Un paquet IPv4 doit être acheminé d'une machine ayant pour adresse IPv4 192.168.2.1 à une machine ayant pour adresse IPv4 192.168.4.1.

Écrire les routes possibles, c'est à dire la liste des routeurs traversés, et le coût de chacune de ces routes, chaque routeur ne pouvant être traversé qu'une seule fois.

- 4.d.** Donner, en la justifiant, la route qui sera empruntée par un paquet IPv4 pour aller d'une machine ayant pour adresse IPv4 192.168.2.1 à une machine ayant pour adresse IPv4 192.168.4.1.

#### 3.4. 4.3.4 Sujet n°4 : 2022, Métropole, J1



J'après 2022, Métropole J1

### Question 1

Une adresse IPv4 est représentée sous la forme de 4 nombres entiers positifs séparés par des points. Chacun de ces 4 entiers peut être représenté sur un octet.

**1.a.** Donner en écriture décimale l'adresse IPv4 correspondant à l'écriture binaire :

11000000.10101000.10000000.10000011

**1.b** Tous les ordinateurs du réseau A ont une adresse IPv4 de la forme : 192.168.128.\_\_\_\_\_, où seul le dernier octet (représenté par \_\_\_\_\_) diffère.

Donner le nombre d'adresses différentes possibles du réseau A.

## Question 2

On rappelle que le protocole RIP cherche à minimiser le nombre de routeurs traversés (qui correspond à la métrique). On donne les tables de routage d'un réseau informatique composé de 5 routeurs (appelés A, B, C, D et E), chacun associé directement à un réseau du même nom, obtenues avec le protocole RIP :

Routeur A	Routeur B	Routeur C	Routeur D	Routeur E
Destination	Métrique			
A	0			
B	1			
C	1			
D	1			
E	2			

Destination	Métrique			
A	1			
B	0			
C	2			
D	1			
E	2			

Destination	Métrique			
A	1			
B	2			
C	0			
D	1			
E	2			

Destination	Métrique			
A	1			
B	1			
C	1			
D	0			
E	1			

Destination	Métrique			
A	2			
B	2			
C	2			
D	1			
E	0			

**2.a.** Donner la liste des routeurs avec lesquels le routeur A est directement relié.

**2.b.** Représenter graphiquement et de manière sommaire les 5 routeurs ainsi que les liaisons existantes entre ceux-ci.

### Question 3

Le protocole OSPF est un protocole de routage qui cherche à minimiser la somme des métriques des liaisons entre routeurs.

Dans le protocole de routage OSPF le débit des liaisons entre routeurs agit sur la métrique via la relation :  $\text{métrique} = \frac{10^8}{\text{débit}}$  dans laquelle le débit est exprimé en bit par seconde ( $\text{bps}$ ).

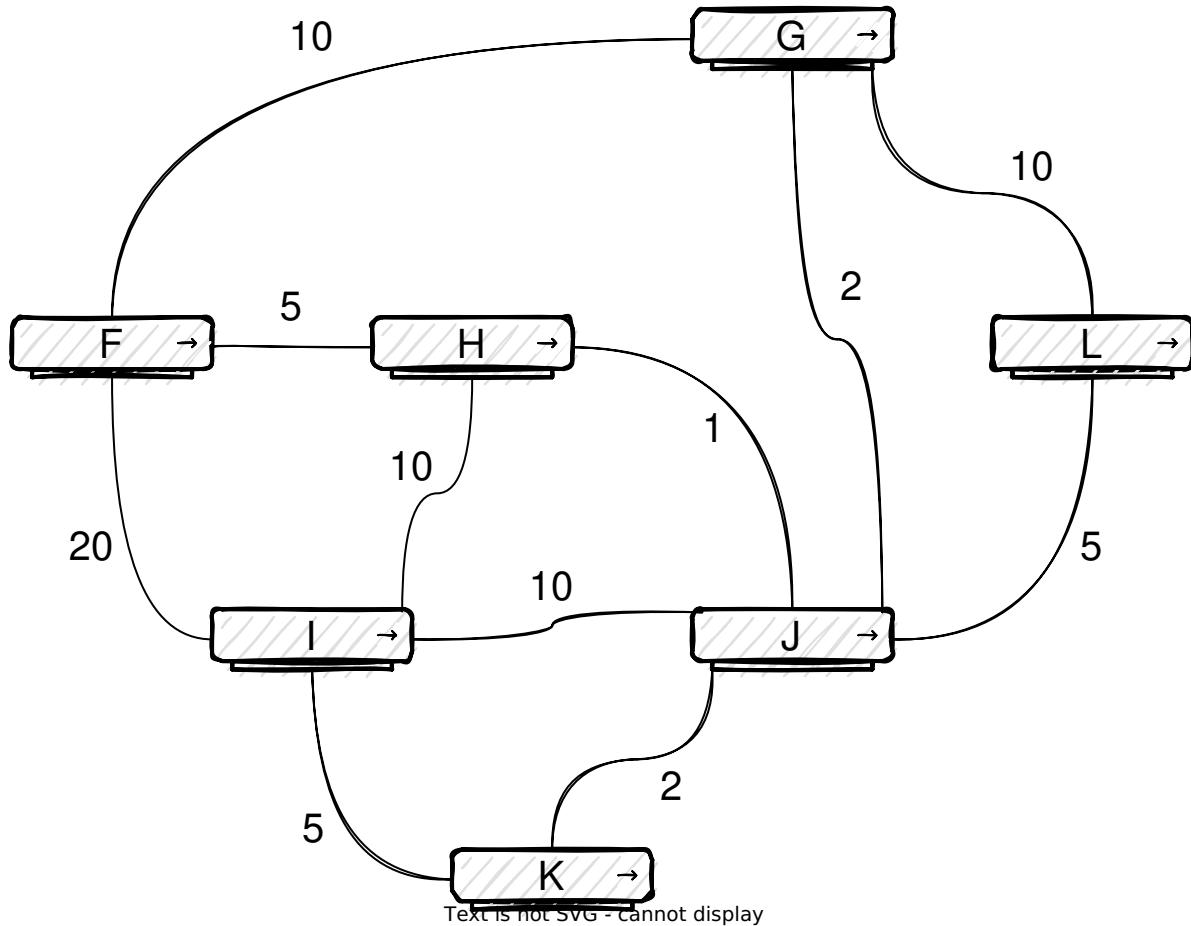
On rappelle qu'un  $\text{kbps}$  est égal à  $10^3 \text{ bps}$  et qu'un  $\text{Mbps}$  est égal à  $10^6 \text{ bps}$ .

Recopier sur votre copie et compléter le tableau suivant :

Débit	$(100 \text{ kbps})$	$(500 \text{ kbps})$	...	$(100 \text{ Mbps})$
Métrique associée	$(1000)$	...	$(10)$	$(1)$

### Question 4

Voici la représentation d'un réseau et la table de routage incomplète du routeur F obtenue avec le protocole OSPF :



#### Routeur F

Destination	Métrique
F	0
G	8
H	5
I	
J	
K	
L	

Les nombres présents sur les liaisons représentent les coûts des routes avec le protocole OSPF.

- 4.a.** Indiquer le chemin emprunté par un message d'un ordinateur du réseau F à destination d'un ordinateur du réseau I. Justifier votre réponse.

**4.b.** Recopier et compléter la table de routage du routeur F.

**4.c.** Citer une unique panne qui suffirait à ce que toutes les données des échanges de tout autre réseau à destination du réseau F transitent par le routeur G. Expliquer en détail votre réponse.

#### 4. 4.4 Thème 3 - Architecture matérielle

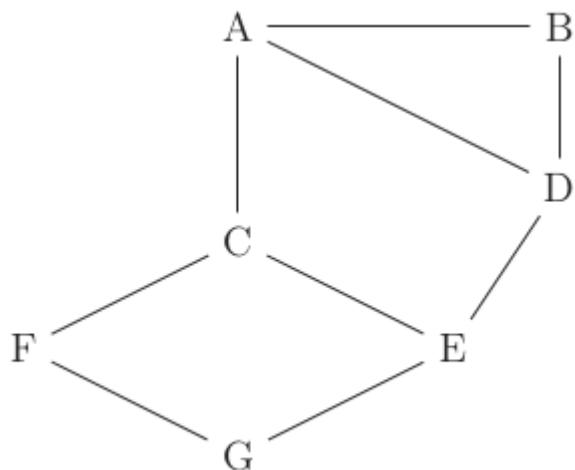
# BAC

# Protocole de Routage

#### 4.1. 4.4.1 Sujet n°1 : sujet zéro

### Sujet zéro

On considère un réseau composé de plusieurs routeurs reliés de la façon suivante :



#### 4.1.1. → Le protocole RIP

Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur la distance, en nombre de sauts, qui le sépare d'un autre routeur. Pour le réseau ci-dessus, on dispose des tables de routage suivantes :

Table de routage du routeur A		
Destination	Routeur suivant	Distance
B	B	1
C	C	1
D	D	1
E	C	2
F	C	2
G	C	3

Table de routage du routeur B		
Destination	Routeur suivant	Distance
A	A	1
C	A	2
D	D	1
E	D	2
F	A	3
G	D	3

Table de routage du routeur C		
Destination	Routeur suivant	Distance
A	A	1
B	A	2
D	E	2
E	E	1
F	F	1
G	F	2

Table de routage du routeur D		
Destination	Routeur suivant	Distance
A	A	1
B	B	1
C	E	2
E	E	1
F	A	3
G	E	2

Table de routage du routeur E		
Destination	Routeur suivant	Distance
A	C	2
B	D	2
C	C	1
D	D	1
F	G	2
G	G	1

Table de routage du routeur F		
Destination	Routeur suivant	Distance
A	C	2
B	C	3
C	C	1
D	C	3
E	G	2
G	G	1

### Question 1

- Le routeur A doit transmettre un message au routeur G, en effectuant un nombre minimal de sauts. Déterminer le trajet parcouru.
- Déterminer une table de routage possible pour le routeur G obtenu à l'aide du protocole RIP.

**Réponse**

1. trajet possible ACFG 'Lecture des table de routage. La distance est de 3.

Destination	Routeur suivant	Distance
A	F	3
B	E	3
C	F	2
D	E	2
E	E	1
F	F	1

**Question 2**

Le routeur C tombe en panne. Reconstruire la table de routage du routeur A en suivant le protocole RIP.

**Réponse**

Destination	Routeur suivant	Distance
B	B	1
D	D	1
E	D	2
F	D	4
G	D	3

**4.1.2. → Le protocole OSPF**

Contrairement au protocole RIP, l'objectif n'est plus de minimiser le nombre de routeurs traversés par un paquet. La notion de distance utilisée dans le protocole OSPF est uniquement liée aux coûts des liaisons.

L'objectif est alors de minimiser la somme des coûts des liaisons traversées.

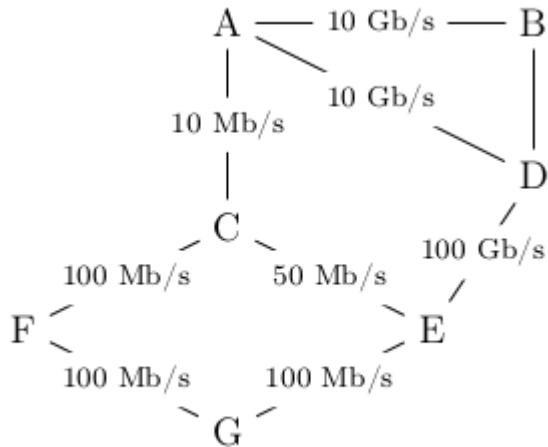
Le coût d'une liaison est donné par la formule suivante :

$$\text{Coût} = \frac{10^8}{d}$$

où  $d$  est la bande passante en bits/s entre les deux routeurs.

On a rajouté sur le graphe représentant le réseau précédent les différents débits des liaisons.

On rappelle que 1 Gb/s = 1 000 Mb/s =  $10^9$  bits/s.



### Question 3

1. Vérifier que le coût de la liaison entre les routeurs A et B est 0,01.
2. La liaison entre le routeur B et D a un coût de 5. Quel est le débit de cette liaison ?

### Réponse

3.1

$A \rightarrow B : 10 \text{ Gb/s}$  soit un coût :  $\frac{10^8}{10 \times 10^9} = 0.01$

3.2

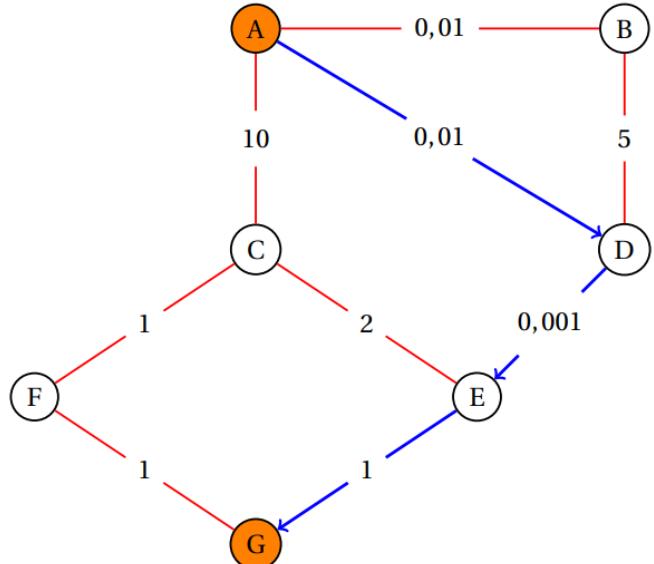
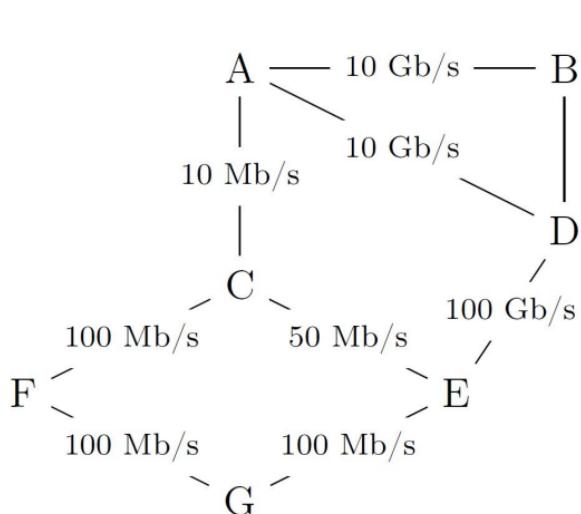
$\frac{10^8}{5} \Rightarrow d = \frac{10^8}{5} = 2 \times 10^7 \text{ b/s}$  soit 20 Mb/s

### Question 4

Le routeur A doit transmettre un message au routeur G, en empruntant le chemin dont la somme des coûts sera la plus petite possible. Déterminer le chemin parcouru. On indiquera le raisonnement utilisé.

**Reponse**

Débit $d$ bits/s	$\text{Coût} = \frac{10^8}{d}$
$d = 10Gb/s = 10^{10} \text{ bits/s}$	$\text{coût} = 10^8 \times 10^{-10} = 10^{-2} = 0,01$
$d = 100Gb/s = 10^{11} \text{ bits/s}$	$\text{coût} = 10^8 \times 10^{-11} = 10^{-3} = 0,001$
$d = 100Mb/s = 10^8 \text{ bits/s}$	$\text{coût} = 10^8 \times 10^{-8} = 1$
$d = 50Mb/s = 5 \times 10^7 \text{ bits/s}$	$\text{coût} = 0,2 \times 10^8 \times 10^{-7} = 2$
$d = 10Mb/s = 10^7 \text{ bits/s}$	$\text{coût} = 10^8 \times 10^{-7} = 10$



Départ	A	B	C	D	E	F	G	Sommet choisi
	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	A
A(0)		0,01(A)	10(A)	0,01(A)	$\infty$	$\infty$	$\infty$	B(A)
B(0,01)			10(A)	5,01(B) 0,01(A)	$\infty$	$\infty$	$\infty$	D(A)
D(0,01)			10(A)		0,011(D)	$\infty$	$\infty$	E(D)
E(0,011)			10(A) 2,011(E)			$\infty$	1,011(E)	G(E)
G(1,011)						2,011(G)		F(G)
F(2,011)			3,011(F) 2,011(E)					C(E)

Le parcourt avec un coût minimal pour aller de A à G est donc ADEG dont le coût est 1,011.

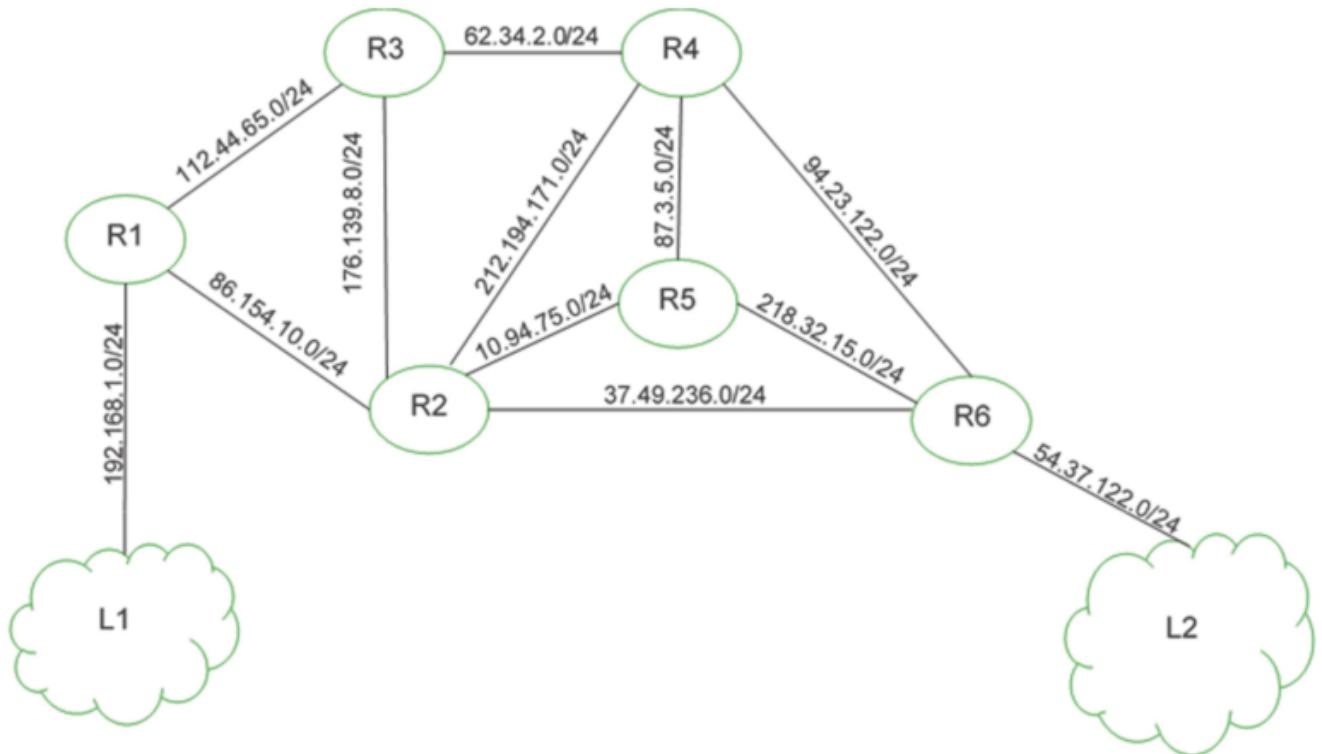
### Correction du tableau de l'algorithme de Dijkstra

#### 4.2. 4.4.2 Sujet n°2



Cet exercice porte sur les réseaux et les protocoles de routage.

On représente ci-dessous un réseau dans lequel R1, R2, R3, R4, R5 et R6 sont des routeurs. Le réseau local L1 est relié au routeur R1 et le réseau local L2 au routeur R6.



#### Rappels et notations

Dans cet exercice, les adresses IP sont composées de 4 octets, soit 32 bits. Elles sont notées X1.X2.X3.X4, où X1, X2, X3 et X4 sont les valeurs des 4 octets, convertis en notation décimale.

La notation X1.X2.X3.X4/n signifie que les n premiers bits de poids forts de l'adresse IP représentent la partie « réseau », les bits suivants représentent la partie « hôte ».

Toutes les adresses des hôtes connectés à un réseau local ont la même partie réseau et peuvent donc communiquer directement. L'adresse IP dont tous les bits de la partie « hôte » sont à 0 est appelée « adresse du réseau ».

On donne également des extraits de la table de routage des routeurs R1 à R5 dans le tableau suivant :

Routeur	Réseau destinataire	Passerelle	Interface
R1	54.37.122.0/24	86.154.10.1	86.154.10.56
R2	54.37.122.0/24	37.49.236.22	37.49.236.23
R3	54.37.122.0/24	62.34.2.8	62.34.2.9
R4	54.37.122.0/24	94.23.122.10	94.23.122.11
R5	54.37.122.0/24	218.32.15.1	218.32.15.2

### Question 1

Un paquet part du réseau local L1 à destination du réseau local L2.

- a. En utilisant l'extrait de la table de routage de R1, vers quel routeur R1 envoie-t-il ce paquet : R2 ou R3 ? Justifier.

- b. A l'aide des extraits de tables de routage ci-dessus, nommer les routeurs traversés par ce paquet, lorsqu'il va du réseau L1 au réseau L2.

### Réponse 1.a

L'extrait de la table de routage de R1 montre que pour atteindre le réseau L2 (57.37.122.0/24) les paquets doivent être envoyés via l'interface 86.154.10.56. Cette interface fait partie du réseau 86.154.10.0/24. Le routeur R2 fait aussi partie de ce réseau. On peut donc affirmer que depuis R1, les paquets seront dirigés vers R2.

### Réponse 1.b

L1 -> R1 -> R2 -> R6 -> L2

### Question 2

La liaison entre R1 et R2 est rompue.

- a. Sachant que ce réseau utilise le protocole RIP (distance en nombre de sauts), donner l'un des deux chemins possibles que pourra suivre un paquet allant de L1 vers L2.  
 b. Dans les extraits de tables de routage ci-dessus, pour le chemin de la question 2.a, quelle(s) ligne(s) sera (seront) modifiée(s) ?

### Réponse 2.a

L1 -> R1 -> R3 -> R4 -> R6 -> L2

### Réponse 2.b

Vu le chemin choisi à la question 2a, seule la ligne R1 sera modifiée (réseau 112.44.65.0 à la place du réseau 86.154.10.0).

### Question 3

On a rétabli la liaison entre R1 et R2.

Par ailleurs, pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût minimal des liaisons) pour effectuer le routage. Le coût des liaisons entre les routeurs est donné par le tableau suivant :

Liaison	R1-R2	R1-R3	R2-R3	R2-R4	R2-R5	R2-R6	R3-R4	R4-R5	R4-R6	R5-R6
Coût	100	100	?	1	10	10	10	1	10	1

a. Le coût  $(C)$  d'une liaison est donné ici par la formule :

$$(C = \frac{10^9}{BP})$$

où  $(BP)$  est la bande passante de la connexion en bps (bit par seconde).

Sachant que la bande passante de la liaison R2-R3 est de 10 Mbps, calculer le coût correspondant.

b. Déterminer le chemin parcouru par un paquet partant du réseau L1 et arrivant au réseau L2, en utilisant le protocole OSPF.

c. Indiquer pour quel(s) routeur(s) l'extrait de la table de routage sera modifié pour un paquet à destination de L2, avec la métrique OSPF.

### Réponse 2.a

$$(C = \frac{10^9}{10^7} = 100)$$

### Réponse 2.b

La route avec le coût minimum (103) est la suivante :

L1 -> R1 -> R2 -> R4 -> R5 -> R6 -> L2

### Réponse 2.c

Les tables de routage R2 et R4 seront modifiées.

 Annexe

R1 :

IP réseau de destination	Passerelle suivante	Interface
192.168.1.0/24	192.168.1.1	Interface 2
10.1.1.0/24	10.1.1.2	Interface 1
0.0.0.0/0	10.1.1.1	Interface 1

R2 :

IP réseau de destination	Passerelle suivante	Interface
10.1.1.0/24	10.1.1.1	Interface 1
10.1.2.0/24	10.1.2.1	Interface 2
10.1.3.0/24	10.1.3.1	Interface 3
192.168.1.0/24	10.1.1.2	Interface 2
172.16.0.0/16	10.1.3.2	Interface 3

R5 :

IP réseau de destination	Passerelle suivante	Interface
10.1.3.0/24	10.1.3.2	Interface 1
10.1.4.0/24	10.1.4.2	Interface 2
10.1.6.0/24	10.1.6.2	Interface 3
10.1.7.0/24	10.1.7.1	Interface 4

R6 :

IP réseau de destination	Passerelle suivante	Interface
172.16.0.0/16	172.16.0.1	Interface 1

## 4.3. 4.4.3 Sujet n°3 : 2022, Métropole, J2



## Après 2022, Métropole, J2, Ex. 3

**Rappels :**

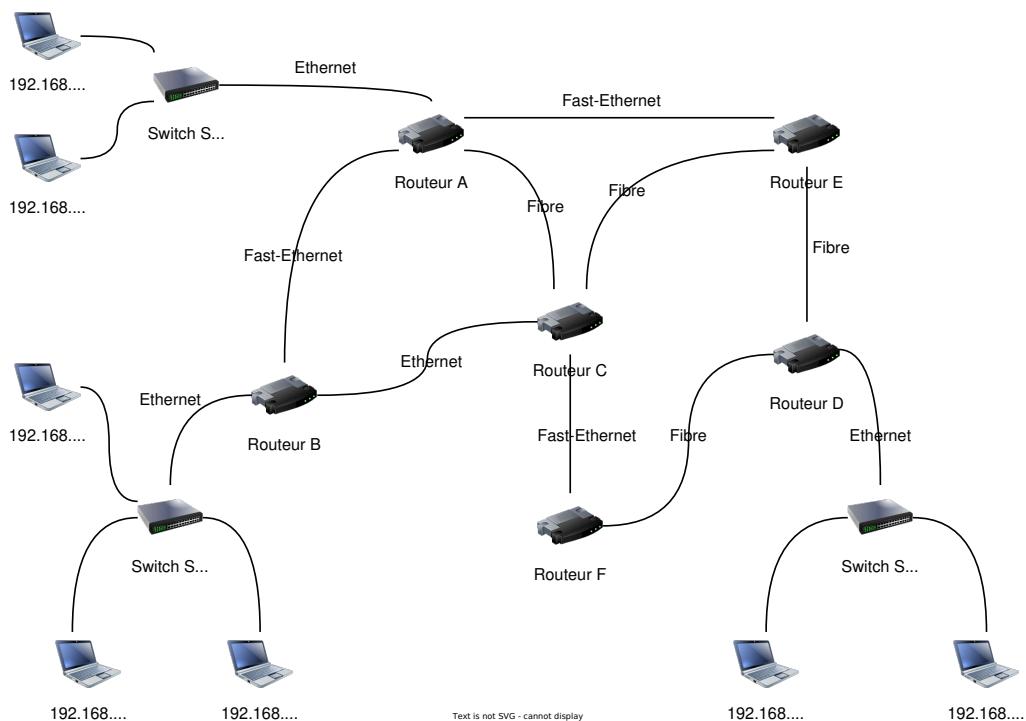
Une adresse IPv4 est composée de 4 octets, soit 32 bits. Elle est notée a.b.c.d, où a, b, c et d sont les valeurs des 4 octets.

La notation a.b.c.d/n signifie que les n premiers bits de l'adresse IP représentent la partie « réseau », les bits qui suivent représentent la partie « machine ».

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 0 est appelée « adresse du réseau ».

L'adresse IPv4 dont tous les bits de la partie « machine » sont à 1 est appelée « adresse de diffusion ».

On considère le réseau représenté sur la ci-dessous :



1. On considère la machine d'adresse IPv4 192.168.1.1 .

### Question 1.a

Donner l'adresse du réseau sur lequel se trouve cette machine.

### Réponse 1.a

On lit sur la figure la dénomination suivante : 192.168.1.0/24 . Les 24 premiers bits, trois octets, représentent l'adresse réseau : celle-ci est donc 192.168.1.0 .

### Question 1.b

Donner l'adresse de diffusion (broadcast) de ce réseau.

### Réponse 1.b

Les 8 derniers bits, le dernier octet, prennent la valeur 1 . Donc l'adresse de diffusion est 192.168.1.255 .

### Question 1.c

Donner le nombre maximal de machines que l'on peut connecter sur ce réseau.

**Réponse 1.c**

On peut connecter  $256 - 2 = 254$  machines sur ce réseau.

**Question 1.d**

On souhaite ajouter une machine sur ce réseau, proposer une adresse IPv4 possible pour cette machine.

**Réponse 1.d**

On propose 192.168.1.17 .

**2**

**2.** La machine d'adresse IPv4 192.168.1.1 transmet un paquet IPv4 à la machine d'adresse IPv4 192.168.4.2 .

**Question 2.a**

Donner toutes les routes pouvant être empruntées par ce paquet IPv4, chaque routeur ne pouvant être traversé qu'une seule fois.

**Réponse**

Les routes possibles sont :

- A → E → D
- A → E → C → F → D
- A → B → C → E → D
- A → B → C → F → D
- A → C → E → D
- A → C → F → D

**Question 2.b**

Expliquer l'utilité d'avoir plusieurs routes possibles reliant les réseaux 192.168.1.0/24 et 192.168.4.0/24 .

**Réponse**

En cas de panne, on pourra utiliser une autre route.

### Question 3

Dans cette question, on suppose que le protocole de routage mis en place dans le réseau est RIP. Ce protocole consiste à minimiser le nombre de sauts.

Le schéma du réseau est celui de la figure ci-dessus.

Les tables de routage utilisées sont données ci-dessous :

Routeur A	Routeur B	Routeur C	Routeur D	Routeur E	Routeur F
-----------	-----------	-----------	-----------	-----------	-----------

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

B	...
---	-----

C	...
---	-----

D	E
---	---

E	...
---	-----

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

C	C
---	---

D	C
---	---

E	C
---	---

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

B	B
---	---

D	E
---	---

E	E
---	---

F	F
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	E
---	---

B	F
---	---

C	F
---	---

E	E
---	---

F	F
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	A
---	---

B	C
---	---

C	C
---	---

D	D
---	---

F	C
---	---

<b>Destination</b>	<b>passee par</b>
--------------------	-------------------

A	C
---	---

B	C
---	---

C	C
---	---

D	D
---	---

E	C
---	---

### Question 3.a

Recopier et compléter sur la copie la table de routage du routeur A.

### Réponse

Table de routage de A :

Destination	Passe par
B	B
C	C
D	E
E	E
F	C

### Question 3.b

Un paquet IP doit aller du routeur B au routeur D. En utilisant les tables de routage, donner le parcours emprunté par celui-ci.

### Réponse

Le paquet suit le trajet suivant :

- B → C (table de routage de B)
- C → E (table de routage de C)
- E → D (table de routage de E)

### Question 3.c

Les connexions entre les routeurs B-C et A-E étant coupées, sur la copie, réécrire les tables de routage des routeurs A, B et C.

### Réponse

Routeur A      Routeur B      Routeur C

Destination	Passe par
B	B
C	C
D	C
E	C
F	C

Destination	Passe par
A	A
C	A
D	A
E	A
F	A

Destination	Passe par
A	A
B	A
D	E
E	E
F	F

### Question 3.d

Déterminer le nouveau parcours emprunté par le paquet IP pour aller du routeur B au routeur D.

### Réponse

Le nouveau parcours est : B → A → C → E → D.

### Question 4

Dans cette question, on suppose que le protocole de routage mis en place dans le réseau est OSPF. Ce protocole consiste à minimiser la somme des coûts des liaisons empruntées.

Le coût d'une liaison est défini par la relation  $\text{coût} = \frac{10^8}{d}$  où  $d$  représente le débit en  $\text{bit}/\text{second}$  et  $\text{coût}$  est sans unité. Le schéma du réseau reste celui du début de l'exercice.

### Question 4.a

Déterminer le coût des liaisons Ethernet ( $(d = 10^7, \text{bit}/\text{s})$ ), Fast-Ethernet ( $(d = 10^8, \text{bit}/\text{s})$ ) et Fibre ( $(d = 10^9, \text{bit}/\text{s})$ ).

### Réponse

Les coûts sont les suivants :

- liaison Ethernet :  $(\frac{10^8}{10^7} = 10)$ ,
- liaison Fast-Ethernet :  $(\frac{10^8}{10^8} = 1)$ ,
- liaison Fibre :  $(\frac{10^8}{10^9} = 0,1)$

### Attention

Dans le protocole OSPF, les coûts sont normalement des nombres entiers strictement positifs (entre  $(1)$  et  $(65,535)$ ). On peut donc aussi arrondir par excès le coût de la liaison Fibre à  $(1)$ .

### Question 4.b

On veut représenter schématiquement le réseau de routeurs à partir du schéma du réseau.

Recopier sur la copie le schéma ci-dessous et tracer les liaisons entre les routeurs en y indiquant leur coût.

```
graph LR
 A((A)) --- B((B))
 A --- C((C))
 A --- E((E))
 B --- C
 C --- E
 E --- D((D))
 D --- F((F))
 C --- F
 LinkStyle 0 stroke-width:0;
 LinkStyle 1 stroke-width:0;
 LinkStyle 2 stroke-width:0;
 LinkStyle 3 stroke-width:0;
 LinkStyle 4 stroke-width:0;
 LinkStyle 5 stroke-width:0;
 LinkStyle 6 stroke-width:0;
 LinkStyle 7 stroke-width:0;
```

## Reponse

Sans arrondi      Avec arrondi

On conserve ici la valeur de \((0,1)\) pour le coût de la liaison Fibre.

```
graph LR
 A["A((A))"] --- |1| B["B((B))"]
 A --- |0,1| C["C((C))"]
 A --- |1| E["E((E))"]
 B --- |10| C
 C --- |0,1| E
 E --- |0,1| D["D((D))"]
 D --- |0,1| F["F((F))"]
 C --- |1| F
```

On arrondit le coût de la liaison Fibre à \((1)\).

```
graph LR
 A["A((A))"] --- |1| B["B((B))"]
 A --- |1| C["C((C))"]
 A --- |1| E["E((E))"]
 B --- |10| C
 C --- |1| E
 E --- |1| D["D((D))"]
 D --- |1| F["F((F))"]
 C --- |1| F
```

## Question 4.c

Un paquet IPv4 doit être acheminé d'une machine ayant pour adresse IPv4 192.168.2.1 à une machine ayant pour adresse IPv4 192.168.4.1 .

Écrire les routes possibles, c'est à dire la liste des routeurs traversés, et le coût de chacune de ces routes, chaque routeur ne pouvant être traversé qu'une seule fois.

### Reponse

Sans arrondi      Sans arrondi

On conserve la valeur de  $\backslash(0,1\backslash)$  pour le coût de la liaison Fibre. On aura donc :

Route	Coût
B → A → E → D	2,1
B → C → A → E → D	11,2
B → A → C → F → D	2,2
B → A → C → E → D	1,3
B → A → E → C → F → D	3,2
B → C → F → D	11,1
B → C → E → D	10,2

On arrondit le coût de la liaison Fibre à  $\backslash(1\backslash)$ . On aura donc :

Route	Coût
B → A → E → D	3
B → C → A → E → D	13
B → A → C → F → D	4
B → A → C → E → D	4
B → A → E → C → F → D	5
B → C → F → D	12
B → C → E → D	12

### Question 4.d

Donner, en la justifiant, la route qui sera empruntée par un paquet IPv4 pour aller d'une machine ayant pour adresse IPv4 192.168.2.1 à une machine ayant pour adresse IPv4 192.168.4.1.

### Reponse

On choisit le chemin de coût minimal :

- si l'on a conservé un coût de  $\backslash(0,1\backslash)$  pour la liaison Fibre, on obtient le chemin B → A → C → E → D pour un coût de  $\backslash(1,3\backslash)$  ;
- si l'on arrondit ce coût à  $\backslash(1\backslash)$ , on obtient le chemin B → A → E → D pour un coût de  $\backslash(3\backslash)$ .

## 4.4. 4.4.4 Sujet 4 : 2022, Métropole, J1

### J'après 2022, Métropole J1

### Question 1

Une adresse IPv4 est représentée sous la forme de 4 nombres entiers positifs séparés par des points. Chacun de ces 4 entiers peut être représenté sur un octet.

#### Question 1.a

Donner en écriture décimale l'adresse IPv4 correspondant à l'écriture binaire : 11000000.10101000.10000000.10000011

#### Réponse

\(1 + 2 + 128 = 131\), ainsi l'adresse est 192.168.128.131 .

#### Question 1.b

Tous les ordinateurs du réseau A ont une adresse IPv4 de la forme : 192.168.128.\_\_\_\_\_, où seul le dernier octet (représenté par \_\_\_\_ ) diffère.

Donner le nombre d'adresses différentes possibles du réseau A.

#### Réponse

Sur les **256** adresses possibles avec 1 octet on trouvera :

- la valeur 0 qui est réservée pour l'adresse IP du réseau ;
- les valeurs 1 à 254 qui peuvent être utilisées pour les adresses des hôtes dans le réseau ;
- la valeur 255 qui est réservée pour l'adresse de diffusion du réseau.

## Question 2

On rappelle que le protocole RIP cherche à minimiser le nombre de routeurs traversés (qui correspond à la métrique). On donne les tables de routage d'un réseau informatique composé de 5 routeurs (appelés A, B, C, D et E), chacun associé directement à un réseau du même nom, obtenues avec le protocole RIP :

Routeur A      Routeur B      Routeur C      Routeur D      Routeur E

Destination	Métrique
-------------	----------

A	0
B	1
C	1
D	1
E	2

Destination	Métrique
-------------	----------

A	1
B	0
C	2
D	1
E	2

Destination	Métrique
-------------	----------

A	1
B	2
C	0
D	1
E	2

Destination	Métrique
-------------	----------

A	1
B	1
C	1
D	0
E	1

Destination	Métrique
-------------	----------

A	2
B	2
C	2
D	1
E	0

### Question 2.a

Donner la liste des routeurs avec lesquels le routeur A est directement relié.

### Réponse

Le routeur A est directement relié aux routeurs **B, C et D**, en effet la valeur de la métrique est de 1 pour ces destinations dans la table de routage du routeur A.

### Question 2.b

Représenter graphiquement et de manière sommaire les 5 routeurs ainsi que les liaisons existantes entre ceux-ci.

### Réponse

```
graph LR
 a((A)) --- b((B))
 a --- c((C))
 a --- d((D))
 b --- d((D))
 c --- d((D))
 d --- e((E))
```

### Question 3

Le protocole OSPF est un protocole de routage qui cherche à minimiser la somme des métriques des liaisons entre routeurs.

Dans le protocole de routage OSPF le débit des liaisons entre routeurs agit sur la métrique via la relation :  $(\text{métrique}) = \frac{10^8}{(\text{débit})}$  dans laquelle le débit est exprimé en bit par seconde ( $\text{bps}$ ).

On rappelle qu'un  $(\text{kbps})$  est égal à  $(10^3 \text{ bps})$  et qu'un  $(\text{Mbps})$  est égal à  $(10^6 \text{ bps})$ .

Recopier sur votre copie et compléter le tableau suivant :

Débit	$(100 \text{ kbps})$	$(500 \text{ kbps})$	...	$(100 \text{ Mbps})$
Métrique associée	$(1000)$	...	$(10)$	$(1)$

### Réponse

Les deux lignes sont inversement proportionnelles.

Débit	\(100 \sim \text{kbps}	\(500 \sim \text{kbps}	\(10 \sim \text{Mbps}	\(100 \sim \text{Mbps}
	\()	\()	\()	\()
Métrique associée	\(1000\)	\(200\)	\(10\)	\(1\)

### Question 4

Voici la représentation d'un réseau et la table de routage incomplète du routeur F obtenue avec le protocole OSPF :

#### Routeur F

Destination	Métrique
F	0
G	8
H	5
I	
J	
K	
L	

Les nombres présents sur les liaisons représentent les coûts des routes avec le protocole OSPF.

### Question 4.a

Indiquer le chemin emprunté par un message d'un ordinateur du réseau F à destination d'un ordinateur du réseau I. Justifier votre réponse.

### Réponse

Le message sera acheminé du réseau F vers le réseau I en passant successivement par les routeurs **H, J et K**. En effet, avec ce trajet, le coût sera égal à **5 + 1 + 2 + 5** soit **13**. Tout autre trajet aura un coût plus élevé.

### Question 4.b

Recopier et compléter la table de routage du routeur F.

**Réponse**

Destination	Métrrique
F	0
G	8
H	5
I	<b>13</b>
J	<b>6</b>
K	<b>8</b>
L	<b>11</b>

**Question 4.c**

Citer une unique panne qui suffirait à ce que toutes les données des échanges de tout autre réseau à destination du réseau F transitent par le routeur G. Expliquer en détail votre réponse.

**Réponse**

Une panne de la liaison **F-H**.

En considérant le routeur I : la liaison est directe avec F, mais pour un coût de 20. Or de I, en passant par les routeurs K, J et G, le coût sera seulement de 19. De fait les routeurs K et J privilégieront également le routeur G. Le routeur J deviendra aussi le routeur le plus économique pour les routeurs H et L.

**⚠** Une panne du **routeur H** n'est pas une réponse acceptable. D'après l'énoncé, tout réseau autre que F doit joindre ce dernier en passant par G.

## 5. T4 - Prog.

### 1. 5.1 C3 Récursivité



#### Programme Terminale

Contenus	Capacités attendues	Commentaires
Récursivité	Ecrire un programme récursif. Analyser le fonctionnement d'un programme récursif.	Des exemples relevant de domaines variés sont à privilégier



#### 1.1. 5.1.1 Première approche

Faites l'essai

### 1.1.1. Principe

En règle générale, un objet est dit récursif s'il se définit à partir de lui-même.  
On trouve donc des acronymes récursifs, comme :

- GNU dans GNU/Linux (GNU is Not Unix),
- le logiciel d'émulation WINE (Wine Is Not an Emulator),
- les cartes bancaires VISA (Visa International Service Association),
- le moteur de recherche Bing (Bing is not Google),
- etc.



### A retenir : Fonction récursive

Une fonction est dite récursive lorsqu'elle fait appel à elle-même dans sa propre définition.

### 1.1.2. Premiers exemples et précautions d'usage

#### Note : No infinite recursion !

Voici trois premiers exemples de fonctions récursives. Dans chaque cas, repérer l'appel récursif à la fonction.

Une seule de ces 3 fonctions est correcte, laquelle?

#### • Fonction 1

##### Script Python

```
def f(n):
 print(n)
 f(n-1)
 print("Hello world!")
```

#### • Fonction 2

##### Script Python

```
def f(n):
 if n == 0:
 print("Hello world!")
 else:
 print(n)
 f(n-1)
```

#### • Fonction 3

##### Script Python

```
def f(n):
 if n == 0:
 print("Hello world!")
 else:
 print(n)
 f(n)
```

### Cas de Base

Lorsqu'on écrit une fonction récursive, le piège classique est de créer une boucle infinie.

Hormis les blagues de geeks d'initiés, la récursivité en informatique ne tolère pas l'auto-référence infinie: il faut prévoir une condition d'arrêt qui traite le cas de base !!!



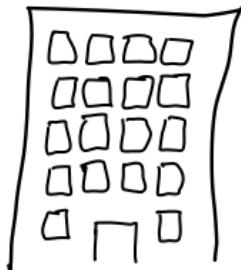
Ce «cas de base» sera aussi appelé «condition d'arrêt», puisque la très grande majorité des algorithmes récursifs peuvent être perçus comme des escaliers qu'on descend marche par marche, jusqu'au sol qui assure notre arrêt.

## Terminaison

Pour s'assurer qu'une fonction récursive se termine, il faut **absolument** que la chaîne d'appel conduise au cas de base.

- si le paramètre de la fonction est un entier, alors l'appel doit se faire avec un **entier strictement inférieur**;
- si le paramètre de la fonction est une liste, alors l'appel doit se faire avec une liste de **longueur strictement inférieure**;
- etc.

Le locataire du 4ème étage  
doit m'acheter mon livre.  
Mais je ne sais pas encore à  
quel prix.



«je vous l'achète le  
double de ce que  
vous donnerait le  
voisin du dessous»



«je vous l'achèterais  
le double de ce que  
vous donnerait le  
voisin du dessous»



«je vous l'achèterais  
le double de ce que  
vous donnerait le  
voisin du dessous»



«je vous l'achèterais  
le double de ce que  
vous donnerait le  
voisin du dessous»



«je vous en  
donnerais 3 euros»



«ok donc je vous en  
donnerais 6 euros»



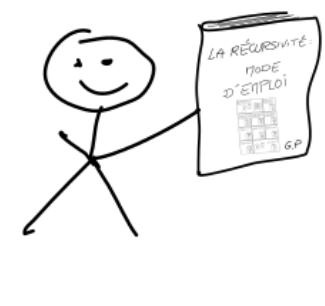
«ok donc je vous en  
donnerais 12 euros»



«ok donc je vous en  
donnerais 24 euros»



«ok, je vous l'achète  
pour 48 euros»



Observez bien la descente puis la remontée de notre vendeur de livre. Le cas de base est ici l'étage 0. Il empêche une descente infinie.

Nous coderons bientôt la fonction donnant le prix du livre en fonction de l'étage.

Pour l'instant, découvrons enfin à quoi ressemble une fonction récursive «bien écrite» :

#### 🐍 Script Python

```
def mystere(n):
 if n == 0 :
 return 0
 else :
 return n + mystere(n-1)
```

Trois choses sont essentielles et doivent se retrouver dans tout programme récursif :

- lignes 2 et 3 : le cas de base (si  $n$  vaut 0 on renvoie vraiment une valeur, en l'occurrence 0)
- ligne 5 : l'appel récursif
- ligne 5 : la décrémentation du paramètre d'appel

#### 🐍 Script Python

```
mystere(5)
15
```

#### 🐍 Script Python

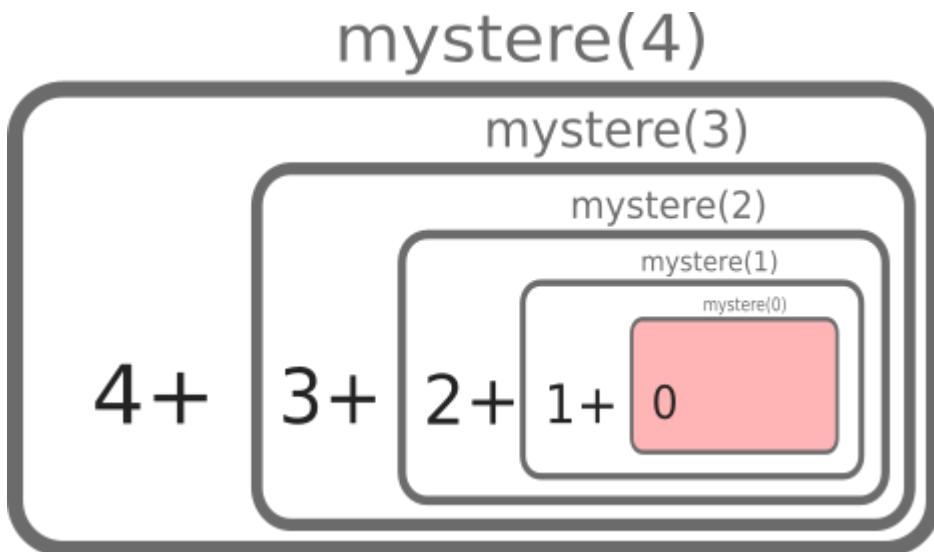
```
mystere(8)
36
```

### Analyse avec Python tutor

Que se passe-t-il lorsqu'on appelle `mystere(4)` ?

$$\begin{aligned} \text{mystere}(4) &= 4 + \text{mystere}(3) \\ &= 4 + (3 + \text{mystere}(2)) \\ &= 4 + (3 + (2 + \text{mystere}(1))) \\ &= 4 + (3 + (2 + (1 + \text{mystere}(0)))) \end{aligned}$$

On voit que l'existence du cas de base pour  $(n=0)$  est primordiale pour éviter la récursion infinie.



Cette fonction `mystere(n)` calcule donc la somme des entiers positifs inférieurs ou égaux à  $(n)$ .

Une anecdote raconte que **Carl Friedrich Gauss** trouva cette valeur de 5050 en quelques secondes, devant son instituteur ébahi. Il venait pour cela d'inventer la formule :  $\frac{1}{2}n(n+1)$

Ici,  $\frac{1}{2}(1+2+3+\dots+100) = \frac{1}{2} \cdot 100 \cdot 101 = 50 \cdot 101 = 5050$

### Connaitre : Somme des n premiers entiers

On souhaite calculer la somme suivante:  $(S = 0 + 1 + 2 + 3 + \dots + (n-1) + n)$

[Version itérative](#)    [Version récursive](#)

En première, on a vu comment construire une fonction *itérative* le permettant, à l'aide d'une boucle `for` (d'où le terme *itératif*) et d'une variable accumulatrice:

#### Script Python

```
def somme_iter(n):
 s = 0
 for k in range(n+1):
 s += k
 return s
somme_iter(100)
5050
```

Une autre façon de voir le problème, c'est de se dire que cette somme peut s'écrire  $(S = n + (n-1) + \dots + 2 + 1 + 0)$  et que c'est la somme de  $n$  et **de la somme des  $(n-1)$  premiers entiers**:  $(S = n + \underbrace{(n-1) + \dots + 3 + 2 + 1 + 0}_{\text{somme des entiers jusqu'à } n-1})$ .

On écrit alors de façon «assez naturelle» la fonction récursive suivante (qui est la fonction mystère précédente) :

#### Script Python

```
def somme_rec(n):
 if n == 0:
 return 0
 else:
 return n + somme(n-1)
somme_rec(100)
5050
```

### Exercice 1 ❤

On considère la fonction `factorielle(n)` (notée  $n!$  en mathématiques), qui calcule le produit d'un entier  $n$  par les entiers positifs qui lui sont inférieurs:  $n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$  Exemple :  $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

S'inspirer des fonctions `somme` précédentes pour écrire deux fonctions `facto_iter` (itératrice) et `facto_rec` (récurse) renvoyant la factorielle d'un nombre entier  $n$  strictement positif.

#### 1.1.3. Mécanisme - Notion de pile

Maintenant qu'on a vu le principe d'une fonction récursive, il faut comprendre comment se passent les appels successifs à la fonction, pour un paramètre différent.

Reprendons l'exemple de la fonction récursive `somme`. Si on appelle cette fonction:

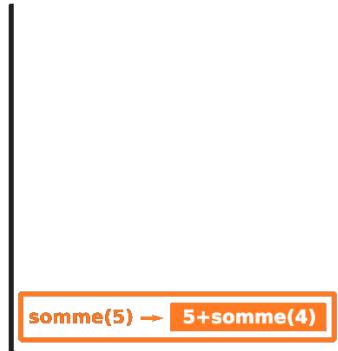
#### Script Python

```
>>> somme(5)
```

Puisque l'argument 5 ne correspond pas au cas de base, la fonction va faire appel à `somme(4)`. Il faut retenir que l'exécution de la fonction `somme` est interrompue (avec l'argument 5) pour rappeler la fonction `somme` (avec l'argument 4)...

Pour gérer ces différents appels, le système utilise une **pile d'exécution** :

(Dans la notion de pile (qui sera traitée plus tard dans le programme de Terminale), seule l'instruction «en haut de la pile» peut être traitée et enlevée (on dit «dépilée»).)



#### 1.1.4. Limitation de la taille de la pile

Nous venons de voir que notre appel à `somme(5)` générait une pile de hauteur 6 (on parlera plutôt de *profondeur* 6). Cette profondeur est-elle limitée ?

##### 🐍 Script Python

```
somme_rec(3000)
```

Dans l'exemple précédent, la pile a une *profondeur* de 6. La profondeur de la pile n'est pas illimitée:

##### 🐍 Script Python

```
>>> somme(3000)
Traceback (most recent call last):
 File "<console>", line 1, in <module>
 File "<tmp 1>", line 5, in somme
 return n + somme(n-1)
 File "<tmp 1>", line 5, in somme
 return n + somme(n-1)
 File "<tmp 1>", line 5, in somme
 return n + somme(n-1)
[Previous line repeated 984 more times]
 File "<tmp 1>", line 2, in somme
 if n == 0:
RecursionError: maximum recursion depth exceeded in comparison
```

Vous venons de provoquer un «débordement de pile», le célèbre **stack overflow**.

De manière générale, les programmes récursifs sont souvent proches de la définition du problème à résoudre et assez naturels à écrire, mais ils sont susceptibles de générer un trop grand nombre d'appels à eux-mêmes et de nécessiter un temps d'exécution trop grand ou un débordement de pile. Il est parfois possible de les optimiser, comme nous le verrons dans le cours concernant la **programmation dynamique**.

Nous reparlerons aussi de récursivité lorsque nous l'inscrirons dans un paradigme plus global de programmation, qui est « **diviser pour régner** » (en anglais *divide and conquer*).

## 1.2. 5.1.2 Premiers Exercices

### Exercice 2

Coder la fonction `prix(etage)` de la BD présentée plus haut.

### Exercice 3

Écrire une fonction récursive `puissance(x,n)` qui calcule le nombre  $(x^n)$ .

### Exercice 4

Dans une grande boite de Pétri contenant un milieu nutritif riche sont déposées 10 bactéries.

On suppose que chaque heure le nombre de bactéries est multiplié par 4.

Ecrire une fonction récursive `nb_bact` qui renvoie le nombre de bactéries au bout de  $(n)$  jours,  $(n)$  étant un entier naturel saisi comme argument.

## 1.3. 5.1.3 Exemples de récursivité double

Les expressions qui définissent une fonction peuvent aussi dépendre de plusieurs appels à la fonction en cours de définition.

### 1.3.1. La suite de Fibonacci

Considérons la suite numérique ainsi définie : -  $(F_0 = 0)$  -  $(F_1 = 1)$  - \$ \forall n \in \mathbb{N}, F\_{n+2} = F\_{n+1} + F\_n\$

On a donc  $(F_2=0+1=1, F_3=F_2+F_1=1+1=2, F_4=F_3+F_2=2+1=3, F_5=F_4+F_3=3+2=5) \dots$

### Exercice 5

Implémenter de façon récursive la suite de Fibonacci.

### Observation de la pile d'exécution

Appelons `F(n)` la fonction calculant de manière récursive le  $n$ -ième terme de la suite. Observons en détail la pile d'exécution lors du calcul de `F(4)`.

#### Script Python

```
from tutor import tutor

fibo(4)
tutor()
```

On s'aperçoit notamment que :

- les appels récursifs ne sont PAS simultanés (rappelons que la simultanéité n'existe théoriquement pas en informatique). On pourrait s'imaginer que la relation  $(F_4=F_3+F_2)$  allait déclencher deux «fils» récursifs calculant respectivement  $(F_3)$  et  $(F_2)$ . Il n'en est rien, on va jusqu'au bout du calcul de  $(F_3)$  avant de s'intéresser à  $(F_2)$ .
- conséquence de la remarque précédente : le calcul de  $(F_2)$  s'effectue 2 fois. Une amélioration future (appelée **mémoïsation**, voir le cours de programmation dynamique) sera de conserver cette valeur de  $(F_2)$  afin d'améliorer les calculs.

On peut y construire par exemple l'arbre d'appel de `fibo(6)` :

#### Script Python

```
from rcviz import viz
```

#### Script Python

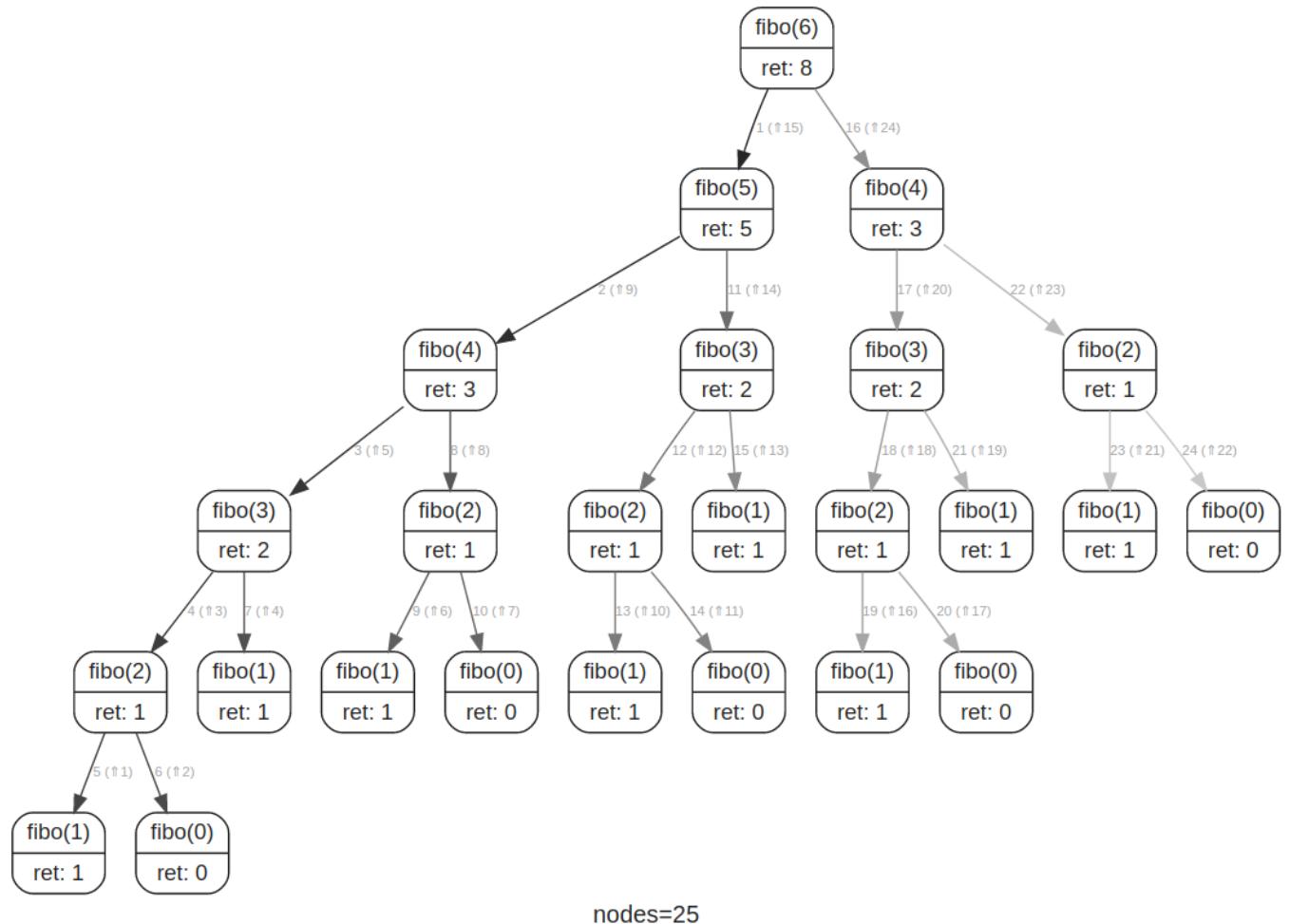
```
@viz
def fibo(n):
 if n == 0:
 return 0
 if n == 1:
 return 1
 else:
 return fibo(n-1)+fibo(n-2)
```

#### Script Python

```
fibo(6)
8
```

#### Texte

```
fibo.callgraph()
```



On y remarque (par exemple) que `fibo(2)` est calculé 5 fois...

#### 1.4. 5.1.4 Exercices

##### Exercice 6

On veut réaliser un château de cartes géants qui prolonge la château de l'image ci-dessous :



On note  $(n)$  le nombre d'étages du château et  $(nb\_cartes(n))$  le nombre de cartes nécessaires pour réaliser un château à  $(n)$  étages.

On admet que l'on peut connaître le nombre  $(nb\_cartes(n+1))$  de cartes nécessaires pour un château à  $(n+1)$  étages si on connaît déjà le nombre  $(nb\_cartes(n))$  en utilisant la relation suivante (appelée relation de récurrence en mathématiques) :

$$[nb\_cartes(n+1)=nb\_cartes(n)+2+3 \times n]$$

(à retrouver pour ceux suivant la spécialité maths)

On veut à partir de ces informations construire une fonction récursive nommée  $(nb\_cartes)$  qui renvoie finalement le nombre  $(nb\_cartes(n))$  si en argument lui est entré le nombre  $(n)$  d'étages voulus au château.

- Ecrire la fonction correspondante 2 Testez votre fonction obtenue.

Vous pouvez utiliser l'image ci-dessus pour connaître quelques valeurs à obtenir.

##### Script Python

```
def nb_cartes(n):
 pass

nb_cartes(15)
345
```

### Exercice 7

Soit  $\{(u_n)\}$  la suite d'entiers définie par :

$$(u_{n+1}) = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$$

avec  $(u_0)$  un entier quelconque plus grand que 1.

Ecrire une fonction récursive `syracuse(u_n)` qui affiche les valeurs successives de la suite  $\{(u_n)\}$  tant que  $\{(u_n)\}$  est plus grand que 1.

La conjecture de Syracuse affirme que, quelle que soit la valeur de  $(u_0)$ , il existe un indice  $(n)$  dans la liste tel que  $(u_n=1)$ ;

Cette conjecture défie toujours les mathématiciens.

### Exercice 8

Voici une fonction mystère nommée `myst` :

#### Script Python

```
def myst(l: list) -> int:
 if l==[]:
 return 0
 else:
 l.pop(0) # suppression du premier terme de la liste l
 return 1+myst(l)
```

1. Pourquoi cette fonction `myst` est une fonction récursive ?
2. Testez cette fonction avec quelques listes.
3. Quel est le rôle de cette fonction `myst` ?

### Exercice 9

Soit  $\{(u_n)\}$  la suite d'entiers définie par :  $(u_n) = \begin{cases} n & \text{si } n = 0 \\ 3u_{n-1} + 2u_{n-2} + 5 & \text{pour tout } n \geq 2 \end{cases}$

Ecrire une fonction récursive `serie(n,a,b)` qui renvoie le  $(n)$ -ième terme de cette suite pour les valeurs de  $(a)$  et  $(b)$  données en paramètres.

### Exercice 10

Un palindrome est une chaîne de caractères qui est identique lue de gauche à droite ou de droite à gauche. Par exemple, la chaîne GIRAFARIG est un palindrome : si on inverse le mot, il reste identique.

Pour coder récursivement un test de palindrome (cf. dessin ci-dessous), il suffit de vérifier que

- les lettres aux extrémités sont les mêmes (les lettres en bleu sur la figure);
- le mot privé de ses deux extrémités est encore un palindrome (en orange sur le dessin), d'où appel récursif.

# GIRAFARIG

**Précaution :** quand on vérifie que le mot privé de ses deux extrémités est encore un palindrome, il faut faire attention à ce que le retrait des extrémités soit possible. Ce problème ne se pose que si le mot a une lettre ou n'a aucune lettre. Dans ces cas, le mot est un palindrome, ce qui donne le cas de base de la récursivité. Remarquons que si le mot a deux lettres, ce n'est pas un cas de base car quand on lui retire ses extrémités, la chaîne devient vide et on tombe sur un cas de base.

**Remarque :**

`mot[1:-1]` est un slice : c'est la chaîne mot amputée de son premier caractère (elle commence à l'indice 1 et se termine juste avant l'indice -1, ce dernier indice référençant le dernier caractère de la chaîne).

## 1.5. 5.1.5 Annexe : dessins récursifs grâce au module turtle

Le module `turtle` permet de faire des tracés basiques. Mais dès l'instant où on met de la récursivité dans le code, les résultats peuvent devenir très surprenants, et aboutir à des structures **fractales**.

### Script Python

```
from turtle import *
ang = 40
def trace(n,l):
 if n == 0 :
 return None
 else :
 forward(l)
 left(ang)
 trace(n-1,0.7*l)
 right(2*ang)
 trace(n-1,0.7*l)
 left(ang)
 forward(-l)

penup()
goto(0,-80)
pendown()
left(90)
speed(0)
trace(5,100)
```

### Texte

```
from turtle import *
ang = 40
def trace(n,l):
 if n == 0 :
 return None
 else :
 forward(l)
 left(ang)
 trace(n-1,0.65*l)
 right(2*ang)
 trace(n-1,0.65*l)
 left(ang)
 forward(-l)

penup()
goto(0,-80)
```

```
pendown()
left(90)
speed(0)

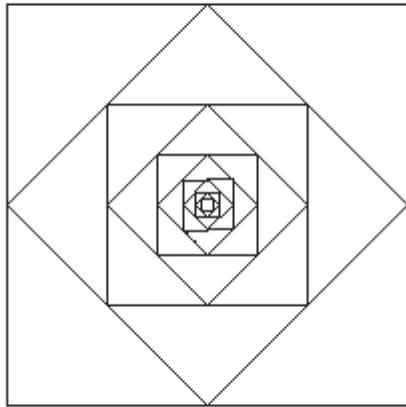
trace(12,100)
mainloop()
```

#### Texte

### Exercice 11 Facultatif

Reproduire le dessin suivant, à l'aide du module `turtle`.

`turtle` est un hommage au langage LOGO inventé par [Seymour Papert](#) au MIT à la fin des années 60.



## 6. T5 Algo.

### 1. 6.1 C7 Algorithmes de tri



#### 1.1. 6.1.1 Préambule

Pourquoi étudier des algorithmes de tri ?

Autant ne pas le cacher, ces algorithmes sont déjà implémentés (quelque soit le langage) dans des fonctions très performantes.

En Python, on utilise la fonction `sort()` :

#### 🐍 Script Python

```
>>> tab = [4, 8, 1, 2, 6]
>>> tab.sort()
>>> tab
[1, 2, 4, 6, 8]
```

Le meilleur de nos futurs algorithmes de tri sera moins efficace que celui de cette fonction `sort()` ...

Malgré cela, il est essentiel de se confronter à l'élaboration manuelle d'un algorithme de tri.

Le tri par insertion est le premier des deux algorithmes de tri que nous allons étudier (nous étudierons aussi le tri par sélection).

Ces deux algorithmes ont pour particularité de :

- ne pas nécessiter la création d'une nouvelle liste. Ils modifient la liste à trier sur place.
- ne pas faire intervenir de fonctions complexes.

#### 1.2. 6.1.2 Activités

##### 1.2.1. ■ Activité 1 : Tri par sélection

1. Commencer par télécharger une application Python :

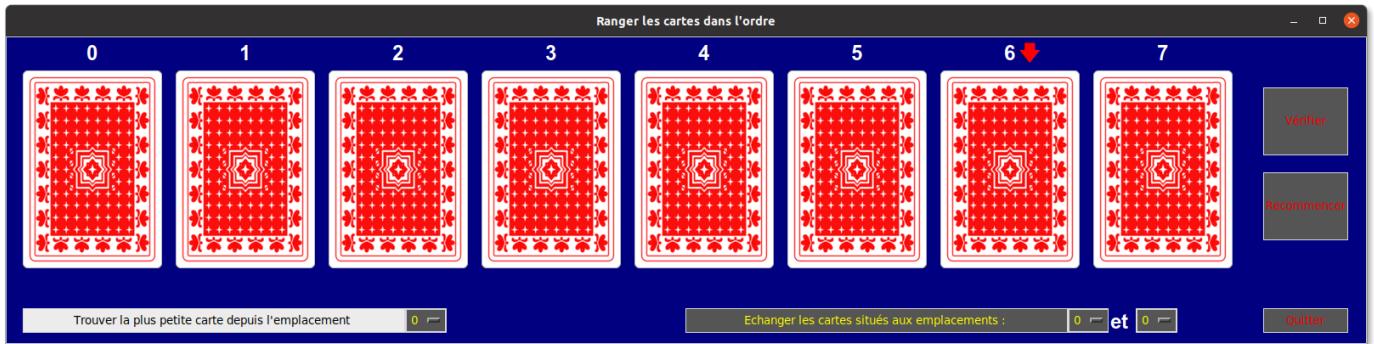
#### Tri par sélection

- Copier ce fichier dans le répertoire de votre choix
- Faire un clic droit sur le fichier compressé et choisir *Extraire ici*
- Lancer le programme Python `activite1.py`.

2. Dans cette activité, on doit ranger des cartes par ordre croissant mais **sans les voir**, on dispose par contre de deux boutons :

- Un bouton Trouver la plus petit carte depuis l'emplacement qui permet de savoir quelle carte est la plus petite à partir de l'emplacement qu'on sélectionne dans le menu déroulant à côté.
- Un bouton Echanger les cartes situés aux emplacements qui permet d'échanger les cartes situés aux emplacements sélectionnés dans les menus déroulants.

Voici une capture d'écran de l'application dans laquelle on vient de sélectionner la plus petite carte depuis l'emplacement 0, elle est alors indiquée par une flèche rouge au-dessus (emplacement 6) :



3. Proposer un algorithme permettant à un ordinateur de ranger une suite de nombres par ordre croissant.

4. Implémentation en python

a. Ecrire une fonction `echange(liste,i,j)` qui échange les éléments d'indice `i` et `j` de la liste `liste` par exemple si `liste=[12,17,10,11,32]` alors après `echange(liste,0,2)` le contenu de `liste` sera `[10,17,12,11,32]`.

b. Ecrire une fonction `min_depuis(liste,i)` qui renvoie le minimum de la liste `liste` à partir de l'indice `i` par exemple `min_depuis([10,17,12,11,32],2)` renvoie `11`.

c. En utilisant ces deux fonctions, proposer une implémentation en Python de l'algorithme du tri par sélection.

#### 1.2.2. ■ Activité 2 : Tri par insertion

1. De même que dans l'activité précédente, commencer par télécharger une application Python :

[Tri par insertion](#)

- Copier ce fichier dans le répertoire de votre choix

- Faire un clic droit sur le fichier compressé et choisir *Extraire ici*

- Lancer le programme Python `activite2.py`.

2. De même que dans l'activité précédente, il faut ranger les cartes dans l'ordre *sans les voir*, on dispose d'un unique bouton permettant d'échanger une carte dont on donne le numéro avec sa voisine *si elles ne sont pas dans le bon ordre*

3. Proposer un algorithme permettant de ranger une liste par ordre croissant en utilisant comme seul "ingrédient" l'échange de deux cartes dont on donne les emplacements.



Bien évidemment, des boucles et des tests seront aussi nécessaires

4. Proposer une implémentation en Python de cet algorithme



On pourra utiliser la fonction `echange` définie dans l'activité précédente.

5. Tester cette fonction

#### 1.3. 6.1.3 Cours

Vous pouvez télécharger une copie au format pdf du diaporama de synthèse de cours présenté en classe :

[Diaporama de cours](#)

**Attention**

Ce diaporama ne vous donne que quelques points de repères lors de vos révisions. Il devrait être complété par la relecture attentive de vos **propres** notes de cours et par une révision approfondie des exercices.

## 1.4. 6.1.4 Cours : tri par insertion

### 1.4.1. Principe et algorithme

Considérons la liste [5,4,9,7,2,1,8,0,6,3]

Voici le fonctionnement de l'algorithme :


**Vidéo Tri par insertion**

### Tri par insertion

#### Explications :

- On traite successivement toutes les valeurs à trier, en commençant par celle en deuxième position.
- Traitement : tant que la valeur à traiter est inférieure à celle située à sa gauche, on échange ces deux valeurs.

### 1.4.2. Codage de l'algorithme

#### Algorithm :

Pour toutes les valeurs, en commençant par la deuxième :

- Tant qu'on trouve à gauche une valeur supérieure et qu'on n'est pas revenu à la première valeur, on échange ces deux valeurs.

## Tri par insertion (version simple) ❤

### 🐍 Script Python

```
def tri_insertion1(liste):
 '''trie en place la liste lst donnée en paramètre'''
 for ind in range(len(liste)-1): #(1)
 pos = ind #(2)
 while pos >= 0 and liste[pos+1] < liste[pos] : #(3)
 liste[pos], liste[pos+1] = liste[pos+1], liste[pos] #(4)
 pos = pos - 1 #(5)
```

1. On commence à 0 et on finit à longueur -1.
2. On «duplique» la variable `ind` en une variable `pos`.  
On se positionne sur l'élément d'indice `pos`. On va faire «reculer» cet élément tant que c'est possible. On ne touche pas à `ind`.
3. Tant qu'on n'est pas revenu au début de la liste et qu'il y a une valeur plus grande à gauche.
4. On échange de place avec l'élément précédent.
5. Notre élément est maintenant à l'indice `pos - 1`.  
La boucle peut continuer.

*Application :*

### 🐍 Script Python

```
>>> maliste = [7, 5, 2, 8, 1, 4]
>>> tri_insertion1(maliste)
>>> maliste
[1, 2, 4, 5, 7, 8]
```

### 💡 vous

Réaliser le tri par insertion de la liste suivante : [27,10,12,8,11]  
Ecrire toutes les étapes.

### 💡 vous

Réaliser le tri par insertion de la liste suivante : [9,6,1,4,8]  
Ecrire toutes les étapes.

### 1.4.3. Complexité de l'algorithme

### 🐍 Script Python

```
def tri_insertion(liste):
 '''trie en place la liste lst donnée en paramètre'''
 for ind in range(1, len(liste)-1): #(1)
 pos = ind #(2)
 while pos >= 0 and liste[pos+1] < liste[pos] : #(3)
```

```
liste[pos], liste[pos+1] = liste[pos+1], liste[pos] #(4)
pos = pos - 1 #(5)
```

1. On commence à 1 et non pas à 0.
2. On «duplicte» la variable `ind` en une variable `pos`.  
On se positionne sur l'élément d'indice `pos`. On va faire «reculer» cet élément tant que c'est possible. On ne touche pas à `ind`.
3. Tant qu'on n'est pas revenu au début de la liste et qu'il y a une valeur plus grande à gauche.
4. On échange de place avec l'élément précédent.
5. Notre élément est maintenant à l'indice `pos - 1`.  
La boucle peut continuer.

#### DÉMONSTRATION

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille  $\lfloor n \rfloor$ .

- boucle `for` : elle s'exécute  $\lfloor n-1 \rfloor$  fois.
- boucle `while` : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à  $\lfloor n-1 \rfloor$ . Or

$$\lfloor 1+2+3+\dots+n-1 = \frac{n(n-1)}{2} \rfloor$$

Le terme de plus haut degré de l'expression  $\lfloor \frac{n(n-1)}{2} \rfloor$  est de degré 2 : le nombre d'opérations effectuées est donc proportionnel au **carré** de la taille des données d'entrée.

Ceci démontre que le tri par insertion est de complexité **quadratique** noté  $\mathcal{O}(n^2)$ .

Dans le cas (rare, mais il faut l'envisager) où la liste est déjà triée, on ne rentre jamais dans la boucle `while` : le nombre d'opérations est dans ce cas égal à  $\lfloor n-1 \rfloor$ , ce qui caractérise une complexité linéaire.

#### 1.4.4. Résumé de la complexité

- dans le meilleur des cas (liste déjà triée) : complexité **linéaire**
- dans le pire des cas (liste triée dans l'ordre décroissant) : complexité **quadratique**

#### 1.4.5. Preuve de la terminaison de l'algorithme

Est-on sûr que notre algorithme va s'arrêter ?

Le programme est constitué d'une boucle `while` imbriquée dans une boucle `for`. Seule la boucle `while` peut provoquer une non-terminaison de l'algorithme. Observons donc ses conditions de sortie :

##### Script Python

```
while pos >= 0 and liste[pos+1] < liste[pos] :
```

La condition `liste[pos+1] < liste[pos]` ne peut pas être rendue fausse avec certitude. Par contre, la condition `pos >= 0` sera fausse dès que la variable `pos` deviendra négative. Or la ligne `pos = pos - 1` nous assure que la variable `pos` diminuera à chaque tour de boucle. La condition `pos >= 0` deviendra alors forcément fausse au bout d'un certain temps.

Nous avons donc prouvé la **terminaison** de l'algorithme.

#### Vocabulaire

On dit que la valeur `pos` est un **variant de boucle**.

C'est une notion théorique (ici illustrée de manière simple par la valeur `pos`) qui permet de prouver la *bonne sortie d'une boucle* et donc la terminaison d'un algorithme.

## POUR ALLER PLUS LOIN : PREUVE DE LA CORRECTION DE L'ALGORITHME

Les preuves de correction sont des preuves théoriques. La preuve ici s'appuie sur le concept mathématique de **récurrence**. Principe du raisonnement par récurrence : une propriété  $\{P(n)\}$  est vraie si :

- $\{P(0)\}$  (par exemple) est vraie
- Pour tout entier naturel  $\{n\}$ , si  $\{P(n)\}$  est vraie alors  $\{P(n+1)\}$  est vraie.

Ici, la propriété serait : « Quand  $\{k\}$  varie entre 0 et `longueur(liste) - 1`, la sous-liste de longueur  $\{k\}$  est triée dans l'ordre croissant.»



On appelle cette propriété un **invariant de boucle**.

*Invariant* signifie qu'elle reste vraie pour chaque boucle.

- quand  $\{k\}$  vaut 0, on place le minimum de la liste en `l[0]`, la sous-liste `l[0]` est donc triée.
- si la sous-liste de  $\{k\}$  éléments est triée, l'algorithme rajoute en dernière position de la liste le minimum de la sous-liste restante, dont tous les éléments sont supérieurs au maximum de la sous-liste de  $\{k\}$  éléments. La sous-liste de  $\{k+1\}$  éléments est donc aussi triée.

## 1.5. 6.1.5 Cours : tri par sélection

### 1.5.1. Animation

Considérons la liste [8,5,2,6,9,3,1,4,8,7]

Voici le fonctionnement de l'algorithme :

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7



### Vidéo Tri par sélection

[Tri par sélection](#)

### 1.5.2. Principe

#### description de l'algorithme

Le travail se fait essentiellement sur les **indices**.

- du premier élément jusqu'à l'avant-dernier :
- on considère que cet élément est l'élément minimum, on stocke donc son indice dans une variable *indice du minimum*.
- on parcourt les éléments suivants, et si on repère un élément plus petit que notre minimum on met à jour notre *indice du minimum*.
- une fois le parcours fini, on échange l'élément de travail avec l'élément minimum qui a été trouvé.

### 1.5.3. Implémentation de l'algorithme

#### Tri par sélection ❤

##### Script Python

```
def tri_selection(lst):
 for k in range(len(lst)-1):
 indice_min = k
 for i in range(k+1, len(lst)):
 if lst[i] < lst[indice_min]:
 indice_min = i
 lst[k], lst[indice_min] = lst[indice_min], lst[k]
```

Vérification :

##### Script Python

```
>>> ma_liste = [7, 5, 2, 8, 1, 4]
>>> tri_selection(ma_liste)
>>> ma_liste
[1, 2, 4, 5, 7, 8]
```

### 1.5.4. Complexité de l'algorithme

#### CALCUL DU NOMBRE D'OPÉRATIONS

Dénombrons le nombre d'opérations, pour une liste de taille  $n$ .

- boucle `for` : elle s'exécute  $n-1$  fois.
- deuxième boucle `for` imbriquée : elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à  $n-1$ .

Or  $(1+2+3+\dots+n-1) = \frac{n(n-1)}{2}$

Ceci est bien un polynôme du second degré, ce qui confirme que la complexité de ce tri est quadratique.

## 1.6. 6.1.6 QCM



**1. On applique l'algorithme du tri par sélection à la liste [9,11,7,16], après la première étape, le contenu de la liste sera :**

Réponses      Correction

- a) [11,9,7,16]
  - b) [7,11,9,16]
  - c) [16,11,7,9]
  - d) Aucune des propositions ci-dessus
- 
- a) [11,9,7,16]
  - b) [7,11,9,16]
  - c) [16,11,7,9]
  - d) Aucune des propositions ci-dessus



**2. On applique l'algorithme du tri par insertion à la liste [9,11,7,16], quel sera le contenu de la liste après le premier échange ?**

Réponses      Correction

- a) [11,9,7,16]
  - b) [9,11,16,7]
  - c) [9,7,11,16]
  - d) Aucune des propositions ci-dessus
- 
- a) [11,9,7,16]
  - b) [9,11,16,7]
  - c) [9,7,11,16]
  - d) Aucune des propositions ci-dessus

**Q. L'algorithme du tri par insertion a une complexité :**

Réponses      Correction

- a) logarithmique
  - b) linéaire
  - c) quadratique
  - d) exponentielle
- 
- a) logarithmique
  - b) linéaire
  - c) quadratique
  - d) exponentielle

**Q. Un programme de tri par insertion prend environ 1 seconde pour trier une liste de  $\backslash(10\backslash,000\backslash)$  éléments, combien de temps prendra-t-il environ pour trier une liste de  $\backslash(100\backslash,000\backslash)$  éléments ?**

Réponses      Correction

- a) 1 seconde
  - b) 10 secondes
  - c) 100 secondes
  - d) 1000 secondes
- 
- a) 1 seconde
  - b) 10 secondes
  - c) 100 secondes
  - d) 1000 secondes



**Quelles sont les deux lignes manquantes dans la fonction ci-dessus qui renvoie le minimum d'une liste non vide :**

```

1 def min_liste(liste):
2 elt_min =
3 for elt in liste:
4 if elt<elt_min:
5
6 return elt_min

```

Réponses      Correction

- a) La ligne 2 est `elt_min=liste[1]` et la ligne 5 est `elt_min=elt`
  - b) La ligne 2 est `elt_min=liste[1]` et la ligne 5 est `elt=elt_min`
  - c) La ligne 2 est `elt_min=liste[0]` et la ligne 5 est `elt=elt_min`
  - d) La ligne 2 est `elt_min=liste[0]` et la ligne 5 est `elt_min=elt`
- 
- a) La ligne 2 est `elt_min=liste[1]` et la ligne 5 est `elt_min=elt`
  - b) La ligne 2 est `elt_min=liste[1]` et la ligne 5 est `elt=elt_min`
  - c) La ligne 2 est `elt_min=liste[0]` et la ligne 5 est `elt=elt_min`
  - d) La ligne 2 est `elt_min=liste[0]` et la ligne 5 est `elt_min=elt`

## 1.7. 6.1.7 Exercices



**Fonction echange(liste,i,j)**

**Script Python**

```
def echange(liste,i,j):
 liste[i],liste[j] = liste[j],liste[i]
```



## Fonctionnement du tri par sélection

Enoncé      Correction

1. Ecrire les étapes du tri par sélection pour la liste [12,19,10,13,11,15,9,14]

2. Même question pour la liste ["P", "R", "O", "G", "R", "A", "M", "M", "E"]

1.

### Script Python

```
[12, 19, 10, 13, 11, 15, 9, 14]
[9, 19, 10, 13, 11, 15, 12, 14]
[9, 10, 19, 13, 11, 15, 12, 14]
[9, 10, 11, 13, 19, 15, 12, 14]
[9, 10, 11, 12, 19, 15, 13, 14]
[9, 10, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 14, 19, 15]
[9, 10, 11, 12, 13, 14, 15, 19]
[9, 10, 11, 12, 13, 14, 15, 19]
```

2.

### Script Python

```
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'E']
['A', 'R', 'O', 'G', 'R', 'P', 'M', 'M', 'E']
['A', 'E', 'O', 'G', 'R', 'P', 'M', 'M', 'R']
['A', 'E', 'G', 'O', 'R', 'P', 'M', 'M', 'R']
['A', 'E', 'G', 'M', 'R', 'P', 'O', 'M', 'R']
['A', 'E', 'G', 'M', 'P', 'O', 'R', 'R', 'R']
['A', 'E', 'G', 'M', 'P', 'O', 'R', 'R', 'R']
['A', 'E', 'G', 'M', 'O', 'P', 'R', 'R', 'R']
['A', 'E', 'G', 'M', 'O', 'P', 'R', 'R', 'R']
['A', 'E', 'G', 'M', 'O', 'P', 'R', 'R', 'R']
```

## Fonctionnement du tri par insertion

## Enoncé      Correction

## Correction

1. Ecrire les étapes du tri par insertion pour la liste [12,19,10,13,11,15,9,14]
  2. Même question pour la liste ["P", "R", "O", "G", "R", "A", "M", "M", "E"]

1

## Script Python

```
[12, 19, 10, 13, 11, 15, 9, 14]
[12, 19, 10, 13, 11, 15, 9, 14]
[12, 10, 19, 13, 11, 15, 9, 14]
[10, 12, 19, 13, 11, 15, 9, 14]
[10, 12, 13, 19, 11, 15, 9, 14]
[10, 12, 13, 11, 19, 15, 9, 14]
[10, 12, 11, 13, 19, 15, 9, 14]
[10, 11, 12, 13, 19, 15, 9, 14]
[10, 11, 12, 13, 15, 19, 9, 14]
[10, 11, 12, 13, 15, 9, 19, 14]
[10, 11, 12, 13, 9, 15, 19, 14]
[10, 11, 12, 9, 13, 15, 19, 14]
[10, 11, 9, 12, 13, 15, 19, 14]
[10, 9, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 15, 14, 19]
[9, 10, 11, 12, 13, 14, 15, 19]
```

2.

## Script Python

## Tri par ordre décroissant

Enoncé      Correction

1. On donne ci-dessous l'implémentation du tri par sélection vu en cours :

### Script Python

```
def echange(liste,i,j):
 liste[i],liste[j] = liste[j],liste[i]

def min_liste(liste,ind):
 elt_min = liste[ind]
 ind_min=ind
 for k in range(ind,len(liste)):
 if liste[k]<elt_min:
 elt_min=liste[k]
 ind_min=k
 return ind_min

def tri_selection(liste):
 longueur = len(liste)
 for ind in range(longueur):
 ind_min = min_liste(liste,ind)
 echange(liste,ind,ind_min)
```

Modifier cette (ces) fonction(s) afin d'effectuer un tri dans l'ordre décroissant.

2. Même question pour l'algorithme du tri par insertion ci-dessous :

### Script Python

```
def tri_insertion(liste):
 for ind in range(1,len(liste)-1):
 j = ind
 while liste[j+1]<liste[j] and j>=0:
 echange(liste,j,j+1)
 j=j-1
```

### Script Python

```
def echange(liste,i,j):
 liste[i],liste[j] = liste[j],liste[i]

def max_liste(liste,ind):
 elt_max = liste[ind]
 ind_max=ind
 for k in range(ind,len(liste)):
 if liste[k]>elt_max:
 elt_max=liste[k]
 ind_max=k
 return ind_max

def tri_selection_inverse(liste):
 longueur = len(liste)
 for ind in range(longueur):
 ind_max = max_liste(liste,ind)
 echange(liste,ind,ind_max)
```



## Tri dans une nouvelle liste

[Enoncé](#)    [Correction](#)

Les algorithmes vus en cours modifient la liste donnée en paramètre, on dit qu'on effectue un *tri en place* c'est à dire directement dans la liste.

1. Modifier la fonction de tri par sélection vu en classe afin d'effectuer le tri en créant une nouvelle liste (et donc sans modifier la liste de départ)
2. Même question pour le tri par insertion



Comme une *nouvelle liste* est créée, on utilisera l'instruction `return` pour la renvoyer vers le programme principal.

### Script Python

```
def tri_selection(liste):
 liste_nouv=[]
 for elt in liste:
 liste_nouv.append(elt)
 longueur = len(liste_nouv)
 for ind in range(longueur):
 ind_min = min_liste(liste_nouv,ind)
 echange(liste_nouv,ind,ind_min)
 return liste_nouv

def tri_insertion(liste):
 liste_nouv=[]
 for elt in liste:
 liste_nouv.append(elt)
 for ind in range(0,len(liste_nouv)-1):
 j = ind
 while liste_nouv[j+1]<liste_nouv[j] and j>=0:
 echange(liste_nouv,j,j+1)
 j=j-1
 return liste_nouv
```

 Liste triée[Enoncé](#)    [Correction](#)

Ecrire une fonction `est_triee` qui prend en argument une `liste` et qui renvoie `True` si `liste` est triée par ordre croissant et `False` dans le cas contraire.

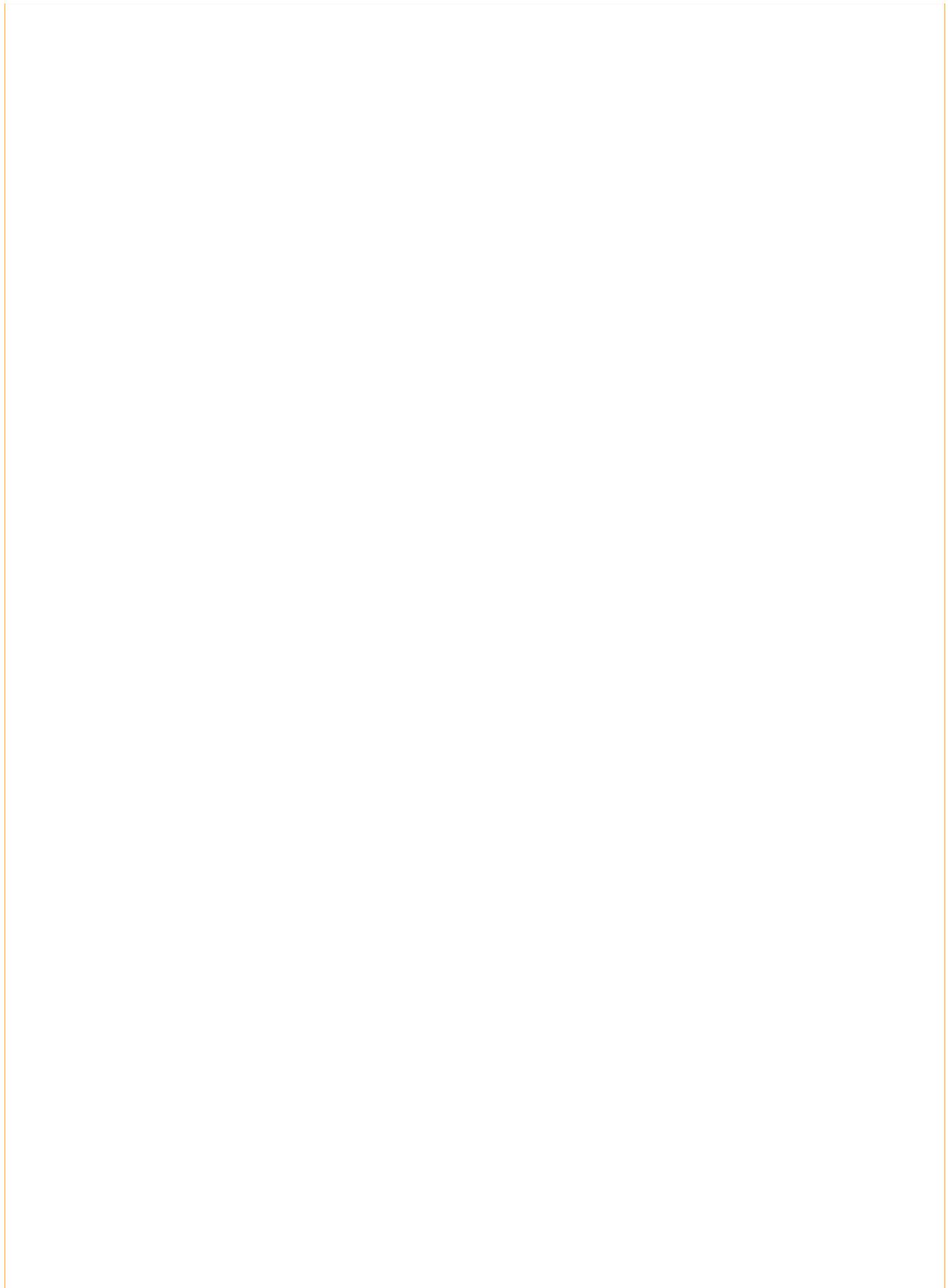
 Attention

On ne doit pas trier la liste, simplement vérifier si elle l'est déjà ou pas.

 Script Python

```
def est_trie(liste):
 long=len(liste)
 for ind in range(long-1):
 if liste[ind+1]<liste[ind]:
 return False
 return True
```

 **Epreuve Pratique**



Écrire une fonction `tri_selection` qui prend en paramètre une liste `tab` de nombres entiers et qui renvoie le tableau trié par ordre croissant.

On utilisera l'algorithme suivant :

- on recherche le plus petit élément du tableau, et on l'échange avec l'élément d'indice 0 ;
- on recherche le second plus petit élément du tableau, et on l'échange avec l'élément d'indice 1 ;
- on continue de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exemple :

#### Script Python

```
>>> tri_selection([1,52,6,-9,12])
[-9, 1, 6, 12, 52]
```

#### Script Python

```
def tri_selection(tab):
 for i in range(len(tab)-1):
 indice_min = i
 for j in range(i+1, len(tab)):
 if tab[j] < tab[indice_min]:
 indice_min = j
 tab[i], tab[indice_min] = tab[indice_min], tab[i]
 return tab
```

#ou version plus découpée, se rapprochant plus de la description de l'algo :

```
def minimum(tab, i):
 ind_minimum = i
 for j in range(i+1, len(tab)):
 if tab[j] < tab[ind_minimum]:
 ind_minimum = j
 return ind_minimum

def echange(tab, i, j):
 tab[i], tab[j] = tab[j], tab[i]

def tri_selection(tab):
 for i in range(len(tab)-1):
 ind_minimum = minimum(tab, i)
 echange(tab, i, ind_minimum)
 return tab
```

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt depuis le début du tableau tous les éléments non rangés et on place en dernière position le plus grand élément.

Exemple avec le tableau : `t = [41, 55, 21, 18, 12, 6, 25]`

- Étape 1 : on parcourt tous les éléments du tableau, on permute le plus grand élément avec le dernier.

Le tableau devient `t = [41, 25, 21, 18, 12, 6, 55]`

- Étape 2 : on parcourt tous les éléments **sauf le dernier**, on permute le plus grand élément trouvé avec l'avant dernier.

Le tableau devient : `t = [6, 25, 21, 18, 12, 41, 55]`

Et ainsi de suite. La code de la fonction `tri_iteratif` qui implémente cet algorithme est donné ci-dessous.

```
1 def tri_iteratif(tab):
2 for k in range(..., 0, -1):
3 imax = ...
4 for i in range(0, ...):
5 if tab[i] > ... :
6 imax = i
7 if tab[max] > ... :
8 ..., tab[imax] = tab[imax], ...
9 return tab
```

#### Script Python

## 2. 6.2 C8 Diviser pour régner



### 2.1. 6.2.1 Activités

#### 2.1.1. ■ Activité 1 : Retour sur l'algorithme de dichotomie



Aide

Cette activité revient sur deux algorithmes de recherche d'un élément dans une liste déjà rencontrés en classe de première.

1. Ecrire une fonction `recherche(x, l)` qui en effectuant un parcours simple de la liste, renvoie `True` ou `False` selon que l'élément `x` se trouve ou non dans la liste `l`.
  2. On suppose maintenant que la **liste est triée**, l'algorithme de recherche par dichotomie vue en classe de première consiste alors à
    - 1 partager la liste en deux listes de longueurs égales (à une unité près)
    - 2 comparer l'élément recherché avec celui situé au milieu de la liste
    - 3 en déduire dans quelle moitié poursuivre la recherche
 On s'arrête lorsque la zone de recherche ne contient plus qu'un élément.
- a. Faire fonctionner "à la main" cet algorithme pour rechercher `6` dans `[1,3,5,7,11,13]`.
  - b. Programmer cet algorithme en version impérative.

### compléter - Dichotomie version impérative ❤️

```

1 def recherche_dichotomique(tab, val) :
2 """
3 renvoie True ou False suivant la présence de la valeur val dans le tableau trié tab.
4 """
5 i_debut = 0
6 i_fin = len(tab) - 1
7 while i_debut <= i_fin :
8 i_centre = (... + ...) // 2 # (1)
9 val_centrale = tab[...] # (2)
10 if val_centrale == val: # (3)
11 return True
12 if val_centrale < val: # (4)
13 i_debut =
14 else : # (5)
15 i_fin = ...
16 return False

```

- a. on prend l'indice central
- b. on prend la valeur centrale
- c. si la valeur centrale est la valeur cherchée...
- d. si la valeur centrale est trop petite...
- e. on ne prend pas la valeur centrale qui a déjà été testée

Exemple d'utilisation :

#### 🐍 Script Python

```

>>> tab = [1, 5, 7, 9, 12, 13]
>>> recherche_dichotomique(tab, 12)
True
>>> recherche_dichotomique(tab, 17)
False

```

À chaque tour de la boucle `while`, la taille de la liste est divisée par 2. Ceci confère à cet algorithme une **complexité logarithmique** (bien meilleure qu'une complexité linéaire).

## Complexité

Pour pouvoir majorer le nombre maximum d'itérations, si le tableau contient  $n$  valeurs, et si on a un entier  $\lfloor k \rfloor$  tel que  $\lfloor n \rfloor \leq 2^k$ , alors puisque qu'à chaque itération, on sélectionne une moitié de ce qui reste :

- au bout d'une itération, une moitié de tableau aura au plus  $\lfloor \frac{2^k}{2} \rfloor = 2^{k-1}$  éléments,
- un quart aura au plus  $\lfloor 2^{k-2} \rfloor$
- et au bout de  $i$  itérations, la taille de ce qui reste à étudier est de taille au plus  $\lfloor 2^{k-i} \rfloor$ .
- En particulier, si l'on fait  $k$  itérations, il reste au plus  $\lfloor 2^{k-k} \rfloor = 1$  valeur du tableau à examiner. On est sûr de s'arrêter cette fois-ci

On a donc montré que si l'entier  $k$  vérifie  $\lfloor n \rfloor \leq 2^k$ , alors l'algorithme va effectuer au plus  $\lfloor n \rfloor$  itérations.

La plus petite valeur est obtenue pour  $\lfloor \log(n) \rfloor = \log_2 k$ .

Ainsi, la complexité de la fonction est de l'ordre du logarithme de la longueur de la liste ( $O(\log_2(n))$ ).  
 $\lfloor \log_2(n) \rfloor$ .

### 3. Dichotomie récursive sans slicing

Il est possible de programmer de manière récursive la recherche dichotomique sans toucher à la liste, et donc en jouant uniquement sur les indices :

## Dichotomie version récursive sans slicing ❤️

```

1 def dicho_rec_2(tab, val, i=0, j=None): # (1)
2 if j is None: # (2)
3 j = len(tab)-1
4 if i > j : :
5 return False
6 m = (i + j) // 2
7 if tab[m] < val :
8 return dicho_rec_2(tab, val, m + 1, j)
9 elif tab[m] > val :
10 return dicho_rec_2(tab, val, i, m - 1)
11 else :
12 return True

```

1. Pour pouvoir appeler simplement la fonction sans avoir à préciser les indices, on leur donne des paramètres par défaut.
2. Il est impossible de donner `j=len(tab)-1` par défaut (car `tab` est aussi un paramètre). On passe donc par une autre valeur (ici `None`) qu'on va ici intercepeter.

Exemple d'utilisation :

### 🐍 Script Python

```

>>> tab = [1, 5, 7, 9, 12, 13]
>>> dicho_rec_2(tab, 12)
True
>>> dicho_rec_2(tab, 17)
False

```

Les algorithmes de dichotomie présentés ci-dessous ont tous en commun de diviser par deux la taille des données de travail à chaque étape. Cette méthode de résolution d'un problème est connue sous le nom de *diviser pour régner*, ou *divide and conquer* en anglais.

Une définition pourrait être :

## Définition ❤

Un problème peut se résoudre en employant le paradigme *diviser pour régner* lorsque :

- il est possible de décomposer ce problème en sous-problèmes **indépendants**.
- la taille de ces sous-problèmes est une **fraction** du problème initial

### Remarques :

- Les sous-problèmes peuvent nécessiter d'être ensuite recombinés entre eux (voir plus loin le tri fusion).

#### 2.1.2. ■ Activité 2 : Tri fusion

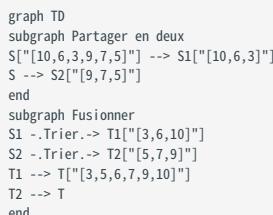
1. Algorithmes de tri vus en première et revus cette année :

- Rappeler rapidement le principe du **tri par sélection** vu en classe de première. Donner les étapes de cet algorithme pour trier la liste [10,6,3,9,7,5]
- Rappeler rapidement le principe du **tri par insertion** vu en classe de première. Donner les étapes de cet algorithme pour trier la liste [10,6,3,9,7,5]
- Quelle est la complexité de ces deux algorithmes ?

2. L'algorithme du **tri fusion** consiste à :

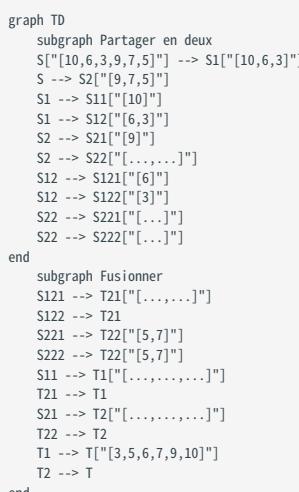
- 1 partager la liste en deux moitiés (à une unité près),
- 2 trier chacune des deux moitiés,
- 3 les fusionner pour obtenir la liste triée.

On a schématisé le tri de la liste [10,6,3,9,7,5] suivant ce principe ci-dessous :



a. Le tri des deux moitiés est lui-même effectué par tri fusion, par conséquent que peut-on dire de cet algorithme ?

b. On a schématisé ci-dessous le fonctionnement complet de l'algorithme pour la liste [10,6,3,9,7,5], recopier et compléter les cases manquantes.



3. Implémentation en Python

- a. Programmer une fonction `partage(l)` qui prend en argument une liste `l` et renvoie les deux moitiés `l1` et `l2` (à une unité près) de `l`. Par exemple `partage([3,7,5])` renvoie `[3]` et `[7,5]`.



- Penser à utiliser les constructions de listes par compréhension
- Les *slices* de Python sont un moyen efficace d'effectuer le partage, mais leur connaissance n'est pas un attendu du programme de terminale. Les élèves intéressés pourront faire leur propre recherche sur le Web.

- b. On donne ci-dessous une fonction `fusion(l1,l2)` qui prend en argument deux listes **déjà triées** `l1` et `l2` et renvoie la liste triée `l` fusion de `l1` et `l2` :

```

1 def fusion(l1,l2):
2 ind1=0
3 ind2=0
4 l = []
5 while ind1<len(l1) and ind2<len(l2):
6 if l1[ind1]<l2[ind2]:
7 l.append(...)
8 ind1+=1
9 else:
10 l.append(...)
11 ind2+=1
12 if ind1==len(l1):
13 for k in range(ind2,len(l2)):
14 l.append(...)
15 else:
16 for k in range(ind1,len(l1)):
17 l.append(...)
18
 return l

```

- i. Recopier et compléter cette fonction.
  - ii. Quel est le rôle des variables `ind1` et `ind2` ?
  - iii. Ajouter un commentaire décrivant le rôle de la boucle `while`.
  - iv. Ajouter un commentaire décrivant le rôle des lignes 12 à 17.
- c. En utilisant les deux fonctions précédentes, écrire une fonction `tri_fusion(l)` qui implémente l'algorithme du tri fusion en Python.

```

1 def tri_fusion(liste):
2 long = len(liste)
3 if long <= 1:
4 return liste
5 else:
6 l1, l2 = partage(liste)
7 l1 = tri_fusion(l1)
8 l2 = tri_fusion(l2)
9
 return fusion(l1,l2)

```



On montre que l'algorithme du tri fusion a une complexité en  $\mathcal{O}(n \log(n))$ , c'est donc un algorithme plus efficace que le tri par insertion ou le tri par sélection qui ont tous les deux une complexité en  $\mathcal{O}(n^2)$ .

## 2.2. 6.2.2 Exercices



### Maximum des éléments d'une liste

On propose l'algorithme suivant pour la recherche du maximum des éléments d'une liste :

- 1 Partager la liste en deux moitiés  $l_1$  et  $l_2$
- 2 Chercher les maximums  $m_1$  de  $l_1$  et  $m_2$  de  $l_2$
- 3 En déduire le maximum  $m$  de  $l$ .

1. Expliquer pourquoi cet algorithme fait partie de la méthode **diviser pour régner**.
2. Cet algorithme est-il récursif ? Justifier.
3. Ecrire une implémentation en Python de cet algorithme.

 **Epreuve Pratique**

Sujet 35 : Exercice 2    Sujet 10 : Exercice 2    Sujet 19 : Exercice 2

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

L'algorithme traite le cas du tableau vide.

L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.

On distingue les trois cas qui renvoient `False` en renvoyant `False,1`, `False,2` et `False,3`.

Compléter l'algorithme de dichotomie donné ci-après.

```

1 def dichotomie(tab, x):
2 """
3 tab : tableau trié dans l'ordre croissant
4 x : nombre entier
5 La fonction renvoie True si tab contient x et False sinon
6 """
7 # cas du tableau vide
8 if ...:
9 return False,1
10 # cas où x n'est pas compris entre les valeurs extrêmes
11 if (x < tab[0]) or ...:
12 return False,2
13 debut = 0
14 fin = len(tab) - 1
15 while debut <= fin:
16 m = ...
17 if x == tab[m]:
18 return ...
19 if x > tab[m]:
20 debut = m + 1
21 else:
22 fin = ...
23 return ...

```

Exemples :

### Script Python

```

>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],27)
(False, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)

```

La fonction `fusion` prend deux listes `L1`, `L2` d'entiers triées par ordre croissant et les fusionne en une liste triée `L12` qu'elle renvoie.

Le code Python de la fonction est

```

1 def fusion(L1,L2):
2 n1 = len(L1)
3 n2 = len(L2)
4 L12 = [0]*(n1+n2)
5 i1 = 0
6 i2 = 0
7 i = 0
8 while i1 < n1 and ... :
9 if L1[i1] < L2[i2]:
10 L12[i] = ...
11 i1 = ...
12 else:
13 L12[i] = L2[i2]
14 i2 = ...
15 i += 1
16 while i1 < n1:
17 L12[i] = ...
18 i1 = i1 + 1
19 i = ...
20 while i2 < n2:
21 L12[i] = ...
22 i2 = i2 + 1
23 i = ...
24 return L12

```

## Exercice n°1 : France 2021

Cet exercice porte sur l'algorithme de tri fusion, qui s'appuie sur la méthode dite de « diviser pour régner ».

### Question 1

[Enoncé](#)    [Solution](#)

- Quel est l'ordre de grandeur du coût, en nombre de comparaisons, de l'algorithme de tri fusion pour une liste de longueur ?
  - Citer le nom d'un autre algorithme de tri. Donner l'ordre de grandeur de son coût, en nombre de comparaisons, pour une liste de longueur .
- Comparer ce coût à celui du tri fusion. Aucune justification n'est attendue.

- la sous-liste de L formée des éléments d'indice strictement inférieur à `len(L)//2` ;
- la sous-liste de L formée des éléments d'indice supérieur ou égal à `len(L)//2` .

On rappelle que la syntaxe `a/b` désigne la division entière de `a` par `b`.

Par exemple,

#### Script Python

```
>>> L = [3, 5, 2, 7, 1, 9, 0]
>>> moitie_gauche(L)
[3, 5, 2]
>>> moitie_droite(L)
[7, 1, 9, 0]
>>> M = [4, 1, 11, 7]
>>> moitie_gauche(M)
[4, 1]
>>> moitie_droite(M)
[11, 7]
```

L'algorithme utilise aussi une fonction `fusion` qui prend en argument deux listes triées `L1` et `L2` et renvoie une liste `L` triée et composée des éléments de `L1` et `L2`.

On donne ci-dessous le code python d'une fonction récursive `tri_fusion` qui prend en argument une liste `L` et renvoie une nouvelle liste triée formée des éléments de `L`.

#### Script Python

```
def tri_fusion(L):
 n = len(L)
 if n<=1 :
 return L
 print(L)
 mg = moitie_gauche(L)
 md = moitie_droite(L)
 L1 = tri_fusion(mg)
 L2 = tri_fusion(md)
 return fusion(L1, L2)
```

## Question 2

[Enoncé](#) [Solution](#)

Donner la liste des affichages produits par l'appel suivant.

### Script Python

```
tri_fusion([7, 4, 2, 1, 8, 5, 6, 3])
```

On s'intéresse désormais à différentes fonctions appelées par tri\_fusion, à savoir moitie\_droite et fusion.

## Question 3

[Enoncé](#) [Solution](#)

Ecrire la fonction moitie\_droite.

## Question 4

[Enoncé](#) [Solution](#)

On donne ci-dessous une version incomplète de la fonction fusion.

```
1 def fusion(L1, L2):
2 L = []
3 n1 = len(L1)
4 n2 = len(L2)
5 i1 = 0
6 i2 = 0
7 while i1 < n1 or i2 < n2 :
8 if i1 >= n1:
9 L.append(L2[i2])
10 i2 = i2 + 1
11 elif i2 >= n2:
12 L.append(L1[i1])
13 i1 = i1 + 1
14 else:
15 e1 = L1[i1]
16 e2 = L2[i2]
17
18
19
20 return L
```

Dans cette fonction, les entiers i1 et i2 représentent respectivement les indices des éléments des listes L1 et L2 que l'on souhaite comparer :

- Si aucun des deux indices n'est valide, la boucle while est interrompue ;
- Si i1 n'est plus un indice valide, on va ajouter à L les éléments de L2 à partir de l'indice i2 ;
- Si i2 n'est plus un indice valide, on va ajouter à L les éléments de L1 à partir de l'indice i1 ;
- Sinon, le plus petit élément non encore traité est ajouté à L et on décale l'indice correspondant.

Écrire sur la copie les instructions manquantes des lignes 17 à 22 permettant d'insérer dans la liste L les éléments des listes L1 et L2 par ordre croissant.

## Exercice n°2 : BAC Polynésie 2021

Cet exercice traite principalement du thème « algorithmique, langages et programmation ». Le but est de comparer le tri par insertion (l'un des algorithmes étudiés en 1ère NSI pour trier un tableau) avec le tri fusion (un algorithme qui applique le principe de « diviser pour régner »).

### 2.2.1. Partie A : Manipulation d'une liste en Python

#### Question A.1

[Enoncé](#)    [Solution](#)

Donner les affichages obtenus après l'exécution du code Python suivant.

#### Script Python

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
notes[3] = 16
print(len(notes))
print(notes)
```

```!!! fabquestion "Question A.2" === "Enoncé" Écrire un code Python permettant d'afficher les éléments d'indice 2 à 4 de la liste notes.

Texte

== "Solution"

2.2.2. Partie B : Tri par insertion

Le tri par insertion est un algorithme efficace qui s'inspire de la façon dont on peut trier une poignée de cartes. On commence avec une seule carte dans la main gauche (les autres cartes sont en tas sur la table) puis on pioche la carte suivante et on l'insère au bon endroit dans la main gauche.

Question B.1

[Enoncé](#) [Solution](#)

Voici une implémentation en Python de cet algorithme. Recopier et compléter les lignes 6 et 7 surlignées (uniquement celles-ci).

```
1 def tri_insertion(liste):
2     """ trie par insertion la liste en paramètre """
3     for indice_courant in range(1,len(liste)):
4         element_a_inserer = liste[indice_courant]
5         i = indice_courant - 1
6         while i >= 0 and liste[i] > ..... :
7             liste[.....] = liste[.....]
8             i = i - 1
9             liste[i + 1] = element_a_inserer
```

On a écrit dans la console les instructions suivantes :

Script Python

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
tri_insertion(notices)
print(notices)
```

On a obtenu l'affichage suivant :

Script Python

```
[3, 7, 8, 9, 12, 14, 17, 18]
```

On s'interroge sur ce qui s'est passé lors de l'exécution de tri_insertion(notices).

Question B.2

[Enoncé](#) [Solution](#)

Donner le contenu de la liste notes après le premier passage dans la boucle for.

Question B.3

[Enoncé](#) [Solution](#)

Donner le contenu de la liste notes après le troisième passage dans la boucle for.

2.2.3. Partie C : Tri fusion

L'algorithme de tri fusion suit le principe de « diviser pour régner ».

- (1) Si le tableau à trier n'a qu'un élément, il est déjà trié.
- (2) Sinon, séparer le tableau en deux parties à peu près égales.
- (3) Trier les deux parties avec l'algorithme de tri fusion.
- (4) Fusionner les deux tableaux triés en un seul tableau.

source : Wikipedia

Question C.1

[Enoncé](#)

Cet algorithme est-il itératif ou récursif ? Justifier en une phrase.

[Solution](#)

Question C.2

[Enoncé](#) [Solution](#)

Expliquer en trois lignes comment faire pour rassembler dans une main deux tas déjà triés de cartes, la carte en haut d'un tas étant la plus petite de ce même tas ;
la deuxième carte d'un tas n'étant visible qu'après avoir retiré la première carte de ce tas.

À la fin du procédé, les cartes en main doivent être triées par ordre croissant.

Une fonction fusionner a été implémentée en Python en s'inspirant du procédé de la question précédente.

Elle prend quatre arguments : la liste qui est en train d'être triée, l'indice où commence la sous-liste de gauche à fusionner, l'indice où termine cette sousliste, et l'indice où se termine la sous-liste de droite.

Question C.3

[Enoncé](#) [Solution](#)

Voici une implémentation de l'algorithme de tri fusion. Recopier et compléter les lignes 8, 9 et 10 soulignées (uniquement celles-ci).

```

1  from math import floor
2
3  def tri_fusion (liste, i_debut, i_fin):
4      if i_debut < i_fin:
5          i_partage = floor((i_debut + i_fin) / 2)
6          tri_fusion(liste, i_debut, .....)
7          tri_fusion(liste, ..... , i_fin)
8          fusionner(liste, ..... , .....)
```

Remarque : la fonction floor renvoie la partie entière du nombre passé en paramètre.

Question C.4

[Enoncé](#) [Solution](#)

Expliquer le rôle de la première ligne du code de la question 3.

permet d'importer la fonction floor() du module math utilisée à la ligne 7

2.2.4. Partie D : Comparaison du tri par insertion et du tri fusion

Voici une illustration des étapes d'un tri effectué sur la liste [3, 41, 52, 26, 38, 57, 9, 49].

[..center}]

Question D.1

[Enoncé](#) [Solution](#)

Quel algorithme a été utilisé : le tri par insertion ou le tri fusion ? Justifier.

Question D.2

[Enoncé](#) [Solution](#)

Identifier le tri qui a une complexité, dans le pire des cas, en $O(n^2)$ et identifier le tri qui a une complexité, dans le pire des cas, en $O(n \log_2 n)$.

Remarque : n représente la longueur de la liste à trier.

Question D.3

[Enoncé](#) [Solution](#)

Justifier brièvement ces deux complexités.

Inversions dans une liste : FRANCE CANDIDAT LIBRE SUJET 1

Cet exercice traite de manipulation de tableaux, de récursivité et du paradigme « diviser pour régner ».

Dans un tableau Python d'entiers tab, on dit que le couple d'indices (i, j) forme une inversion lorsque $i < j$ et $tab[i] > tab[j]$. On donne ci-dessous quelques exemples.

- Dans le tableau [1, 5, 3, 7], le couple d'indices (1,2) forme une inversion car $5 > 3$.
Par contre, le couple (1,3) ne forme pas d'inversion car $5 < 7$. Il n'y a qu'une inversion dans ce tableau.
- Il y a trois inversions dans le tableau [1, 6, 2, 7, 3], à savoir les couples d'indices (1, 2), (1, 4) et (3, 4).
- On peut compter six inversions dans le tableau [7, 6, 5, 3] : les couples d'indices (0, 1), (0, 2), (0, 3), (1, 2), (1, 3) et (2, 3).

On se propose dans cet exercice de déterminer le nombre d'inversions dans un tableau quelconque.

Questions préliminaires

Question 1

[Enoncé](#) [Solution](#)

Expliquer pourquoi le couple (1, 3) est une inversion dans le tableau [4, 8, 3, 7].

Question2

[Enoncé](#) [Solution](#)

Justifier que le couple (2, 3) n'en est pas une.

2.2.5. Partie A : Méthode itérative

Le but de cette partie est d'écrire une fonction itérative nombre_inversion qui renvoie le nombre d'inversions dans un tableau. Pour cela, on commence par écrire une fonction fonction1 qui sera ensuite utilisée pour écrire la fonction nombre_inversion.

Question A.1

[Enoncé](#) [Solution](#)

On donne la fonction suivante.

Script Python

```
def fonction1(tab, i):
    nb_elem = len(tab)
    cpt = 0
    for j in range(i+1, nb_elem):
        if tab[j] < tab[i]:
            cpt += 1
    return cpt
```

- Indiquer ce que renvoie la fonction1(tab, i) dans les cas suivants.
- Cas n°1 : tab = [1, 5, 3, 7] et i = 0.
 - Cas n°2 : tab = [1, 5, 3, 7] et i = 1.
 - Cas n°3 : tab = [1, 5, 2, 6, 4] et i = 1.
- Expliquer ce que permet de déterminer cette fonction.

Question A.2

[Enoncé](#) [Solution](#)

En utilisant la fonction précédente, écrire une fonction nombre_inversion(tab) qui prend en argument un tableau et renvoie le nombre d'inversions dans ce tableau.

On donne ci-dessous les résultats attendus pour certains appels.

Texte

```
>>> nombre_inversions([1, 5, 7])
0
>>> nombre_inversions([1, 6, 2, 7, 3])
3
>>> nombre_inversions([7, 6, 5, 3])
6
```

Question A.3

[Enoncé](#) [Solution](#)

Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ?

Aucune justification n'est attendue.

2.2.6. Partie B : Méthode récursive

Le but de cette partie est de concevoir une version récursive de la fonction nombre_inversion.

On définit pour cela des fonctions auxiliaires.

Question B.1

[Enoncé](#) [Solution](#)

Donner le nom d'un algorithme de tri ayant une complexité meilleure que quadratique.

Dans la suite de cet exercice, on suppose qu'on dispose d'une fonction tri(tab) qui prend en argument un tableau et renvoie un tableau contenant les mêmes éléments rangés dans l'ordre croissant.

Question B.2

[Enoncé](#) [Solution](#)

Écrire une fonction moitie_gauche(tab) qui prend en argument un tableau tab et renvoie un nouveau tableau contenant la moitié gauche de tab. Si le nombre d'éléments de tab est impair, l'élément du centre se trouve dans cette partie gauche.

On donne ci-dessous les résultats attendus pour certains appels.

Script Python

```
>>> moitie_gauche([])
[]
>>> moitie_gauche([4, 8, 3])
[4, 8]
>>> moitie_gauche ([4, 8, 3, 7])
[4, 8]
```

Dans la suite, on suppose qu'on dispose de la fonction moitie_droite(tab) qui renvoie la moitié droite sans l'élément du milieu.

Question B.3

[Enoncé](#) [Solution](#)

On suppose qu'une fonction nb_inv_tab(tab1, tab2)a été écrite. Cette fonction renvoie le nombre d'inversions du tableau obtenu en mettant bout à bout les tableaux tab1 et tab2, à condition que tab1 et tab2 soient triés dans l'ordre croissant.

On donne ci-dessous deux exemples d'appel de cette fonction :

Script Python

```
>>> nb_inv_tab([3, 7, 9], [2, 10])
3
>>> nb_inv_tab([7, 9, 13], [7, 10, 14])
3
```

En utilisant la fonction nb_inv_tab et les questions précédentes, écrire une fonction récursive nb_inversions_rec(tab) qui permet de calculer le nombre d'inversions dans un tableau. * Cette fonction renverra le même nombre que nombre_inversions(tab) de la partie A. On procédera de la façon suivante :

- Séparer le tableau en deux tableaux de tailles égales (à une unité près).
- Appeler récursivement la fonction nb_inversions_rec pour compter le nombre d'inversions dans chacun des deux tableaux.
- Trier les deux tableaux (on rappelle qu'une fonction de tri est déjà définie).
- Ajouter au nombre d'inversions précédemment comptées le nombre renvoyé par la fonction nb_inv_tab avec pour arguments les deux tableaux triés.



Cart de tour d'une image

1. Pour faire tourner une image carré de côté $\backslash(2^n)$ pixels d'un quart de tour à gauche, on propose la méthode suivante :

- Diviser l'image en quatre quarts Q1,Q2,Q3,Q4

| | |
|-----------|-----------|
| Q1 | Q2 |
| Q3 | Q4 |

- Faire tourner chacun des quarts d'un quart de tour à gauche

| | |
|-----------|-----------|
| Q1 | Q2 |
| Q3 | Q4 |

- Permuter chaque quart afin de le placer correctement

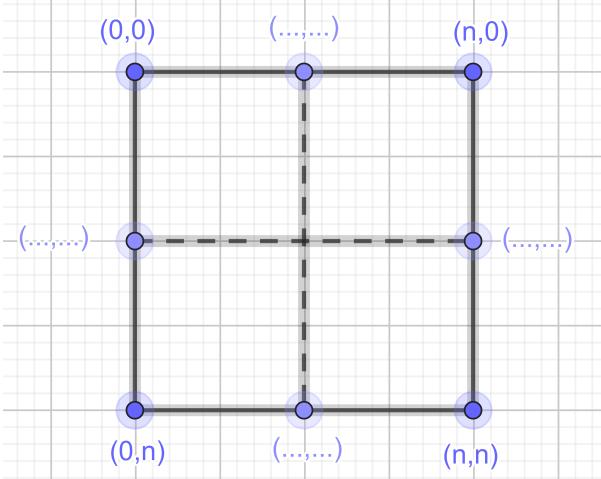
| | |
|-----------|-----------|
| Q2 | Q4 |
| Q1 | Q3 |

Expliquer pourquoi cette méthode est une illustration de la technique diviser pour régner.

2. C'est algorithme est-il du type itératif ou récursif ? Justifier.

3. Découpage de l'image en quatres quart à l'aide du module pil de manipulation d'images

- a. On a représenté une image carré de n pixels de côté avec le système de coordonnées d'une image dans le module pil. Quelles sont les coordonnées manquantes ?



- b. La méthode `crop` du module pil permet d'extraire une portion rectangulaire d'une image en donnant les coordonnées des coins supérieur gauche et inférieur droit du rectangle. Compléter la fonction Python suivante qui prend en entrée une image et retourne les quatre quart de cette image.

Script Python

```
from PIL import Image
def partage_quart(image):
    n = image.width
    if n > 1:
        q1 = image.crop((0,0,n/2,n/2))
        q2 = image.crop((...,0,...,0))
        q3 = image.crop((0,...,0,...))
        q4 = image.crop((...,0,...,0))
    return q1,q2,q3,q4
```

- c. Tester cette fonction (on pourra utiliser cette image carré)

Aide

- La création d'une image dans pil à partir d'un fichier s'effectue à l'aide de :

Script Python

```
img_test = Image.open("mettre ici le nom du fichier")
```

- La visualisation d'une image s'effectue à l'aide de :

Script Python

```
img_test.show()
```

- d. Ajouter une instruction `assert` permettant de vérifier que l'image est carré (c'est à dire `image.width==image.height`)

- e. Ajouter une instruction `assert` permettant de vérifier que `n` est pair.

4. Compléter puis tester la fonction python qui implémente l'algorithme décrit à la question 1.

Script Python

```
def quart_tour(image):
    n = image.width
    # Partage de l'image en quatre quart
    if n>1:
        q1,q2,q3,q4 = partage_quart(image)
```

```
# Rotation de chacun des quarts
rq1 = quart_tour(q1)
rq2 = quart_tour(q2)
rq3 = quart_tour(q3)
rq4 = quart_tour(q4)
# Reconstruction de l'image
resultat = Image.new('RGB',image.size)
resultat.paste(rq2,(0,0))
resultat.paste(....,(n//2,0))
resultat.paste(rq1,(....,...))
resultat.paste(....,(....,...))
return resultat
else:
    return image
```

7. BAC

1. 7.1 2021

1.1. 7.1.1 2021 Correction épreuves écrites

Remarques :

- les sujets sont classés dans l'ordre alphabétique de leur repère,
- chaque sujet comporte 5 exercices,
- si un exercice est corrigé son numéro est indiqué en vert, sinon en rouge

| Repère | Centre | Jour | Téléchargement | Correction |
|-------------|--------------------------------|------|----------------|------------|
| 21-NSIJ1AN1 | Amérique du nord | 1 | 21-NSIJ1AN1 | ① ② ③ ④ ⑤ |
| 21-NSIJ1G11 | Etranger | 1 | 21-NSIJ1G11 | ① ② ③ ④ ⑤ |
| 21-NSIJ1ME1 | Métropole | 1 | 21-NSIJ1ME1 | ① ② ③ ④ ⑤ |
| 21-NSIJ1ME2 | Métropole candidats libres | 1 | 21-NSIJ1ME2 | ① ② ③ ④ ⑤ |
| 21-NSIJ1ME3 | Métropole session de septembre | 1 | 21-NSIJ1ME3 | ① ② ③ ④ ⑤ |
| 21-NSIJ2G11 | Etranger | 2 | 21-NSIJ2G11 | ① ② ③ ④ ⑤ |
| 21-NSIJ2ME1 | Métropole | 2 | 21-NSIJ2ME1 | ① ② ③ ④ ⑤ |
| 21-NSIJ2ME2 | Métropole candidats libres | 2 | 21-NSIJ2ME2 | ① ② ③ ④ ⑤ |
| 21-NSIJ2ME3 | Métropole session de septembre | 2 | 21-NSIJ2ME3 | ① ② ③ ④ ⑤ |
| 21-NSIJ2PO1 | Polynésie | 2 | 21-NSIJ2PO1 | ① ② ③ ④ ⑤ |

1.2. 7.1.2 2021 : Epreuves écrites

1.2.1. Amérique du nord - jour 1 : 21-NSIJ1AN1

[21-NSIJ1AN1 !\[\]\(f4faeb491fb7969817afdfd3aa8a26a8_img.jpg\)](#)

- Exercice 1** : bases de données relationnelles et langage SQL
- Exercice 2** : routage, processus et systèmes sur puces
- Exercice 3** : tableaux et programmation de base en Python
- Exercice 4** : arbres binaires et algorithmes associés
- Exercice 5** : notion de pile, de file et programmation de base en Python

1.2.2. Etranger - jour 1 : 21-NSIJ1G11

21-NSIJ1G11 

- **Exercice 1** : programmation objet (code de César)
- **Exercice 2** : structures de données (dictionnaires)
- **Exercice 3** : arbres binaires de recherche
- **Exercice 4** : réseau
- **Exercice 5** : structure de données (piles)

1.2.3. Métropole - jour 1 : 21-NSIJ1ME1

21-NSIJ1ME1 

- **Exercice 1** : arbres binaires de recherche
- **Exercice 2** : gestion des processus, opérateurs booléens
- **Exercice 3** : base de données et langage SQL
- **Exercice 4** : algorithme de tri fusion et méthode diviser pour régner
- **Exercice 5** : réseaux et protocoles de routage

1.2.4. Métropole candidats libres - jour 1 : 21-NSIJ1ME2

21-NSIJ1ME2 

- **Exercice 1** : bases de données
- **Exercice 2** : notions de piles et programmation orientée objet
- **Exercice 3** : gestion des processus et protocoles de routage
- **Exercice 4** : algorithme et programmation en Python
- **Exercice 5** : manipulation de tableaux, récursivité, méthode "diviser pour régner"

1.2.5. Métropole session de septembre - jour 1 : 21-NSIJ1ME3

21-NSIJ1ME3 

- **Exercice 1** : protocoles de communication, réseau, protocoles de routage
- **Exercice 2** : algorithmique (recherche dichotomique), langages et programmation (récursivité)
- **Exercice 3** : bases de données et langage SQL
- **Exercice 4** : Structure de données (programmation objet), langages et programmation
- **Exercice 5** : structures de données (arbre, arbre binaire, pile)

1.2.6. Etranger - jour 2 : 21-NSIJ2G11

21-NSIJ2G11 

- **Exercice 1 :** structures de données : piles
- **Exercice 2 :** programmation python, tuples et liste
- **Exercice 3 :** Conversion décimal/binaire, table de vérité, codages des caractères
- **Exercice 4 :** Base de données
- **Exercice 5 :** programmation Python : commande d'un bandeau de diodes à l'aide d'un raspberry

1.2.7. Métropole - jour 2 : 21-NSIJ2ME1

21-NSIJ2ME1 

- **Exercice 1 :** arbres et programmation orientée objet
- **Exercice 2 :** base de données relationnelles
- **Exercice 3 :** réseaux et protocoles de routage
- **Exercice 4 :** gestion des processus et des ressources
- **Exercice 5 :** structure de données linéaires

1.2.8. Métropole candidats libres - jour 2 : 21-NSIJ2ME2

21-NSIJ2ME2 

- **Exercice 1 :** bases de données relationnelles et langage SQL
- **Exercice 2 :** gestion des processus et des ressources par un système d'exploitation
- **Exercice 3 :** arbres binaires de recherche et programmation orientée objet
- **Exercice 4 :** programmation et récursivité
- **Exercice 5 :** programmation

1.2.9. Métropole session de septembre - jour 2 : 21-NSIJ2ME3

21-NSIJ2ME3 

- **Exercice 1 :** réseau, protocoles de communication et de routages
- **Exercice 2 :** structure de données, langages et programmation
- **Exercice 3 :** base de données
- **Exercice 4 :** programmation orientée objet, langages et programmation
- **Exercice 5 :** traitement de données en table (CSV), langages et programmation"

1.2.10. Polynésie - jour 2 : 21-NSIJ2PO1

21-NSIJ2PO1 

- **Exercice 1 :** algorithmique et programmation (*algorithmes de tri*)
- **Exercice 2 :** données en table, bases de données
- **Exercice 3 :** arbres binaires de recherche et programmation orientée objet
- **Exercice 4 :** routage, architecture matérielle
- **Exercice 5 :** données en table, bases de données

2. 7.2 2022

2.1. 7.2.1 2022 Correction épreuves écrites

Remarques :

- les sujets sont classés dans l'ordre alphabétique de leur repère,
- chaque sujet comporte 5 exercices,
- si un exercice est corrigé son numéro est indiqué en vert, sinon en rouge

| Repère | Centre | Jour | Téléchargement | Correction |
|-------------|--------------------------------|------|----------------|------------|
| 22-NSIJ1G11 | Etranger | 1 | 22-NSIJ1G11 | ① ② ③ ④ ⑤ |
| 22-NSIJ1ME1 | Métropole | 1 | 22-NSIJ1ME1 | ① ② ③ ④ ⑤ |
| 22-NSIJ1PO1 | Polynésie | 1 | 22-NSIJ1PO1 | ① ② ③ ④ ⑤ |
| 22-NSIJ2G11 | Etranger | 2 | 22-NSIJ2G11 | ① ② ③ ④ ⑤ |
| 22-NSIJ2ME1 | Métropole | 2 | 22-NSIJ2ME1 | ① ② ③ ④ ⑤ |
| 22-NSIJ1AN1 | Amérique du nord | 1 | 22-NSIJ1AN1 | ① ② ③ ④ ⑤ |
| 22-NSIJ2AN1 | Amérique du nord | 2 | 22-NSIJ2AN1 | ① ② ③ ④ ⑤ |
| 22-NSIJ1JA1 | Asie-Pacifique | 1 | 22-NSIJ1JA1 | ① ② ③ ④ ⑤ |
| 22-NSIJ2JA1 | Asie-Pacifique | 2 | 22-NSIJ2JA1 | ① ② ③ ④ ⑤ |
| 22-NSIJ1LR1 | Mayotte et réseau AEFE | 1 | 22-NSIJ1LR1 | ① ② ③ ④ ⑤ |
| 22-NSIJ2LR1 | Mayotte et réseau AEFE | 2 | 22-NSIJ2LR1 | ① ② ③ ④ ⑤ |
| 22-NSIJ1ME3 | Métropole session de septembre | 1 | 22-NSIJ1ME3 | ① ② ③ ④ ⑤ |
| 22-NSIJ1AS1 | Amérique du sud | 1 | 22-NSIJ1AS1 | ① ② ③ ④ ⑤ |
| 22-NSIJ2AS1 | Amérique du sud | 2 | 22-NSIJ2AS1 | ① ② ③ ④ ⑤ |

2.2. 7.2.2 2022 : Epreuves écrites

2.2.1. Etranger - jour 1 : 22-NSIJ1G11

22-NSIJ1G11 

- Exercice 1** : structures de données (listes, p-uplets et dictionnaires)
- Exercice 2** : structures de données (files et la programmation objet en langage python)
- Exercice 3** : structures de données (dictionnaires)
- Exercice 4** : les bases de données
- Exercice 5** : architecture matérielle des ordinateurs, les réseaux et sur les protocoles de routage

2.2.2. Métropole - jour 1 : 22-NSIJ1ME1

22-NSIJ1ME1 

- **Exercice 1 :** structures de données
- **Exercice 2 :** bases de données
- **Exercice 3 :** représentations binaires et protocoles de routage
- **Exercice 4 :** parcours des arbres binaires, diviser pour régner, récursivité
- **Exercice 5 :** programmation orientée objet

2.2.3. Polynésie - jour 1 : 22-NSIJ1PO1

22-NSIJ1PO1 

- **Exercice 1 :** programmation et récursivité
- **Exercice 2 :** architecture matérielle, ordonnancement et expressions booléennes
- **Exercice 3 :** base de données, modèle relationnel, langage SQL
- **Exercice 4 :** structures de données, piles
- **Exercice 5 :** algorithmique, algorithme sur les arbres binaires

2.2.4. Etranger - jour 2 : 22-NSIJ2G11

22-NSIJ2G11 

- **Exercice 1 :** langages et programmation (récursivité)
- **Exercice 2 :** structure de données (dictionnaires)
- **Exercice 3 :** base de données
- **Exercice 4 :** structures de données, programmation objet
- **Exercice 5 :** architectures matérielles, systèmes d'exploitation et réseaux (protocoles de routage)

2.2.5. Métropole - jour 2 : 22-NSIJ2ME1

22-NSIJ2ME1 

- **Exercice 1 :** arbres binaires de recherche, la programmation orientée objet et la récursivité
- **Exercice 2 :** structures de données
- **Exercice 3 :** réseaux et protocoles de routage
- **Exercice 4 :** base de données relationnelles et langage SQL
- **Exercice 5 :** programmation objet et méthode diviser pour régner

2.2.6. Amérique du nord - jour 1 : 22-NSIJ1AN1

22-NSIJ1AN1 

- **Exercice 1 :** bases de données relationnelles et langage SQL
- **Exercice 2 :** réseaux et protocoles de routage
- **Exercice 3 :** arbres binaires de recherche et algorithmes associés
- **Exercice 4 :** chaînes de caractères, tableau et programmation de base en Python
- **Exercice 5 :** files, tableaux et algorithmes associés

2.2.7. Amérique du nord - jour 2 : 22-NSIJ2AN1

22-NSIJ2AN1 

- **Exercice 1 :** listes, arbres binaires de recherche et programmation orientée objet
- **Exercice 2 :** systèmes d'exploitation, gestion des processus par un système d'exploitation
- **Exercice 3 :** bases de données relationnelles et le langage SQL
- **Exercice 4 :** arbres binaires et algorithmes associés
- **Exercice 5 :** tableaux à deux dimensions et la programmation Python en général

2.2.8. Asie-Pacifique - jour 1 : 22-NSIJ1JA1

22-NSIJ1JA1 

- **Exercice 1 :** algorithmique, chaînes de caractères, complexité
- **Exercice 2 :** base de données
- **Exercice 3 :** systèmes d'exploitation
- **Exercice 4 :** programmation objet en langage Python
- **Exercice 5 :** programmation Python

2.2.9. Asie-Pacifique - jour 2 : 22-NSIJ2JA1

22-NSIJ2JA1 

- **Exercice 1 :** systèmes d'exploitation Linux
- **Exercice 2 :** arbres binaires de recherche
- **Exercice 3 :** structures de données, programmation
- **Exercice 4 :** bases de données et langage SQL
- **Exercice 5 :** Exécution de programmes, recherche et corrections de bugs

2.2.10. Mayotte et réseau AEFE - jour 1 : 22-NSIJ1LR1

22-NSIJ1LR1 

- **Exercice 1** : structures de données (*listes, piles et files*)
- **Exercice 2** : structures de données (*programmation objet*)
- **Exercice 3** : bases de données relationnelles et langage *SQL*
- **Exercice 4** : algorithmique (*arbres binaires en profondeurs préfixe et infixé*)
- **Exercice 5** : réseau, protocoles de routage, langage et programmation

2.2.11. Mayotte et réseau AEFE - jour 2 : 22-NSIJ2LR1

22-NSIJ2LR1 

- **Exercice 1** : structures de données (*pile*)
- **Exercice 2** : bases de données
- **Exercice 3** : représentation binaire d'un entier relatif, systèmes d'exploitation
- **Exercice 4** : arbres binaires de recherche
- **Exercice 5** : algorithmes et programmation Python

2.2.12. Métropole session de septembre - jour 1 : 22-NSIJ1ME3

22-NSIJ1ME3 

- **Exercice 1** : algorithmique, arbres binaires de recherche et leurs parcours
- **Exercice 2** : programmation orientée objet, itérations et récursivité
- **Exercice 3** : bases de données relationnelles et langage *SQL*
- **Exercice 4** : architecture matérielle, gestion de processus et réseaux
- **Exercice 5** : notion de file et programmation en Python

2.2.13. Amérique du sud - jour 1 : 22-NSIJ1AS1

22-NSIJ1AS1 

- **Exercice 1** : bases de données
- **Exercice 2** : programmation et algorithmes de tri
- **Exercice 3** : arbres binaires
- **Exercice 4** : gestion des processus et des ressources par un système d'exploitation
- **Exercice 5** : réseaux et protocoles de routage

2.2.14. Amérique du sud - jour 2 : 22-NSIJ2AS1

22-NSIJ2AS1 

- **Exercice 1 :** programmation, algorithmique et complexité
- **Exercice 2 :** réseaux et routage
- **Exercice 3 :** base de données
- **Exercice 4 :** programmation en Python, récursivité et méthode diviser pour régner
- **Exercice 5 :** arbres binaires, programmation orientée objet et récursivité

8. Annales du Bac E.P.

1. 8.1 Épreuve pratique

1.1. 8.1.1 Modalités

Textes réglementaires

- https://www.education.gouv.fr/bo/20/Special2/MENE2001797N.htm?cid_bo=149244

- Durée : 1 heure
- L'épreuve pratique donne lieu à une note sur **8 points**, qui s'ajouteront aux 12 points de l'épreuve écrite.

La partie pratique consiste en la résolution de **deux exercices sur ordinateur**, chacun étant noté sur **4 points**.

Le candidat est évalué sur la base d'un dialogue avec un professeur-examinateur. Un examinateur évalue au maximum quatre élèves. L'examinateur ne peut pas évaluer un élève qu'il a eu en classe durant l'année en cours. L'évaluation de cette partie se déroule au cours du deuxième trimestre pendant la période de l'épreuve écrite de spécialité.

Premier exercice

Le premier exercice consiste à programmer un algorithme figurant explicitement au programme, ne présentant pas de difficulté particulière, dont on fournit une spécification. Il s'agit donc de restituer un algorithme rencontré et travaillé à plusieurs reprises en cours de formation. Le sujet peut proposer un jeu de test avec les réponses attendues pour permettre au candidat de vérifier son travail.

Deuxième exercice

Pour le second exercice, un programme est fourni au candidat. Cet exercice ne demande pas l'écriture complète d'un programme, mais permet de valider des compétences de programmation suivant des modalités variées : le candidat doit, par exemple, compléter un programme « à trous » afin de répondre à une spécification donnée, ou encore compléter un programme pour le documenter, ou encore compléter un programme en ajoutant des assertions, etc.

1.2. 8.1.2 Banque d'exercices

Textes réglementaires

- <https://eduscol.education.fr/2661/banque-des-epreuves-pratiques-de-specialite-nsi>

La banque d'exercices est publique et peut être téléchargée en un pdf unique [ici](#).

2. 8.2 Epreuve Pratique - Consigne

2.1. 8.2.1 Déroulement de l'épreuve pratique

Chaque session d'épreuve pratique dure une heure. Une fois l'épreuve lancée, l'examinateur **laisse les candidats en autonomie pendant 5 à 10 minutes**, puis **l'examinateur passe ensuite à tour de rôle auprès de chaque candidat afin d'installer un dialogue** avec ce candidat.

Durant l'épreuve, l'examinateur doit s'adresser aux candidats en faisant preuve de discrétion, afin de ne pas gêner les autres candidats ou afin de ne pas leur procurer une aide non demandée.

Chaque heure d'épreuve est suivie d'un quart d'heure de Préparation/Accueil. Durant ce quart d'heure :

- l'examinateur remplit les fiches d'évaluation individuelles des candidats ;
- le référent informatique installe sur les postes les nouveaux sujets ;
- l'examinateur accueille les candidats de la session suivante, vérifie leur identité et leur fait signer la feuille d'émargement.

2.2. 8.2.2 Evaluation du candidat

2.2.1. Grille d'évaluation

Lors de l'évaluation, il est attendu une évaluation précise des compétences qui peut prendre appui sur l'exemple de grille d'évaluation de la note de service du 16-3-2021 jointe en annexe 1.

Cette **grille** est un **exemple de grille de compétence** pour aider les examinateurs ; elle n'a **aucun caractère prescriptif**. Chaque exercice est noté sur 4 points et doit faire l'objet d'une notation. Les notes des deux exercices sont exprimées à 0,5 point près. Pour chaque exercice, le professeur examinateur peut attribuer au candidat des notes du types 3 - 3,5 - 4 mais pas 3,75. La note sur 8 points attribuée au candidat est la somme de ces deux notes. Cette somme est laissée telle quelle sans être arrondie. Par exemple, un candidat qui a obtenu 3 points sur 4 au premier exercice et 3,5 points sur 4 au deuxième exercice se voit attribué la note de 6,5 points sur 8.

L'aspect oral fait pleinement partie de l'épreuve pratique. Il est donc conseillé de poser une ou plusieurs questions sur le code pour chaque exercice.

Concernant la notation des exercices, il est recommandé d'appliquer le barème suivant :

- **3 points pour la programmation**
- **1 point pour l'expression orale.**

Pour la programmation, il faut donner au minimum 2,5 points si le code est cohérent même si le programme ne « tourne » pas.

| Critères d'évaluation | Définition du critère | Très insuffisant | Insuffisant | Satisfaisant | Très satisfaisant |
|--|---|---|--|---|---|
| Connaissance des savoir-faire techniques | Connaissance des concepts de base | Besoin permanent d'assistance | A besoin de consignes complémentaires et d'assistance ponctuelle | A rarement besoin de consignes complémentaires | Travaille de façon autonome |
| Qualité de mise en œuvre | Niveau de conformité des opérations réalisées | Fait fréquemment des erreurs, exige une surveillance permanente | Produit un travail qu'il faut contrôler régulièrement | Fait des erreurs minimes qu'il ou elle parvient à verbaliser et propose des solutions | Travaille sans erreur |
| Qualité du dialogue | Justification | Pas de réponse | Pas clair | Relativement clair mais manque parfois de précision | Démontre une capacité à reformuler pour bien se faire comprendre. |

3. 8.3 2022

3.1. 8.3.1 Année 2022 : Sujets d'épreuves pratiques

Attention

Des erreurs d'énoncé figurent dans certains des sujets, dans la correction ils sont signalés par une mention  **Bug**. De la même façon, certains sujets contiennent des techniques de programmation problématiques indiquées par une mention  Attention.

| Numéro | Lien de téléchargement | Thème exercice 1 | Thème exercice 2 | Code fourni | Correction |
|--------|------------------------|--|--|--|------------|
| 1 | Sujet N°1 | Recherche d'occurrences | Rendu de monnaie récursif |  Code | 2022-S01 |
| 2 | Sujet N°2 | Calcul d'une moyenne | Triangle de Pascal |  Code | 2022-S02 |
| 3 | Sujet N°3 | Codage par différence | Arbre binaire et expression arithmétique |  Code | 2022-S03 |
| 4 | Sujet N°4 | Entiers consécutifs dans un tableau | Codage d'une image en liste de liste |  Code | 2022-S04 |
| 5 | Sujet N°5 | Recherche du minimum et du maximum | POO : cartes et paquet de cartes |  Code | 2022-S05 |
| 6 | Sujet N°6 | Valeur et indice du maximum dans une liste | Recherche textuelle |  Code | 2022-S06 |
| 7 | Sujet N°7 | Conversion binaire/ décimal | Tri à bulles |  Code | 2022-S07 |
| 8 | Sujet N°8 | Recherche de la première occurrence | Insertion dans une liste triée |  Code | 2022-S08 |
| 9 | Sujet N°9 | Suite de Collatz | Codage d'un mot |  Code | 2022-S09 |
| 10 | Sujet N°10 | Nombre d'occurrence avec un dictionnaire | Fusion de deux listes triées |  Code | 2022-S10 |
| 11 | Sujet N°11 | Recherche dichotomique | Code de César |  Code | 2022-S11 |
| 12 | Sujet N°12 | Calcul d'une moyenne | Séparation des 0 et des 1 dans une liste |  Code | 2022-S12 |
| 13 | Sujet N°13 | Rendu de monnaie | POO : gestion d'une file |  Code | 2022-S13 |
| 14 | Sujet N°14 | Mots correspondants à un motif | Recherche d'un cycle |  Code | 2022-S14 |
| 15 | Sujet N°15 | Nombre de répétitions d'un élément | Conversion en binaire |  Code | 2022-S15 |
| 16 | Sujet N°16 | Maximum d'un élément dans une liste | Structure de données : piles |  Code | 2022-S16 |
| 17 | Sujet N°17 | Nombre de mots dans une phrase | POO : arbre binaire de recherche |  Code | 2022-S17 |
| 18 | Sujet N°18 | Minimum d'une liste de températures | Palindrome |  Code | 2022-S18 |
| 19 | Sujet N°19 | Multiplications avec uniquement | Recherche dichotomique |  Code | 2022-S19 |

| Numéro | Lien de téléchargement | Thème exercice 1 | Thème exercice 2 | Code fourni | Correction |
|--------|------------------------|--|--|--|------------|
| | | additions et soustractions | | | |
| 20 | Sujet N°20 | Ou exclusif entre deux tableaux | POO : Test de carrés magiques |  Code | 2022-S20 |
| 21 | Sujet N°21 | Multiplications avec uniquement additions et soustractions | Recherche dichotomique dans un tableau trié |  Code | 2022-S21 |
| 22 | Sujet N°22 | Ecriture d'une chaîne de caractères à l'envers | Crible d'Eratosthène |  Code | 2022-S22 |
| 23 | Sujet N°23 | Maximum des valeurs d'un dictionnaire | POO : pile pour noter une expression arithmétique |  Code | 2022-S23 |
| 24 | Sujet N°24 | Maximum des éléments d'une liste | POO : expression bien parenthésée et piles |  Code | 2022-S24 |
| 25 | Sujet N°25 | Traitement de données en tables | Recherche récursive dans un tableau |  Code | 2022-S25 |
| 26 | Sujet N°26 | Minimum des éléments d'une liste | Séparation des 0 et des 1 dans une liste |  Code | 2022-S26 |
| 27 | Sujet N°27 | Taille d'un arbre binaire représenté par un dictionnaire | Tri par sélection |  Code | 2022-S27 |
| 28 | Sujet N°28 | Calcul de moyenne | Conversion decimal en binaire |  Code | 2022-S28 |
| 29 | Sujet N°29 | Termes de la suite de Fibonacci | Recherche de maximum dans une liste |  Code | 2022-S29 |
| 30 | Sujet N°30 | Fusion de deux listes déjà triées | Conversion numération romaine |  Code | 2022-S30 |
| 31 | Sujet N°31 | Nombre d'occurrence d'un élément dans une liste | Rendu de monnaie |  Code | 2022-S31 |
| 32 | Sujet N°32 | Dernière occurrence d'un élément dans une liste | POO : adresse IP |  Code | 2022-S32 |
| 33 | Sujet N°33 | Conversion binaire décimal | Tri par insertion |  Code | 2022-S33 |
| 34 | Sujet N°34 | Lettre la plus fréquente dans un texte | Représentation d'une image par une liste de listes |  Code | 2022-S34 |
| 35 | Sujet N°35 | Calcul d'une moyenne | Recherche dichotomique |  Code | 2022-S35 |

| Numéro | Lien de téléchargement | Thème exercice 1 | Thème exercice 2 | Code fourni | Correction |
|--------|----------------------------|---|---|--|--------------------------|
| 36 | Sujet N°36 | Dernière occurrence d'un élément dans une liste | Calcul de la distance entre deux points |  Code | 2022-S36 |
| 37 | Sujet N°37 | Vérification si une liste est triée ou non | Comptabilisation de votes (dictionnaires) |  Code | 2022-S37 |
| 38 | Sujet N°38 | Tri par sélection | Jeu du nombre mystère |  Code | 2022-S38 |
| 39 | Sujet N°39 | Calcul d'une moyenne | Représentation d'une image par une liste de liste |  Code | 2022-S39 |
| 40 | Sujet N°40 | Recherche d'un élément dans une liste | Calcul de moyennes (dictionnaires) |  Code | 2022-S40 |

3.2. 8.3.2 Corrigés épreuves pratiques

3.2.1. Corrigé sujet 01 - Année : 2022

Sujet 01 - 20222 

EXERCICE 1

```
1 def recherche(caractere,mot):
2     occurrence = 0
3     for c in mot:
4         if c == caractere:
5             occurrence += 1
6     return occurrence
```

Commentaires

C'est un exercice classique de parcours d'un itérable en comptant les occurrences d'apparition d'une valeur. Un parcours par élément suffit, les indices des occurrences n'étant pas utilisées.

EXERCICE 2

```
1 Pieces = [100,50,20,10,5,2,1]
2 def rendu_glouton(arendre, solution=[], i=0):
3     if arendre == 0:
4         return solution #(1)
5     p = Pieces[i]
6     if p <= arendre : #(2)
7         solution.append(p)
8         return rendu_glouton(arendre - p, solution, i)
9     else :
10        return rendu_glouton(arendre, solution, i+1) #(3)
```

1. Lorsqu'il n'y a plus rien à rendre on renvoie la solution
2. Si la pièce est plus petite que la somme à rendre on l'ajoute à la solution et on diminue la somme à rendre
3. La pièce dépasse la somme à rendre, on relance le processus en regardant la pièce suivante dans la liste

Attention

- Le code fourni utilise un objet mutable (une liste) comme paramètre par défaut d'une fonction :

Script Python

```
def rendu_glouton(arendre, solution=[], i=0):
```

C'est une très mauvaise pratique car source d'erreurs, en effet la variable `solution` étant mutable elle est modifiée par la fonction lors d'un premier appel et ne sera donc plus vide lors des appels suivants. Pour constater le problème, faire plusieurs appels à cette fonction sans spécifier les arguments ayant des valeurs par défaut. Pour une solution à ce problème, on pourra par exemple consulter [ce site](#)

3.2.2. Corrigé sujet 02 - Année : 2022

Sujet 02 - 20222 

EXERCICE 1

```

1 def moyenne(donnees):
2     somme_notes = 0
3     somme_coefficients = 0
4     for d in donnees:
5         note = d[0]
6         coefficient = d[1]
7         somme_notes += note*coefficient
8         somme_coefficients += coefficient
9     return somme_notes/somme_coefficients

```

 Commentaires

Bien comprendre la façon dont les données sont organisées, c'est une liste dont chaque élément est un tuple (couple, coefficient). Ainsi dans l'exemple de l'énoncé : `donnees=[(15,2),(9,1),(12,3)]` la première donnée notée `d` est le couple `(15,2)` et donc la première note est `d[0]` et le premier coefficient `d[1]`.

EXERCICE 2

```

1 def pascal(n):
2     C= [[1]] #(1)
3     for k in range(1,n+1):
4         Ck = [1] #(2)
5         for i in range(1,k):
6             Ck.append(C[k-1][i-1]+C[k-1][i]) #(3)
7         Ck.append(1) #(2)
8         C.append(Ck)
9     return C

```

1. La variable C est la liste des lignes du triangle de Pascal et la toute première ligne du triangle de Pascal contient un unique 1
2. Toutes les lignes (sauf la première) commencent et se terminent par un 1.
3. On construit la ligne en utilisant la relation donnée dans l'énoncé.
4. Toutes les lignes (sauf la première) commencent et se terminent par un 1.

 Commentaire

Le sujet avantage probablement les élèves faisant la spécialité mathématiques en terminale puisqu'ils auront déjà rencontré le triangle de Pascal ainsi que la relation de Pascal :

$$\binom{k}{i} = \binom{k-1}{i-1} + \binom{k-1}{i}$$

qui sert de base à la construction de la $(k\text{-ième})$ ligne du triangle de Pascal à partir de la ligne précédente. On peut d'ailleurs signaler que la génération des coefficients binomiaux est un des algorithmes prévus au programme de terminale en spécialité mathématiques.

3.2.3. Corrigé sujet 03 - Année : 2022

Sujet 03 - 20222 

EXERCICE 1

```

1 def delta(liste):
2     codage=[liste[0]]
3     for i in range(1,len(liste)):
4         codage.append(liste[i]-liste[i-1])
5     return codage

```

 **Commentaires**

- On construit le codage en partant du premier élément de la liste. Les autres éléments sont les différences entre deux éléments consécutifs de la liste de départ.
- L'écriture de cette fonction peut aussi se faire (de façon plus concise) en utilisant les listes par compréhension :

 **Script Python**

```
def delta(liste):
    return [liste[i]-liste[i-1] if i>0 else liste[i] for i in range(len(liste))]
```

EXERCICE 2

 **Bug**

Le code fourni semble contenir une erreur, en effet, pour le compléter on utilise un `if True` à la ligne 23 ! On devrait donc soit se passer de la ligne 23, soit réécrire cette fonction.

```

1 class Noeud:
2     def __init__(self, g, v, d):
3         self.gauche = g
4         self.valeur = v
5         self.droit = d
6
7     def __str__(self):
8         return str(self.valeur)
9
10    def est_une_feuille(self):
11        '''Renvoie True si et seulement si le noeud est une feuille'''
12        return self.gauche is None and self.droit is None
13
14
15    def expression_infixe(e):
16        s = "" #(1)
17        if e.gauche is not None: #(2)
18            s = '(' + s + expression_infixe(e.gauche)
19            s = s + str(e.valeur)
20        if e.droit is not None: #(3)
21            s = s + expression_infixe(e.droit) + ')'
22        if True : #(4)
23            return s
24

```

- La variable `s` va contenir l'expression arithmétique
- Si le noeud contient un fils gauche, on construit l'expression associée et on ajoute la valeur du noeud à la suite.
- On construit la partie droite de l'expression (si elle existe)
- Si on atteint cette ligne, l'expression a été construite en totalité, il reste à la renvoyer. Il ne devrait pas y avoir de `if` !

 **Commentaire**

Sujet assez difficile *en plus d'être buggé* et qui utilise diverses notions du programme (arbre, récursivité) et qui présente de plus un aspect mathématique.

3.2.4. Corrigé sujet 04 - Année : 2022

Sujet 04 - 20222 

EXERCICE 1

```

1  def recherche(liste):
2      consecutifs = []
3      for i in range(len(liste)-1):
4          if liste[i+1]==liste[i]+1:
5              consecutifs.append((liste[i],liste[i+1]))
6      return consecutifs

```

 **Commentaires**

- La condition `liste[i+1]==liste[i]+1` permet de tester que deux éléments consécutifs de la liste sont deux entiers qui se suivent.
- On peut utiliser les définitions de liste par compréhension :

 **Script Python**

```
def recherche(liste):
    return [(liste[i],liste[i+1]) for i in range(len(liste)-1) if liste[i+1]==liste[i]+1]
```

EXERCICE 2

```

1  def propager(M, i, j, val):
2      if M[i][j]==val: #(1)
3          return
4
5      M[i][j]=val
6
7      # L'élément en haut fait partie de la composante
8      if ((i-1) >= 0 and M[i-1][j] == 1): #(2)
9          propager(M, i-1, j, val)
10
11     # L'élément en bas fait partie de la composante
12     if ((i+1) < len(M) and M[i+1][j] == 1): #(2)
13         propager(M, i+1, j, val)
14
15     # L'élément à gauche fait partie de la composante
16     if ((j-1) >= 0 and M[i][j-1] == 1): #(2)
17         propager(M, i, j-1, val)
18
19     # L'élément à droite fait partie de la composante
20     if ((j+1) < len(M) and M[i][j+1] == 1): #(2)
21         propager(M, i, j+1, val)

```

- C'est la condition d'arrêt de la récursivité, on ne relance pas la propagation sur les cases voisines
- On relance la propagation à partir de la case voisine si celle-ci est dans la grille (première partie de la condition) et aussi dans la même composante (seconde partie de la condition)
- On relance la propagation à partir de la case voisine si celle-ci est dans la grille (première partie de la condition) et aussi dans la même composante (seconde partie de la condition)
- On relance la propagation à partir de la case voisine si celle-ci est dans la grille (première partie de la condition) et aussi dans la même composante (seconde partie de la condition)
- On relance la propagation à partir de la case voisine si celle-ci est dans la grille (première partie de la condition) et aussi dans la même composante (seconde partie de la condition)

 **Commentaire**

Le `return` ligne 3 (équivalent à un `return None`) permet de mettre fin à la récursivité. On peut faire autrement et éviter d'utiliser `return` d'autant plus que cette fonction modifie une liste en place mais ne renvoie pas de valeur.

3.2.5. Corrigé sujet 05 - Année : 2022

Sujet 05 - 20222 

EXERCICE 1

 **Bug**

La fonction à écrire s'appelle `RechercheMinMax` dans l'énoncé (avec le `R` majuscule) mais devient `rechercheMinMax` avec un `r` minuscule dans les appels.

```
1 def RechercheMinMax(tab):
2     if tab==[]:
3         return {'min':None,'max':None}
4     mini,maxi = tab[0],tab[0]
5     for elt in tab:
6         if elt<mini: mini=elt
7         if elt>maxi: maxi=elt
8     return {'min' : mini, 'max' : maxi}
```

 **Commentaires**

Encore une recherche classique de minimum et de maximum, le résultat est renvoyé sous la forme d'un dictionnaire.

EXERCICE 2

```

1  class Carte:
2      """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à 13)"""
3      def __init__(self, c, v):
4          assert 1<=c<=4, "La couleur est entre 1 et 4"
5          assert 1<=v<=13, "La valeur est entre 1 et 13" #(1)
6          self.Couleur = c
7          self.Valeur = v
8
9      """Renvoie le nom de la Carte As, 2, ... 10,
10     Valet, Dame, Roi"""
11     def getNom(self):
12         if ( self.Valeur > 1 and self.Valeur < 11):
13             return str( self.Valeur)
14         elif self.Valeur == 11:
15             return "Valet"
16         elif self.Valeur == 12:
17             return "Dame"
18         elif self.Valeur == 13:
19             return "Roi"
20         else:
21             return "As"
22
23     """Renvoie la couleur de la Carte (parmi pique, coeur, carreau, trefle)"""
24     def getCouleur(self):
25         return ['pique', 'coeur', 'carreau', 'trefle'][self.Couleur - 1]
26
27 class PaquetDeCarte:
28     def __init__(self):
29         self.contenu = []
30
31     """Remplit le paquet de cartes"""
32     def remplir(self):
33         self.contenu = [Carte(couleur,valeur) for couleur in range(1, 5) for valeur in range(1, 14)] #(2)
34
35     """Renvoie la Carte qui se trouve à la position donnée"""
36     def getCarteAt(self, pos):
37         assert 0<=pos<52, "Le numéro de la carte doit être entre 0 et 51"
38         if 0 <= pos < 52 :
39             return self.contenu[pos]

```

1. Ce sont les instructions `assert` permettant de vérifier que la couleur est entre 1 et 4 inclus et la valeur entre 1 et 13 inclus.
2. On utilise une définition de liste par compréhension pour parcourir les couleurs et valeurs possibles.
3. Un jeu de cartes contient 52 cartes, la position d'une carte est entre 0 et 51 inclus.

Attention

- Le sujet demande d'ajouter des instructions `assert` dans la méthode `getCarteAt` mais cette méthode teste déjà que le numéro de la carte est entre les limites imposées. Le `assert` pourrait vérifier ici que `pos` est bien une variable de type `int`.
- Les *docstring* devraient être placées juste après la ligne `def` de définition des méthodes et pas avant.
- Il serait pertinent d'utiliser un dictionnaire afin d'associer valeur et nom de d'une carte : `{1 : 'As', 2:'2', ..., 12: 'Dame',13 : 'Roi'}`

3.2.6. Corrigé sujet 06 - Année : 2022

Sujet 06 - 20222 

EXERCICE 1

```

1  def maxi(tab):
2      if tab==[]: return None,None
3      indice_maxi,maxi = 0, tab[0]
4      for indice in range(1,len(tab)):
5          if tab[indice]>maxi:
6              indice_maxi,maxi = indice,tab[indice]
7      return maxi,indice_maxi

```

 **Commentaires**

- Rien n'est indiqué pour la liste vide, on a choisi de renvoyer le couple `(None,None)` dans ce cas
- Un parcours par indice est nécessaire puisqu'on a besoin de la position du maximum.

EXERCICE 2

```

1  def recherche(gene, seq_adn):
2      n = len(seq_adn)
3      g = len(gene)
4      i = 0 #(1)
5      trouve = False
6      while i < n-g and trouve == False : #(2)
7          j = 0
8          while j < g and gene[j] == seq_adn[i+j]:
9              j += 1 #(3)
10         if j == g:
11             trouve = True
12         i+=1 #(4)
13     return trouve
14
15 print(recherche("AATC", "GTACAAATCTTGC"))
16 print(recherche("AGTC", "GTACAAATCTTGC"))

```

1. C'est l'indice `i` de parcours de la chaîne, initialisé à 0
2. La recherche continue tant que `i` est inférieure à la longueur de la chaîne (`n`) - la longueur du motif (`g`) et que la motif n'a pas été trouvé
3. On a une correspondance, `j` est l'indice de parcours du motif, on continue à chercher en avançant dans le motif `j=j+1`.
4. On passe à l'indice suivant de la chaîne.

 **Attention**

- La recherche textuelle n'est pas au programme de l'épreuve de Bac.
- Le double parcours avec un indice parcourant la chaîne et un autre le motif présente sans doute une difficulté.
- Au lieu de `trouve == False` (ligne 6), on peut écrire `not trouve` qui est sans doute plus parlant.

3.2.7. Corrigé sujet 07 - Année : 2022

Sujet 07 - 20222 

EXERCICE 1

```

1 def conv_bin(n):
2     liste_bit=[n%2]
3     n=n//2
4     while n!=0:
5         liste_bit.append(n%2)
6         n=n//2
7     liste_bit.reverse()
8     return liste_bit,len(liste_bit)

```

 **Commentaires**

1. L'exemple de l'énoncé est mal choisi, en effet l'écriture binaire de 9 : \((9_{10})=1001_2\)\) étant un palindrome (identique à l'envers), cet exemple ne permet pas de détecter un éventuel oubli de l'utilisation de `reverse`.
2. Dans la correction, l'initialisation `liste_bit=[n%2]` permet de traiter le cas de 0.

EXERCICE 2

```

1 def tri_bulles(T):
2     n = len(T)
3     for i in range(len(T)-1,0,-1): #(1)
4         for j in range(i): #(2)
5             if T[j] > T[j+1]: #(3)
6                 temp = T[j] #3)
7                 T[j] = T[j+1]
8                 T[j+1] = temp
9     return T

```

1. On parcourt la liste à l'envers à l'aide de l'indice `i` (le dernier élément de `T` a pour indice `len(T)-1`)
2. On teste si deux éléments consécutifs ne sont pas dans le bon ordre
3. Si oui, alors on les échange

 **Attention**

- Cet exercice demande de coder `le tri à bulles` qui n'est pourtant pas au programme. Son principe est de faire remonter les plus petits éléments de la liste vers le début en les échangeant avec leur voisins.
- Les lignes 6,7 et 8 permettant d'échanger `T[j]` et `T[j+1]` en utilisant la variable temporaire `temp`, on pourrait écrire plus simplement : `T[j],T[j+1]=T[j+1],T[j]`
- Le `return T` peut laisser penser qu'on veut récupérer la liste triée, alors que `T` est modifiée et triée par la fonction puisque mutable.

3.2.8. Corrigé sujet 08 - Année : 2022

Sujet 08 - 20222 

EXERCICE 1

```

1 def recherche_elt(tab):
2     for i in range(len(tab)):
3         if tab[i]==elt:
4             return i
5     return -1

```

 **Commentaires**

Exercice classique de recherche dans une liste, un parcours par les indices s'impose puisqu'on renvoie la position de l'élément dans la liste.

EXERCICE 2

```

1 def insere(a, tab):
2     l = list(tab) #l contient les mêmes éléments que tab
3     l.append(a)
4     i = len(tab)-1 #(1)
5     while a < l[i] and i >= 0: #(2)
6         l[i+1] = l[i] #(3)
7         l[i] = a
8         i = i - 1 #(4)
9     return l

```

1. C'est l'indice auquel se trouve l'élément précédent celui qui a été inséré. Au début on insère à la fin (en position `len(tab)`) donc celui qui précède a l'indice `len(tab)-1`.
2. On échange `a` avec le précédent tant qu'il est inférieur et que le début de liste n'est pas atteint.
3. Cette ligne et la suivante permettent d'effectuer l'échange.
4. Décrémentation de la position à tester.

 **Attention**

- Pour compléter le code, il faut comprendre l'algorithme mis en oeuvre pour insérer au bon emplacement :
 - a. Ajouter à la fin
 - b. Remonter l'élément en l'échangeant avec son voisin de gauche tant qu'il est inférieur à ce dernier
- La ligne 2 a pour but de faire une copie du tableau `tab` afin de ne pas le modifier. On aurait pu utiliser `copy`.

3.2.9. Corrigé sujet 09 - Année : 2022

Sujet 09 - 20222 

EXERCICE 1

```

1 def calcul(k):
2     resultat=[k]
3     while k!=1:
4         if k%2==0:
5             k=k//2
6         else:
7             k=3*k+1
8         resultat.append(k)
9     return resultat
10
11 print(calcul(7))

```

 Commentaires

- Le sujet avantage les élèves ayant suivi l'enseignement de spécialités mathématiques. Les suites définies par récurrence comme celle de l'énoncé y sont vues dès la classe de première.
- Le test permettant de savoir si un entier `2` est pair s'écrit `n%2 == 0`, c'est à dire qu'on teste qu'en divisant par 2 il reste 0.
- Attention à la ligne 5 à bien utiliser la division euclidienne `//` et pas la division décimale `\` (sinon le résultat obtenu serait alors un flottant).

EXERCICE 2

```

1 dico = {"A":1, "B":2, "C":3, "D":4, "E":5, "F":6, "G":7, \
2     "H":8, "I":9, "J":10, "K":11, "L":12, "M":13, \
3     "N":14, "O":15, "P":16, "Q":17, "R":18, "S":19, \
4     "T":20, "U":21, "V":22, "W":23, "X":24, "Y":25, "Z":26}
5
6 def est_parfait(mot) :
7     #mot est une chaîne de caractères (en lettres majuscules)
8     code_c = ""
9     code_a = 0 #(1)
10    for c in mot :
11        code_c = code_c + str(dico[c]) #(2)
12        code_a = code_a + dico[c] #(3)
13    code_c = int(code_c) #(4)
14    if code_c%code_a==0 : #(4)
15        mot_est_parfait = True
16    else :
17        mot_est_parfait = False
18    return [code_a, code_c, mot_est_parfait]

```

- Bien comprendre que le `code_a` est un entier (addition des entiers), tandis que `code_c` est une chaîne de caractères (concaténation).
- Les valeurs du dictionnaires sont des entiers pour concaténer avec la chaîne `code_c` il faut convertir en `str`
- On addition des entiers, pas de conversion nécessaire
- Teste si le code additionné divise le code concaténé.

 Attention

- Les lignes 14,15,16, et 17 se résument à `mot_est_parfait = (code_c%code_a==0)`
- Même si cela est autorisé en Python, le changement de type d'une variable tel que celui effectué ligne 13 (où `int_c` qui était une chaîne de caractères devient un entier) est largement considéré comme une mauvaise pratique de programmation.
- On aurait pu se passer du dictionnaire en utilisant les fonctions `ord` et `chr`.

3.2.10. Corrigé sujet 10 - Année : 2022

Sujet 10 - 20222 

EXERCICE 1

```

1  def occurrence_lettres(phrase):
2      occ = {}
3      for caractere in phrase:
4          if caractere in occ:
5              occ[caractere] += 1
6          else:
7              occ[caractere]=1
8      return occ

```

 Commentaire

- Bon exercice pour revoir l'utilisation des dictionnaires.
- Le terme *occurrence* est utilisé avec le sens habituellement attribué à *nombre d'occurrences*
- Dans le nom de la fonction dans l'énoncé, *occurence* s'écrit avec un seul *r*, ce serait deux sans faute d'orthographe.

EXERCICE 2

```

1  def fusion(L1,L2):
2      n1 = len(L1)
3      n2 = len(L2)
4      L12 = [0]^(n1+n2)
5      i1 = 0
6      i2 = 0
7      i = 0
8      while i1 < n1 and i2<n2 : #(1)
9          if L1[i1] < L2[i2]:
10              L12[i] = L1[i1] #(2)
11              i1 = i1 + 1
12          else:
13              L12[i] = L2[i2]
14              i2 = i2 + 1
15          i += 1
16      while i1 < n1:
17          L12[i] = L1[i1] #(3)
18          i1 = i1 + 1
19          i = i + 1
20      while i2 < n2:
21          L12[i] = L2[i2]
22          i2 = i2 + 1
23          i = i + 1
24      return L12

```

1. *i1* est l'indice de parcours de *L1* (de longueur *n1*) *i2* est l'indice de parcours de *L2* (de longueur *n2*)
2. On se trouve dans le cas où le plus petit élément se trouve dans *L1*, c'est donc lui qui est ajouté à liste fusionnée *L12*.
3. On a atteint la fin de l'une des listes, il reste donc à ajouter les éléments restants de l'autre liste.

3.2.11. Corrigé sujet 11 - Année : 2022

Sujet 11 - 20222 

EXERCICE 1

```

1 def recherche(tab,elt):
2     ind_debut = 0
3     ind_fin = len(tab)-1
4     while ind_fin > ind_debut:
5         ind_milieu = (ind_fin+ind_debut)//2
6         if tab[ind_milieu]==elt:
7             return ind_milieu
8         elif tab[ind_milieu]>elt:
9             ind_fin=ind_milieu-1
10        else:
11            ind_debut=ind_milieu+1
12    return -1

```

 **Commentaire**

- Bien qu'au programme (méthode diviser pour régner), cet exercice est bien plus difficile que ceux proposés habituellement en exercice 1 (recherche simple, recherche de maximum, calcul de moyennes, ...).
- En cas de difficultés, revenir au chapitre de première sur la [recherche par dichotomie](#)

EXERCICE 2

```

1 ALPHABET='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2
3 def position_alphabet(lettre):
4     return ALPHABET.find(lettre)
5
6 def cesar(message, decalage):
7     resultat = ''
8     for lettre in message : #(1)
9         if lettre in ALPHABET :
10             indice = (position_alphabet(lettre) + decalage)%26 #(2)
11             resultat = resultat + ALPHABET[indice]
12         else:
13             resultat = resultat + lettre #(3)
14     return resultat

```

1. La variable de parcours du message s'appelle `lettre` comme indiqué à la ligne suivante !
2. On ajoute le decalage à la position de la lettre le `%26` sert ensuite à s'assurer qu'on retombe entre 0 et 25.
3. Si la lettre n'est pas dans l'alphabet, on la laisse telle quelle

 **Attention**

1. La variable `lettre` de parcours du message peut contenir justement autre chose qu'une lettre (ponctuation, espace, ...)
2. La méthode `find` de recherche de l'indice de la lettre dans l'alphabet pourrait s'écrire sans utiliser la méthode `find` (par exemple avec `ord`)

3.2.12. Corrigé sujet 12 - Année : 2022

Sujet 12 - 20222 

EXERCICE 1

```

1  def moyenne(tab):
2      assert tab!=[], "erreur"
3      somme = 0
4      for valeur in tab:
5          somme = somme + valeur
6      return somme/len(tab)

```

 **Commentaire**

- Le traitement demandé dans le cas d'un tableau vide n'est pas explicite, on ne sait pas s'il faut juste faire un `print(erreur)` ou plutôt utiliser un `assert` (comme dans cette correction)

EXERCICE 2

```

1  def tri(tab):
2      #i est le premier indice de la zone non triee, j le dernier indice.
3      #Au debut, la zone non triee est le tableau entier.
4      i= 0
5      j= len(tab)-1 #(1)
6      while i != j :
7          if tab[i]== 0:
8              i = i + 1 #(2)
9          else :
10             valeur = tab[j]
11             tab[j] = tab[i] #(3)
12             tab[i] = valeur
13             j= j-1 #(4)
14     return tab

```

- Le dernier élément d'un tableau `tab` a pour indice `len(tab)-1`
- Si l'élément est un 0, on incrémente le début de la zone non triée d'indice `i`. La zone non triée diminue "par la gauche".
- Sinon, on échange cette valeur avec la fin de zone non triée (située en indice `j`).
- La zone non triée diminue "par la droite", on décrémente donc sa fin d'indice `j`

 **Attention**

Les lignes 10 à 12 qui permettent d'échanger `tab[j]` et `tab[i]` en utilisant la variable temporaire `valeur` peuvent être simplifiées en remarquant que `tab[i]` vaut forcément 1 dans cette branche du `if`.

3.2.13. Corrigé sujet 13 - Année : 2022

Sujet 13 - 20222 

EXERCICE 1

```

1 def rendu(somme_a_rendre):
2     n1 = somme_a_rendre//5
3     somme_a_rendre = somme_a_rendre%5
4     n2 = somme_a_rendre//2
5     n3 = somme_a_rendre%2
6     return [n1,n2,n3]

```

 Commentaire

- Bien qu'au programme de première ([algorithme glouton](#)), cet exercice est bien plus difficile que ceux proposés habituellement en exercice 1 (recherche simple, recherche de maximum, calcul de moyennes, ...).
- La correction proposée ici utilise les trois variables `n1`, `n2` et `n3` proposées dans l'énoncé. Dans un cadre plus général, une boucle travaillant sur une liste de pièces serait préférable. Voir par exemple la [correction proposée ici](#).

EXERCICE 2

```

1 class Maillon :
2     def __init__(self,v) :
3         self.valeur = v
4         self.suivant = None
5
6 class File :
7     def __init__(self) :
8         self.dernier_file = None
9
10    def enfile(self,element) :
11        nouveau_maillon = Maillon(element) #(1)
12        nouveau_maillon.suivant = self.dernier_file
13        self.dernier_file = nouveau_maillon #(2)
14
15    def est_vide(self) :
16        return self.dernier_file == None
17
18    def affiche(self) :
19        maillon = self.dernier_file
20        while maillon != None : #(3)
21            print(maillon.valeur)
22            maillon = maillon.suivant #(4)
23
24    def defile(self) :
25        if not self.est_vide() :
26            if self.dernier_file.suivant == None :
27                resultat = self.dernier_file.valeur
28                self.dernier_file = None
29                return resultat
30            maillon = self.dernier_file #(5)
31            while maillon.suivant != None :
32                maillon = maillon.suivant
33            resultat = maillon.suivant.valeur
34            maillon.suivant = None
35            return resultat
36        return None

```

- Le constructeur de la classe `Maillon` prend en argument la valeur `v` (appelée ici `element`)
- Comme indiqué dans l'énoncé, l'attribut `dernier_file` doit contenir le dernier maillon enfilé.
- La fin de file est atteinte lorsque le maillon vaut `None`
- Passage au maillon suivant
- On commence au dernier maillon en s'assurant avant que le maillon suivant n'est pas `None`

Attention

1. L'implémentation d'une file proposée ici est problématique, pour défiler, il faut partir de la fin (seul accès à la file), la remonter en entier afin de défiler le premier.
2. Faire un schéma de l'implémentation proposée peut aider à la compréhension du code donné dans l'énoncé.

3.2.14. Corrigé sujet 14 - Année : 2022

Sujet 14 - 20222 

EXERCICE 1

```

1 def correspond(mot,mot_a_trous):
2     for indice in range(len(mot)):
3         if mot_a_trous[indice]!="*" and mot[indice]!=mot_a_trous[indice]:
4             return False
5     return True

```

 Commentaire

Le sujet ne précise pas le comportement attendu si `mot` et `mot_a_trous` n'ont pas la même longueur.

EXERCICE 2

```

1 def est_cyclique(plan):
2     """
3         Prend en paramètre un dictionnaire 'plan' correspondant
4         à un plan d'envoi de messages entre 'N' personnes A, B, C,
5         D, E, F ... (avec N <= 26).
6         Renvoie True si le plan d'envoi de messages est cyclique
7         et False sinon.
8     """
9     personne = 'A'
10    N = len(plan)
11    for i in range(N-1): #(1)
12        if plan[personne] == 'A': #(2)
13            return False
14        else:
15            personne = plan[personne] #(3)
16    return True

```

1. Attention à ne pas parcourir en totalité le dictionnaire (sinon on trouvera forcément un cycle). On s'arrête donc à l'avant dernier.
2. C'est le fait de retomber sur la personne de départ (donc `'A'`) qui indique qu'on a trouvé un cycle.
3. Passage à la personne suivante (revoir si besoin les dictionnaires)

3.2.15. Corrigé sujet 15 - Année : 2022

Sujet 15 - 20222 

EXERCICE 1

```

1  def nb_repetitions_elt(tab):
2      nb_rep = 0
3      for x in tab:
4          if x==elt:
5              nb_rep += 1
6      return nb_rep

```

 **Commentaire**

C'est un exercice classique de parcours d'un itérable en comptant les occurrences d'apparition d'une valeur. Un parcours par élément suffit, les indices des occurrences n'étant pas utilisées.

EXERCICE 2

```

1  def binaire(a):
2      bin_a = str(a%2) #(1)
3      a = a // 2
4      while a != 0 :
5          bin_a = str(a%2) + bin_a #(2)
6          a = a//2 #(3)
7      return bin_a

```

1. On rappelle que `a%2` est le reste dans la division euclidienne de `a` par `2` et donc le premier chiffre (à droite) de son écriture binaire.
2. Ajout du nouveau chiffre (en le convertissant en chaîne de caractères)
3. On rappelle que `a//2` est le quotient dans la division euclidienne de `a` par `2`.

 **Attention**

L'illustration de l'algorithme proposé dans l'énoncé laisse penser que la condition d'arrêt de l'algorithme est `a==1`.

3.2.16. Corrigé sujet 16 - Année : 2022

Sujet 16 - 20222 

EXERCICE 1

```

1 def maxi(tab):
2     indice_maximum, maximum = 0, tab[0]
3     for indice in range(len(tab)):
4         if tab[indice]>maximum:
5             indice_maximum,maximum = indice, tab[indice]
6     return maximum, indice_maximum

```

 **Commentaire**

L'énoncé n'indique pas le comportement attendu dans le cas de la liste vide.

EXERCICE 2

```

1 def positif(T):
2     T2 = list(T) #1
3     T3 = []
4     while T2 != []:
5         x = T2.pop()
6         if x >= 0: #(2)
7             T3.append(x)
8     T2 = []
9     while T3 != []:
10        x = T3.pop()
11        T2.append(x)
12    print('T = ',T)
13    return T2

```

1. Copie *indépendante* de la liste de départ avec la technique de l'énoncé : `list`.
2. On dépile en totalité `T2`, les éléments positifs sont empilés dans `T3`.
3. On dépile `T3` et on empile son contenu dans `T2`, afin de les remettre dans l'ordre de départ.

 **Attention**

1. Bien comprendre que le sujet se limite à l'interface habituelle d'une pile (`empile` avec `append`, `dépile` avec `pop` et `est_vide` avec `==[]`).
2. La ligne 8 est inutile, `T2` est vide puisqu'on a quitté la boucle.

3.2.17. Corrigé sujet **17** - Année : 2022**Sujet 17 - 20222** 

EXERCICE 1

```

1 def nombre_de_mots(phrase):
2     nb_espace = 0
3     for caractere in phrase:
4         if caractere==" ":
5             nb_espace+=1
6     if phrase[-1]==".":
7         return nb_espace+1
8     else:
9         return nb_espace

```

 **Commentaire**

1. Il faut avoir remarqué que le nombre de mots est égal:
 - au nombre d'espace si la phrase se termine par "!" ou "?",
 - au nombre d'espace plus un si la phrase se termine par un ".".
2. Les exemples de l'énoncé ne testent que la fin avec un point d'exclamation.
3. Dans le corrigé, on a compté le nombre d'espace en effectuant un parcours de la phrase, on pouvait aussi utiliser la méthode `count` des chaînes de caractères.

EXERCICE 2

```

1  class Noeud:
2      """
3          Classe implémentant un noeud d'arbre binaire
4          disposant de 3 attributs :
5          - valeur : la valeur de l'étiquette,
6          - gauche : le sous-arbre gauche.
7          - droit : le sous-arbre droit.
8      """
9      def __init__(self, v, g, d):
10         self.valeur = v
11         self.gauche = g
12         self.droite = d
13
14     class ABR:
15         """
16             Classe implémentant une structure
17             d'arbre binaire de recherche.
18         """
19
20         def __init__(self):
21             """Crée un arbre binaire de recherche vide"""
22             self.racine = None
23
24         def est_vide(self):
25             """Renvoie True si l'ABR est vide et False sinon."""
26             return self.racine is None
27
28         def parcours(self, tab = []):
29             """
30                 Renvoie la liste tab complétée avec tous les
31                 éléments de
32                 l'ABR triés par ordre croissant.
33             """
34             if self.est_vide():
35                 return tab
36             else:
37                 self.racine.gauche.parcours(tab)
38                 tab.append(self.racine.valeur) #1)
39                 self.racine.droite.parcours(tab)
40             return tab
41
42         def insere(self, element):
43             """Insère un élément dans l'arbre binaire de recherche."""
44             if self.est_vide():
45                 self.racine = Noeud(element, ABR(), ABR())
46             else:
47                 if element < self.racine.valeur:
48                     self.racine.gauche.insere(element)
49                 else :
50                     self.racine.droite.insere(element)
51
52         def recherche(self, element):
53             """
54                 Renvoie True si element est présent dans l'arbre
55                 binaire et False sinon.
56             """
57             if self.est_vide():
58                 return False #2)
59             else:
60                 if element < self.racine.valeur:
61                     return self.racine.gauche.recherche(element) #3)
62                 elif element > self.racine.valeur:
63                     return self.racine.droite.recherche(element)
64                 else:
65                     return True

```

1. On parcours à gauche, on ajoute la valeur de la racine puis on parcourt à droite.
2. Si l'arbre est vide alors l'élément ne s'y trouve pas !
3. Si l'arbre n'est pas vide, on compare avec la valeur de la racine. Si ce n'est pas la valeur cherchée on recherche à droite ou à gauche suivant les cas.

Attention

- Le code fourni utilise un objet mutable (une liste) comme paramètre par défaut de la méthode de parcours :

 **Script Python**

```
def parcours(self, tab = []):
```

C'est une très mauvaise pratique car source d'erreurs, en effet la variable `tab` étant mutable elle est modifiée par la fonction lors d'un premier appel et ne sera donc plus vide lors des appels suivants. Pour une solution à ce problème, on pourra par exemple consulter [ce site](#)

3.2.18. Corrigé sujet 18 - Année : 2022

Sujet 18 - 20222 

EXERCICE 1

```

1 def mini(releve,date):
2     indice_mini, temp_mini = 0, releve[0]
3     for i in range(len(releve)):
4         if releve[i]<temp_mini:
5             indice_mini,temp_mini = i,releve[i]
6     return temp_mini,date[indice_mini]

```

 **Commentaire**

Bien comprendre que les années et les températures moyennes correspondantes sont aux mêmes indices dans les deux listes. On recherche donc l'indice de la température moyenne et on l'utilise pour retrouver l'année correspondante.

EXERCICE 2

```

1 def inverse_chaine(chaine):
2     result = ""
3     for caractere in chaine:
4         result = caractere + result #(1)
5     return result
6
7 def est_palindrome(chaine):
8     inverse = inverse_chaine(chaine)
9     return inverse==chaine #(2)
10
11 def est_nbre_palindrome(nbre):
12     chaine = str(nbre) #(3)
13     return est_palindrome(chaine)

```

1. Le dernier caractère à être ajouté doit être au début, il faut donc écrire `result = caractere + result` et pas `result = result + caractere`
2. Cela est équivalent à écrire :

 **Script Python**

```

if inverse==chaine:
    return True
else:
    return False

```

3. Conversion en chaînes de caractères afin de pouvoir utiliser `est_palindrome`.

3.2.19. Corrigé sujet 19 - Année : 2022

Sujet 19 - 20222 

EXERCICE 1

```

1  def multiplication(a,b):
2      produit=0
3      for i in range(abs(a)):
4          produit += abs(b)
5      if (a>0 and b<0) or (a<0 and b>0):
6          return -produit
7      else:
8          return produit

```

 Commentaires

1. On peut rappeler la règle des signes pour un produit :
 - un produit est négatif si les deux facteurs ne sont pas de même signe (ligne 5)
 - et positif sinon.
2. Si `a` et `b` sont deux entiers positifs :
$$a \times b = \underbrace{b + b + \dots + b}_{\text{a termes}}$$
3. L'énoncé pourrait aiguiller vers l'utilisation de `abs` (valeur absolue) pour traiter les problèmes de signe.

EXERCICE 2

```

1  def chercher(T,n,i,j):
2      if i < 0 or j>len(T)-1: #(1)
3          print("Erreur")
4          return None
5      if i > j :
6          return None
7      m = (i+j) // 2 #(2)
8      if T[m] < n :
9          return chercher(T, n, m+1 , j) #(3)
10     elif T[m]>n :
11         return chercher(T, n, i , m-1)
12     else:
13         return m

```

1. L'indice du dernier élément d'un tableau `T` est `len(T)-1`.
2. Calcul de l'indice au milieu
3. On peut chercher *après* l'indice `m` à cause du strictement inférieur dans le test de comparaison `T[m]<n`.

3.2.20. Corrigé sujet 20 - Année : 2022

Sujet 20 - 20222 

EXERCICE 1

```

1  def xor(a,b):
2      resultat = []
3      for i in range(len(a)):
4          if a[i]==b[i]:
5              resultat.append(0)
6          else:
7              resultat.append(1)
8      return resultat

```

 Commentaire

On peut aussi utiliser une définition de listes par compréhension.

EXERCICE 2

```

1  class Carré:
2      def __init__(self, tableau = [[]]):
3          self.ordre = len(tableau)
4          self.valeurs = tableau
5
6      def affiche(self):
7          '''Affiche un carré'''
8          for i in range(self.ordre):
9              print(self.valeurs[i])
10
11     def somme_ligne(self, i):
12         '''Calcule la somme des valeurs de la ligne i'''
13         return sum(self.valeurs[i])
14
15     def somme_col(self, j):
16         '''calcule la somme des valeurs de la colonne j'''
17         return sum([self.valeurs[i][j] for i in range(self.ordre)])
18
19     def est_magique(carre):
20         n = carre.ordre
21         s = carre.somme_ligne(0)
22
23         #test de la somme de chaque ligne
24         for i in range(1,n): #(1)
25             if carre.somme_ligne(i) != s:
26                 return False
27
28         #test de la somme de chaque colonne
29         for j in range(n):
30             if carre.somme_col(j) != s: #(2)
31                 return False
32
33         #test de la somme de chaque diagonale
34         if sum([carre.valeurs[k][k] for k in range(n)]) != s: #(3)
35             return False
36         if sum([carre.valeurs[k][n-1-k] for k in range(n)]) != s:
37             return False
38         return True #(4)

```

1. Par la peine de tester la ligne d'indice 0, elle a servi à calculer la somme de référence `s` (ligne 21), on commence donc à 1.
2. On utilise la méthode `somme_col` de la classe `Carré`
3. La diagonale principale se caractérise par des indices de lignes et de colonne identiques.
4. Si on atteint cette ligne, tous les tests ont été passé avec succès, le carré est magique !

Attention

- Le code fourni utilise un objet mutable (une liste) comme paramètre par défaut d'une fonction :

 **Script Python**

```
def __init__(self, tableau = [[]]):
```

C'est une très mauvaise pratique car source d'erreurs, en effet la variable `tableau` étant mutable elle est modifiée par la fonction lors d'un premier appel et ne sera donc plus vide lors des appels suivants. Pour une solution à ce problème, on pourra par exemple consulter [ce site](#)

3.2.21. Corrigé sujet 21 - Année : 2022

Sujet 21 - 20222 

EXERCICE 1

```

1  def multiplication(a,b):
2      produit=0
3      for i in range(abs(a)):
4          produit += abs(b)
5      if (a>0 and b<0) or (a<0 and b>0):
6          return -produit
7      else:
8          return produit

```

 Commentaires

1. On peut rappeler la règle des signes pour un produit :
 - un produit est négatif si les deux facteurs ne sont pas de même signe (ligne 5)
 - et positif sinon.
2. Si `a` et `b` sont deux entiers positifs :
$$a \times b = \underbrace{b + b + \dots + b}_{\text{a termes}}$$
3. L'énoncé pourrait aiguiller vers l'utilisation de `abs` (valeur absolue) pour traiter les problèmes de signe.

EXERCICE 2

```

1  def dichotomie(tab, x):
2      """
3          tab : tableau d'entiers trié dans l'ordre croissant
4          x : nombre entier
5          La fonction renvoie True si tab contient x et False sinon
6      """
7
8      debut = 0
9      fin = len(tab) - 1
10     while debut <= fin:
11         m = (debut+fin)//2 #(1)
12         if x == tab[m]:
13             return True
14         if x > tab[m]:
15             debut = m + 1
16         else:
17             fin = m-1 #(2)
18     return False

```

1. Calcul de l'indice se trouvant "au milieu" entre `debut` et `fin`.
2. Ici `x < tab[m]` (la cas d'égalité est traité avant), donc l'indice de `fin` de recherche est avant `m`.

3.2.22. Corrigé sujet 22 - Année : 2022

Sujet 22 - 20222 

EXERCICE 1

```

1 def renverse(chaine):
2     chaine_inverse = ""
3     for caractere in chaine:
4         chaine_inverse = caractere + chaine_inverse
5     return chaine_inverse

```

 **Réponse**

On peut proposer une version utilisant la méthode `join` des listes, en effet ajouter des éléments à une liste est plus efficace que d'ajouter des caractères à une chaîne de caractères.

EXERCICE 2

```

1 def crible(N):
2     """renvoie un tableau contenant tous les nombres premiers plus petit que N"""
3     premiers = []
4     tab = [True] * N
5     tab[0], tab[1] = False, False
6     for i in range(2, N):
7         if tab[i] == True: #(1)
8             premiers.append(i)
9             for multiple in range(2*i, N, i): #(2)
10                tab[multiple] = False #3
11
12 return premiers

```

1. C'est le cas où le nombre est `i` premier
2. Ce sont les multiples de `i`, on parcourt donc avec un pas de `i`
3. Les multiples ne sont pas des nombres premiers.

 **Attention**

Le crible d'Eratosthène proposé ici fait partie des algorithmes proposés dans l'option mathématiques expertes.

3.2.23. Corrigé sujet 23 - Année : 2022

Sujet 23 - 20222 

EXERCICE 1

```

1 def max_dico(dico):
2     max_like = 0
3     for pseudo in dico:
4         if dico[pseudo]>max_like:
5             max_pseudo = pseudo
6             max_like=dico[pseudo]
7     return max_pseudo,max_like

```

 Commentaires

1. Dans cette correction on a choisi d'initialiser le maximum possible à 0.
2. L'énoncé parle de "la clé du dictionnaire associée à la valeur maximale" mais plusieurs clés distinctes peuvent être associées à cette valeur maximale {'Bob':102,'Alice':102} .

EXERCICE 2

```

1 class Pile:
2     """Classe définissant une structure de pile."""
3     def __init__(self):
4         self.contenu = []
5
6     def est_vide(self):
7         """Renvoie le booléen True si la pile est vide, False sinon."""
8         return self.contenu == []
9
10    def empiler(self, v):
11        """Place l'élément v au sommet de la pile"""
12        self.contenu.append(v)
13
14    def depiler(self):
15        """
16            Retire et renvoie l'élément placé au sommet de la pile,
17            si la pile n'est pas vide.
18            """
19        if not self.est_vide():
20            return self.contenu.pop()
21
22
23    def eval_expression(tab):
24        p = Pile()
25        for element in tab: #(1)
26            if element != '+' and element != '*': #(2)
27                p.empiler(element)
28            else:
29                if element == "+": #(3)
30                    resultat = p.depiler() + p.depiler()
31                else:
32                    resultat = p.depiler() * p.depiler()
33                p.empiler(resultat) #(4)
34
35        return resultat

```

1. Le nom de la variable de parcours est indiqué juste en dessous : element
2. On suit l'algorithme de l'énoncé : si l'élément n'est pas un opérateur, alors on l'empile.
3. Si c'est un opérateur, alors on effectue l'opération. C'est soit l'addition, soit la multiplication car on s'est limité à ces deux opérations.
4. Le résultat est empilé comme indiqué dans l'algorithme donné dans l'énoncé.

3.2.24. Corrigé sujet 24 - Année : 2022

Sujet 24 - 202222 

EXERCICE 1

```

1 def maxliste(liste):
2     maxi = liste[0]
3     for elt in liste:
4         if elt > maxi:
5             maxi=elt
6     return maxi

```

 Commentaires

L'énoncé précise que la liste est non vide, on peut donc se permettre d'initialiser le maximum courant avec le premier élément de la liste.

EXERCICE 2

```

1 class Pile:
2     """ Classe définissant une pile """
3     def __init__(self, valeurs=[]):
4         self.valeurs = valeurs
5
6     def est_vide(self):
7         """Renvoie True si la pile est vide, False sinon"""
8         return self.valeurs == []
9
10    def empiler(self, c):
11        """Place l'élément c au sommet de la pile"""
12        self.valeurs.append(c)
13
14    def depiler(self):
15        """Supprime l'élément placé au sommet de la pile, à condition qu'elle soit non vide"""
16        if self.est_vide() == False:
17            self.valeurs.pop()
18
19
20    def parentheseage (ch):
21        """Renvoie True si la chaîne ch est bien parenthésée et False sinon"""
22        p = Pile()
23        for c in ch:
24            if c == "(": #(1)
25                p.empiler(c)
26            elif c == ")": #(2)
27                if p.est_vide():
28                    return False
29                else:
30                    p.depiler()
31        return p.est_vide() #(3)

```

1. On suit l'algorithme donné dans l'énonce : si on rencontre une parenthèse ouvrante alors on l'empile
2. Si c'est une parenthèse fermante, on dépile dans le cas où la pile est vide, l'expression est mal parenthésée.
3. Si à la fin du parcours la pile n'est pas vide, l'expression est mal parenthésée.

Attention

- Le code fourni utilise un objet mutable (une liste) comme paramètre par défaut d'une fonction :

 **Script Python**

```
def __init__(self, valeurs=[]):
```

C'est une très mauvaise pratique car source d'erreurs, en effet la variable `valeurs` étant mutable elle est modifiée par la fonction lors d'un premier appel et ne sera donc plus vide lors des appels suivants. Pour une solution à ce problème, on pourra par exemple consulter [ce site](#)

- Dans l'énoncé au format `pdf`, le `else` de la ligne 29 n'est pas correctement indenté, le problème n'apparaît pas dans le fichier `.py` fourni avec le sujet. Probablement en lien avec ce souci, le `elif` de la ligne 26 pourrait être un `else`.

3.2.25. Corrigé sujet 25 - Année : 2022

Sujet 25 - 20222 

EXERCICE 1

```

1 def selection_enclos(table_animaux,num_enclos):
2     resultat = []
3     for animal in table_animaux:
4         if animal['enclos']==num_enclos:
5             resultat.append(animal)
6     return resultat

```

 **Commentaires**

Revoir le chapitre [traitement de données en tables](#) du programme de première en cas de difficultés (et aussi l'utilisation des dictionnaires)

EXERCICE 2

```

1 def trouver_intrus(tab, g, d):
2     """
3     Renvoie la valeur de l'intrus situé entre les indices g et d
4     dans la liste tab où
5     tab vérifie les conditions de l'exercice,
6     g et d sont des multiples de 3.
7     """
8     if g == d:
9         return tab[g] #(1)
10    else:
11        nombre_de_triplets = (d - g)// 3
12        indice = g + 3 * (nombre_de_triplets // 2)
13        if tab[indice]==tab[indice+1] : #(2)
14            return trouver_intrus(tab,indice+3,d) #(3)
15        else:
16            return trouver_intrus(tab,g,indice)

```

- La zone de recherche se limite à un élément : l'intrus. On peut mettre indifféremment `tab[g]` ou `tab[d]`, ils sont égaux puisque `g=d` ici.
- On suit l'algorithme proposé dans l'énoncé et on compare l'élément du milieu à son voisin de droite
- Attention à la légère différence de traitement dans un cas où passe à `indice+3` car les éléments sont situés **strictement** avant l'intrus

 **Attention**

Sujet sans doute difficile et qui présente un algorithme nouveau (bien que semblable à une recherche par dichotomie)

3.2.26. Corrigé sujet 26 - Année : 2022

Sujet 26 - 20222 

EXERCICE 1

```

1 def RechercheMin(tab):
2     if tab==[]: return None
3     indice_mini,mini = 0, tab[0]
4     for indice in range(1,len(tab)):
5         if tab[indice]<mini:
6             indice_mini,mini = indice,tab[indice]
7     return indice_mini

```

 **Commentaires**

- Rien n'est indiqué pour la liste vide, on a choisi de renvoyer `None` dans ce cas
- Un parcours par indice est nécessaire puisqu'on a besoin de la position du minimum.

EXERCICE 2

```

1 def separe(tab):
2     i = 0
3     j = len(tab)-1 #(1)
4     while i < j :
5         if tab[i] == 0 :
6             i = i + 1 #(2)
7         else :
8             tab[i], tab[j] = tab[j],tab[i] #(3)
9             j = j-1
10    return tab

```

1. `i` et `j` sont les indices délimitant la partie non encore triée du tableau, au début c'est donc le tableau entier. Et on rappelle que l'indice du dernier élément d'une liste `tab` est `len(tab)-1`.
2. Si on rencontre un 0, la zone non triée diminue "par la gauche", donc on incrémente `i`
3. Dans le cas contraire, la zone non triée diminue "par la droite". On a rencontré un 1, on le positionne donc à l'extrême droite de la zone non triée en l'échangeant avec la valeur situé à cet endroit.

3.2.27. Corrigé sujet 27 - Année : 2022

Sujet 27 - 20222 

EXERCICE 1

```

1  def taille(arbre,lettre):
2      if arbre[lettre]==['','']:
3          return 1
4      elif arbre[lettre][0]=='':
5          return 1+taille(arbre,arbre[lettre][1])
6      elif arbre[lettre][1]=='':
7          return 1+taille(arbre,arbre[lettre][0])
8      else:
9          return 1+taille(arbre,arbre[lettre][0])+taille(arbre,arbre[lettre][1])

```

 **Commentaires**

1. La correction suit les indications du sujet en traitant les 4 cas. On peut faire autrement (et plus simplement).
2. Ce sujet est sans doute difficile car il utilise une représentation des arbres binaires inhabituelle, en plus de mélanger diverses notions du programme (récursivité, arbre, dictionnaire, listes)

EXERCICE 2

```

1  def tri_iteratif(tab):
2      for k in range( len(tab)-1 , 0, -1): #(1)
3          imax = 0
4          for i in range(0 , k ): #(2)
5              if tab[i] > tab[imax] :
6                  imax = i
7              if tab[imax] > tab[k] :
8                  tab[k] , tab[imax] = tab[imax] , tab[k]
9      return tab

```

1. L'indice du dernier élément d'un tableau `tab` est `len(tab)-1`. On parcourt ici dans l'ordre inverse (revoir l'instruction `range` si nécessaire)
2. Cette portion du programme est une recherche classique de maximum.
3. Ici on échange le maximum trouvé avec l'élément d'indice `k`

 **Attention**

1. En dépit du nom `tri_itératif`, il fallait reconnaître ici l'algorithme du tri par sélection.
2. Les listes étant mutables, `tab` est modifiée par la fonction (tri en place), alors que le `return` finale peut laisser penser qu'on veut récupérer un "nouveau tableau".
3. Le test ligne 7 peut paraître surprenant mais comme on a cherché le maximum entre les indices `0` et `k-1`, on doit vérifier qu'il ne se trouve pas à l'indice `k` (dans ce cas l'échange n'est pas nécessaire). On aurait pu chercher entre `0` et `k` et éviter ce test.

3.2.28. Corrigé sujet 28 - Année : 2022

Sujet 28 - 20222 

EXERCICE 1

```

1 def moyenne(tab):
2     somme = 0
3     for valeur in tab:
4         somme = somme + valeur
5     return somme/len(tab)

```

 **Commentaires**

C'est un exercice classique de parcours d'un itérable. Un parcours par élément suffit.

EXERCICE 2

```

1 def dec_to_bin(a):
2     bin_a = str(a%2) #(1)
3     a = a//2
4     while a != 0 : #(2)
5         bin_a = str(a%2) + bin_a #(3)
6         a = a // 2
7     return bin_a

```

1. C'est algorithme des divisions successives, on initialise avec le premier chiffre (donc le reste dans division euclidienne de `a` par 2)
2. L'algorithme s'arrête lorsque `a` vaut 0
3. L'algorithme donne l'ordre inverse (du dernier ou premier), on ajoute donc les chiffres successivement obtenu *devant* l'écriture binaire.

3.2.29. Corrigé sujet 29 - Année : 2022

Sujet 29 - 20222 

EXERCICE 1



1. La fonction s'appelle `fibonacci` (un n et deux c) et on parle de la suite de Fibonacci (un c et deux n). L'orthographe correcte est Fibonacci.

```

1 def fibonacci(n):
2     d = {1 : 1, 2 : 1}
3     for k in range(3, n+1):
4         d[k] = d[k-1] + d[k-2]
5     return d[n]

```



On peut aussi utiliser une liste, mais les indices sont alors décalés (le premier élément d'une liste a pour indice 0 alors que le premier élément de la suite a pour indice 1. L'utilisation d'un dictionnaire simplifie le problème.

EXERCICE 2

```

1 Liste_eleves = ['a','b','c','d','e','f','g','h','i','j']
2 Liste_notes = [1, 40, 80, 60, 58, 80, 75, 80, 60, 24]
3
4 def meilleures_notes():
5     note_maxi = 0
6     nb_eleves_note_maxi = 0 #(1)
7     liste_maxi = []
8
9     for compteur in range(len(Liste_eleves)):
10        if Liste_notes[compteur] == note_maxi: #(2)
11            nb_eleves_note_maxi = nb_eleves_note_maxi + 1
12            Liste_maxi.append(Liste_eleves[compteur])
13        if Liste_notes[compteur] > note_maxi:
14            note_maxi = Liste_notes[compteur]
15            nb_eleves_note_maxi = 1 #(3)
16            Liste_maxi = [Liste_eleves[compteur]]
17
18    return (note_maxi,nb_eleves_note_maxi,liste_maxi)

```

1. On travaille avec trois variables : la note maximale courante, le nombre d'élèves ayant cette note et la liste de ces élèves.
2. Lors du parcours de la liste de note, si on trouve une note égale à la note maximale alors on ajoute l'élève concernée à la liste de ceux ayant la meilleure note
3. Si on trouve une note supérieur au maximum courant, cette note devient le nouveau maximum et un seul élève a cette note, `liste_maxi` et `nb_eleves_note_maxi` sont donc mis à jour en conséquence



L'intérêt d'une fonction est de pouvoir être appelée au sein d'un même programme avec des arguments différents. Par conséquent, `liste_eleves` et `liste_notes` devraient être des arguments de la fonction `meilleures_notes` et pas des variables globales du programme. Cette façon de procéder est clairement une mauvaise pratique de programmation.

3.2.30. Corrigé sujet 30 - Année : 2022

Sujet 30 - 20222 

EXERCICE 1

```

1 def fusion(tab1,tab2):
2     i1,i2 = 0,0
3     tab = []
4     while i1<len(tab1) and i2<len(tab2):
5         if tab1[i1]<tab2[i2]:
6             tab.append(tab1[i1])
7             i1 += 1
8         else:
9             tab.append(tab2[i2])
10            i2 += 1
11    tab = tab + tab1[i1:] + tab2[i2:]
12    return tab

```

 **Commentaire**

Même si le tri fusion est au programme de terminale, l'exercice est sans doute difficile. On rappelle que pour fusionner deux listes déjà triées, on peut :

- parcourir les deux listes en comparant leurs éléments, le plus petit est placé dans la liste fusion et on progresse dans le parcours de la liste correspondante
- lorsque la fin d'une deux listes est atteinte on complète avec les éléments de l'autre.

EXERCICE 2

 **Bug**

Il y a des différences notables (indentation, lignes,) entre le code python qui figure sur le sujet au format pdf et le code python fourni avec le sujet sous forme d'un fichier .py

```

1 def rom_to_dec (nombre):
2     """ Renvoie l'écriture décimale du nombre donné en chiffres romains """
3
4     dico = {"I":1, "V":5, "X":10, "L":50, "C":100, "D":500, "M":1000} #(1)
5     if len(nombre) == 1:
6         return dico[nombre] #(2)
7
8     else:
9         ### on supprime le premier caractère de la chaîne contenue dans la variable nombre
10        ### et cette nouvelle chaîne est enregistrée dans la variable nombre_droite
11        nombre_droite = nombre[1:]
12
13
14        if dico[nombre[0]] >= dico[nombre[1]]:
15            return dico[nombre[0]] + rom_to_dec(nombre_droite) #(3)
16        else:
17            return rom_to_dec(nombre_droite)-dico[nombre[0]] #(4)

```

1. On complète avec les valeurs des chiffres romains (données dans l'énoncé).
2. Si le nombre contient un unique chiffre, on renvoie sa valeur.
3. Ici, il faut ajouter la valeur de la première lettre car elle est supérieure à la valeur de la lettre suivante.
4. Ici, il faut soustraire car elle est inférieure à la valeur de la lettre suivante.

AAttention

- Le sujet utilise les *slices* (`nombre_droite = nombre[1:]`) qui ne sont pas au programme.

3.2.31. Corrigé sujet 31 - Année : 2022

Sujet 31 - 20222 

EXERCICE 1

```

1 def recherche(a,t):
2     nb_occurrence = 0
3     for elt in t:
4         if elt==a:
5             nb_occurrence +=1
6     return nb_occurrence

```

 **Commentaires**

C'est un exercice classique de parcours d'un itérable en comptant les occurrences d'apparition d'une valeur. Un parcours par élément suffit, les indices des occurrences n'étant pas utilisées.

EXERCICE 2

```

1 def rendu_monnaie_centimes(s_due, s_versee):
2     pieces = [1, 2, 5, 10, 20, 50, 100, 200]
3     rendu = [] #(1)
4     a_rendre = s_versee - s_due #(2)
5     i = len(pieces) - 1
6     while a_rendre > 0 : #(3)
7         if pieces[i] <= a_rendre :
8             rendu.append(pieces[i]) #(4)
9             a_rendre = a_rendre - pieces[i]
10        else :
11            i = i-1
12    return rendu

```

1. La liste des pièces à rendre, initialisée à []
2. La somme à rendre, initialisé à `s_versee - s_due`
3. La condition d'arrêt, plus rien à rendre
4. C'est l'algorithme glouton classique pour le rendu de monnaie (les pieces sont rangées dans l'ordre). Si la pièce est inférieure à la somme à rendre, on l'ajoute au rendu et on diminue la somme à rendre. Sinon on passe à la pièce suivante.

 **Attention**

1. On utilise ici une liste de pièces classées par ordre croissant de valeurs, cela oblige donc à commencer par la fin de la liste. C'est ce qui explique le parcours de la liste "à l'envers" : initialisation de `i` à `len(pièces)-1` puis décrémentation de `i`.
2. La fonction utilise deux arguments `s_versee` et `s_due` pour calculer la somme à rendre (`s_versee-s_due`), on pourrait directement une fonction qui prend en argument la somme à rendre.

3.2.32. Corrigé sujet 32 - Année : 2022

Sujet 32 - 20222 

EXERCICE 1

```

1 def recherche(elt,tab):
2     for i in range(len(tab)-1,-1,-1):
3         if elt==tab[i]:
4             return i
5     return -1

```

 **Commentaires**

Le sujet demande de rechercher la **dernière** occurrence, la correction proposée ici parcourt la liste à l'envers et renvoie la première occurrence rencontrée. Il faut donc dans ce cas savoir à écrire un parcours à l'envers à l'aide de `range`. On peut faire aussi parcourir dans le sens normal jusqu'à la fin et mettre la jour l'indice à chaque fois qu'on rencontre la valeur:

 **Script Python**

```

def recherche(elt,tab):
    indice = -1
    for i in range(len(tab)):
        if tab[i] == elt:
            indice = i
    return indice

```

EXERCICE 2

```

1 class AdresseIP:
2
3     def __init__(self, adresse):
4         self.adresse = adresse
5
6     def liste_octet(self):
7         """renvoie une liste de nombres entiers,
8             la liste des octets de l'adresse IP"""
9         return [int(i) for i in self.adresse.split(".")]
10
11    def est_reservee(self):
12        """renvoie True si l'adresse IP est une adresse
13            réservée, False sinon"""
14        return self.liste_octet()[3]==0 or self.liste_octet()[3]==#(1)
15
16    def adresse_suivante(self):
17        """renvoie un objet de AdresseIP avec l'adresse
18            IP qui suit l'adresse self
19            si elle existe et False sinon"""
20        if self.liste_octet()[3] < 254:
21            octet_nouveau = self.liste_octet()[3] + 1
22            return AdresseIP('192.168.0.' + str(octet_nouveau)) #(2)
23        else:
24            return False

```

1. Le dernier octet est le 4ème élément (donc celui d'indice 3) de la liste renvoyée par la méthode `liste_octet`. L'adresse est réservée lorsque ce dernier octet vaut 0 ou 255.
2. Attention à la conversion de type, pour concaténer le début de l'adresse avec le dernier octet

 **Attention**

La méthode `split` des chaînes de caractères est utilisée sans explications ni exemples dans la méthode `liste_octet`. Bien comprendre que cette méthode permet de convertir une adresse comme "192.168.0.0" (une chaîne de caractères) en `[192,168,0,0]` (liste d'entiers).

3.2.33. Corrigé sujet 33 - Année : 2022

Sujet 33 - 20222 

EXERCICE 1

```

1 def convertir(T):
2     poids = len(T)-1
3     valeur = 0
4     for elt in T:
5         valeur += 2**poids * elt
6         poids -=1
7     return valeur

```

 Commentaires

On peut aussi effectuer un parcours par indice (cela invite d'utiliser la variable `poids`)

EXERCICE 2

```

1 def tri_insertion(L):
2     n = len(L)
3
4     # cas du tableau vide
5     if L==[]:
6         return L
7
8     for j in range(1,n):
9         e = L[j]
10        i = j
11
12        # A l'etape j, le sous-tableau L[0,j-1] est trie
13        # et on insere L[j] dans ce sous-tableau en determinant
14        # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].
15        while i > 0 and L[i-1] > L[j]: #(1)
16            i = i - 1
17
18        # si i != j, on decale le sous tableau L[i,j-1] d'un cran
19        # vers la droite et on place L[j] en position i
20        if i != j:
21            for k in range(j,i,-1): #(2)
22                L[k] = L[k-1]
23            L[i] = e #(3)
24
25    return L

```

1. On se contente de suivre les indications données en commentaire.
2. On décale les éléments de façon à laisser libre l'emplacement d'indice `i`.
3. On a sauvegardé dans `e` la valeur à insérer.

 Attention

1. On peut regretter les noms de variables courts et donc fort peu explicites.
2. L'insertion dans le début de liste se fait souvent en échangeant le nombre avec son voisin de droite tant qu'il lui est inférieur (ou que le début de liste n'est pas atteint)
3. Le commentaire ligne 18 parle du "sous tableau `L[i:j-1]`", il faut comprendre les éléments du tableau dont les indices sont entre `i` et `j-1`. De même ligne 12 pour le `sous tableau L[0:j-1]`.
4. Cette fonction fait un tri en place et modifie donc la liste `L`, le `return` final laisse cependant croire qu'on a construit un nouveau table qu'on souhaite renvoyer.

3.2.34. Corrigé sujet 34 - Année : 2022

Sujet 34 - 20222 

EXERCICE 1

 Bug

Il y a une faute de frappe dans la variable `alphabet` de l'énoncé (une virgule en trop dans la valeur `'o,'`)

```

1  alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
2      'j', 'k', 'l', 'm', 'n', 'o,', 'p', 'q', 'r',
3      's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
4
5  def occurrence_max(ch):
6      occurrence = [0]*26
7      for caractère in ch:
8          if caractère in alphabet:
9              index_caractère = alphabet.index(caractère)
10             occurrence[index_caractère] += 1
11     indice_max_occurrence = 0
12     max_occurrence = occurrence[0]
13     for i in range(0, len(occurrence)):
14         if occurrence[i]>max_occurrence:
15             max_occurrence = occurrence[i]
16             indice_max_occurrence = i
17     return alphabet[indice_max_occurrence]
```

 Commentaires

1. Le problème du nombre d'occurrence maximal d'un caractère dans un texte peut aussi se résoudre en utilisant un dictionnaire dont les clés sont les caractères et les valeurs les nombres d'occurrences.
2. On utilise dans la correction proposée ici, la méthode `index` qui renvoie l'indice de la première apparition d'un élément dans une liste. On peut aussi rechercher cet indice en effectuant un parcourt de la liste.

EXERCICE 2

```

1 def nbLig(image):
2     '''renvoie le nombre de lignes de l'image'''
3     return len(image) #(1)
4
5 def nbCol(image):
6     '''renvoie la largeur de l'image'''
7     return len(image[0]) #(2)
8
9 def negatif(image):
10    '''renvoie le negatif de l'image sous la forme
11       d'une liste de listes'''
12    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))]
13    # on cree une image de 0 aux memes dimensions que le parametre image
14    for i in range(len(image)):
15        for j in range(nbCol(image)): #(3)
16            L[i][j] = 255-image[i][j] #(4)
17    return L
18
19 def binaire(image, seuil):
20    '''renvoie une image binarisee de l'image sous la forme
21       d'une liste de listes contenant des 0 si la valeur
22       du pixel est strictement inferieure au seuil
23       et 1 sinon'''
24    L = [[0 for k in range(nbCol(image))] for i in range(nbLig(image))] # on cree une image de 0 aux memes dimensions que le parametre image
25    for i in range(len(image)):
26        for j in range(nbCol(image)):
27            if image[i][j] < seuil : #(5)
28                L[i][j] = 0
29            else:
30                L[i][j] = 1
31    return L

```

1. Comme indiqué dans l'énoncé, une image est une listes de listes (toutes de même longueur).la hauteur de l'image est le nombres de listes
2. La largeur de l'image est la longueur d'une sous-liste (elles ont toute la même longueur).
3. On utilise la fonction `nbCol` définie plus haut
4. Se référer à l'énoncé : la somme du pixel et de son négatif doit faire 255.
5. On applique l'algorithme donné en commentaire : le pixel est mis à 0 s'il est inférieur au seuil et à 1 sinon.

Attention

1. On peut regretter que le programme définit les fonctions `nbLig` et `nbCol` afin de récupérer les dimensions d'une image mais ne les utilise pas pour parcourir l'image (voir ligne 14 et 25)

3.2.35. Corrigé sujet 35 - Année : 2022

Sujet 35 - 20222 

EXERCICE 1

```

1 def moyenne(tab):
2     somme = 0
3     for valeur in tab:
4         somme = somme + valeur
5     return somme/len(tab)

```

 **Commentaires**

Exercice classique de parcours d'une liste, un parcours par élément suffit les indices n'étant pas utilisés.

EXERCICE 2

```

1 def dichotomie(tab, x):
2     """
3         tab : tableau trié dans l'ordre croissant
4         x : nombre entier
5         La fonction renvoie True si tab contient x et False sinon
6     """
7     # cas du tableau vide
8     if tab==[]: #(1)
9         return False,1
10    # cas où x n'est pas compris entre les valeurs extrêmes
11    if (x < tab[0]) or (x>tab[len(tab)-1]):
12        return False,2 #(2)
13
14    debut = 0
15    fin = len(tab) - 1
16    while debut <= fin:
17        m = (debut+fin)//2 #(3)
18        if x == tab[m]:
19            return True
20        if x > tab[m]:
21            debut = m + 1
22        else:
23            fin = m-1
24    return False,3 #(4)

```

1. Comme indiqué au dessus en commentaire : c'est le cas du tableau vide !
2. Valeur cherchée en dehors des valeurs extrêmes, on rappelle que l'indice du dernier élément d'un tableau `tab` est `len(tab)-1`.
3. Calcul de l'indice situé au milieu entre les deux indices de recherche.
4. L'énoncé demande de renvoyer `False,3` dans ce cas.

 **Attention**

1. On utilise de façon préférentielle des fonctions ayant toujours le même type de sortie, ici on a parfois un couple (booléen, entier) (comme `False,2`) parfois un booléen seul.

3.2.36. Corrigé sujet 36 - Année : 2022

Sujet 36 - 20222 

EXERCICE 1

```

1 def recherche_elt(tab):
2     for i in range(len(tab)-1,-1,-1):
3         if elt==tab[i]:
4             return i
5     return len(tab)

```

 Commentaires

Le sujet demande de rechercher la **dernière** occurrence, la correction proposée ici parcourt la liste à l'envers et renvoie la première occurrence rencontrée. Il faut donc dans ce cas savoir à écrire un parcours à l'envers à l'aide de `range`. On peut faire aussi parcourir dans le sens normal jusqu'à la fin et mettre la jour l'indice à chaque fois qu'on rencontre la valeur:

 Script Python

```

def recherche(tab, n):
    indice_solution = len(tab)
    for i in range(len(tab)):
        if tab[i] == n:
            indice_solution = i
    return indice_solution

```

EXERCICE 2

```

1 from math import sqrt # import de la fonction racine carree
2
3 def distance(point1, point2):
4     """ Calcule et renvoie la distance entre deux points. """
5     return sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2) #(1)
6
7 assert distance((1, 0), (5, 3)) == 5.0, "erreur de calcul"
8
9 def plus_courte_distance(tab, depart):
10    """ Renvoie le point du tableau tab se trouvant a la plus
11    courte distance du point depart."""
12    point = tab[0]
13    min_dist = distance(point, depart) #(2)
14    for i in range(1, len(tab)): #(3)
15        if distance(tab[i], depart) < min_dist:
16            point = tab[i]
17            min_dist = distance(tab[i], depart)
18    return point

```

1. Un point est un tuple `(abscisse,ordonnée)` donc `point[0]` contient l'abscisse et `point[1]` l'ordonnée.
2. On initialise le minimum à la distance entre le point de départ et le premier point de la liste (celui d'indice 0)
3. Algorithme classique de recherche du minimum

 Attention

1. L'exercice demande aussi d'ajouter une ou des préconditions à la fonction `distance`. Les points ayant des coordonnées entières on peut proposer : `assert type(point[0])==int and type(point[1])==int`. On pourrait aussi s'assurer que `point` est un tuple de longueur 2.
2. On doit être très prudent en utilisant les tests d'égalité avec des flottants tels que celui de la ligne 7

3.2.37. Corrigé sujet 37 - Année : 2022

Sujet 37 - 20222 

EXERCICE 1

```

1 def verifie(liste):
2     for i in range(len(liste)-1):
3         if liste[i]>liste[i+1]:
4             return False
5     return True

```

 **Commentaire**

On se contente de vérifier que chaque élément est bien inférieur à celui qui le suit dans le tableau.

EXERCICE 2

```

1 urne = ['A', 'A', 'A', 'B', 'C', 'B', 'C', 'B']
2
3 def depouille(urne):
4     resultat = {} #(1)
5     for bulletin in urne:
6         if bulletin in resultat: #(2)
7             resultat[bulletin] = resultat[bulletin] + 1
8         else:
9             resultat[bulletin] = 1
10    return resultat
11
12 def vainqueur(election):
13     vainqueur = ''
14     nmax = 0
15     for candidat in election:
16         if election[candidat] > nmax : #(3)
17             nmax = election[candidat]
18             vainqueur = candidat
19     liste_finale = [nom for nom in election if election[nom] == nmax] #(4)
20     return liste_finale

```

1. Initialisation à un dictionnaire vide.
2. Si la clé existe dans le dictionnaire on incrémente sa valeur de 1, sinon on ajoute cette clé avec la valeur 1 (c'est le premier vote pour ce groupe)
3. Algorithme classique de recherche du maximum en parcourant toutes les valeurs
4. On construit donc ici par compréhension la liste des candidats (car il peut y en avoir plusieurs) ayant le nombre de votes maximales

 **Attention**

La variable `vainqueur` définie à la ligne 15 (et qui porte le même nom que la fonction) peut laisser penser qu'il y en a un seul ! Alors qu'on construit justement une liste pour gérer les cas d'ex-aequo, cette variable n'a en fait aucune utilité.

3.2.38. Corrigé sujet 38 - Année : 2022

Sujet 38 - 20222 

EXERCICE 1

```

1 def minimum(tab, i):
2     ind_minimum = i
3     for j in range(i+1, len(tab)):
4         if tab[j] < tab[ind_minimum]:
5             ind_minimum = j
6     return ind_minimum
7
8 def echange(tab, i, j):
9     tab[i], tab[j] = tab[j], tab[i]
10
11 def tri_selection(tab):
12     for i in range(len(tab)):
13         ind_minimum = minimum(tab, i)
14         echange(tab, i, ind_minimum)
15     return tab

```

 Commentaires

C'est l'algorithme classique du tri par sélection, les explications de l'énoncé sont peut être ambiguës, on rappelle que ce tri consiste pour `i` indice de parcours du tableau à :

- rechercher le minimum à *partir de la position i*
- échanger ce minimum avec l'élément d'indice `i`

EXERCICE 2

```

1 from random import randint
2
3 def plus_ou_moins():
4     nb_mystere = randint(1,99) #1
5     nb_test = int(input("Proposez un nombre entre 1 et 99 : "))
6     compteur = 0 #(2)
7
8     while nb_mystere != nb_test and compteur < 10 : #(3)
9         compteur = compteur + 1
10        if nb_mystere > nb_test:
11            nb_test = int(input("Trop petit ! Testez encore : "))
12        else:
13            nb_test = int(input("Trop grand ! Testez encore : "))
14
15        if nb_mystere == nb_test:
16            print ("Bravo ! Le nombre était ",nb_mystere)
17            print("Nombre d'essais: ",compteur)
18        else:
19            print ("Perdu ! Le nombre était ",nb_mystere)

```

1. L'aide sur la fonction `randint` est donnée dans l'énoncé
2. `compteur` est la variable comptabilisant le nombre de tentatives du joueur
3. Le jeu se poursuit tant que le nombres de tentatives est inférieur à 10 et que la bonne réponse n'a pas été donnée.

3.2.39. Corrigé sujet 39 - Année : 2022

Sujet 39 - 20222 

EXERCICE 1

```

1 def moyenne(tab):
2     somme = 0
3     for valeur in tab:
4         somme = somme + valeur
5     return somme/len(tab)

```

 Commentaires

Exercice classique de parcours d'une liste, un parcours par élément suffit les indices n'étant pas utilisés. Le sujet ne précise pas le comportement à adopter si la liste est vide.

EXERCICE 2

```

1 coeur = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], \
2     [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0], \
3     [0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0], \
4     [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0], \
5     [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0], \
6     [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], \
7     [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0], \
8     [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0], \
9     [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0], \
10    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], \
11    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], \
12    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
13
14 def affiche(dessin):
15     ''' affichage d'une grille : les 1 sont represente par
16     des "", les 0 par deux espaces " " '''
17     for ligne in dessin:
18         for col in ligne:
19             if col == 1:
20                 print(" ",end="")
21             else:
22                 print("  ",end="")
23         print()
24
25
26 def zoomListe(liste_depart,k):
27     '''renvoie une liste contenant k fois chaque
28     element de liste_depart'''
29     liste_zoom = [] #(1)
30     for elt in liste_depart :
31         for i in range(k):
32             liste_zoom.append(elt)
33     return liste_zoom
34
35 def zoomDessin(grille,k):
36     '''renvoie une grille ou les lignes sont zoomees k fois
37     ET repetees k fois'''
38     grille_zoom=[]
39     for elt in grille:
40         liste_zoom = zoomListe(elt,k) #(2)
41         for i in range(k):
42             grille_zoom.append(liste_zoom)
43     return grille_zoom

```

- On suit l'algorithme de l'énoncé en dupliquant chaque élément de la liste de départ k fois grâce à la boucle `for i in range(k)`
- On zoom chaque ligne en utilisant la fonction précédente

 Attention

Les commentaires sur le code python fourni et ceux écrits dans le sujet au format `.pdf` ne correspondent pas (fautes dans frappe dans le code au format `.py`)

3.2.40. Corrigé sujet 40 - Année : 2022

Sujet 40 - 20222

EXERCICE 1

```

1  def recherche_elt(tab):
2      liste_indexe=[]
3      for i in range(len(tab)):
4          if tab[i]==elt:
5              liste_indexe.append(i)
6      return liste_indexe

```

Commentaires

1. Exercice classique de parcours de listes, en utilisant ici les indices.
2. On peut utiliser une définition de liste par compréhension :

Script Python

```
def recherche_elt(tab):
    return [i for i in range(len(tab)) if tab[i] == elt]
```

EXERCICE 2

```

1  resultats = {"Dupont": {"DS1": [15, 5, 4],
2                           "DS1": [14, 5, 1],
3                           "DS2": [13, 4],
4                           "PROJET1": [16, 3],
5                           "DS3": [14, 4]},
6   "Durand": {"DS1": [6, 4],
7                           "DS1": [14, 5, 1],
8                           "DS2": [8, 4],
9                           "PROJET1": [9, 3],
10                          "IE1": [7, 2],
11                          "DS3": [8, 4],
12                          "DS4": [15, 4]}}
13
14
15  def moyenne(nom):
16      if nom in resultats: #(1)
17          notes = resultats[nom]
18          total_points = 0 #(2)
19          total_coefficients = 0
20          for valeurs in notes.values(): #(3)
21              note, coefficient = valeurs
22              total_points = total_points + note * coefficient #(4)
23              total_coefficients = coefficient + coefficient
24          return round(total_points / total_coefficients, 1 )
25      else:
26          return -1

```

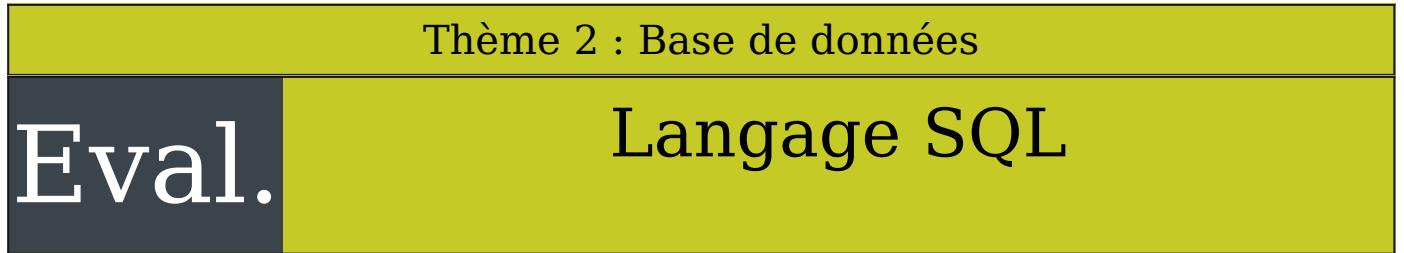
1. On vérifie que l'élève est bien présent dans les données (sinon on renvoie -1)
2. Pour calculer la moyenne il nous faut la somme des notes (pondérées par leur coefficient) et la somme des coefficients. On commence par initialiser ces deux sommes à 0
3. Parcours par valeur du dictionnaire des notes de l'élèves (on rappelle que les clés sont les types d'épreuve, on en a pas besoin ici)
4. A partir d'ici c'est l'algorithme classique du calcul d'une moyenne

Attention

L'exercice est sans doute difficile, il faut comprendre la façon dont les données sont organisées. Il s'agit d'une liste de dictionnaire ayant pour clé les noms des élèves et comme valeur un dictionnaire qui lui a pour clé les types d'épreuve et les valeurs une liste contenant la note et son coefficient.

9. Evaluations

1. 9.1 SQL : Devoir n°1



1.1. 9.1.1 D'après 2021, France, J2,

L'énoncé de cet exercice utilise les mots du langage SQL suivants : SELECT FROM, WHERE, JOIN ON, INSERT INTO VALUES, UPDATE, SET, DELETE, COUNT, AND, OR .

On considère dans cet exercice une gestion simplifiée des emprunts des ouvrages d'un CDI. La base de données utilisée sera constituée de trois relations (ou tables) nommées `Eleves`, `Livres` et `Emprunts` selon le schéma relationnel suivant :

- `Livres` (isbn (CHAR 13), titre (CHAR), auteur (CHAR))
- `Emprunts` (idEmprunt (INT), #idEleve (INT), #isbn (CHAR 13), dateEmprunt (DATE), dateRetour (Date))
- `Eleves` (idEleve (INT), nom (CHAR), prenom (CHAR), classe (CHAR))

Dans ce schéma relationnel, un attribut souligné indique qu'il s'agit d'une clé primaire.

Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère. Ainsi, l'attribut `idEleve` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `idEleve` de la relation `Eleves`. De même l'attribut `isbn` de la relation `Emprunts` est une clé étrangère qui fait référence à la clé primaire `isbn` de la relation compléter `Livres`.

1. Expliquer pourquoi le code SQL ci-dessous provoque une erreur.

Requête SQL

```
INSERT INTO Eleves VALUES (128, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (200, 'Dupont', 'Jean', 'T1') ;
INSERT INTO Eleves VALUES (128, 'Dubois', 'Jean', 'T2') ;
```

2. Dans la définition de la relation `Emprunts`, qu'est-ce qui assure qu'on ne peut pas enregistrer un emprunt pour un élève qui n'a pas encore été inscrit dans la relation `Eleves` ?

3. Écrire une requête SQL qui renvoie les titres des ouvrages de Molière détenus par le CDI.

4. Décrire le résultat renvoyé par la requête ci-dessous.

Requête SQL

```
SELECT COUNT(*)
FROM Eleves
WHERE classe = 'T2' ;
```

5. Camille a emprunté le livre « *Les misérables* ». Le code ci-dessous a permis d'enregistrer cet emprunt.

Requête SQL

```
INSERT INTO Emprunts
VALUES (640, 192, '9782070409228', '2020-09-15', NULL);
```

Camille a restitué le livre le 30 septembre 2020. Recopier et compléter la requête ci-dessous de manière à mettre à jour la date de retour dans la base de données.

Requête SQL

```
UPDATE Emprunts  
SET .....  
WHERE ..... ;
```

6. Décrire le résultat renvoyé par la requête ci-dessous.

Requête SQL

```
SELECT DISTINCT nom, prenom  
FROM Eleves, Emprunts  
WHERE Eleves.idEleve = Emprunts.idEleve  
AND Eleves.classe = 'T2' ;
```

7. Écrire une requête SQL qui permet de lister les noms et prénoms des élèves qui ont emprunté le livre « *Les misérables* ».

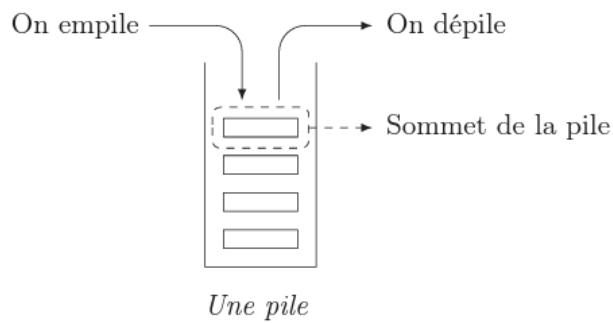
2. 9.2 DS 0011

DS

Devoir : Pile -File et Routage



On rappelle qu'une pile est une structure de données abstraite fondée sur le principe «dernier arrivé, premier sorti» :



On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous :



Structure de données abstraite : Pile

Opérations :

- `creer_pile_vide` : $\lambda(\text{varnothing}) \rightarrow \text{Pile}$
- `creer_pile_vide()` renvoie une file vide
- `est_vide` : $\text{Pile} \rightarrow \text{Booléen}$
- `est_vide(pile)` renvoie True si pile est vide, False sinon
- `empiler` : $\text{Pile}, \text{Elément} \rightarrow \text{Rien}$
- `empiler(pile, element)` ajoute element au sommet de la pile
- `depiler` : $\text{Pile} \rightarrow \text{Elément}$
- `depiler(pile)` renvoie l'élément eau sommet de la pile en le retirant de la pile

1. On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut) :

| |
|---|
| 4 |
| 2 |
| 5 |
| 8 |

Quel serait le contenu de la pile Q après l'exécution de la suite d'instruction suivante ?

🐍 Script Python

```
Q=creer_pile_vide()
while not est_vide(P):
    empiler(Q, depiler(P))
```

a: On appelle **hauteur** d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile P et renvoie sa hauteur. Après appel de cette fonction, la pile P doit avoir retrouvé son état d'origine.

Exemple : si P est la pile de la question 1 : `hauteur_pile(P)` = 4.

Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les ??? par les bonnes instructions.

🐍 Script Python

```
def hauteur_pile(P):
    Q = creer_pile_vide ()
    n = 0
    while not (est_vide (P)):
        ???
        x = depiler(P)
        empiler(Q,x)
    while not (est_vide(Q)):
        ???
        empiler(P,x)
    return ???
```

b. Créer une fonction `max_pile` ayant pour paramètre une pile P et un entier i. Cette fonction renvoie la position j de l'élément maximum de la pile P.

Après appel de cette fonction, la pile P devra avoir retrouvé son état d'origine. La position du sommet de la pile est 1.

Exemple : si P est la pile de la question 1 : `max_pile(P,2)` = 1.

3. Créer une fonction `retourner` ayant pour paramètres une pile P et un entier j. Cette fonction inverse l'ordre des j derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

Exemple :

si P est la pile de la question 1(a), après l'appel de `retourner(P, 3)`, l'état de la pile P sera :

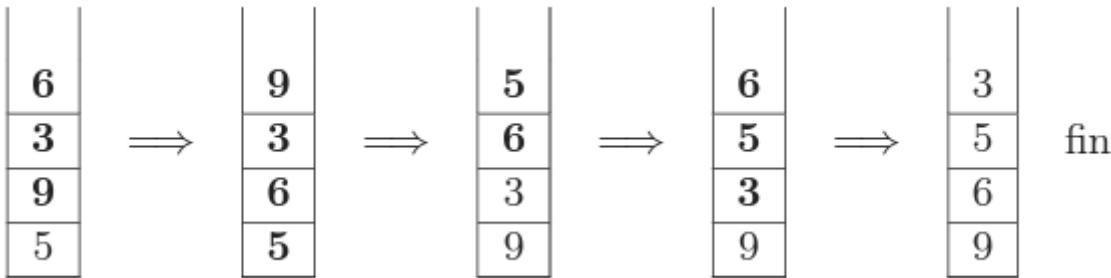
| |
|---|
| 5 |
| 2 |
| 4 |
| 8 |

4. L'objectif de cette question est de trier une pile de crêpes. On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

Le principe est le suivant :

- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :

Créer la fonction `tri_crepes` ayant pour paramètre une pile P. Cette fonction trie la pile P selon la méthode du tri crêpes et ne renvoie rien.

On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile P est

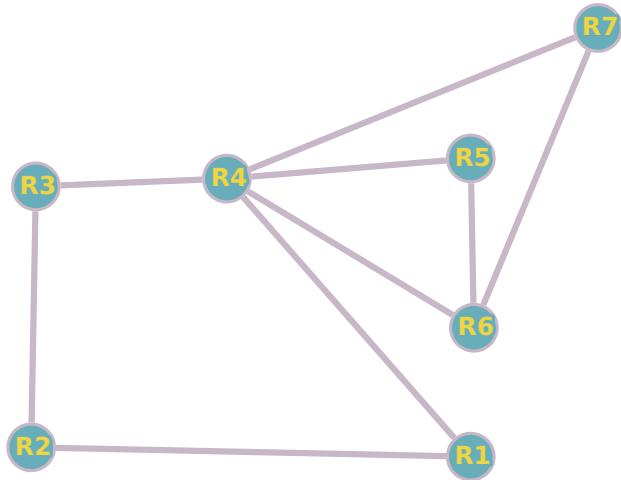
| |
|----|
| 7 |
| 14 |
| 12 |
| 5 |
| 8 |

,après l'appel de `tri_crepes(P)` , la pile P devient

| |
|----|
| 5 |
| 7 |
| 8 |
| 12 |
| 14 |



On considère le réseau suivant composé de sept routeurs.



On donne les tables de routage préalablement construites ci-dessous avec le protocole RIP (Routing Information Protocol). Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur, la distance, en nombre de sauts, qui le sépare d'un autre routeur.

Voir ANNEXE

1. Le routeur R2 doit envoyer un paquet de données au routeur R7 qui accuse réception.
Déterminer le chemin parcouru par le paquet de données ainsi que celui parcouru par l'accusé de réception.
2. a. Indiquer la faiblesse que représente ce réseau en cas de panne du routeur R4.
b. Proposer une solution pour y remédier.
3. Dans cette question uniquement, on décide de rajouter un routeur R8 qui sera relié aux routeurs R2 et R6.
a. Donner la table de routage pour R8 qui minimise le nombre de saut.
b. Donner une nouvelle table de routage pour R2.

4. Pour la suite de l'exercice on considère le réseau sans le routeur R8.

Il a été décidé de modifier les règles de routage de ce réseau en appliquant le protocole de routage OSPF qui prend en compte la bande passante.

Ce protocole attribue un coût à chaque liaison afin de privilégier le choix de certaines routes plus rapide. Plus le coût est faible, plus le lien est intéressant.

Le coût d'une liaison est calculé par la formule :

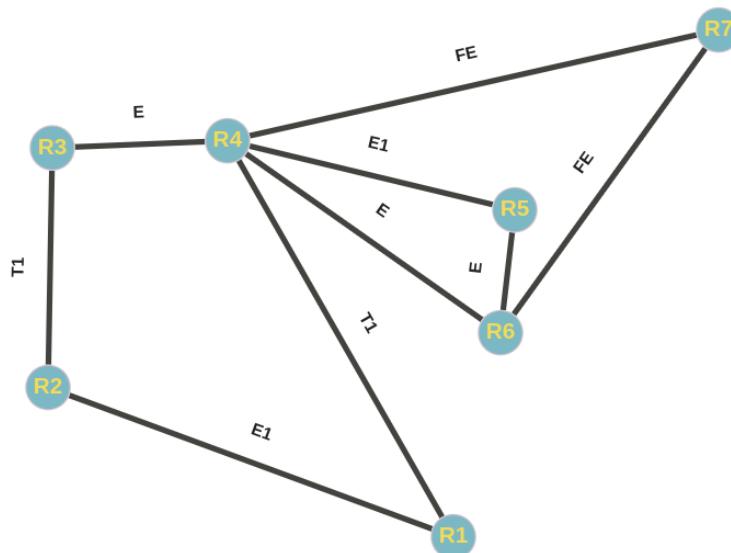
$$\text{Coût} = \frac{10^8}{\text{bit/s}} \times \text{bande passante du lien en bit/s}$$

Voici le tableau référençant les coûts des liaisons en fonction du type de liaison entre deux routeurs :

| Type de liaison | bande passante | Coût |
|-------------------|----------------|------|
| FastEthernet (FE) | ? | 1 |
| Ethernet (E) | 10 Mb/s | ? |
| (E1) | 2,048 Mb/s | 49 |
| (T1) | 1.544 Mb/s | 65 |

On rappelle que $1 \text{ Mb/s} = 1000 \text{ kb/s} = 10^6 \text{ bit/s}$.

- a. Déterminer la bande passante de FastEthernet (FE) et justifier que le coût du réseau de type Ethernet (E) est de 10.
- b. On précise sur le graphe ci-dessous les types de liaison dans notre réseau :



Le coût d'un chemin est la somme des coûts des liaisons rencontrées. Donner en justifiant le chemin le moins coûteux pour relier R2 à R5. Préciser le coût.

ANNEXE :

Table de routage de R1

| Destination | Lien | Distance |
|-------------|------|----------|
| R2 | R2 | 1 |
| R3 | R4 | 2 |
| R4 | R4 | 1 |
| R5 | R4 | 2 |
| R6 | R4 | 2 |
| R7 | R4 | 2 |

Table de routage de R2

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R1 | 1 |
| R3 | R3 | 1 |
| R4 | R1 | 2 |
| R5 | R3 | 3 |
| R6 | R3 | 3 |
| R7 | R1 | 3 |

Table de routage de R3

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R2 | 2 |
| R2 | R2 | 1 |
| R4 | R4 | 1 |
| R5 | R4 | 2 |
| R6 | R4 | 2 |
| R7 | R4 | 2 |

Table de routage de R4

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R1 | 1 |
| R2 | R3 | 2 |
| R3 | R3 | 1 |
| R5 | R5 | 1 |
| R6 | R6 | 1 |
| R7 | R7 | 1 |

Table de routage de R5

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R4 | 2 |
| R2 | R4 | 3 |
| R3 | R4 | 2 |
| R4 | R4 | 1 |
| R6 | R6 | 1 |
| R7 | R6 | 2 |

Table de routage de R6

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R4 | 2 |
| R2 | R4 | 3 |
| R3 | R4 | 2 |
| R4 | R4 | 1 |
| R5 | R5 | 1 |
| R7 | R7 | 1 |

Table de routage de R7

| Destination | Lien | Distance |
|-------------|------|----------|
| R1 | R4 | 2 |
| R2 | R4 | 3 |
| R3 | R4 | 2 |
| R4 | R4 | 1 |
| R5 | R4 | 2 |
| R6 | R6 | 1 |

10. Extras

1. 10.1 Projet

Projet

Pygame : Initiation



1.1. 10.1.1 Preamble

Pygame est un package de Python facilitant la création de jeux basés une interface graphique. Vous pouvez :

- l'installer sur votre distribution Python, par `pip3 install pygame`.
- le tester directement via <https://repl.it/>, en choisissant `pygame` dans la liste des langages proposés.

Script Python

```
import pygame, sys
from pygame.locals import *

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

while continuer:
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_RIGHT:
                continuer = False

    pygame.display.flip()

pygame.quit()
```

Commentaires

- Durant tout le code, notre scène de travail sera l'objet `ecran`, dans lequel nous viendrons coller de nouveaux éléments.

Éléments structurants d'un code pygame :

- `pygame.init()` effectue une initialisation globale de tous les modules `pygame` importés. À mettre au début du code.
- `while continue` : comme très souvent dans les jeux, la structure essentielle est une boucle infinie dont on ne sortira que par un appui sur la flèche bas où `continue` passe en `False`.

1.2. 10.1.2 Apparition d'un personnage

1.2.1. Téléchargement de l'image

Nous allons travailler avec le sprite ci-dessous, nommé `perso.png`.



Téléchargez-le pour le mettre dans le même dossier que votre code `pygame`. A redéfinir avec une bonne dimension.

Vous pouvez trouver sur internet un grand nombre de sprites libres de droits, au format `png` (donc gérant la transparence), dans de multiples positions (ce qui permet de simuler des mouvements fluides). Ici nous travaillerons avec un sprite unique.

1.2.2. Importation de l'image dans la fenêtre

Script Python

```
perso = pygame.image.load("Paragoomba.png").convert_alpha()
```

La fonction `convert_alpha()` est appelée pour que soit correctement traité le canal de transparence (canal *alpha*) de notre image.

1.2.3. Affichage de l'image

À ce stade, `perso` est un objet `pygame` de type `Surface`.

Afin de facilement pouvoir le déplacer, nous allons stocker la position de cet objet dans une variable `position_perso`, qui sera de type `rect`.

Script Python

```
position_perso = perso.get_rect()
```

Pour afficher cette image, nous allons venir le superposer aux éléments graphiques déjà dessinés (en l'occurrence : rien) avec l'instruction `blit()` :

Script Python

```
fenetre.blit(perso, position_perso)
```

► récapitulatif du code

Script Python

```
import pygame, sys
from pygame.locals import *

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
```

```

position_perso = perso.get_rect()

while continuer:
    ecran.blit(perso, position_perso)
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_RIGHT:
                continuer = False

    pygame.display.flip()

pygame.quit()

```

1.3. 10.1.3 Gestion des évènements

En informatique, un événement peut être une entrée clavier (soit l'appui soit le relâchement d'une touche), le déplacement de votre souris, un clic (encore une fois, appui ou relâchement, qui seront traités comme deux événements distincts). Un bouton de votre joystick peut aussi engendrer un événement, et même la fermeture de votre fenêtre est considéré comme un événement !

Pour Pygame, un événement est représenté par un type et divers autres attributs que nous allons détailler dans ce chapitre.

De plus, il faut savoir que chaque événement créé est envoyé sur une file (ou queue), en attendant d'être traité. Quand un événement entre dans cette queue, il est placé à la fin de celle-ci. Vous l'aurez donc compris, le premier événement transmis à Pygame sera traité en premier ! Cette notion est très importante, puisque nous allons nous en servir sous peu !

Ce type de queue est dit FIFO.

1.3.1. Comment les capturer ?

On utilise le module event de Pygame.

Voici ce que nous dit la documentation à propos de ce module :

Ficher/Masquer la documentation

- pygame.event
- pygame module for interacting with events and queues
- pygame.event.pump — internally process pygame event handlers
- pygame.event.get — get events from the queue
- pygame.event.poll — get a single event from the queue
- pygame.event.wait — wait for a single event from the queue
- pygame.event.peek — test if event types are waiting on the queue
- pygame.event.clear — remove all events from the queue
- pygame.event.event_name — get the string name from and event id
- pygame.event.set_blocked — control which events are allowed on the queue
- pygame.event.set_allowed — control which events are allowed on the queue
- pygame.event.get_blocked — test if a type of event is blocked from the queue
- pygame.event.set_grab — control the sharing of input devices with other applications
- pygame.event.get_grab — test if the program is sharing input devices
- pygame.event.post — place a new event on the queue
- pygame.event.Event — create a new event object
- pygame.event.EventType — pygame object for representing SDL events
- **event**

Et comme on peut le voir, le module event ne permet pas que d'intercepter des événements. Il nous permet aussi de créer des événements. Et même d'en bloquer !

- Lorsqu'un programme pygame est lancé, la variable interne pygame.event.get() reçoit en continu les événements des périphériques gérés par le système d'exploitation.
- Nous allons nous intéresser aux événements de type KEYDOWN (touche de clavier appuyée) ou de type MOUSEBUTTONDOWN (boutons de souris appuyé).

1.3.2. Évènements clavier

EXEMPLE DE CODE

La structure de code pour détecter l'appui sur une touche de clavier est, dans le cas de la détection de la touche «Flèche droite» :

Script Python

```
for event in pygame.event.get():
    if event.type == KEYDOWN:
        if event.key == K_RIGHT:
            print("flèche droite appuyée")
```

La touche (en anglais *key*) «Flèche Droite» est appelée `K_RIGHT` par pygame.

Le nom de toutes les touches peut être retrouvé à l'adresse [pygame ref](#)

Remarque : c'est grâce à la ligne initiale

Script Python

```
from pygame.locals import *
```

que la variable `K_RIGHT` (et toutes les autres) est reconnue.

PROBLÈME DE LA RÉMANENCE

Quand une touche de clavier est appuyée, elle le reste un certain temps. Parfois volontairement (sur un intervalle long) quand l'utilisateur décide de la laisser appuyée, mais aussi involontairement (sur un intervalle très court), lors d'un appui «classique». Il existe donc toujours un intervalle de temps pendant lequel la touche reste appuyée. Que doit faire notre programme pendant ce temps ? Deux options sont possibles :

- **option 1** : considérer que la touche appuyée correspond à un seul et unique évènement, quelle que soit la durée de l'appui sur la touche.
- **option 2** : considérer qu'au bout d'un certain délai, la touche encore appuyée doit déclencher un nouvel évènement.

Par défaut, `pygame` est réglé sur l'option 1. Néanmoins, il est classique pour les jeux vidéos de vouloir que «laisser la touche appuyée» continue à faire avancer le personnage. Nous allons donc faire en sorte que toutes les 50 millisecondes, un nouvel appui soit détecté si la touche est restée enfoncée. Cela se fera par l'expression :

🐍 Script Python

```
pygame.key.set_repeat(50)
```

1.3.3. Évènements souris

EXEMPLE DE CODE

La structure de code pour détecter l'appui sur un bouton de la souris est, dans le cas de la détection du bouton de gauche (le bouton 1) :

🐍 Script Python

```
for event in pygame.event.get():
    if event.type == MOUSEBUTTONDOWN and event.button == 1:
        print("clic gauche détecté")
```

RÉCUPÉRATION DES COORDONNÉES DE LA SOURIS

Le tuple `(abscisse, ordonnée)` des coordonnées de la souris sera récupéré avec l'instruction `pygame.mouse.get_pos()`.

1.3.4. Déplacement du personnage

Le déplacement d'un personnage se fera toujours par modification de ses coordonnées (et visuellement, par effacement de la dernière position).

Ce déplacement pourra être :

- absolu : on donne de nouvelles coordonnées au personnage.
- relatif : on indique de combien le personnage doit se décaler par rapport à sa position initiale.

1.3.5. Déplacement absolu

Pour afficher le personnage à la position `(100,200)`, on écrira :

🐍 Script Python

```
position_perso.topleft = (100, 200)
```

où `position_perso` est l'objet de type `rect` contenant les coordonnées.



Coder un script pour déplacer Paragoomba à la souris (Paragoomba doit toujours suivre la souris) (`MOUSEMOTION`)

rection

Script Python

```
import pygame, sys
from pygame.locals import *
from random import randint

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
position_perso = perso.get_rect()
position_perso.topleft = (100,200)

while continuer:
    pygame.draw.rect(ecran, (10,186,181), (0, 0, 640, 480))
    for event in pygame.event.get():
        if event.type == pygame.MOUSEMOTION:
            position_perso = event.pos
        if event.type == KEYDOWN:
            if event.key == K_RIGHT:
                continuer = False
    ecran.blit(perso, position_perso)
    pygame.display.flip()

pygame.quit()
```



Coder un script pour dessiner un rectangle sur l'écran au relâchement d'un bouton de la souris.

rection

Script Python

```
import pygame, sys
from pygame.locals import *
from random import randint

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
position_perso = perso.get_rect()
position_perso.topleft = (100,200)

largeur = 10
hauteur = 10
couleur = (200, 80, 20)

while continuer:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
            x, y = event.pos
            pygame.draw.rect(ecran, couleur, (x, y, largeur, hauteur))
        if event.type == KEYDOWN:
            if event.key == K_RIGHT:
                continuer = False
    ecran.blit(perso, position_perso)
    pygame.display.flip()
```

1.4. 10.1.4 Le jeu de tennis (à la Pong)

Pour la création du jeu de tennis, nous allons organiser notre code en plusieurs fichiers :

- un fichier tennis.py qui sera le programme principal
- un fichier constantes.py qui contiendra les constantes utilisées par les autres fichiers (hauteur et largeur de fenêtre, couleurs), certaines fonctions...

1.4.1. Les packages utilisés

On commence par introduire les packages (bibliothèques) qui seront utilisées :

Script Python

```
import pygame
from pygame.locals import *
from constantes import *
```

1.4.2. Les constantes du jeu : fichier constantes.py

On définit la hauteur et la largeur de la fenêtre ainsi que l'abscisse du mur qui sera situé à droite.

On définit également un jeu de couleurs.

■ Fichier constantes.py

🐍 Script Python

```

import pygame
from pygame.locals import *
from constantes import *

# initialisation de l'écran de jeu
pygame.init()

police = pygame.font.SysFont("Arial", 25)
fonte = pygame.font.Font(None, 30)

# Initialise la fenêtre de jeu
largeur_ecran = 600
hauteur_ecran = 400

screen = pygame.display.set_mode((largeur_ecran, hauteur_ecran))
pygame.display.set_caption("Tennis")

# variables d'état

hauteur_raquette=50

largeur_raquette =10
dist_mur = 20 # distance du mur au bord de la raquette

raquette_G_x = dist_mur
raquette_G_y = 50

ball_x = int(largeur_ecran / 2)
ball_y = int(hauteur_ecran / 2)
ball_speed_x = -4
ball_speed_y = -4
ball_rayon = 10

score = 0
vie=2

COORD_X_MUR = largeur_ecran-20

# Definit des couleurs RGB
BLACK = [0, 0, 0]
WHITE = [255, 255, 255]
GREEN = [24, 161, 80]
RED = [255, 0, 0]
BLUE = [30 , 36 , 161]
ORANGE = [196 , 92 , 54]

#fonctions permettant de dessiner la balle et les deux raquettes
def Raquette(x, y):
    R = (x,y,largeur_raquette,hauteur_raquette)
    pygame.draw.rect(screen, WHITE, R, 0)

def Balle(x,y):
    pygame.draw.circle(screen, WHITE, (x,y),10, 0)

def Mur():
    R = (COORD_X_MUR,0,20,hauteur_ecran)
    pygame.draw.rect(screen, GREEN, R, 0)

def touche_clavier():
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            # Ctrl-C pour quitter le jeu
            if event.key == pygame.K_c and pygame.key.get_mods() & pygame.KMOD_CTRL:
                quitter()
            # retourner la touche pressée
            return event.key
    # sinon, ne rien retourner (valeur nulle)
    return None

```

```

def attente():
    while touche_clavier() == None:
        pygame.display.update()

# initialisation de l'écran de jeu
pygame.init()

def affiche_texte_centre(texte, y=-1, couleur=None):
    if couleur == None:
        couleur = ORANGE
    rendu = fonte.render(texte, True, couleur)
    rectangle = rendu.get_rect()
    if y == -1:
        rectangle.center = ((largeur_ecran) / 2 , (hauteur_ecran) / 2)
    else:
        rectangle.center = ((largeur_ecran) / 2 , y)
    screen.blit(rendu, rectangle)

def affiche_texte(texte, x, y, couleur=None):
    if couleur == None:
        couleur = WHITE
    rendu = fonte.render(texte, True, couleur)
    rectangle = rendu.get_rect()
    rectangle.center = (x, y)
    screen.blit(rendu, rectangle)

```

1.4.3. Le jeu

On crée la fenêtre de jeu, on utilise la fonte courante et on charge les sons qui seront utilisé pour le jeu :

- la musique d'ambiance music.mp3
- et le bruit de verre brisé glass_break.wav qui indique la fin du jeu

Le jeu se compose de trois parties :

- l'écran d'accueil qui indique quelles sont les touches pour jouer
- le jeu de tennis
- la fin de partie qui affiche le score obtenu par le joueur

Tennis.py

Script Python

```

import pygame
from pygame.locals import *
from constantes import *

# Gestion du rafraîchissement de l'écran
clock = pygame.time.Clock()

# Cache le pointeur de la souris
pygame.mouse.set_visible(0)

#####
# écran d'accueil
#####
screen.fill(BLACK)
affiche_texte_centre("Appuyez sur une touche pour commencer", 100)
affiche_texte_centre("Flèche haut pour faire monter la raquette", 140)
affiche_texte_centre("Flèche bas pour faire descendre la raquette", 170)

attente()

# Le jeu continue tant que l'utilisateur ne ferme pas la fenêtre
Termine = False

# Boucle principale de jeu
while not Termine:
    # récupère la liste des événements du joueur
    event = pygame.event.Event(pygame.USEREVENT)

    # dessine le mur de droite

    # EVENEMENTS
    # détecte le clic sur le bouton close de la fenêtre
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            Termine = True

    # récupère la liste des touches claviers appuyées sous la forme d'une liste de booléens
    KeysPressed = pygame.key.get_pressed()

    # LOGIQUE
    # déplacement du palet gauche

    if KeysPressed[pygame.K_UP]:
        raquette_G_y -= 3

    if KeysPressed[pygame.K_DOWN]:
        raquette_G_y += 3

    if raquette_G_y < 0 :
        raquette_G_y = 0

    if raquette_G_y > hauteur_ecran - hauteur_raquette :
        raquette_G_y = hauteur_ecran - hauteur_raquette

    # Déplacement de la balle
    ball_x += ball_speed_x
    ball_y += ball_speed_y

    if ball_y < ball_rayon or ball_y > hauteur_ecran - ball_rayon :
        ball_speed_y *= -1

    # collision entre la balle et le palet de gauche
    if ball_x < dist_mur + largeur_raquette + ball_rayon :
        if ball_y > raquette_G_y and ball_y < raquette_G_y + hauteur_raquette :
            ball_speed_x *= -1
            score+=1

```

```

# collision avec les murs gauche et droit
if ball_x < ball_rayon :
    ball_x = int(largeur_ecran / 2)
    ball_y = int(hauteur_ecran / 2)
    vie-=1

if ball_x > largeur_ecran - ball_rayon :
    ball_speed_x *= -1

# AFFICHAGE
# Dessine le fond
screen.fill(BLACK)
Mur()
Raquette(raquette_G_x, raquette_G_y)

Balle(ball_x,ball_y)

# dessine le texte dans une zone de rendu à part
texte = "Votre score : " + str(score) + " Vie : " + str(vie)
if score == 15 :
    texte = "Joueur GAGNANT"
    Termine=True
if vie<=0:
    texte = 'PERDU'
    Termine=True

zone = police.render( texte, True, GREEN)
# affiche la zone de rendu au dessus de fenetre de jeu
screen.blit(zone,(280,10))

# Bascule l'image dessinée à l'écran
pygame.display.flip()

# Demande à pygame de se caler sur 30 FPS
clock.tick(30)

screen.fill( 'black')
texte = "Votre score est de {} points".format(score)
affiche_texte_centre(texte,150)
affiche_texte_centre("Appuyez sur une touche pour terminer", 200)
attente()

# Ferme la fenêtre
del(police)
pygame.quit()

```

1.4.4. Le jeu Pong en lui-même



Faites votre propre jeu avec une deuxième raquette, meilleur gestion des rebonds, changement de vitesses....

1.5. 10.1.5 SNAKE en Python, le plus simplement possible

1.5.1. Version 0

PYGAME

On importe pygame avec :



```
import pygame
from pygame.locals import *
```

Le second import sert à quitter le jeu proprement.

CONSTANTES

On crée quelques constantes :

Script Python

```
HAUTEUR = 600 # hauteur de la fenetre
LARGEUR = 600 # largeur de la fenetre
BLOC = 20

# Les couleurs utilisees
NOIR = (... , ... , ...) # fond
ROUGE = (... , ... , ...) # pomme
JAUNE = (... , ... , ...) # tete
VERT = (... , ... , ...) # corps
CYAN = (... , ... , ...) # texte

FPS = 30
```

INITIALISATION

On initialise le jeu :

Script Python

```
pygame.init()
horloge = pygame.time.Clock()
fenetre = pygame.display.set_mode((LARGEUR, HAUTEUR))
pygame.display.set_caption('Snake')

pygame.display.update()
```

BOUCLE INFINIE

Tous les jeux comportent une boucle infinie. Celle-ci ne contient pas grand chose :

- quitter le jeu,
- remplir la fenêtre de noir
- faire avancer l'horloge
- mettre à jour les affichages

BOUCLE INFINIE

Script Python

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                pygame.quit()
    fenetre.fill(NoIR)
    horloge.tick(FPS)
    pygame.display.update()
```

BOUCLE INFINIE

- La boucle `for event in...` permet de récupérer les événements "cliquer sur la croix" ou "appuyer sur Escape" et quitte le jeu dans ce cas.
- Ensuite on dessine la fenêtre, remplie de noir
- On fait avancer l'horloge
- On affiche tout ça

1.5.2. Version 1

Ecrire du texte

Cette fonction nous permettra d'écrire facilement le score

Script Python

```
def drawText(text, font, surface, x, y):
    textobj = font.render(text, 1, CYAN)
    textrect = textobj.get_rect()
    textrect.topleft = (x, y)
    surface.blit(textobj, textrect)
```

Taille de la police, valeur du score

Script Python

```
font = pygame.font.SysFont(None, 48)
```

et

Script Python

```
score = 0
```

Le serpent

Le serpent est une double liste

Script Python

```
snake = [[3, 3], [2, 3], [1, 3]]
```

Le premier élément est sa tête, elle est en [3,3] ensuite vient son corps. Il commence donc avec une taille de 3.

Dessiner le serpent

Dans la boucle infinie, avant l'horloge :

Script Python

```
for elt in snake[1:]:
    pygame.draw.rect(fenetre, VERT, (elt[0] * BLOC, elt[1] * BLOC, BLOC, BLOC))
pygame.draw.rect(fenetre, JAUNE, (snake[0][0] * BLOC, snake[0][1] * BLOC, BLOC, BLOC))
```

Le corps est vert et la tête jaune.

Afficher le score

On utilise notre fonction créée plus tôt :

Script Python

```
drawText(str(score), font, fenetre, 0.2*LARGEUR, 0.2*HAUTEUR)
```

1.5.3. Version 2

On ne capturait que "Escape" et le clic sur la croix. On ajoute les flèches.

Capturer les touches du jeu

Diminuer la vitesse de rafraîchissement

Script Python

```
FPS = 5
```

DÉPLACER LE SERPENT

On commence par créer une direction (= la vitesse) en dehors de la boucle infinie

 Script Python

```
direction = (1, 0)
```

DÉPLACER LE SERPENT

Chaque pression d'une flèche change la direction :

 Script Python

```
key = pygame.key.get_pressed()
if key:
    if key[pygame.K_UP]:
        direction = (... , ...)
    if key[pygame.K_DOWN]:
        direction = (... , ...)
    if key[pygame.K_LEFT]:
        direction = (... , ...)
    if key[pygame.K_RIGHT]:
        direction = (... , ...)
```

DÉPLACER LE SERPENT

Ensuite la tête.

C'est l'ancienne tête, qui s'est déplacée :

 Script Python

```
head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]
```

DÉPLACER LE SERPENT

Le corps se déplace.

1. On ajoute la tête au début :

 direction Script Python

```
snake.insert(0, head)
```

1. On perd un élément de fin :

 direction Script Python

```
snake.pop(-1)
```

1.5.4. Version 4

LA MORT DU SERPENT

Il meurt :

- s'il quitte l'écran
- si sa tête est dans son corps

LA MORT DU SERPENT

Direction

Script Python

```
if head in snake[1:] or head[0] < 0 or head[0] > LARGEUR / BLOC - 1 or head[1] < 0 or head[1] > HAUTEUR / BLOC - 1:
    pygame.quit()
```

1.5.5. Version 5

FLUIDITÉ

Le jeu n'est pas fluide.

On va mettre à jour les éléments du jeu toutes les 1.5 secondes et afficher 30 frames par secondes.

Il nous faut deux variables supplémentaires :

1. Une valeur pour décider quand mettre à jour
2. Un compteur

FLUIDITÉ

Script Python

```
FPS = 30
MAJ = 10

# ...

# juste avant la boucle infinie
compteur = 0
```

FLUIDITÉ

Dans la boucle infinie

Script Python

```
if compteur == MAJ:
    compteur = 0
    head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]
    # mettre les autres événements concernant le serpent

    # On augmente le compteur
    # tout à la fin de la boucle infinie
    compteur += 1
```

NOURRITURE

On crée d'abord une nouvelle liste :

Script Python

```
pomme = [8, 3]
```

NOURRITURE

On dessine la pomme comme la tête, mais en rouge

Direction

Script Python

```
pygame.draw.rect(fenetre, ROUGE, (pomme[0] * BLOC, pomme[1] * BLOC, BLOC, BLOC))
```

NOURRITURE

Puis on détecte la collision avec la pomme.

En cas de collision :

1. Le score augmente

2. Une nouvelle pomme est créée.

La boucle `while` empêche la pomme d'apparaître sur le serpent

NOURRITURE

rection**Script Python**

```
from random import randint
# ...

# Dans la boucle infinie
if snake[0] == pomme:
    score += 1
    while pomme in snake:
        pomme = [randint(0, LARGEUR / BLOC - 1),
                  randint(0, HAUTEUR / BLOC - 1)]
```

NOURRITURE

S'il n'y a pas de collision le serpent diminue, sinon il conserve sa taille

Script Python

```
else:
    snake.pop(-1)
```

1.5.6. Conclusion

C'est terminé...

Snake en 100 lignes (peu commentées) avec le minimum d'instructions. On peut faire beaucoup plus court mais c'est déjà très simple

- Python permet notamment de créer des jeux,
- Créer un jeu avec Pygame n'est pas difficile,
- Il nous faut quelques constantes, quelques éléments de jeu (serpent, tête)
- Une boucle infinie dans laquelle
- On lit les saisies de l'utilisateur
- On effectue les calculs (nouvelle tête, collisions etc.)
- On met à jour les éléments graphiques

Code final

Script Python

```
"""
Snake simple
Snake réalisé "simplement" en Pygame avec Python 3.
Nécessite Pygame et Python 3.7
"""
# -*- coding: utf-8 -*-

# pour choisir où faire apparaître la nouvelle pomme
from random import randint
# la librairie pygame
import pygame
# afin de quitter le jeu proprement
from pygame.locals import *

# Les dimensions de la fenêtre
HAUTEUR = 600 # hauteur de la fenetre
LARGEUR = 600 # largeur de la fenetre
# Ainsi que celle d'un carré à l'écran : 20 pixels
BLOC = 20

# Les couleurs utilisées
NOIR = (0, 0, 0) # fond
ROUGE = (193, 68, 51) # pomme
JAUNE = (208, 210, 62) # tête
VERT = (97, 195, 73) # corps
CYAN = (51, 133, 193) # texte

# Vitesse de rafraîchissement du jeu
FPS = 60
# On effectue les calculs toutes les 15 frames
MAJ = 15

#####
##### Fonctions #####
#####

def drawText(text, font, surface, x, y):
    """
    Dessine du texte à l'écran
    @param text: (str) le texte
    @param font: (pygame.font) la police
    @param surface: (pygame.surface) la surface sur laquelle écrire
    @param x, y: (int) les coordonnées du texte
    """

    textobj = font.render(text, 1, CYAN)
    textrect = textobj.get_rect()
    textrect.topleft = (x, y)
    surface.blit(textobj, textrect)

#####
##### GAME INITIALISATION #####
#####

# pygame
# les éléments indispensables sont init, time.Clock() et un
# display
pygame.init()
horloge = pygame.time.Clock()
fenetre = pygame.display.set_mode((LARGEUR, HAUTEUR))
# titre de la fenêtre
pygame.display.set_caption('Snake')

# taille et type de la fonte
font = pygame.font.SysFont(None, 48)

# on met immédiatement à jour avant de commencer
pygame.display.update()

#####
##### GAME LOOP #####
#####

# les éléments du jeu
```

```

# le serpent est un tableau à 2 dimensions.
# le premier est la tête, les suivants le corps
# chaque élément est une liste de coordonnées [abs, ord]
snake = [[3, 3], [2, 3], [1, 3]]
direction = (1, 0)
pomme = [8, 3]

# le score est un entier
score = 0

# compteur de frame pour les mises à jour
compteur = 0

while True:
    # Saisies de l'utilisateur

    # quitter le jeu
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()

        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                pygame.quit()

    # déplacer le serpent
    key = pygame.key.get_pressed()
    if key:
        if key[pygame.K_UP]:
            # on change la direction vers le haut
            direction = (0, -1)
        if key[pygame.K_DOWN]:
            # on change la direction vers le bas
            direction = (0, 1)
        if key[pygame.K_LEFT]:
            # on change la direction vers la gauche
            direction = (-1, 0)
        if key[pygame.K_RIGHT]:
            # on change la direction vers la droite
            direction = (1, 0)

    # Calculs
    # ils ne sont effectués que toutes les 15 frames
    if compteur == MAJ:
        # on reset le compteur
        compteur = 0

        # la nouvelle tête est l'ancienne, déplacée dans la direction
        head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]

        # on l'insère au début
        snake.insert(0, head)

        # collision tête / pomme
        if snake[0] == pomme:
            # on augmente le score
            score += 1
            # on déplace la pomme en dehors du corps
            while pomme in snake:
                # nécessaire de tirer plusieurs fois si on n'a
                # pas de chance !
                pomme = [randint(0, LARGEUR / BLOC - 1),
                         randint(0, HAUTEUR / BLOC - 1)]
        else:
            # si le serpent n'a pas mangé la pomme, il diminue
            snake.pop(-1)

        # mort du serpent
        # s'il touche son corps
        # ou s'il quitte l'écran
        if head in snake[1:] \
           or head[0] < 0 \
           or head[0] > LARGEUR / BLOC - 1 \
           or head[1] < 0 \
           or head[1] > HAUTEUR / BLOC - 1:
            pygame.quit()

    # Graphiques

    # d'abord on remplit de noir pour cacher les images précédentes
    fenetre.fill(NOIR)
    # Ensuite on dessine le corps en vert
    for elt in snake[1:]:
        pygame.draw.rect(fenetre, VERT,
                         (elt[0] * BLOC, elt[1] * BLOC, BLOC, BLOC))

```

```
# la tête en jaune
pygame.draw.rect(fenetre, JAUNE,
    (snake[0][0] * BLOC, snake[0][1] * BLOC, BLOC, BLOC))

# la pomme en rouge
pygame.draw.rect(fenetre, ROUGE,
    (pomme[0] * BLOC, pomme[1] * BLOC, BLOC, BLOC))

# le score dans le coin de l'écran
drawText(str(score), font, fenetre, 0.2 * LARGEUR, 0.2 * HAUTEUR)

# On met pygame à jour
# en avançant l'horloge
horloge.tick(FPS)
# en dessinant les éléments
pygame.display.update()
# et comptant les frames
compteur += 1
```

1.6. 10.1.6 Mini-Tetris

Un peu d'histoire

Tetris est un jeu vidéo entre arcade et puzzle, conçu par Aleksei Pajitnov en juin 1984. Le succès devient planétaire et tous les consoles qui suivront posséderont leur version de Tetris. Il fait partie des jeux les plus addictifs de l'époque, avec Pacman.

Code mini-tetris - Le début

Script Python

```

import pygame
import copy
import random

# initialisation de l'écran de jeu
pygame.init()

# Définit des couleurs RGB
NOIR = (0, 0, 0)
VERT = (0, 255, 0)
ROUGE = (255, 0, 0)
BLEU = (0, 0, 255)
GRIS = (128,128,128)
CYAN = (0,255,255)
JAUNE = (255,255,0)
ORANGE= (255,150,0)
VERT = (0,255,255)
MAUVE = (180,80,255)
LCoul = [ NOIR, GRIS, CYAN, JAUNE, MAUVE, ORANGE, BLEU, ROUGE, VERT ]

# PIECES
P_I = [ [0,2,0],
        [0,2,0],
        [0,2,0] ]

P_T = [ [0,0,0],
        [4,4,4],
        [0,4,0] ]

P_O = [ [3,3,0],
        [3,3,0],
        [0,0,0] ]

P_L = [ [0,0,0],
        [5,5,5],
        [5,0,0] ]

P_J = [ [0,0,0],
        [6,6,6],
        [0,0,6] ]

P_Z = [ [7,7,0],
        [0,7,7],
        [0,0,0] ]

P_S = [ [0,8,8],
        [8,8,0],
        [0,0,0] ]

LP = [ P_I, P_T, P_O, P_L, P_J, P_Z, P_S]

def AffPiece():
    P = LP[idpiece]
    for dx in range(3):
        for dy in range(3):
            c = P[dy][dx]
            if c != 0:
                idcoul = int(c)
                xx = (px+dx) * LCASE
                yy = (py+dy) * LCASE
                R = (xx,yy,LCASE,LCASE)
                pygame.draw.rect(screen,LCoul[idcoul],R)

# DECORS
LIGNE_VIDE = [1,1] + [0]*11 + [1]*2
DECOR = []
for i in range(16):
    DECOR.append(LIGNE_VIDE.copy())
DECOR.append([1]*15)
DECOR.append([1]*15)

LCASE = 20
def AfficheDecor():
    for y in range(len(DECOR)):
        for x in range(len(DECOR[0])):
            
```

```

        xx = x * LCASE
        yy = y * LCASE
        id = DECOR[y][x]

        pygame.draw.rect(screen,LCOUL[id],(xx,yy,LCASE,LCASE))
        pygame.draw.rect(screen,NOIR,(xx,yy,LCASE,LCASE),1)

# Initialise la fenêtre de jeu
screenWidth = 300
screenHeight = 360
screen = pygame.display.set_mode((screenWidth,screenHeight))
pygame.display.set_caption("MINI TETRIS")

# Gestion du rafraîchissement de l'écran
clock = pygame.time.Clock()
# Cache le pointeur de la souris
pygame.mouse.set_visible(0)

# variables d'état
# pièce courante
idpiece = 1
px = 6
py = 0
rot = 0
#Touche
KKeyDown = 0
KeyUp = 0
KeyLeft = 0
KeyRight = 0

#compteur d'affichage
comptage = 0

# Le jeu continue tant que l'utilisateur ne ferme pas la fenêtre
Termine = False

# Boucle principale de jeu
while not Termine:
    # récupère la liste des événements du joueur
    event = pygame.event.Event(pygame.USEREVENT)

    # EVENEMENTS
    # détecte le clic sur le bouton close de la fenêtre
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            Termine = True

    # récupère la liste des touches claviers appuyées sous la forme d'une liste de booléens
    KeysPressed = pygame.key.get_pressed()

    # LOGIQUE
    # déplacement de la pièce
    comptage += 1
    if comptage % 20 == 0 :
        py += 1

    if KeysPressed[pygame.K_UP] and KeyUp == 0:
        pass

    if KeysPressed[pygame.K_LEFT] and KeyLeft == 0:
        pass

    if KeysPressed[pygame.K_RIGHT] and KeyRight == 0:
        pass

    if KeysPressed[pygame.K_DOWN] and KeyDown == 0:
        pass

    KeyDown = KeysPressed[pygame.K_DOWN]
    KeyUp = KeysPressed[pygame.K_UP]
    KeyLeft = KeysPressed[pygame.K_LEFT]
    KeyRight = KeysPressed[pygame.K_RIGHT]

    # AFFICHAGE
    # Dessine le fond
    AfficheDecor()
    AffPiece()

```

```
# Bascule l'image dessinée à l'écran
pygame.display.flip()

# Demande à pygame de se caler sur 30 FPS
clock.tick(30)

# Ferme la fenêtre
pygame.quit()
```

Le jeu est fonctionnel que dans une petite partie : une pièce descend mais il est impossible de la déplacer.

1.6.1. Présentation du code

Les constantes couleurs :

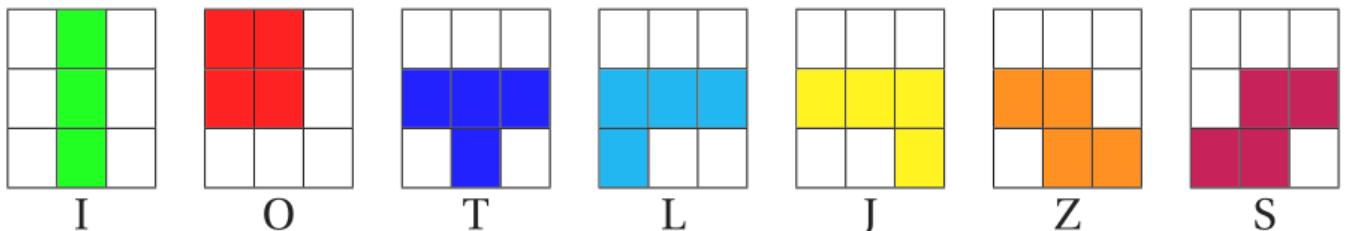
Script Python

```
# Définit des couleurs RGB
NOIR = (0, 0, 0)
VERT = (0, 255, 0)
ROUGE = (255, 0, 0)
BLEU = (0, 0, 255)
GRIS = (128, 128, 128)
CYAN = (0, 255, 255)
JAUNE = (255, 255, 0)
ORANGE = (255, 150, 0)
MAUVE = (180, 80, 255)
LCoul = [ NOIR, GRIS, CYAN, JAUNE, MAUVE, ORANGE, BLEU, VERT ]
```

Le fond noir est associé à la valeur d'indice 0, les murs gris à la valeur d'indice 1 et chaque pièce du jeu est associée avec la couleur d'indice compris entre 2 et 8.

LES DIFFÉRENTES PIÈCES :

Pour simplifier les algorithmes, on va utiliser des combinaisons de carrés qui s'inscrivent dans une grille 3 \(\times\) 3



Toutes les pièces sont stockées dans des listes de 3 \(\times\) 3 éléments. Une valeur nulle correspond à une case vide et une valeur non nulle indique une case pleine ainsi que sa couleur. L'ensemble des pièces est stocké dans une liste nommée LP :

Script Python

```
# PIECES
P_I = [ [0,2,0],
        [0,2,0],
        [0,2,0] ]

P_O = [ [3,3,0],
        [3,3,0],
        [0,0,0] ]

P_T = [ [0,0,0],
        [4,4,4],
        [0,4,0] ]

P_L = [ [0,0,0],
        [5,5,5],
        [5,0,0] ]

P_J = [ [0,0,0],
        [6,6,6],
        [0,0,6] ]

P_Z = [ [7,7,0],
        [0,7,7],
        [0,0,0] ]
```

```
P_S = [ [0,8,8],
        [8,8,0],
        [0,0,0]]]

LP = [ P_I, P_O, P_T, P_L, P_J, P_Z, P_S]
```

Les variables d'état sont présentées ci-dessous.

🐍 Script Python

```
# variables d'état
# pièce courante
idpiece = 1
px = 6
py = 0
rot = 0
#Touches
KeyDown = 0
KeyUp = 0
KeyLeft = 0
KeyRight = 0
```

La variable `idpiece` indique l'indice de la pièce courante dans la liste `LP`. Ainsi le jeu démarre avec la pièce T.

Les variables `px`, `py` et `rot` indiquent la position en (x) , et (y) de la pièce dans la grille, ainsi que sa rotation : 0 pour 0° et 1 pour 90° .

Et quatre variables pour stocker l'état précédent des touches fléchées : enfoncé ou non. L'intérêt des deux dernières est de pouvoir détecter les appuis sur ces touches.

1.6.2. Affichage du décors :

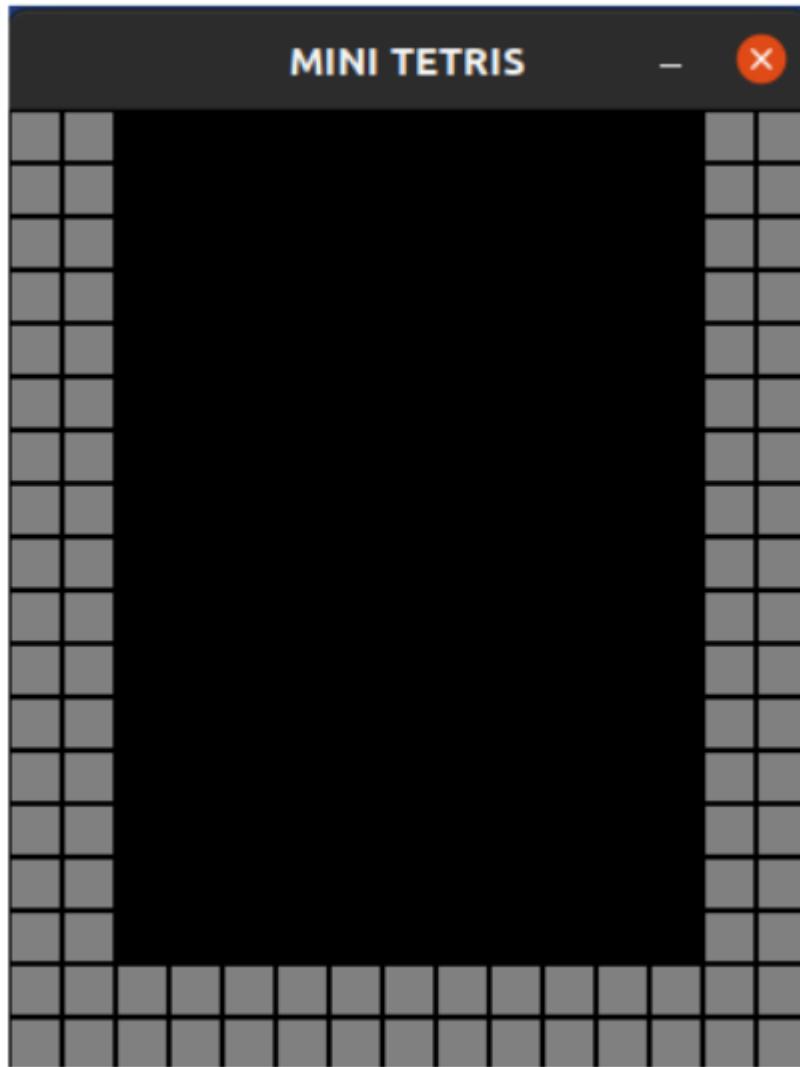
🐍 Script Python

```
# DECORS
LIGNE_VIDE = [1,1] + [0]*11 + [1]*2
DECOR = []
for i in range(16):
    DECOR.append(LIGNE_VIDE.copy())
DECOR.append([1]*15)
DECOR.append([1]*15)

LCASE = 20
def AfficheDecor():
    for y in range(len(DECOR)):
        for x in range(len(DECOR[0])):
            xx = x * LCASE
            yy = y * LCASE
            id = DECOR[y][x]
            pygame.draw.rect(screen,LCOUL[id],(xx,yy,LCASE,LCASE))
            pygame.draw.rect(screen,NOIR,(xx,yy,LCASE,LCASE),1)
```

Le décor est stocké dans une grille de 16 cases de large pour 18 de haut. Comme la largeur des cases fait 20 pixels, on a donc une fenêtre de taille 300 (times) 360 pixels. On définit une constante `LIGNE_VIDE` composée de 2 colonnes sur la gauche et sur la droite, qui marquent les bords avec des cases grises, donc de couleur associée 1. Les 11 cases centrales vides sont remplies avec la valeur 0. Le décor est défini comme une liste de 16 lignes.

Les 16 premières sont des lignes vides, et les 2 dernières sont remplies de 1. Pour créer les 16 lignes vides, nous utilisons la liste `LIGNE_VIDE` qu'on copie avec la fonction `copy()`. Ceci est très important car chaque ligne doit être indépendante !



Les valeurs sont stockées dans une liste de listes intitulée `DECOR`. Ainsi `len(DECOR)` correspond au nombre de lignes et `len(DECOR[0])` au nombre de colonnes du jeu.

En écrivant `DECOR[y][x]` on accède à l'indice de couleur pour la case de coordonnées (x,y). L'origine du décor (0,0) est positionnée en haut à gauche de l'écran.

La variable `LCASE` définit la largeur d'une case en pixels. Pour dessiner entièrement la grille, on utilise un double boucle en x et y.

On dessine un carré plein grâce à la première fonction `draw.rect()`, puis les bords noirs avec le deuxième appel.

1.6.3. Déplacement des pièces :

On déplace la pièce courante avec une technique particulière. On utilise une variable comptage qui comptabilise le nombre d'affichages effectués. Le test effectué est : `comptage % 20 == 0`, ce qui produit 20 affichages. Comme on est à 30 FPS, cela se produit toutes les 0,66 seconde. à ce moment-là, on fait descendre la pièce d'une ligne vers le bas. (A ce niveau aucune collision n'est pas gérée)

🐍 Script Python

```
# LOGIQUE
# déplacement de la pièce
comptage += 1
if comptage % 20 == 0 :
    py += 1
```

Dans la partie gérant la logique du jeu, on trouve cette ligne

Script Python

```
if KeysPressed[pygame.K_UP] and KeyUp == 0:  
    pass
```

La variable `KeyUp` stocke l'état de la touche [Flèche Haut] lors de l'affichage précédent. Dans cette condition, on détecte si le joueur vient d'appuyer sur cette touche. Pour l'instant cette condition ne déclenche rien mais cela va changer par la suite.



Gestion de la rotation

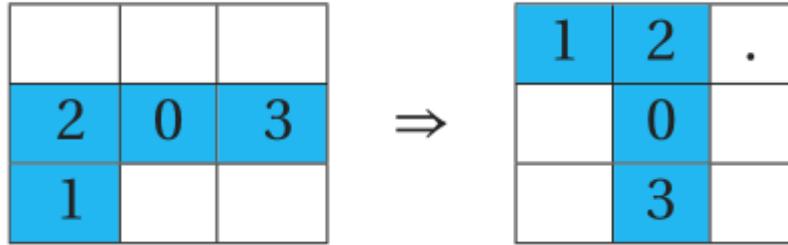
Vous allez gérer la rotation de la pièce courante. Tout d'abord après la condition gérant l'appui sur la touche [Flèche Haut], vous allez modifier la valeur de la variable `rot`. Chaque appui doit augmenter la variable `rot` de 1. Il serait judicieux d'appliquer un modulo 4 pour faire en sorte que cette variable ne puisse prendre que des valeurs entre 0 et 3. Dans le jeu original, les pièces ne tournent que dans un sens.

[Question 1](#) [Question 2](#) [Question 3](#)

Créez une fonction `Rot90Droite(P)` qui, à partir d'une pièce 3 \times 3 tourne cette pièce de 90° . La pièce P correspond à une liste de listes, cette pièce ne doit pas être modifiée. Vous allez construire une nouvelle pièce et la retourner. Voici quelques conseils :

- Pour créer une nouvelle pièce, vous pouvez l'initialiser à partir d'une liste de listes contenant des 0 ou appliquer la fonction `copy.deepcopy()` sur la pièce P actuelle. Le contenu n'a pas d'importance, car de toute façon, il va être écrasé
- Il faut programmer la rotation de 90° . Voici un exemple avec la pièce P en entrée et la pièce R à calculer à droite.

Dans tous les cas, la case centrale ne change pas.



- Option 1 : écrivez une instruction pour chacune des huit cases. Par exemple, pour la case 1 en haut à gauche : `R[0][0]=P[2][0]`, et pour la case 2 : `R[0][1]=P[1][0]`.
- Option 2 : faites une liste des positions des huit cases des bords, ceci en tournant dans le sens des aiguilles d'une montre : `L=[(0,0), (1,0), (2,0), (2,1) (2,2), (1,2) ...]` Ainsi en créant une boucle `for` d'indice `i` allant de 0 à 7, vous savez que la case à la position `R[i]` doit être initialisée avec la case `L[(i-2)%8]`.

Créez une fonction `Rotn(P,nb)` qui calcule la pièce P après nb rotations.

Pour cela :

- Initialiser une pièce 3 \times 3 sous forme de liste de listes. Il est judicieux d'utiliser la fonction `copy.deepcopy()` pour cloner la pièce P, car si la variable `nb` vaut 0, il n'y aura aucune rotation effectuée et c'est la copie de la pièce initiale qui sera retournée.
- Effectuez autant de rotations que nécessaire. Pour cela utiliser la fonction `Rot90Droite()`.
- Retournez le résultat.

Modifier la fonction `AffPiece()` pour qu'elle tienne compte de la variable `rot` et affichez la pièce en tenant compte de ce paramètre. Maintenant, lorsque vous appuyez sur la touche [Flèche Haut], vous devez voir la pièce tourner.



Déplacement latéraux

Question 1 Question 2

Écrivez une fonction `DetectCollision()` qui détermine suivant une pièce, une rotation et une position (x,y) données s'il y a collision avec le décor ou non. Voici quelques conseils :

- Appliquer la rotation sur la pièce pour obtenir sa bonne orientation
- Créez une double boucle d'indices dx et dy pour parcourir les cases de la pièce.
- Comparer chaque case `(dx,dy)` de la pièce avec la case `(x+dx,y+dy)` du décor. Si les deux cases sont non vides, alors il y a collision, et retournez vrai dans ce cas.

Complétez le code gérant l'appui sur les touches `[Flèche Droite]` et `[Flèche Gauche]`. Par exemple, lors de l'appui sur `[Flèche Gauche]`, vérifiez d'abord que la futur place de la pièce n'est pas en collision avec le décor. Si aucune collision n'est détectée, alors modifier la position de la pièce en faisant : `\(px=1\)`.



Gestion de la descente

[Question 1](#) [Question 2](#) [Question 3](#) [Question 4](#) [Question 5](#)

Écrivez une fonction `FusionDecor()` qui, suivant une pièce, une rotation et une position (x,y) donnée, fixe cette pièce dans le décor. Cette fonction est comparable à la fonction `DetectCollision()`, sauf qu'il n'y a pas à faire de test, mais juste un transfert des cases colorées de la pièce vers les cases de la grille.

Écrivez une fonction `NextPiece()` qui initialise une nouvelle pièce. Pour cela, grâce au package `random`, choisissez une pièce au hasard. Sa position sera forcément la ligne 0 et au milieu de la grille, c'est-à-dire à l'abscisse 6. Par contre vous pouvez choisir sa rotation aléatoirement.

Tous les 20 affichages, la pièce courante descend automatiquement d'une ligne, gérez la collision avec le décor. Lorsque la pièce est susceptible de descendre, examinez si sa position futur produit une collision. Dans ce cas-là, elle ne doit pas descendre, car elle est stoppée par quelque chose. Appelez cette fonction `FusionDecor()` pour figer la pièce.

Après cela générez une nouvelle pièce.

Vous pouvez maintenant gérer l'appui sur la touche `[Flèche BAS]`. Le mécanisme est identique à celui de la descente automatique.

Il reste un mécanisme à mettre en place : le retrait des lignes pleines. Cet événement peut arriver après la fusion d'une pièce avec le décor. Il se peut qu'une ou plusieurs lignes pleines apparaissent. Écrivez une fonction `RetraitLigne()` dont l'objectif est de retirer l'ensemble des lignes pleines du décor.

- Créez une boucle `for` avec un indice partant de 0 jusqu'à 15 compris. Les deux dernières lignes ne doivent pas être traitées.
- Faites un calcul pour trouver la valeur de l'indice qui parcourt les lignes en sens inverse, c'est-à-dire de l'indice 15 à 0.
- Testez la ligne associée à ce nouvel indice pour savoir si elle est pleine :
- Pour cela, il suffit de détecter si une valeur 0 est présente dans la ligne courante. Utilisez le test `0 in MaLigneCourante` qui retourne Vrai ou Faux
- Si la ligne est pleine, retirez-la grâce à la fonction `DECOR.pop(index)`.
- Une fois le parcours terminé, des lignes ont pu être supprimées. Ainsi tant que le nombre de lignes dans la liste `DECOR` est insuffisant, rajoutez des lignes vides à l'indice 0 grâce à la fonction `DECOR.insert(0,...)`. Pensez à insérer une ligne vide qui soit indépendante de la constante `LIGNE_VIDE` définie dans le programme.

11. Conseils de travail

1. 11.1 Conditions matérielles

Lorsqu'on ne travaillera pas sur les notebook de capytale, il est conseillé de travailler avec 3 espaces :

- l'écran de l'ordinateur partitionné avec les 2 premiers espaces: ce site et un IDE (Spyder ou Thonny par exemple);

Est en codant qu'on apprend à coder

Tous les exemples de code dans le cours doivent être **retapés** (résistez à l'envie du copier-coller).

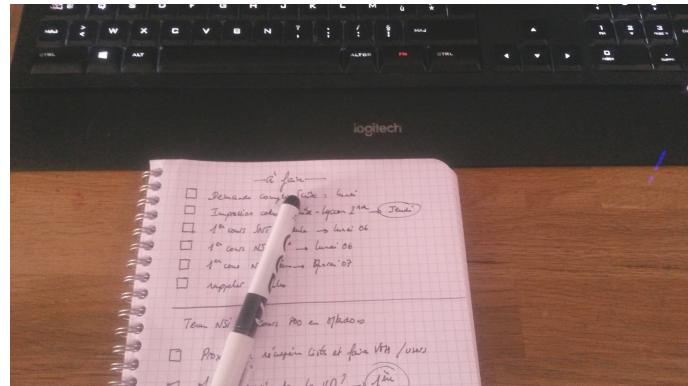
Cela permet de :

- mémoriser la syntaxe ;
- vérifier si le code proposé ne contient pas d'erreur ;

et le plus important :

- **faire ses propres tests et modifications** pour vérifier sa bonne compréhension.

- et un troisième espace essentiel : un cahier à spirale et un stylo !



2. 11.2 Dossiers, fichiers et versionning

Cette année en NSI nous allons manipuler un certain nombre de fichiers qui sont dans capytale sur l'ENT. Pensez à les "étiqueter", de plus il est possible de les télécharger sur clé usb, dans ce cas il est important de les nommer et les classer de façon rigoureuse pour les retrouver rapidement et les partager.

Conseils

- Utilisez des dossiers pour classer vos fichiers. Il n'y en a jamais assez.
- Prenez l'habitude de faire plusieurs sauvegardes de vos documents (sur le réseau du lycée, sur l'ENT, sur clé USB).
- Évitez dans les noms de fichiers et de dossiers **les espaces** (utilisez plutôt `_`) ainsi que **les caractères accentués** et les caractères spéciaux.
- Un nom de fichier doit être parlant (un peu comme une variable en fait). On évitera de nommer ses codes Python `python1.py`, `python2.py`, `python3.py`, etc. Mais plutôt `NSI_T4_tri_selection.py` par exemple pour un exercice de programmation sur le tri par selection au thème 4.
- Lorsqu'on travaille sur un projet ou un programme conséquent, il peut être utile de conserver des archives du travail à plusieurs étapes de l'élaboration, plutôt que de ne conserver que la dernière version. À cet effet on pourra numérotier : `NSI_projet_morpion_v1.py`, puis `NSI_projet_morpion_v2.py`, `NSI_projet_morpion_v3.py`, etc.

3. 11.3 Usage du clavier

Utiliser le clavier est souvent bien plus pratique et surtout plus rapide qu'utiliser la souris. Encore faut-il apprendre à l'apprivoiser...

La sélection au clavier

Outre les touches `DEBUT` et `FIN` qui permettent d'atteindre rapidement le début ou la fin d'une ligne, les flèches directionnelles servent évidemment à se déplacer dans du texte. Mais combinées:

- à la touche `CTRL` : elles permettent de se déplacer de mot en mot;
- à la touche `MAJ` : elles permettent de sélectionner un caractère;
- aux touches `MAJ` et `CTRL` : elles permettent de sélectionner une mot.

De même, en se plaçant en début d'une ligne et en combinant la touche `MAJ` et `FIN`, on sélectionne la ligne entière.

Les raccourcis clavier

Il existe de très nombreux raccourcis clavier qui permettent d'exécuter des tâches courantes sans passer par les menus du logiciel. Certains sont (quasi-)universels, c'est-à-dire que ce sont les mêmes sur tous les logiciels, d'autres sont spécifiques à chaque logiciel. Il est important d'en connaître quelques-uns pour être plus efficace.

Les universels IDE Navigateur Web

- La triplète magique `CTRL+X`, `CTRL+C`, `CTRL+V` pour couper, copier, coller;
- `CTRL+O` pour ouvrir un fichier
- `CTRL+N` pour créer un nouveau document;
- `CTRL+S` pour sauvegarder le document en cours;
- `CTRL+MAJ+S` pour sauvegarder en précisant le nom du fichier;
- `CTRL+Z` pour annuler la dernière action, `CTRL+Y` ou `CTRL+MAJ+Z` pour la rétablir;
- `CTRL+W` pour fermer un onglet;
- `CTRL+Q` ou `ALT+F4` pour fermer le logiciel;
- `CTRL+A` pour sélectionner tout (All).

À chercher de suite lorsqu'on utilise un nouvel IDE, les raccourcis pour les actions suivantes (entre parenthèses ceux de Thonny):

- exécuter le code (`F5`)
- commenter/décommenter une ligne (`CTRL+M`)
- `CTRL+T` pour ouvrir un nouvel onglet;
- `CTRL+H` pour ouvrir l'historique;
- combiner `CTRL` + clic pour forcer l'ouverture d'un lien dans un nouvel onglet;
- combiner `MAJ` + clic pour forcer l'ouverture d'un lien dans une nouvelle fenêtre;

Sources

- site de Cédric Gouygou