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

- ⭐⭐⭐ Advent of code 2022 ⭐⭐⭐ a démarré !
  - vous pouvez joindre le leaderboard NSI-Murat avec le code 2817159-e8bb5792.
  - quelques conseils ici

Actualités

- **Thème en cours** : Les arbres.

Epreuve BAC 2023

- 2 «sujets 0» correspondant à la nouvelle maquette du bac version 2023 (3 exos) ont été publiés. À retrouver
  - sujet A.
  - sujet B.
- 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.

1. Progression sur l'année

1.1. Liste des chapitres

	Titre	Durée
	C1- Le Modéle relationnel	1

	Titre	Durée
	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
	C9- Les Dictionnaires - Révision	1
	C10- Les arbres	2
	C11- Algorithmes sur les arbres	2
	C12- arbres binaires de recherche	1
	C13- Gestion Processus	1

1.2. Liste des sujets BAC

	BAC	Le
	Sujet BAC 3- Langage SQL	SQL
	Sujet BAC 4- Programmation Orientée Objet	POO
	Sujet BAC 5- Piles et Files	Struc. Données
	Sujet BAC 6- Protocole de routage	Réseaux
	Sujet BAC 8- Diviser pour régner	Algo
	Sujet BAC 10- Les arbres - Partie 1	Struc. Données

	BAC	Le
	Sujet BAC 11- Algorihtmes Arbres Binaires	Algo
	Sujet BAC 12- Arbres Binaires de Recherche	Algo
	Sujet BAC 13- Gestion Processus	os

Note

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.

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

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. ➡ 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. ➡ 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. ➡ Thème : Langage de programmation(LP)

☑ Récursivité
☐ Modularité
☐ Mise au point des programmes. Gestion des bugs.

2.5. ➡ Thème : Algorithmique(A)

☐ Algorithmes sur les arbres binaires et sur les arbres binaires de recherche.

☑ Méthode « diviser pour régner »

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

Fiche pour calculer sa note au BAC 2023

Je suis en première générale en 2021-2022.
Comment calculer ma note de baccalauréat ?

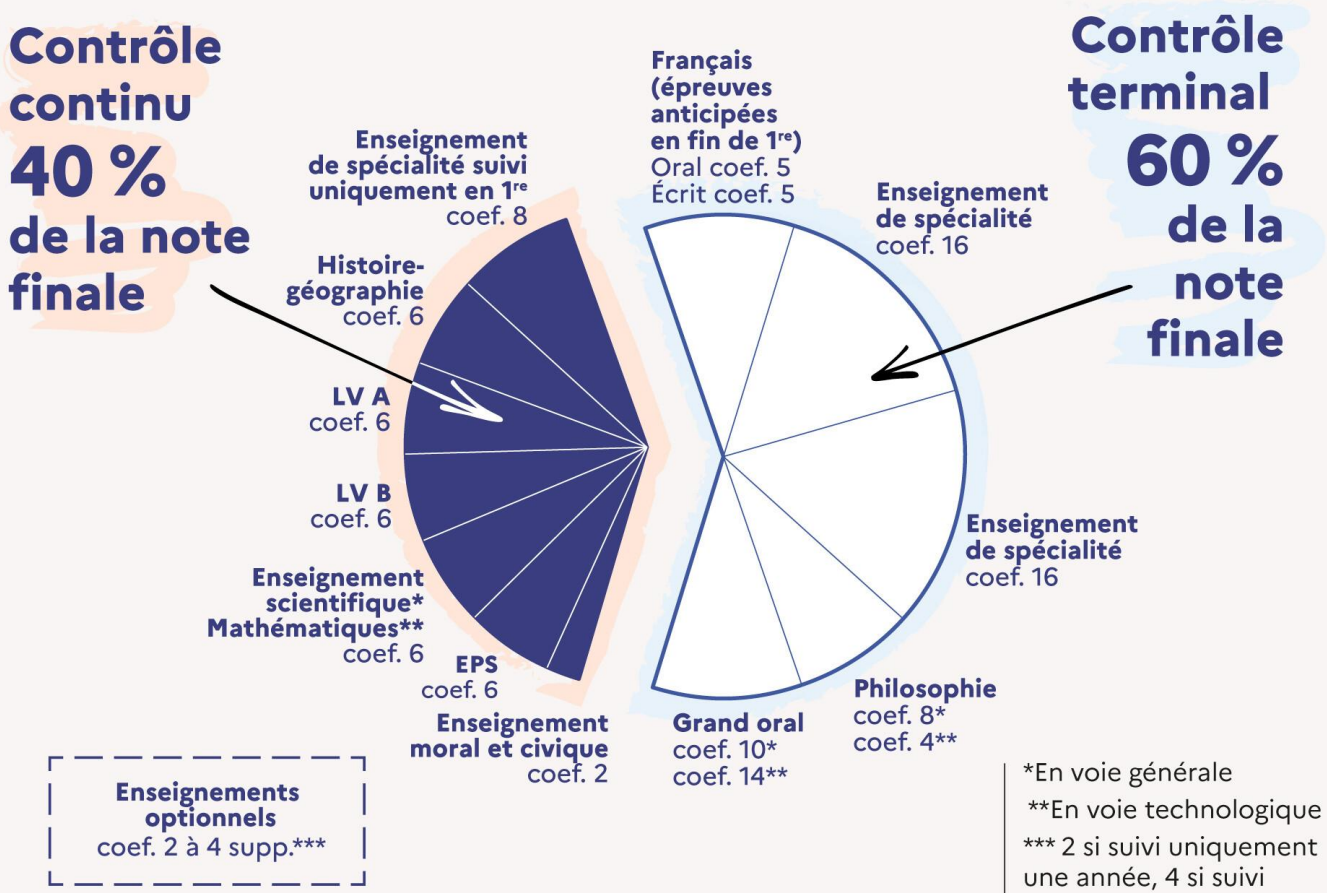
Rentrez vos notes dans ce tableau,
multipliez-les par leur coefficient et divisez le total par 100.

Pour les élèves des établissements publics et privés sous contrat ainsi que des établissements français homologués à l'étranger.

	Note/20 (1re)	Coef. (1re)	Note de 1re multipliée par coef.	Note/20 (Tle)	Coef. (Tle)	Note de Tle multipliée par coef.	**Note totale**

BACCALAURÉAT GÉNÉRAL ET TECHNOLOGIQUE

Répartition de la note finale



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

MENJS - Octobre 2021

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. 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**.

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 appelera **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.
```

3. Mise en pratique

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 caractéristiquec d'une voiture
        annee, couleur, vitesse max, age """
        self.annee = annee
        self.couleur = coul
        self.vitesse_max = vmax
        self.age = 2022 - self.annee
```

3.1.1. Création et initialisation d'un objet en Python

A 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 caractéristiquec 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, "Citroên","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
Citroên
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.

Q1

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'>
```

3.1.2. Ecriture 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 numerateur
        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.

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 aimerait écrire.

🐍 **Script Python**

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

🐍 **Script Python**

```
r1  # évaluation
```

```
<__main__.Rationnel object at 0xc72088>
```

🐍 **Script Python**

```
print(r1)  # affichag
```

```
<__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 numerateur
        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({str.num}, {str.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 **atributs** 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)`.

POO : TD

Thème 1 - Structure de données

05 TD : Programmation Orientée Objet (POO)

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. Exercice 2 : Class Player

Exercice 2

Écrire une classe `Player` qui :

- ne prendra aucun argument lors de son instanciation.
- 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**

```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**

```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(10000000)
Vous avez déposé 10000000 euros
Solde actuel du compte : 10000950 euros
```

🐍 **Script Python**

3. Exercice 3 : Class Pokemon

Exercice 4 Pokemon et POO

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 *
```

```python
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**

```python
#Ecrire le code ici !!!

assert aqua.defense == 15
```

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**

```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 : *> Créer une méthode `se_reposer(self)` qui redonne tous ses points de vie au Pokemon.*

🐍 **Script Python**

```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
```

3.3. Interaction avec un autre Pokemon

Un pokemon 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 :*** *> Créer une méthode `dominer(self, adversaire)` prenant en paramètre une autre instance de la classe `Pokemon`, qui renvoie `True` si le Pokemon a un type qui domine celui du Pokemon donné en paramètre et `False` sinon.*

🐍 **Script Python**

```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.dominer(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.attaquer(flamichou)
assert flamichou.pvActuels == 32
aqua.attaquer(rocamon)
assert rocamon.pvActuels == 74
```

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 : *> Créer 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.attaquer(p2)
        print(pokemon1.nom + ': ' + pokemon1.pvActuels + 'hp' + ' | ' + pokemon2.nom + ': ' + pokemon2.pvActuels +
'hp')
        p2.attaquer(p1)
        print(pokemon1.nom + ': ' + pokemon1.pvActuels + 'hp' + ' | ' + pokemon2.nom + ': ' + 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 : *> Compléter 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.attaquer(p2)
        print('\t' + pokemon1.nom + ': ' + str(pokemon1.pvActuels) + 'hp' + ' | ' + pokemon2.nom + ': ' +
str(pokemon2.pvActuels) + 'hp')
        p2.attaquer(p1)
        print('\t\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: 81hp | Rocamon: 50hp
        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**

POO : TD

Thème 1 - Structure de données

BAC

TD : Sujet BAC Programmation Orientée Objet (POO)

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**

```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)
```

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.

Réponse :

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).

Réponse :

Modification de la méthode `reserver`:

🐍 **Script Python**

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

ou

🐍 **Script Python**

```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`.

Réponse :

🐍 **Script Python**

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

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`.

Réponse :

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

🐍 **Script Python**

```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 ».

1. 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.

Réponse :

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**

```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**

```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()` .

a. 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**

```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 chaines 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 méhode `mystere` renvoie une liste à laquelle ont été ajoutés les noms des chambres du gite 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 gite "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` des 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)

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**

```python
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
        - son arome
        - 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
```

```python
    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.

Réponse :

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

🐍 **Script Python**

```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**

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

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

Réponse :

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

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

Réponse :

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.

Réponse :

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

🐍 **Script Python**

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

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

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

🐍 **Script Python**

```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.

Réponse :

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

🐍 **Script Python**

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

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

Réponse :

Fonction `dépiler` :

🐍 **Script Python**

```python
def depiler(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**

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

ou bien, en plus explicite :

🐍 **Script Python**

```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**

```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))
```

Réponse :

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 génèrerait 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**

```python
class Yaourt:
    """ Classe définissant un yaourt caractérisé par :
        - son arome
        - 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

Sujet BAC 4 : Programmation Orientée Objet

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

P.O.O

- 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 :

```
class Joueur:
    def __init__(self, pseudo, identifiant, equipe):
        " Appelle le constructeur et initialise "
        self.pseudo = pseudo
        self.equipe = equipe
        self.id = identifiant
        self.nb_de_tirs_emis = 0
        self.liste_id_tirs_recus = []
        self.est_actif = True

    def tire(self):
        " Méthode déclenchée par l'appui sur la gâchette "
        if self.est_actif:
            self.nb_de_tirs_emis += 1

    def est_determine(self):
        " Le joueur réalise-t-il un grand nombre de tirs ? "
        return self.nb_de_tirs_emis > 500  # Un booléen est renvoyé.

    def subit_un_tir(self, id_recu):
        " Méthode déclenchée par les capteurs de la veste "
        if self.est_actif:
            self.est_actif = False
            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"}`

Réponse

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 `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.

Réponse

🐍 **Script Python**

```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`.

Réponse

🐍 **Script Python**

```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**

```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.

Réponse

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**

```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**

```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
```

2. Exercice 2 : D'après 2022, Centres étrangers, J2, Ex. 4

P.O.O

- 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 chaine 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"`

Réponse

`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
```

Réponse

🐍 **Script Python**

```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.

Réponse

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`.

Réponse

🐍 **Script Python**

```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
```

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. Ecire plusieurs implémentations d'une même structure de données.	L'abstraction des structures de données est introduite aprés plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deuc piles).
Listes, piles, files : structures linéaires. Dictionnaires, index et clé.	Distinguer des structures par le jeu des méthodes qui les caractèrisent. 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.

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 déja 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.

2. Structures de données linéaires

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.

2.1.1. 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**.

2.1.2. 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.

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

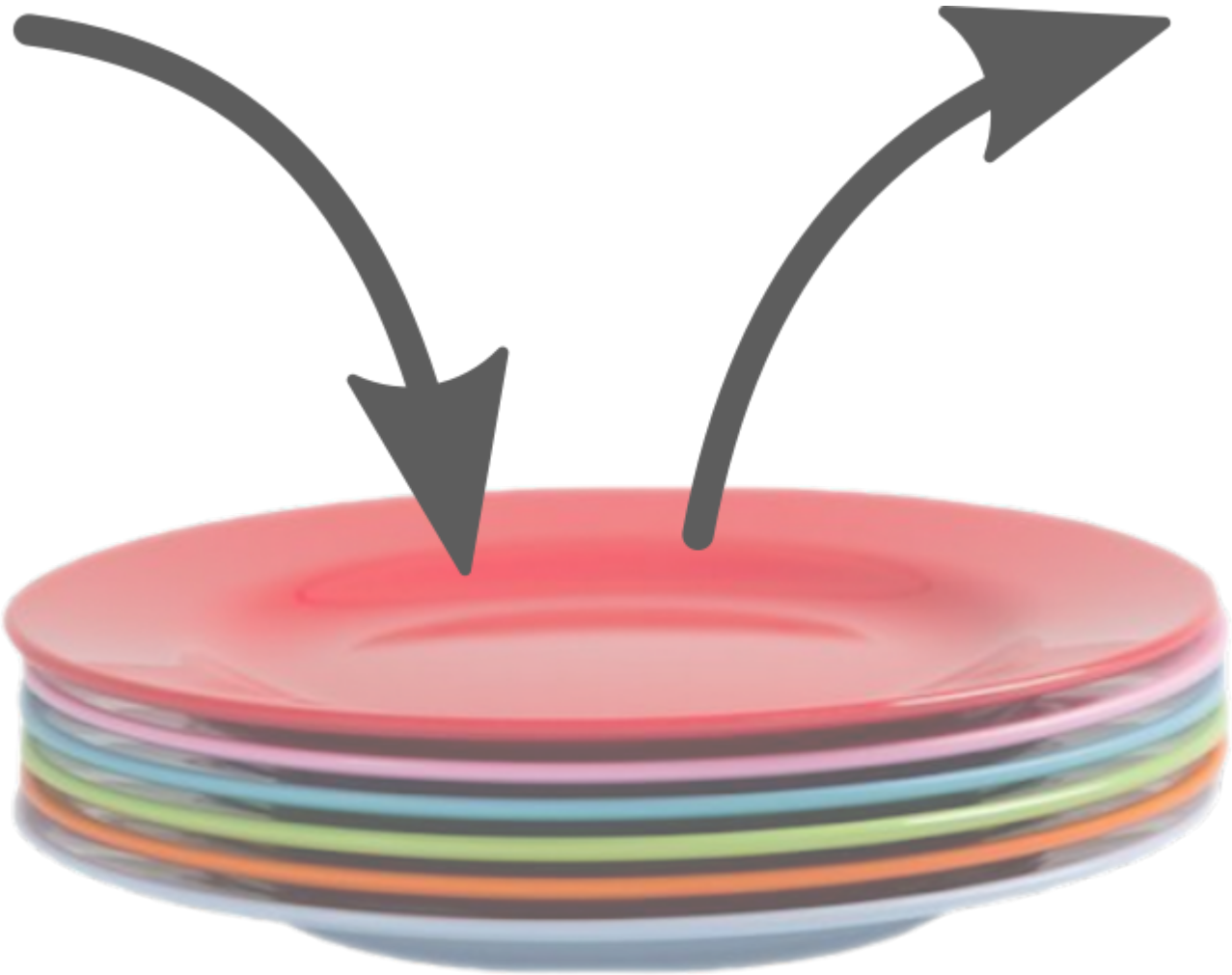
La nature des données ne fait pas tout. Il faut aussi s'intéresser à la manière dont on voudra les traiter :

- 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 ?

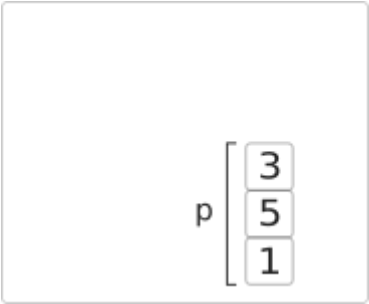
Lorsque ces problématiques d'entrée/sortie n'interviennent pas, la structure «classique» de liste est adaptée. Mais lorsque celle-ci est importante, il convient de différencier la structure de **pile** de celle de **file**.

2.2.1. Les piles (*stack*)



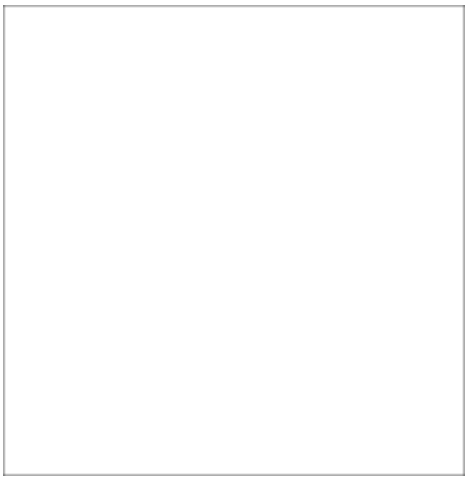
Une structure de **pile** (penser à une pile d'assiette) est associée à la méthode **LIFO** (Last In, First Out) :

- 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.

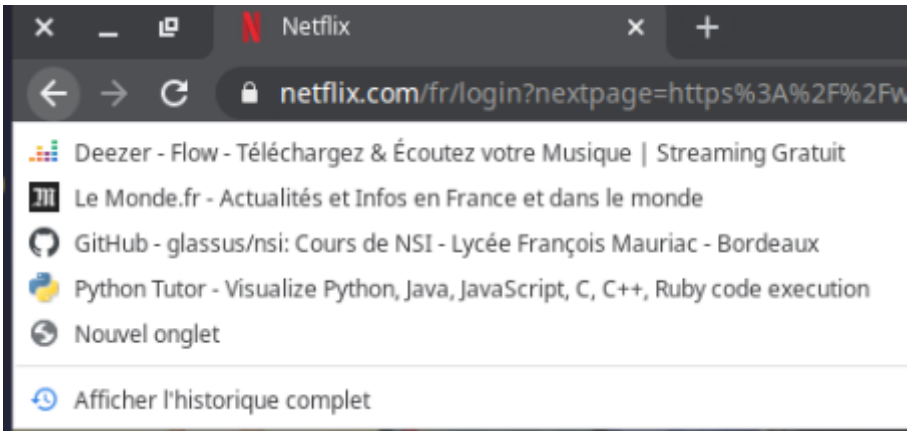


Exemples de données stockées sous forme de pile

- 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 :



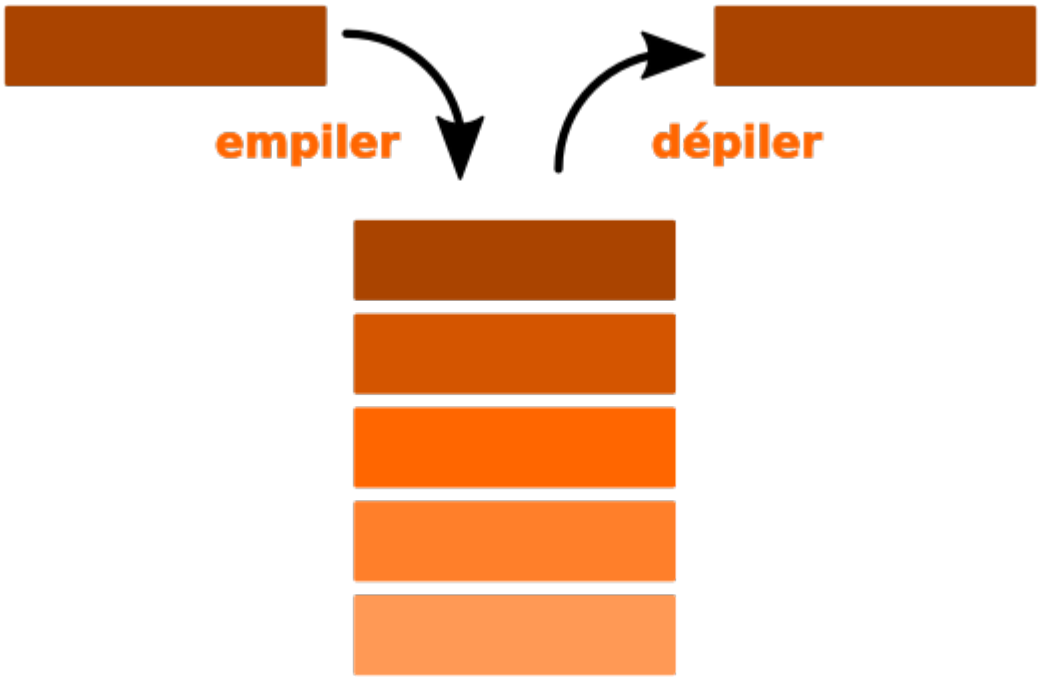
- historiques d'annulation d'instructions (Ctrl+Z)

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 (`depiler` , `pop`) et le renvoyer

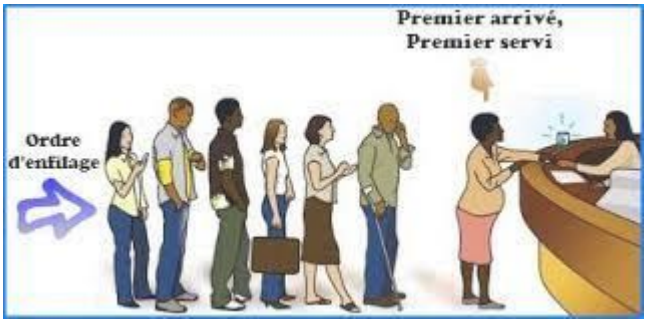
Ces opérations doivent être réalisées **en temps constant**, soit en \(O(1)\).

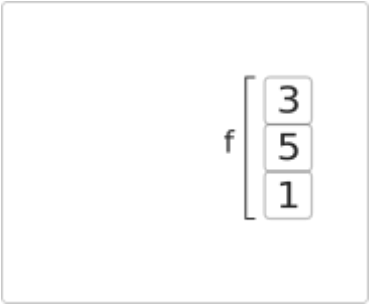


2.3. Les files

A connaitre

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...

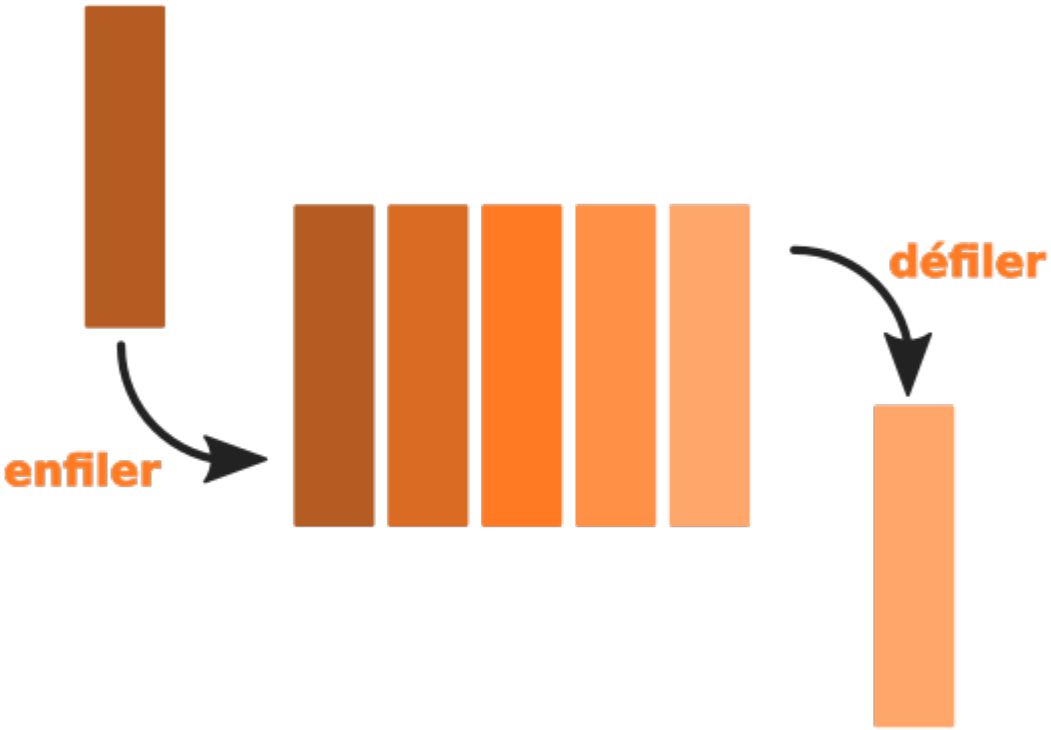
- gestion des processus (plus tard...)
- parcours en largeur d'un arbre/graphe (plus tard...)

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 \(O(1)\).

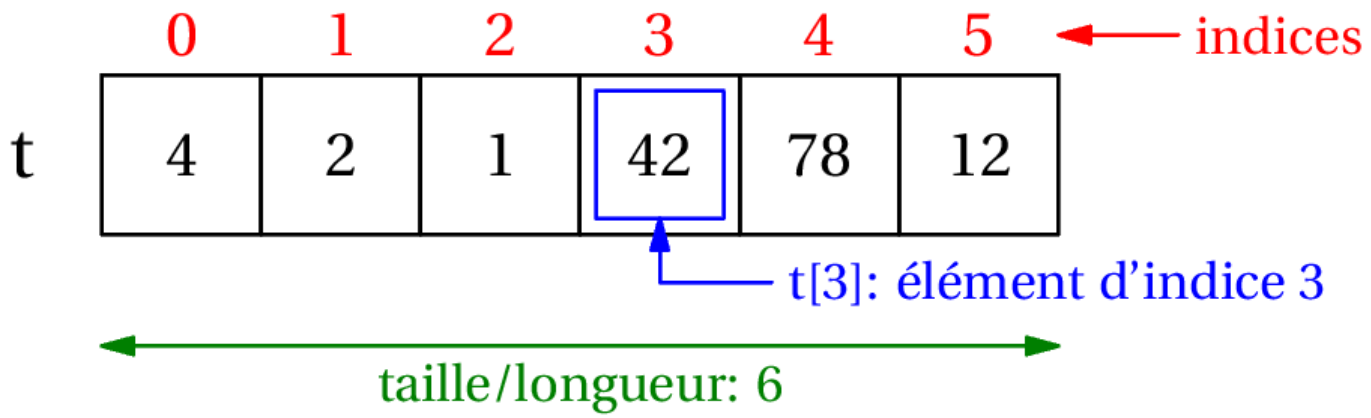


3. Listes chaînées

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 :

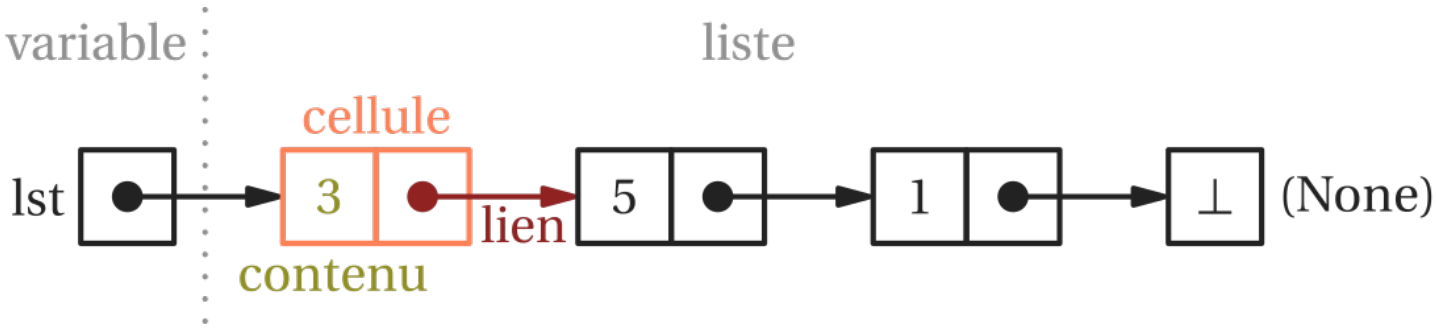
- création d'un tableau de taille donnée
- accès à un élément à partir de son indice en temps constant (\(O(1)\))
- modification d'un élément à partir de son indice en temps constant (\(O(1)\))

En revanche, l'insertion d'un élément dans le tableau impose de décaler tous les élements d'un indice, elle se fait donc en temps linéaire, soit \(O(n)\)...

3.2. Liste chaînée

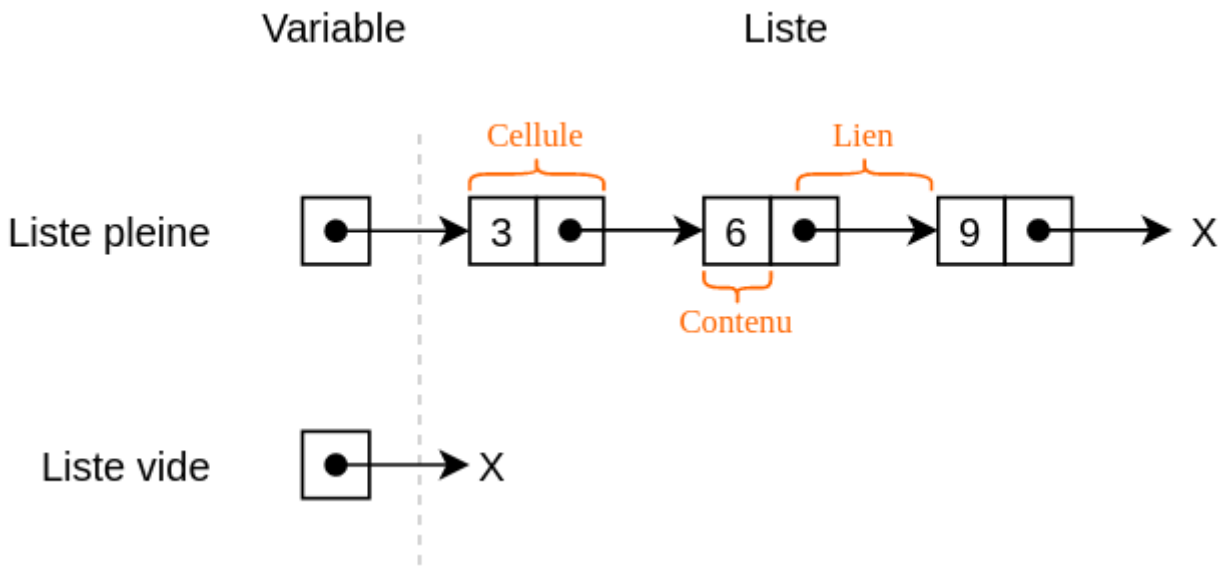
Avec une structure de **liste (chaînée)**, on représente à nouveau une séquence d'éléments (à nouveau le plus souvent homogènes), mais les données ne sont pas nécessairement séquentielles en mémoire. On dispose en revanche d'un moyen permettant de passer d'un élément au suivant, d'où le terme *chaîné*.

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.

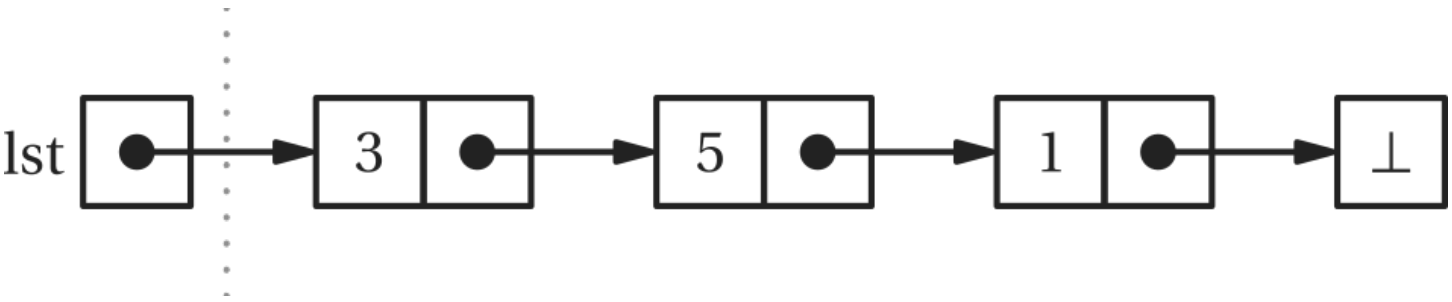


Interface :

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 \(O(1)\).



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 (en \(O(n)\)) et donc non constant.

3.2.1. 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.

3.2.2. Exemple d'implémentation minimale d'une liste chaînée

Exemple : implémentation d'une liste chainée **en POO** 💛

🐍 **Script Python**

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

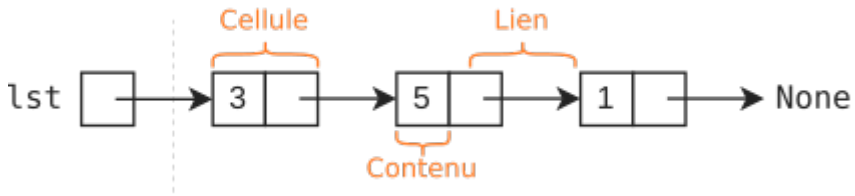
Cette implémentation rudimentaire permet bien la création d'une liste :

🐍 **Script Python**

```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...

3.2.3. 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**

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

Implémentation

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

🐍 **Script Python**

```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 renvoie_tete(self):
        t=self.tete.contenu
        self.tete=self.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.renvoie_tete()))
        l.reverse()
        for v in l:
            self.ajoute_tete(v)
        l.reverse()
        return ' -> '.join(l)
```

Vérification de la question 2

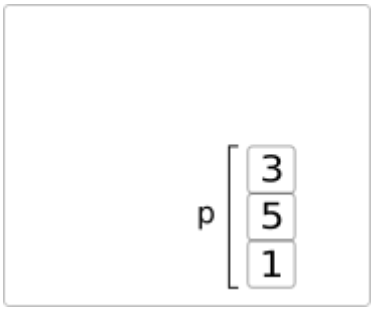
🐍 **Script Python**

```
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.renvoie_tete() )

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

lst.ajoute_tete(2)

print(lst)
print("valeur renvoyé : ",lst.renvoie_tete() )
print(lst)
print("valeur renvoyé : ",lst.renvoie_tete() )
print(lst)
print("valeur renvoyé : ",lst.renvoie_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
```

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 a minima :

- 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**.

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() #
```

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()`

4.2.1. À l'aide du type `list` de Python

Exercice

Créer la classe ci-dessus. Le type `list` de Python est parfaitement adapté. Des renseignement 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**

```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)

4.2.2. 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**

```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**

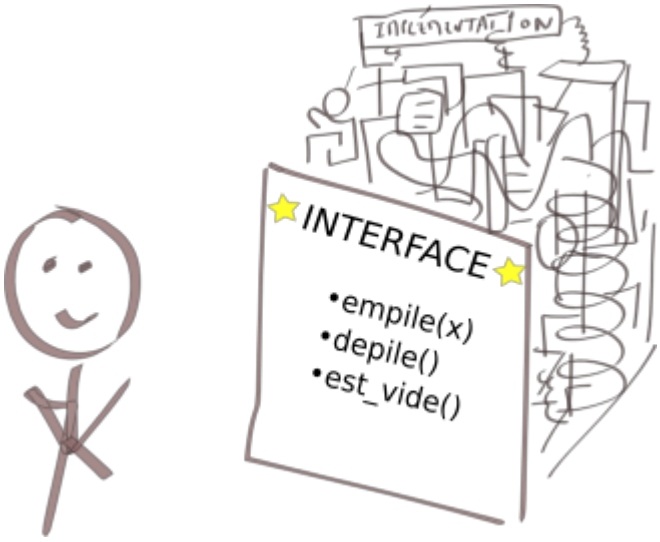
```python
class Pile:
    def __init__(self):
        self.data = None

    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.



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**

```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**

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

🐍 **Script Python**

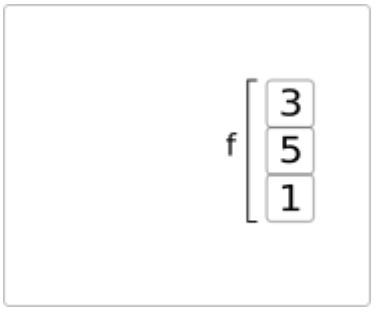
```python
back()
adresse_courante
```

Correction et commentaires

🐍 **Script Python**

```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. Les Files



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

5.1. 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
```

5.2. 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**

```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 = "|"             # convenablement la file avec print(p)
        for k in self.data :
            s = s + str(k) + "|"
        return s
```

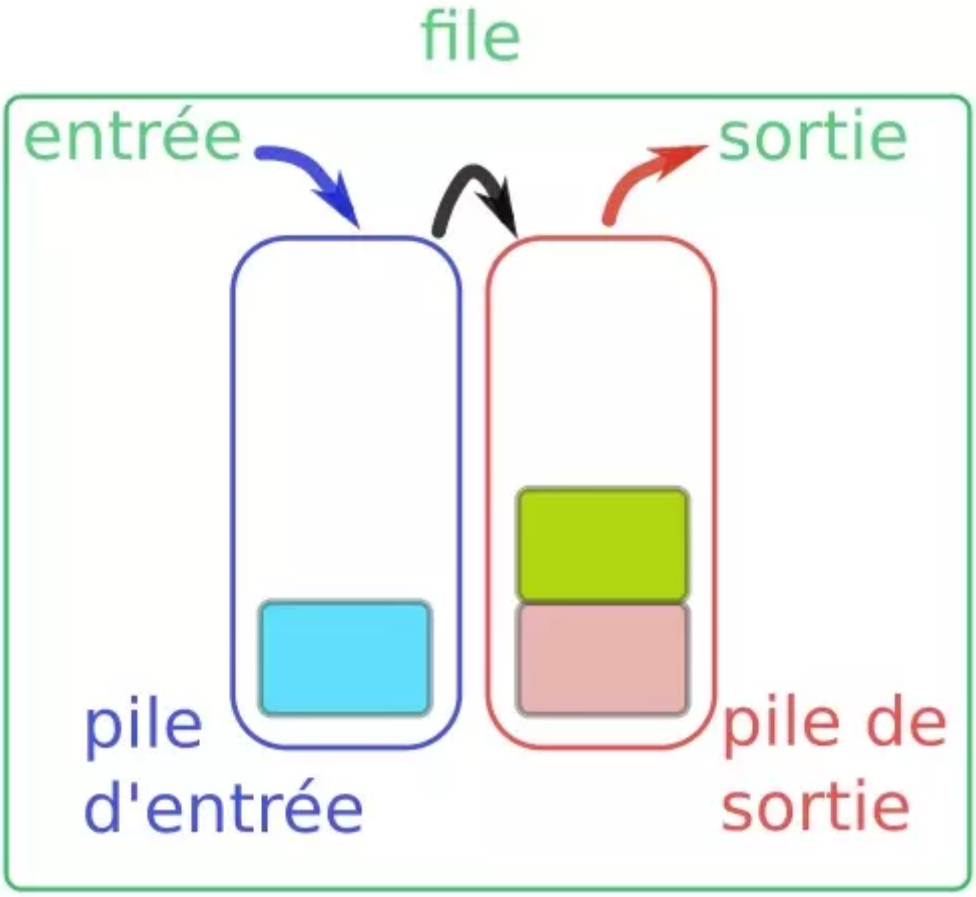
Remarque :
Notre implémentation répond parfaitement à l'interface qui était demandée. Mais si le «cahier des charges» obligeait à ce que les opérations `enfile()` et `defile()` aient lieu en temps constant (en \(O(1)\)), notre implémentation ne conviendrait pas.

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.

5.3. 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 1ère cellule
        return v

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

```python
            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**

```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**

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

f.defile()
```

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**

```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**

```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 ...
```

Sujet BAC 5 : Piles et Files

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

D'après 2022, Métropole, J1, Ex. 1

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	Structure de données
`"(2 + 3) × (18/(4 + 2))"`	`()(())`

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 : " ((()()"
- Expression simplifiée C : "(()))("

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.

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 dépilée) 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><em></p></em>"
↑
```

```
flowchart TD
   A[" <br> <br> <br> <br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD
   A[" <br> <br> <br>&lt;p&gt;<br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD

A[" <br> <br>&lt;em&gt;<br>&lt;p&gt;<br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD
   A[" <br> <br> <br>&lt;p&gt;<br>=====<br>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><em></em></p>"
↑
```

```
flowchart TD
   A[" <br> <br> <br> <br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD
   A[" <br> <br> <br>&lt;p&gt;<br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD

A[" <br> <br>&lt;em&gt;<br>&lt;p&gt;<br>=====<br>Pile"]
```

Parcours de l'expression

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

```
flowchart TD
   A[" <br> <br> <br>&lt;p&gt;<br>=====<br>Pile"]
```

`<em>` et `</em>` se correspondent.

Parcours de l'expression

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

```
flowchart TD
   A[" <br> <br> <br> <br>=====<br>Pile"]
```

`<p>` et `</p>` se correspondent.

Parcours de l'expression

```
"<p><em></em></p>"
            ↑
```

```
flowchart TD
   A[" <br> <br> <br> <br>=====<br>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.

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.

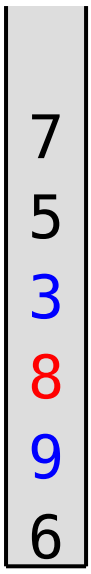
Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

Voici un exemple détaillé de déroulement d'une partie.

Premier parcours de la pile Deuxième parcours Troisième parcours Quatrième parcours

7
5
4
3
8
9
6

7
5
4
3
8
9
6



- 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.

7
5
3
9
6

7
3
9
6

7
3
9
6

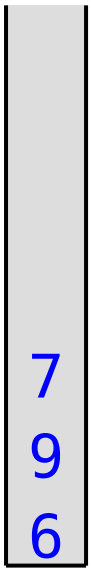
- On recommence à partir du haut de la pile : on retire le 5.
- Le triplet suivant (3, 9 et 6) n'entraîne pas de suppression.
- Il ne reste plus que deux valeurs à étudier (9 et 6) : on a terminé le deuxième parcours.

7
3
9
6

7
9
6

- On recommence en haut de la pile avec 7, 2 et 9 : on retire le 2.
- Il ne reste que le 9 et le 6 : on a terminé le

troisième parcours.



- On recommence en haut de la pile avec 7, 9 et 6. La pile est inchangée.
- La pile n'a pas été modifiée lors de ce parcours : la partie est terminée et cette pile n'est pas gagnante.

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

Réponse

Il s'agit d'une pile gagnante :

Premier parcours **Second parcours**

4
9
8
7
4
2

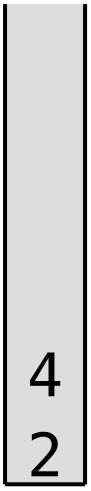
4
8
7
4
2

4
8
4
2

- 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.

4
8
4
2

4
4
2



- 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

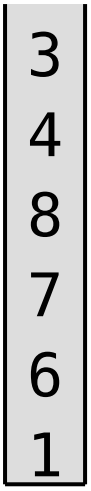
Réponse

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 4 2 1		
5 5 4 2 1		

4
5
4
9
2
0

4
0



L'int[illegible] d'une pile est proposée ci-dessous :

- [illegible]ile_vide() renvoie une pile vide,
- [illegible](p) renvoie True si p est vide, False sinon,
- [illegible](p, element) ajoute element au sommet de p,
- [illegible]p) retire l'élément au sommet de p et le renvoie,
- [illegible](p) renvoie l'élément au sommet de p sans le retirer de p,
- [illegible]: renvoie le nombre d'éléments de p.

Dans [illegible]te 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 dépilés 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.

```
def reduire_triplet_au_sommet(p):
    haut = depiler(p)
    milieu = depiler(p)
    bas = sommet(p)
    est_reduit = ...
    if haut % 2 != ...:
        empiler(p, ...)
        ...
    empiler(p, ...)
    return ...
```

Réponse

```python
def reduire_triplet_au_sommet(p):
    haut = depiler(p)
    milieu = depiler(p)
    bas = sommet(p)
    est_reduit = True
    if haut % 2 != bas % 2:
        empiler(p, milieu)
        est_reduit = False
    empiler(p, haut)
    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 :

```python
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    reduction_pendant_parcours = False
    while taille(p) >= 3:
        if ...:
            reduction_pendant_parcours = ...
        e = depiler(p)
        empiler(q, e)
    while not est_vide(q):
        ...
        ...
    return ...
```

Résultat

```python
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    reduction_pendant_parcours = False
    while taille(p) >= 3:
        if reduire_triplet_au_sommet(p):
            reduction_pendant_parcours = True
        e = depiler(p)
        empiler(q, e)
    while not est_vide(q):
        e = depiler(q)
        empiler(p, e)
    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.

```python
def jouer(p):
    if parcourir_pile_en_reduisant(...):
        ...(...)
```

Résultat

```python
def jouer(p):
    if parcourir_pile_en_reduisant(p):
        jouer(p)
```

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 apparait 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 dépilant 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

```python
def est_triee(self):
    if not self.est_vide():
        e1 = self.depiler()
        while not self.est_vide():
            e2 = self.depiler()
            if e1 > e2 :
                return False
            e1 = e2
    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 dépilée, 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`. • `q` est vide ; • `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 :

- `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 chaines 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**

```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
}
```

Réponse

```mermaid
%%{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**

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

```
    repertoire = ...
return repertoire
```

Réponse

🐍 **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 ?

Réponse

- 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**

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

Réponse

🐍 **Script Python**

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

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

Réponse

🐍 **Script Python**

```python
def ajoute_repertoire(racine, adr, nom_repertoire):
    repertoire = parcourt(racine, adr)
    repertoire[nom_repertoire] = dict()
```

4.a.

`isinstance` 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**

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

Réponse

🐍 **Script Python**

```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**

```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
```

C9 Les Dictionnaires - Révision

Contenus	Capacités attendues	Commentaires
Dictionnaires, index et clé.	Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	

Ref1, Ref2

1. Révision

- Un dictionnaire est une structure de donnée gardant en mémoire des informations sous la forme (clé, valeur).
- Le but d'un dictionnaire est d'être capable d'accéder rapidement à une valeur à partir de sa clé.

1.1. Créer un dictionnaire

Il y a deux manière de créer un dictionnaire vide.

🐍 **Script Python**

```
diconul = {}
print(diconul)
```

```
{}
```

ou

🐍 **Script Python**

```
diconul = dict()
print(diconul)
```

```
{}
```

On peut créer un dictionnaire

- directement

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
print("dico1: ", dico1)
```

dico1: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- champ par champ

🐍 **Script Python**

```
dico2 = {} # Un dictionnaire vide
dico2["nom"] = "simpson"
dico2["prénom"] = "Homer"
dico2["année"] = 1989
print("dico2: ", dico2)
```

dico2: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- avec la méthode **`update()`**

🐍 **Script Python**

```
dico3 = {}
dico3.update({"nom": "simpson",'prénom': 'Homer','année': 1989})
print("dico3: ", dico3)
```

dico3: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- À partir d'une liste de liste ou d'une liste de tuple

🐍 **Script Python**

```
#liste = [("nom", "simpson"), (["prénom", "Homer"]), (["année", 1989])]
liste = [["nom", "simpson"], ["prénom", "Homer"], ["année", 1989]]
dico4 = dict(liste)
print("dico4: ", dico4)
```

dico4: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- À partir de deux listes (ou tuple)

🐍 **Script Python**

```
liste_keys = ["nom", "prénom", "année"]
liste_values = ["simpson", "Homer", 1989]
# On utilise une compréhension de liste
dico5 = {key: value for key, value in zip(liste_keys,liste_values)}
print("dico5: ", dico5)
```

dico5: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

1.2. Lire et modifier des valeurs des valeurs

🐍 **Script Python**

```
print(dico1["nom"])
```

simpson

🐍 **Script Python**

```
"surnom" in dico1.keys() # teste si "surnom" est une clé
# identique à "surnom" in dico1 # teste si "surnom" est une clé
```

False

🐍 **Script Python**

```
"Homer" in dico1.values() # teste si Homer est une valeur
True
```

True

🐍 **Script Python**

```
dico1["année"]=1990    # On change l'année de naissance
dico1
```

{'nom': 'simpson', 'prénom': 'Homer', 'année': 1990}

1.3. Parcourir un dictionnaire

- Par clés

🐍 **Script Python**

```
for cle in dico1.keys():
    # ou plus simplement for key in dico1:
    print(f"la clé est {cle} et sa valeur est {dico1[cle]}.")


la clé est nom et sa valeur est simpson.
la clé est prénom et sa valeur est Homer.
la clé est année et sa valeur est 1990.
```

- par valeurs

🐍 **Script Python**

```
for valeur in dico1.values():
    print(valeur)

simpson
Homer
1990
```

- Par couple (clé, valeurs)

🐍 **Script Python**

```
for key,value in dico1.items():
    print(f"La clé est {key} et sa valeur {value}")

La clé est nom et sa valeur simpson
La clé est prénom et sa valeur Homer
La clé est année et sa valeur 1990
```

1.4. Suppression d’un élément

- Méthode **`pop()`**
  Cette méthode retourne la valeur

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
val=dico1.pop('nom')
print(dico1)
print(val)

{'prénom': 'Homer', 'année': 1989}
simpson
```

- Méthode **`popitem()`**
  Cette méthode retourne le tuple (clé, valeur).

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
print(dico1)
val=dico1.popitem()
print(dico1)
val=dico1.popitem()
print(val)
print(dico1)


{'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}
{'nom': 'simpson', 'prénom': 'Homer'}
('prénom', 'Homer')
{'nom': 'simpson'}
```

- Instruction **`del`**

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
del dico1["prénom"]
dico1

{'nom': 'simpson', 'année': 1989}
```

1.5. Les méthodes

- **`.copy()`** : Copie d'un dictionnaire car une variable contient l'adresse d'un dictionnaire et non le dictionnaire lui même.

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
dico2 = dico1
dico1['prénom']='lisa'
print(dico1)
print(dico2)

{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
```

Attention comme pour les liste quand on écrit `dico2 = dico1`, c'est l'emplacement de la mémoire du dictionnaire dico1 qui est copié et non le dictionnaire.

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
dico2 = dico1.copy()
dico1['prénom']='lisa'
print(dico1)
print(dico2)

{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
{'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}
```

- **`.pop(key[, default])`** : Supprime key si key est dans le dictionnaire sinon renvoie default.
- **`.popitem()`** : Supprime et renvoie une paire (key, value) du dictionnaire.

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('nom')
print(var)
print(dico1)
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.popitem()
print(var)
print(dico1)


simpson
{'prénom': 'Homer', 'année': 1989}
('année', 1989)
{'nom': 'simpson', 'prénom': 'Homer'}
```

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('surnom')
```

```
print(var)


Traceback (most recent call last):
  File "<input>", line 2, in <module>
KeyError: 'surnom'
```

La clé **surnom** n'existe pas et il y a une erreur.
Pour gérer les erreurs on utilise la condition `try ... except`

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
try:
  var=dico1.pop('surnom')
  print(var)
except:
  print("Il n'y a pas de clé surnom")


Il n'y a pas de clé surnom
```

On peut aussi utiliser le deuxième argument de la méthode `.pop()`

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('surnom',"Il n'y a pas surnom")
print(var)

var=dico1.pop('nom',"Il n'y a pas nom")
print(var)


Il n'y a pas surnom
simpson
```

Dans tous les cas utilisez la fonction `help()` pour en savoir plus

🐍 **Script Python**

```
dico= {"nom": "simpson", "prénom": "Homer", "année": 1989}
help(dico.pop)
```

🐍 **Script Python**

```
  Help on built-in function pop:

  pop(...) method of builtins.dict instance
    D.pop(k[,d]) -> v, remove specified key and return the corresponding value.

    If the key is not found, return the default if given; otherwise,
    raise a KeyError.
```

- `.keys()` Les clés.

- **`.values()`** Les valeurs.
- **`.items()`** (clés, valeurs).

2. TP

Exo

Enoncé **Solution**

Ecrire une fonction `dicoCarre(n)` programme Python qui permet de créer à partir d'un entier n , un dictionnaire formé des entiers de 1 à n et de leurs carrées.

Exemple pour n = 7 le dictionnaire sera de la forme:

🐍 **Script Python**

```
>>>dicoCarre(7)
{1: 1, 2: 4, 3: 9, 4: 16,
5: 25 , 6: 36 , 7: 49}
```

🐍 **Script Python**

```
def creer_dico(n:int)->dict:
    dico={}
    for k in range(1,n+1):
        dico[k]=k**2
    return dico

dcarre=creer_dico(10)
print(dcarre)
```

🐍 **Script Python**

```
assert  creer_dico(7) == {1: 1, 2: 4, 3: 9, 4: 16, 5: 25 , 6: 36 , 7: 49}
```

Exo

Enoncé **Solution**

Écrire une fonction `dicoPaireImpaire(L)` en Python qui prends en paramètre une liste de nombres entiers et qui renvoie un dictionnaire dont les clés sont les entiers de la liste et dont les valeurs sont `'pair'` ou `'impair'` selon la parité du nombre.

🐍 **Script Python**

```python
def dicoPaireImpaire(L):
    dico={}
    for elt in L:
        if elt%2==0:
            dico[elt]='pair'
        else:
            dico[elt]='impair'
    return dico

dicoPaireImpaire([1,5,7,8,9,12,15,456,87])
```

Exo

On considère trois dictionnaires Pythons qui regroupe la totalité du matériels informatiques. Écrire une fonction `concaDico(dico1,dico2,dico3)` Python qui regroupe en concaténant ces trois dictionnaires en un seul et le renvoie.

Exemple:

🐍 **Script Python**

```
dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}
>>>concaDico(dicPC,dicPhone,dicTablette)
{"PC HP": 11 , "PC Acer": 7 , "PC Lenovo": 17 ,
"PC Del": 23,"Phone Sumsung": 22 , "Phone
Iphone": 9 , "Phone Other": 13, "Tablette
Sumsung": 15 , "Tablette Other": 13 }
```

🐍 **Script Python**

```
#Version en une fonction

def concaDico(dico1,dico2,dico3):
    dicTotal={}
    for cle,val in dico1.items():
        if cle in dicTotal:
            dicTotal["PC"+" "+cle]+=val
        else:
            dicTotal["PC"+" "+cle]=val
    for cle,val in dico2.items():
        if cle in dicTotal:
            dicTotal["Phone"+" "+cle]+=val
        else:
            dicTotal["Phone"+" "+cle]=val
    for cle,val in dico3.items():
        if cle in dicTotal:
            dicTotal["Tablette"+" "+cle]+=val
        else:
            dicTotal["Tablette"+" "+cle]=val

    return dicTotal

dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
```

```
dicTablette = {"Sumsung": 15 , "Other": 13}

dicTotal=concaDico(dicPC,dicPhone,dicTablette)

print(dicTotal)
```

`{'PC HP': 11, 'PC Acer': 7, 'PC Lenovo': 17, 'PC Del': 23, 'Phone Sumsung': 22, 'Phone Iphone': 9, 'Phone Other': 13, 'Tablette Sumsung': 15, 'Tablette Other': 13}`

🐍 **Script Python**

```
#Version en deux fonctions

def ajout(dico,dicTotal,nom):
    for cle,val in dico.items():
        if cle in dicTotal:
            dicTotal[nom+" "+cle]+=val
        else:
            dicTotal[nom+" "+cle]=val
    return dicTotal


def concaDico(dico1,dico2,dico3):
    dicTotal={}
    ajout(dico1,dicTotal,"PC")
    ajout(dico2,dicTotal,'Phone')
    ajout(dico3,dicTotal,"Tablette")
    return dicTotal

dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}

dicTotal=concaDico(dicPC,dicPhone,dicTablette)
```

Ecrire une fonction qui regroupe et calcule la somme des apareils par marque.

🐍 **Script Python**

```
dicPC = {"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone = {"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}
```

```python
def ajout(dico,dicTotal):
    for cle,val in dico.items():
        if cle in dicTotal:
            dicTotal[cle] += val
        else:
            dicTotal[cle] = val
    return dicTotal


def sommeDico(dico1,dico2,dico3):
    dicTotal={}
    ajout(dico1,dicTotal)
    ajout(dico2,dicTotal)
    ajout(dico3,dicTotal)
    return dicTotal

dicTotal=sommeDico(dicPC,dicPhone,dicTablette)
```

🐍 **Script Python**

```python
assert dicTotal ==  {'HP': 11, 'Acer': 7, 'Lenovo': 17, 'Del': 23, 'Sumsung': 37, 'Other': 26, 'Iphone': 9}
```

Exo

Enoncé **Solution**

1. Ecrire une fonction `diviseur(a)` qui prend un entrier `a` en argument et renvoie la liste de ses diviseurs.

Exemple:

🐍 **Script Python**

```
>>>diviseur(14)
[1,2,7,14]
```

1. Ecrire une fonction `dicoDiviseur(L)` en Python qui prend une liste de nombres entiers et renvoe un dictionnaire dont les clés sont les entiers saisis et dont les valeurs sont les listes des diviseurs des nombres saisis.

Exemple :

🐍 **Script Python**

```
>>>dicoDiviseur([2 , 7 , 11 , 5 , 3 , 19 , 14 , 9 , 1 , 4])
d = {2 : [1,2] , 7: [1,7] , 14: [1,2,7,14] , 9: [1,3,9] , 11: [1,11] , 5: [1,5]  , 3: [1,3]  , 19: [1,19]  , 1: [1] , 4: [1,2,4] }
```

🐍 **Script Python**

```python
def diviseur(a):
    tab=[]
    for k in range(1,a+1):
        if a%k==0:
            tab.append(k)
    return tab


def dicoDiviseur(L):
    dico={}
    for elt in L:
        rep=diviseur(elt)
        dico[elt]=rep
    return dico

dicoDiviseur([2 , 7 , 11 ,
5 , 3 , 19 , 14 , 9 , 1 , 4])
```

Exo

Enoncé **Solution**

Écrire une fonction `dicoInverse(texte)` en Python qui prend un texte en argument et renvoye un dictionnaire dont les clés sont les mots du texte saisi et les valeurs sont les inverses des mots qui composent le texte.

Exemple :

🐍 **Script Python**

```
>>>dicoInverse("Python est un facile" )
{'Python': 'nohtyp' , 'est': 'tse' , 'facile': 'elicaf'}
```

🐍 **Script Python**

```python
def decomposPhrase(phrase):
    mot=[phrase[0]]
    pos=0
    for i in range(1,len(phrase)):
        if phrase[i]==" ":
            mot.append(phrase[i])
            pos+=1
        else:
            mot[pos]+=phrase[i]
    return mot

print(decomposPhrase("Python est un facile" ))

def inverse_mot(mot):
    mot_inv=""
    for lettre in mot:
        mot_inv=lettre.lower()+mot_inv
    return mot_inv

print(inverse_mot("Python"))

def dicoInverse(phrase):
    dico={}
    for mot in decomposPhrase(phrase):
```

```
        dico[mot]=inverse_mot(mot)
    return dico

dicoInverse("Python est un facile" )
```

Exo

Enoncé Solution

Vous trouverez sur https://www.liste-de-mots.com/ liste des points de chaque lettres de l'alphabet.

Le module ***string*** permet d'obtenir les lettres de l'alphabet.

🐍 **Script Python**

```
import string
print(string.ascii_uppercase)
print(list(string.ascii_uppercase))
```

📋 **Texte**

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z']
```

1. Créer un dictionnaire `dico_scrabble` dont les clés sont lettres de l'alphabet et les valeurs sont 1
2. Copier et modifier alors le dictionnaire `dico_scrabble` dont les clés sont lettres de l'alphabet et les valeurs sont les points de ces lettres au scrabble.
3. Créer une fonction **`points_mot(mot:str)->int`** qui renvoie le nombre de points d'un mot

1.

🐍 **Script Python**

```
dico_scrabble = dict()  # ou {}
for i in range (65, 65+26):
    dico_scrabble[chr(i)] = 1
```

🐍 **Script Python**

```
assert dico_scrabble == {'A': 1, 'B': 1, 'C': 1, 'D': 1, 'E': 1, 'F': 1, 'G': 1, 'H': 1, 'I': 1, 'J': 1, 'K': 1, 'L': 1,
                         'M': 1, 'N': 1, 'O': 1, 'P': 1, 'Q': 1, 'R': 1, 'S': 1, 'T': 1, 'U': 1, 'V': 1, 'W': 1, 'X': 1, 'Y': 1, 'Z': 1}
```

1.

🐍 **Script Python**

```
dico_scrabble = {'A': 1, 'B': 3, 'C': 3, 'D': 2, 'E': 1, 'F': 4, 'G': 2, 'H': 4, 'I': 1, 'J': 8, 'K': 10, 'L': 1, 'M': 2,
                 'N': 1, 'O': 1, 'P': 3, 'Q': 8, 'R': 1, 'S': 1, 'T': 1, 'U': 1, 'V': 4, 'W': 10, 'X': 10, 'Y': 10, 'Z': 10, '*': 0}
```

🐍 **Script Python**

```
assert  dico_scrabble['A'] == 1
assert  dico_scrabble['Z'] == 10
assert  dico_scrabble['*'] == 0
```

1.

🐍 **Script Python**

```
def points_mot(mot:str)->int:
    pt = 0
    for i in mot:
        pt += dico_scrabble[i.upper()]
    return pt
```

🐍 **Script Python**

```
assert points_mot('YACK') == 24
assert points_mot('yack') == 24
assert points_mot('NERUDA') == 7
assert points_mot('Neruda') == 7
```

Exo

Enoncé Solution

On considère le dictionnaire suivant dont les clés sont les noms des élèves et les valeurs des clés sont les moyennes générales obtenues en passant l'examen final:

🐍 **Script Python**

```
lyceens =
{"lyceen_1": 13,
"lyceen_2": 17,
"lyceen_3": 9,
"lyceen_4": 15,
"lyceen_5": 8,
      "lyceen_6": 14,
"lyceen_7": 16,
"lyceen_8": 12,
"lyceen_9": 13,
"lyceen_10": 15,
      "lyceen_11": 14,
"lyceen_112": 9,
"lyceen_13": 10,
"lyceen_14": 12,
"lyceen_15": 13,
      "lyceen_16": 7,
"lyceen_17": 12,
"lyceen_18": 15,
"lyceen_19": 9,
"lyceen_20": 17}
```

Écrire un programme Python qui partitionne ce dictionnaire en deux sous dictionnaires:

1. `etudiantAdmis` dont les clés sont les étudiants admis et

les valeurs des clés sont les moyennes obtenues (moyenne supérieurs ou égales à 10).

2. `etudiantNonAdmis` dont les clés sont les étudiants non admis et les valeurs des clés sont les moyennes obtenues (moyenne strictement inférieur à 10).

🐍 **Script Python**

```
# on crée  deux dictionnaires vides un pour les admis et l'autre pour les non admis
# on crée  deux dictionnaires vides un pour les admis et l'autre pour les non admis
etudiantAdmis = dict()
etudiantNonAdmis = dict()
for k,v in lyceens.items():
    if v < 10:
etudiantNonAdmis[k] = v
    else:
etudiantAdmis[k] = v
```

🐍 **Script Python**

```
assert  etudiantAdmis == {'lyceen_1': 13, 'lyceen_2': 17,
```

```
'lyceen_4': 15,
'lyceen_6': 14,
'lyceen_7': 16,
'lyceen_8': 12,
'lyceen_9': 13,
'lyceen_10': 15,
'lyceen_11': 14,
'lyceen_13': 10,
'lyceen_14': 12,
'lyceen_15': 13,
'lyceen_17': 12,
'lyceen_18': 15,
'lyceen_20': 17}
assert
etudiantNonAdmis
== {'lyceen_3': 9,
'lyceen_5': 8,
'lyceen_112': 9,
'lyceen_16': 7,
'lyceen_19': 9}
```

2.

Vo es expériences ci-dessus : le temps de recherche dans le dictionnaire est pr du nombre d'entrées dans ce dictionnaires, car en multipliant le nombre de co est resté pratiquement identique alors que dans le cas de la recherche dans ur portionnel à la longueur du tableau.

Le ne structure de données optimisée pour la recherche sur les clés.

Le fonction de hachage.

3. Complément hash (hors programme)

La notion de Hachage est omniprésente en informatique et est au cœur du fonctionnement des dictionnaires. Le hachage est un mécanisme permettant de transformer la clé en un nombre unique permettant l'accès à la donnée, un peu à la manière d'un indice dans un tableau.

3.1. Définition d'une fonction de hachage

Une fonction de hachage est une fonction qui va calculer une empreinte unique à partir de la donnée fournie en entrée. Elle doit respecter les règles suivantes :

- La longueur de l'empreinte (valeur retournée par la fonction de hachage) doit être toujours la même, indépendamment de la donnée fournie en entrée.
- Connaissant l'empreinte, il ne doit pas être possible de reconstituer la donnée d'origine
- des données différentes doivent donner dans la mesure du possible des empreintes différentes.
- des données identiques doivent donner des empreintes identiques.

3.2. Quelques utilisations du hachage

L'utilisation la plus courante est le stockage des mots de passe dans un système informatique un peu sécurisé.
En effet, lorsqu'on crée un compte sur un service en ligne, le mot de passe ne doit pas être stocké en clair, une empreinte est générée afin que si le service est piraté et que les comptes sont dérobés, il ne soit pas possible de reconstituer le mot de passe à partir de l'empreinte. Voici un exemple de fonctionnement d'une fonction de hachage.

Nous utiliserons le hachage **sha256**.

🐍 **Script Python**

```python
from hashlib import sha256

h1=sha256(b"Simpson Homer")   # b"Simpson Homer" est un objet de type  bytes
print (h1.hexdigest())
```

```
8d167108dff6b68905c7154c04152a1218511d6b84bc839395e7d653222912f3
```

🐍 **Script Python**

```python
h2=sha256(b"Simpson Homer")
print(h2.hexdigest())
```

```
8d167108dff6b68905c7154c04152a1218511d6b84bc839395e7d653222912f3
```

🐍 **Script Python**

```python
h3=sha256(b"Simpson  Homer")
print(h3.hexdigest())
```

```
5c9fe022bd1d6dd652e32e0cd871920f7212124cba31532442211599fe634adf
```

Ce n'est pas une valeur au hasard. Un hash donne une même valeur pour un même texte.

Voici un texte qui vous permettra de comprendre le principe des fonctions de hachages : c'est quoi le hachage.

Pour avoir quelques idées sur le principe des tables de hachages, je vous recommande le visionnage de ces deux vidéos:

On constate bien sur cet exemple que :

- un petit changement dans la valeur d'entrée fournit une empreinte totalement différentes
- toutes les empreintes ont la même longueur 64 caractères hexadécimal donc 32 octets et donc \(32\times8=256\) bit.

Video

Vous avez déjà compris que l'algorithme de recherche dans une table de hachage a une complexité \(O(1)\) (le temps de recherche ne dépend pas du nombre d'éléments présents dans la table de hachage), alors que la complexité de l'algorithme de recherche dans un tableau non trié est \(O(n)\).

Comme l'implémentation des dictionnaires s'appuie sur les tables de hachage, on peut dire que l'algorithme de recherche d'un élément dans un dictionnaire a une complexité $O(1)$ alors que l'algorithme de recherche d'un élément dans un tableau non trié a une complexité $O(n)$.

C10 Les arbres

1. Activités

1.1. ■ Activité 1 : *Vocabulaire*

1. Après execution d'une commande dans un terminal Linux on a obtenu l'affichage ci-dessous :

```
.
├── Maths
├── NSI
│   ├── Cours
│   │   └── Systèmes
│   ├── Exercices
│   │   └── Python
│   ├── Histoire
│   └── Projet
└── SVT
```

   a. De quelle commande s'agissait-il ? (Rechercher la commande sur internet) Quelle est sa traduction en français ?

   b. Que sont `Maths` et `NSI` pour le système d'exploitation ?

   c. Un point (**.**) figure en haut du schéma, quelle en est la signification ?

   d. Que signifie le lien entre `NSI` et `Projet` ?

2. Le schéma ci-dessus est un exemple d'**arbre** (on parle d'ailleurs de l'arborescence des dossiers dans un système d'exploitation). Un arbre est constitué de **noeuds** et un lien entre deux noeuds s'appelle une **arête**.

   a. Citer trois noeuds de l'arbre ci-dessus et donner un exemple d'arête.

   b. Quel noeud est le répertoire parent de `Systèmes` ? On dira dans le vocabulaire des arbres que le noeud `Systèmes` est un **fils** de ce noeud.

   c. Citer tous les fils du noeud `NSI`.

   d. Dans un arbre, un seul et unique noeud n'est le fils de personne, on l'appelle la **racine** de l'arbre. De qui s'agit-il ici ?

   e. Nommer les noeuds n'ayant aucun fils (on les appelle **feuilles** de l'arbre).

   f. Une **branche** est une suite (finie) de noeuds depuis la racine vers une feuille. Donner une branche de cette arbre constituée de trois noeuds.

   g. Donner la **taille** de cet arbre (c'est à dire son nombre de noeuds).

   h. Donner l'**arité** de cet arbre, c'est à dire le nombre maximal de fils qu'un noeud peut avoir.

   i. Donner la **hauteur** de cet arbre c'est à dire le nombre maximal de noeuds dans une branche.

Attention

La hauteur d'un arbre est parfois définie comme le plus grand nombre d'arêtes dans une branche. Dans les sujets de bac faire attention à la définition utilisée qui est normalement donnée dans l'énoncé.

1.2. ■ Activité 2 : *Arbre binaire*

Arbre binaire

Un **arbre binaire** est un arbre dont tous les noeuds ont au plus deux fils.

```
graph TD
A["A"] --> B["B"]
A --> C["C"]
subgraph sous arbre droit
    C --> W[" "]
    C --> G["G"]
    W --> W1[" "]
    W --> W2[" "]
    G --> J["J"]
    G --> K["K"]
end
subgraph sous arbre gauche
    B --> D["D"]
    B --> F["F"]
    D --> E["E"]
    D --> H["H"]
    F --> I["I"]
    F --> T[" "]
end
style W opacity:0
linkStyle 2 stroke:#FFFFFF,stroke-width:0px
linkStyle 4 stroke:#FFFFFF,stroke-width:0px
linkStyle 5 stroke:#FFFFFF,stroke-width:0px
linkStyle 13 stroke:#FFFFFF,stroke-width:0px
style W1 opacity:0
style W2 opacity:0
style T opacity:0
style W opacity:0
```

1. Quelle est la taille de cet arbre ?
2. Listez les noeuds internes puis les feuilles.
3. Quelle est la hauteur de cet arbre ?
4. Quels sont les fils du noeud B ?
5. Quel est le sous-abre gauche du noeud B ?

6. Quel est le sous-abre gauche du noeud C ?

1.3. ■ Activité 3 : *Relation entre la hauteur et la taille d'un arbre binaire*

1. On note h la hauteur d'un arbre binaire et n sa taille, on suppose dans la suite que $n \geq 2$. En rappelant les définitions de la hauteur et de la taille d'un arbre, justifier que $n \geq h$.
2. On numérote les noeuds d'un arbre binaire de la suivante :
   - la racine porte le numéro 1,
   - le fils gauche d'un noeud porte le numéro de son père suivi d'un 0,
   - le fils droit d'un noeud porte le numéro de son père suivi d'un 1.

On a entamé la numérotation de l'arbre binaire ci-dessous, recopier cet arbre et compléter cette numérotation.

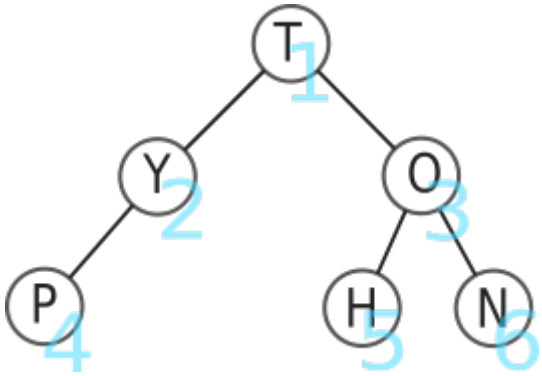
```
graph TD
A["A : 1"] --> B["B : 10"]
A --> C["C : 11"]
B --> D["D : ..."]
B --> E["E : 101"]
C --> V1[" "]
C --> F["F: ..."]
V1 --> V2[" "]
V1 --> V3[" "]
D --> G["G: ..."]
D --> H["H: ..."]
F --> I["I: ..."]
F --> V2[" "]
E --> V4[" "]
E --> V5[" "]
style V1 opacity:0;
linkStyle 4 stroke:#FFFFFF,stroke-width:0px
style V2 opacity:0;
linkStyle 6 stroke:#FFFFFF,stroke-width:0px
style V3 opacity:0;
linkStyle 7 stroke:#FFFFFF,stroke-width:0px
style V2 opacity:0;
linkStyle 11 stroke:#FFFFFF,stroke-width:0px
style V4 opacity:0;
linkStyle 12 stroke:#FFFFFF,stroke-width:0px
style V5 opacity:0;
linkStyle 13 stroke:#FFFFFF,stroke-width:0px
```

1. Justifier que sur un niveau donné de l'arbre tous les numéros de noeud ont le même nombre de caractères.
2. En déduire en fonction de h, le nombre de caractères formant le numéro des feuilles
3. En utilisant vos connaissances sur la numérotation binaire d'un entier positif, prouver que $n \leq 2^h - 1$.

1.4. ■ Activité 4 : *Parcours en largeur (*BFS : Breadth First Search*) d'abord d'un arbre binaire*

Parcours en largeur d'abord (BFS)

Le parcours en largeur d'abord est un parcours étage par étage (de haut en bas) et de gauche à droite.



L'ordre des lettres parcourues est donc T-Y-O-P-H-N.

> Un parcours en largeur d'abord n'est pas récursif.

1. Indiquez dans quel ordre les noeuds sont explorés dans le cas d'un parcours en largeur de l'arbre `A1` suivant.

```
graph TD
A["2"] --> B["8"]
A --> C["9"]
B --> D["4"]
B --> E["5"]
C --> V1[" "]
C --> F["3"]
V1 --> V2[" "]
V1 --> V3[" "]
style V1 opacity:0;
linkStyle 4 stroke:#FFFFFF,stroke-width:0px
style V2 opacity:0;
linkStyle 6 stroke:#FFFFFF,stroke-width:0px
style V3 opacity:0;
linkStyle 7 stroke:#FFFFFF,stroke-width:0px
style V2 opacity:0;
```

2. Indiquez dans quel ordre les noeuds sont explorés dans le cas d'un parcours en largeur de l'arbre `A2` suivant.

```
graph TD
A["1"] --> B["2"]
A --> C["5"]
B --> D["3"]
B --> E["4"]
```

```
C --> V1["6"]
C --> F[" "]
V1 --> V2["7"]
V1 --> V3["8"]
D --> G[" "]
D --> H[" "]
F --> I[" "]
F --> V6[" "]
E --> V4[" "]
E --> V5[" "]
style F opacity:0;
linkStyle 5 stroke:#FFFFFF,stroke-width:0px
style G opacity:0;
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
style H opacity:0;
linkStyle 9 stroke:#FFFFFF,stroke-width:0px
style I opacity:0;
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
style V6 opacity:0;
linkStyle 11 stroke:#FFFFFF,stroke-width:0px
style V4 opacity:0;
linkStyle 12 stroke:#FFFFFF,stroke-width:0px
style V5 opacity:0;
linkStyle 13 stroke:#FFFFFF,stroke-width:0px
```

1.5. ■ Activité 5 : *Parcours en profondeur d'un arbre binaire*

Note

Dans le cas où on explore complètement l'un des deux sous-arbres avant le second on parle d'un **parcours en profondeur**. On utilise le terme *profondeur* car dans ce cas on tente toujours de visiter le noeud le plus éloigné de la racine à condition qu'il soit le fils d'un noeud déjà visité.
On distingue trois ordres particuliers pour explorer en profondeur les sous-arbres gauche, droit et la racine du noeud courant :

- **ordre préfixe** : le noeud courant est traité, puis son sous-arbre gauche et son sous-arbre droit.
- **ordre infixe** : le noeud courant est traité entre son sous-arbre gauche et son sous-arbre droit.
- **ordre suffixe** : le noeud courant est traité après son sous-arbre gauche et son sous-arbre droit.

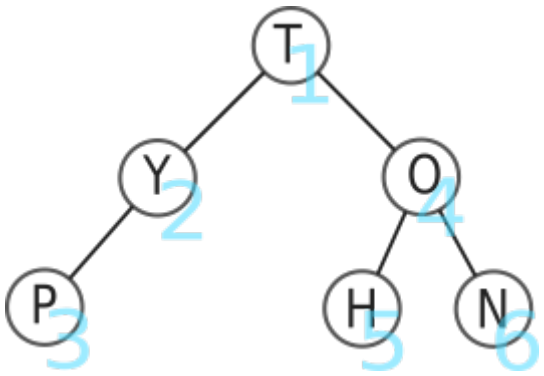
1.5.1. Parcours préfixe

Le parcours **préfixe** est un parcours **en profondeur d'abord**.

Méthode du parcours préfixe :

(parfois aussi appelé *préordre*)

- Chaque nœud est visité avant que ses fils le soient.
- On part de la racine, puis on visite son fils gauche (et éventuellement le fils gauche de celui-ci, etc.) avant de remonter et de redescendre vers le fils droit.



L'ordre des lettres parcourues est donc T-Y-P-O-H-N.

Vidéo explicative d'un parcours préfixe

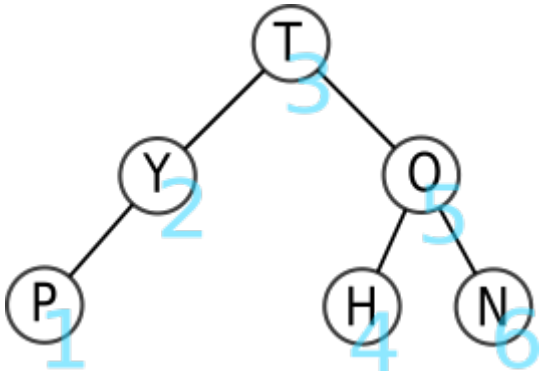
1.5.2. Parcours infixe

Parcours infixe

Le parcours **infixe** est aussi un parcours en profondeur d'abord.

Méthode du parcours infixe : (parfois aussi appelé *en ordre*)

- Chaque nœud est visité **après son fils gauche mais avant son fils droit**.
- On part donc de la feuille la plus à gauche et on remonte par vagues sucessives. Un nœud ne peut pas être visité si son fils gauche ne l'a pas été.



L'ordre des lettres parcourues est donc P-Y-T-H-O-N.

Vidéo explicative d'un parcours infixe

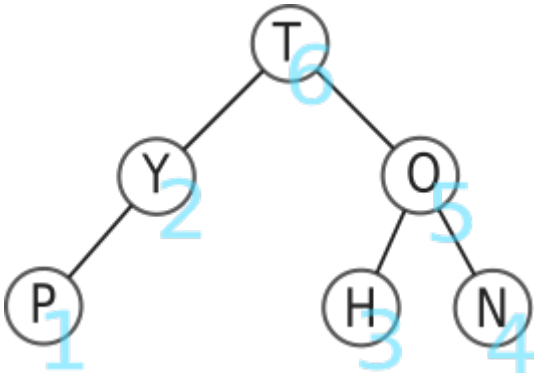
1.5.3. Parcours suffixe

Parcours suffixe

Le parcours **suffixe** est aussi un parcours en profondeur d'abord.

Méthode du parcours suffixe : (parfois aussi appelé *post ordre*)

- Chaque nœud est visité **après ses fils le soient**.
- On part donc de la feuille la plus à gauche, et on ne remonte à un nœud père que si ses fils ont tous été visités.



L'ordre des lettres parcourues est donc P-Y-H-N-O-T.

1.5.4. Comment ne pas se mélanger entre le pré / in / post fixe ?

- *pré* veut dire *avant*
- *in* veut dire *au milieu*

- *post* veut dire *après*

Ces trois mots-clés parlent de la place du **père** par rapport à ses fils.
Ensuite, il faut toujours se souvenir qu'on traite le fils gauche avant le fils droit.

- préfixe : le père doit être le premier par rapport à ses fils.
- infixe : le père doit être entre son fils gauche (traité en premier) et son fils droit.
- postfixe : le père ne doit être traité que quand ses deux fils (gauche d'abord, droite ensuite) l'ont été.

Un parcours préfixe commencera toujours par la racine, alors qu'un parcours postfixe finira toujours par la racine. Dans un parcours infixe, la racine sera «au milieu» (pas nécessairement parfaitement).

1.5.5. Exercices d'application

Exercice

Énoncé | **Corr. largeur** | **Corr. préfixe** | **Corr. infixe** | **Corr. postfixe**

```
graph TD
    A(1) --> B(2)
    B --> C(4)
    B --> D(5)
    A --> E(3)
    D --> F(7)
    D --> G(8)
    E --> H(" ")
    E --> I(6)
    I --> J(9)
    I --> K(" ")

linkStyle 6 stroke-width: 0px;
    style H opacity:0;

linkStyle 9 stroke-width: 0px;
    style K opacity:0;
```

Donner le rendu de chaque parcours :

1. Parcours en largeur

2. Parcours préfixe
3. Parcours infixe
4. Parcours postfixe

largeur : 1 2 3 4 5 6 7 8 9

préfixe : 1 2 4 5 7 8 3 6 9

infixe : 4 2 7 5 8 1 3 9 6

postfixe : 4 7 8 5 2 9 6 3 1

Exercice

Énoncé | **Corr. largeur** | **Corr. préfixe** | **Corr. infixe** | **Corr. postfixe**

```
graph TD
    A(9) --> B(8)
    B --> C(6)
    B --> D(2)
    A --> E(7)
    D --> F(1)
    D --> G(" ")
    E --> H(" ")
    E --> I(5)
    I --> J(4)
    I --> K(3)

linkStyle 5 stroke-width: 0px;
    style G opacity:0;

linkStyle 6 stroke-width: 0px;
    style H opacity:0;
```

Donner le rendu de chaque parcours :

1. Parcours en largeur

2. Parcours préfixe
3. Parcours infixe
4. Parcours postfixe

largeur : 9 8 7 6 2 5 1 4 3

préfixe : 9 8 6 2 1 7 5 4 3

infixe : 6 8 1 2 9 7 4 5 3

postfixe : 6 1 2 8 4 3 5 7 9

2. 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.

3. Exercices

3.1. ■ Exercice 1 : *DOM*

On considère le code html suivant :

HTML

```html
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <title>Exemple</title>
</head>
<body>
   <h1> Un titre </h1>
   <p> Bonjour ! </p>
   <table>
      <tr>
```

```
        <td> Case 1</td>
        <td> Case 2</td>
      </tr>
    </table>
  </body>
  </html>
```

Ce document peut se représenter par un arbre :

```
graph TD
HTML["html"] --> HEAD["head"]
HTML["html"] --> BODY["body"]
HEAD --> META["meta"]
HEAD --> TITLE["title"]
BODY --> H1["h1"]
BODY --> P["p"]
BODY --> TABLE["table"]
TABLE --> TR["tr"]
TR --> TD1["td"]
TR --> TD2["td"]
```

1. Que représentent les noeuds de cet arbre ?
2. Que signifie du point de vue du html une arête de cet arbre ?
3. Quelle est la racine de cet arbre ?
4. Donner les feuilles de cet arbre.
5. Donner une branche de cet arbre.
6. Donner l'arité de cet arbre.
7. Donner la hauteur de cet arbre.

3.2. ■ Exercice 2 : *Vocabulaire sur les arbres*

On considère l'arbre ci-dessous :

```
graph TD
A["A"] --> L["L"]
A --> G["G"]
L --> O["O"]
L --> R["R"]
G --> V1[" "]
G --> I["I"]
O --> T["T"]
O --> H["H"]
R --> M["M"]
R --> E["E"]
I --> S["S"]
I --> V2[" "]
style V1 opacity:0;
linkStyle 4 stroke:#FFFFFF,stroke-width:0px
style V2 opacity:0;
linkStyle 11 stroke:#FFFFFF,stroke-width:0px
```

1. Justifier qu'il s'agit d'un arbre binaire. Quelle est sa taille ?

2. Quelle est sa racine ? Nommer les feuilles.
3. Donner une branche de cet arbre.
4. Donner la hauteur de cet arbre.
5. Donner une branche de longueur 3 dans cet arbre
6. On considère le sous arbre gauche de la racine, quel est l'arité de chaque noeud ? Que peut-on dire du sous arbre gauche ?
7. Même question pour le sous arbre droit de la racine.

3.3. ■ Exercice 3 : *Représenter un arbre binaire*

1. Dessiner tous les arbres binaires à 3 noeuds. Attention à bien distinguer le fils droit du fils gauche.
2. Dessiner un arbre binaire de taille 4 et de profondeur 4.
3. Dessiner un arbre binaire de taille 4 et de profondeur 3.
4. Quelle est la taille maximale d'un arbre binaire de hauteur 10 ? Justifier

> **Aide**
>
> On pourra utiliser sans justification, la formule vue en cours qui permet d'encadrer la taille d'un arbre en fonction de sa hauteur.

3.4. ■ Exercice 4 : *Représenter un arbre binaire par un triplet*

On rappelle qu'on peut définir un arbre binaire de façon récursive, en effet, un arbre binaire est soit vide (noté Δ), soit un triplet $(etiquette,sag,sad)$ où sag (pour sous arbre gauche) et sad (pour sous arbre droit) sont eux-mêmes des arbres binaires.

On considère les arbres suivants :

- $A = (7,\Delta,\Delta)$
- $B = (2,(3,\Delta,(9,\Delta,\Delta)),(10,\Delta,\Delta))$
- $C = (4,(1,(8,\Delta,\Delta),(7,\Delta,\Delta)),(3,(6,\Delta,\Delta),(5,\Delta,\Delta)))$

Pour chacun de ces arbres :

1. Faire un schéma pour le représenter.
2. Donner sa taille, sa hauteur, son nombre de feuilles.

Sujet BAC 10 : Les arbres - Partie 1

1. 2022, Polynésie, J1, Ex. 5

BAC : Construction d'arbres binaires

On manipule ici les arbres binaires avec trois fonctions :

- `est_vide(A)` renvoie `True` si l'arbre binaire `A` est vide, `False` s'il ne l'est pas ;
- Pour un arbre binaire `A` **non vide** :
  - `sous_arbre_gauche(A)` renvoie le sous-arbre à gauche de `A` ;
  - `sous_arbre_droite(A)` renvoie le sous-arbre à droite de `A`.

L'arbre binaire renvoyé par les fonctions `sous_arbre_gauche` et `sous_arbre_droite` peut éventuellement être l'arbre vide.

On définit la hauteur d'un arbre binaire de la façon suivante :

- la hauteur de l'arbre vide est \(0\) ;
- sinon, la hauteur est égale à \(1 + M\), où \(M\) est la plus grande des hauteurs de ses sous-arbres (à gauche et à droite).

1.a. Donner la hauteur de l'arbre ci-dessous.

```
graph TD
   N0( ) --> N2( )
   N0   --> N1( )
   N2  --> N5( )
   N2  --> N6( )
   linkStyle 3 stroke-width:0px;
   style N6 opacity:0;
```

Réponse

Avec cette définition, la hauteur de cet arbre binaire est 3.

1.b. Dessiner sur la copie un arbre binaire de hauteur \(5\).

Réponse

Avec cette définition, voici un arbre binaire de hauteur \(5\).

```
graph TD
    N0( ) --> N1( )
    N0   --> N2( )
    N1  --> N3( )
    N1  --> N4( )
    N2  --> N5( )
    N2  --> N6( )
    N4  --> N7( )
    N4  --> N8( )
    N7  --> N9( )
    N7  --> N10( )
    linkStyle 7 stroke-width:0px;
    style N8 opacity:0;
    linkStyle 4 stroke-width:0px;
    style N5 opacity:0;
```

La hauteur d'un arbre est calculée par l'algorithme récursif suivant :

```
1   Algorithme hauteur(A) :
2       si A vide :
3           renvoyer ...
4       sinon:
5           renvoyer 1 + max(
6               hauteur(sous_arbre_gauche(A)),
7               ...
8           )
```

2. Recopier sur la copie les lignes 3 et 7 en complétant les points de suspension.

Réponse

```
Algorithme hauteur(A) :
    si A vide :
        renvoyer 0
    sinon:
        renvoyer 1 + max(
            hauteur(sous_arbre_gauche(A)),
            hauteur(sous_arbre_droite(A)),
        )
```

On considère un arbre binaire `R` dont on note `G` le sous-arbre à gauche et `D` le sous-arbre à droite. On suppose que `R` est de hauteur \(5\) et `G` de hauteur \(3\).

3.a. Justifier le fait que `D` n'est pas l'arbre vide et déterminer sa hauteur.

Réponse

Si `D` est égal à l'arbre vide, alors la hauteur de `R` est égale à `1 + hauteur(G)` qui est égal à \(1+3=4\), or `R` est de hauteur \(5\). Contradiction.

Ainsi `D` n'est pas l'arbre vide.

Dans ce cas `1 + max(hauteur(G), hauteur(D))` est égal à \(4\). D'où

- `1 + max(3, hauteur(D))` est égal à \(5\).
- `max(3, hauteur(D))` est égal à \(4\).
- `hauteur(D)` est égal à \(4\).

3.b. Illustrer cette situation par un dessin.

Réponse

- Cet arbre est de hauteur \(5\),
- son sous arbre à gauche est de hauteur \(3\),
- son sous arbre à droite est de hauteur \(4\).

```
graph TD
   N0( ) --> N1( )
   N0   --> N2( )
   N1  --> N3( )
   N1  --> N4( )
   N2  --> N5( )
   N2  --> N6( )
   N4  --> N7( )
   N4  --> N8( )
   N6  --> N9( )
   N6  --> N10( )
   N9  --> N11( )
   N9  --> N12( )
   linkStyle 7 stroke-width:0px;
   style N8 opacity:0;
   linkStyle 4 stroke-width:0px;
   style N5 opacity:0;
```

Soit un arbre binaire non vide de hauteur `h`. On note `n` le nombre de nœuds de cet arbre. On admet que \(h \leqslant n \leqslant 2^h - 1\).

4.a. Vérifier ces inégalités sur l'arbre binaire de la question **1.a.**.

Réponse

Dans la question **1.a.**, l'arbre binaire possède $n = 4$ nœuds et a une hauteur $h = 3$.

On a bien $3 \leqslant 4 \leqslant 2^3 - 1$ qui s'écrit aussi $3 \leqslant 4 \leqslant 7$

4.b. Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant `h` nœuds.

Réponse

Il **suffit**, par exemple, de construire un arbre binaire où pour chaque nœud, soit le sous arbre à gauche est vide, soit celui à droite.

- Cela peut être toujours celui à gauche qui est vide, on parle alors d'arbre peigne à droite.
- Cela peut être toujours celui à droite qui est vide, on parle alors d'arbre peigne à gauche.

4.c. Expliquer comment construire un arbre binaire de hauteur `h` quelconque ayant $2^h - 1$ nœuds.

Indication : $2^h - 1 = 1+2+4+...+2^{h-1}$.

Réponse

Il **faut**, dans ce cas, construire un arbre binaire complet ; les sous-arbres vides sont tous à la même profondeur.

L'objectif de la fin de l'exercice est d'écrire le code d'une fonction `fabrique(h, n)` qui prend comme paramètres deux nombres entiers positifs `h` et `n` tels que $h < n < 2^h - 1$, et qui renvoie un arbre binaire de hauteur `h` à `n` nœuds.

Pour cela, on utilise les deux fonctions suivantes :

- `arbre_vide()`, qui renvoie un arbre vide ;
- `arbre(gauche, droite)` qui renvoie l'arbre fils à `gauche` et le fils à `droite`.

5. Recopier sur la copie l'arbre binaire ci-dessous et numéroter ses nœuds de 1 en 1 en commençant à 1, en effectuant un parcours en profondeur préfixe.

```
graph TD
   N0( ) --> N2( )
   N0   --> N1( )
   N1 --> N3( )
   N1 --> N4( )
   N2  --> N5( )
   N2  --> N6( )
   linkStyle 5 stroke-width:0px;
   style N6 opacity:0;
   N3 --> N7( )
   N3 --> N8( )
   N4 --> N9( )
   N4 --> N10( )
```

Réponse

```
graph TD
   N0(1) --> N2(9)
   N0   --> N1(2)
   N1 --> N3(3)
   N1 --> N4(6)
   N2  --> N5(10)
   N2  --> N6( )
   linkStyle 5 stroke-width:0px;
   style N6 opacity:0;
   N3 --> N7(4)
   N3 --> N8(5)
   N4 --> N9(7)
   N4 --> N10(8)
```

La fonction `fabrique` ci-dessous a pour but de répondre au problème posé.

🐍 **Script Python**

```python
def fabrique(h, n):
    if n == 0:
        return ...
    else:
        reste = n - 1                    # 1 pour la racine du sous-arbre
        n_gauche = min(reste, ...)       # le plus possible à gauche
        n_droite = reste - n_gauche      # la suite à droite
        return arbre(
            fabrique(..., n_gauche),
            ...
        )
```

6. Recopier sur la copie les lignes 3, 6 et 9 en complétant les points de suspension.

Réponse

```python
def arbre_vide():
    return []

def arbre(gauche, droite):
    return [gauche, droite]

def fabrique(h, n):
    if n == 0:
        return arbre_vide()
    else:
        reste = n - 1                    # 1 pour la racine du sous-arbre
        n_gauche = min(reste, 2**(h-1) - 1)  # le plus possible à gauche
        n_droite = reste - n_gauche         # la suite à droite
        return arbre(
            fabrique(h-1, n_gauche),
            fabrique(h-1, n_droite)
        )

def taille(arbre):
    if arbre == []:
        return 0
    else:
        gauche, droite = arbre
        return 1 + taille(gauche) + taille(droite)

def hauteur(arbre):
    if arbre == []:
        return 0
    else:
        gauche, droite = arbre
        return 1 + max(hauteur(gauche), hauteur(droite))


for h in range(4):
    print("Hauteur", h)
    for n in range(h, 2**h):
        n_sav = n
        arbre_hn = fabrique(h, n)
        print(arbre_hn)
        assert (h, n_sav) == (hauteur(arbre_hn), taille(arbre_hn))
    print()
```

2. 2022, Métropole, J1, Ex. 4

BAC : Somme des valeurs d'un arbre binaire

Cet exercice traite du calcul de la somme d'un arbre binaire. Cette somme consiste à additionner toutes les valeurs numériques contenues dans les nœuds de l'arbre.

L'arbre utilisé dans les parties A et B est le suivant :

```
graph TD
   N0(3) --> N1(6)
   N0    --> N2(2)
   N1    --> N4(7)
   N1    --> N5(4)
   N2    --> N6(9)
   N2    --> N7(1)
```

2.1. Partie A : Parcours d'un arbre

1. Donner la somme de l'arbre précédent. Justifier la réponse en explicitant le calcul qui a permis de l'obtenir.

Réponse

La somme de l'arbre est \(3 + 6 + 2 + 7 + 4 + 9 + 1 = 32\)

2. Indiquer la lettre correspondante aux noms « racine », « feuille », « nœud », « SAG » (Sous Arbre à Gauche) et « SAD » (Sous Arbre à Droite). Chaque lettre A, B, C, D et E ne devra être utilisée qu'une seule fois.

```
flowchart TD
   subgraph G0 ["Arbre avec des lettres à associer
                &nbs

      N0(3) --> N1(6)
      N0    --> N2(2)
      subgraph G2 [ ]
         N2    --> N6(9)
         N2    --> N7(1)
      end
      subgraph G3 [ ]
         N1    --> N4(7)
         N1    --> N5(4)
      end

      A>A] -.- N0
      B>B] -.- N1
      C>C] -.- N4
      G3 -.- D>D]
      G2 -.- E>E]
   end
```

Réponse

Lettre	Nom
A	racine
B	nœud
C	feuille
D	SAG
E	SAD

3. Parmi les quatre propositions A, B, C et D ci-dessous, donnant un parcours en largeur de l'arbre, une seule est correcte. Indiquer laquelle.

- **Proposition A** : 7 - 6 - 4 - 3 - 9 - 2 - 1
- **Proposition B** : 3 - 6 - 7 - 4 - 2 - 9 - 1
- **Proposition C** : 3 - 6 - 2 - 7 - 4 - 9 - 1
- **Proposition D** : 7 - 4 - 6 - 9 - 1 - 2 - 3

Réponse

On lit chaque niveau, de la gauche vers la droite.

- 3, puis
- 6, 2, puis
- 7, 4, 9, 1.

La proposition **C** est la bonne.

4. Écrire en langage Python la fonction `somme` qui prend en paramètre une liste de nombres et qui renvoie la somme de ses éléments.

Exemple : `somme([1, 2, 3, 4])` est égale à 10.

Réponse

Plusieurs réponses possibles

Itération sans indice

🐍 **Script Python**

```python
def
somme(une_liste):
    resultat = 0
    for un_entier in
une_liste:
        resultat +=
un_entier
    return resultat
```

Itération avec indice

Recommandée.

🐍 **Script Python**

```python
def
somme(une_liste):
    resultat = 0
    for i in
range(len(une_liste)):
        resultat +=
une_liste[i]
    return resultat
```

5. `re)` pourrait se traduire en langage naturel par :

📋 **Pseudo-code**

```
parcourir(A):
    L = liste_vide
    F = file_vide
    enfiler A dans F
    Tant que F n'est pas vide
        défiler S de F
        ajouter la valeur de la racine de S dans L
        Pour chaque sous arbre SA non vide de S
            enfiler SA dans F
    renvoyer L
```

Donner le type de parcours obtenu grâce à la fonction `parcourir`.

Réponse

Si, à un moment du traitement, la file ne contient que des éléments d'un certain niveau, **puis** (éventuellement) du niveau suivant, alors on enfile pendant le traitement des éléments du niveau suivant, ce qui fait que cette propriété est conservée.

Au départ, la propriété est de mise avec un seul élément. Elle le restera pendant tout le parcours, ainsi on traite les éléments niveau par niveau.

Il s'agit d'un **parcours en largeur**.

2.2. Partie B : Méthode « diviser pour régner »

6. Parmi les quatre propositions A,B, C et D ci-dessous, indiquer la seule proposition correcte. En informatique, le principe diviser pour régner est associé à :

- **Proposition A** : diviser une fonction en deux fonctions de plus petit code.
- **Proposition B** : utiliser plusieurs modules
- **Proposition C** : séparer les informations en fonction de leur type
- **Proposition D** : découper un problème initial en sous-problèmes, à résoudre, puis combiner leurs solutions

Réponse

Proposition D

En informatique, diviser pour régner (du latin « *Divide ut imperes* », *divide and conquer* en anglais) est une technique algorithmique consistant à :

- Diviser : découper un problème initial en sous-problèmes ;
- Régner : résoudre les sous-problèmes (récursivement ou directement s'ils sont assez petits) ;
- Combiner : calculer une solution au problème initial à partir des solutions des sous-problèmes.

7. L'arbre présenté dans le problème peut être décomposé en racine et sous arbres :

```
graph TD
   N0(3) --> N1(6)
   N0    --> N2(2)
   subgraph "SAD
                &nbs
```

```
    N2    --> N6(9)
    N2    --> N7(1)
    end
    subgraph "SAG
                &nbs
    N1    --> N4(7)
    N1    --> N5(4)
    end
```

Indiquer dans l'esprit de « diviser pour régner » l'égalité donnant la somme d'un arbre en fonction de la somme des sous arbres et de la valeur numérique de la racine.

Réponse

`somme(arbre)` est égal à `valeur_racine(arbre) + somme(SAG(arbre)) + somme(SAD(arbre))`

8. Écrire en langage Python une fonction récursive `somme(arbre)`. Cette fonction renvoie la somme de l'arbre passé en paramètre.

Les fonctions suivantes sont disponibles :

- `est_vide(arbre)` : détermine si `arbre` est vide et renvoie un booléen `True` ou `False`.
- `valeur_racine(arbre)` : renvoie la valeur numérique de la racine de `arbre` ;
- `arbre_gauche(arbre)` : renvoie le sous arbre à gauche de `arbre` ;
- `arbre_droite(arbre)` : renvoie le sous arbre à droite de `arbre`.

Réponse

🐍 **Script Python**

```python
def somme(arbre):
    if est_vide(arbre):
        return 0
    else:
        return (
            valeur_racine(arbre)
            + somme(arbre_gauche(arbre))
            + somme(arbre_droite(arbre))
        )
```

3. 2022 Asie J1 - Ex3

BAC : Arbres binaires

Les premiers travaux concernant l’aide à la décision médicale se sont développés pendant les années soixante-dix parallèlement à l’avènement de l’informatique dans le secteur médical. L’arbre de décision est une technique décisionnelle fréquemment employée pour rechercher la meilleure stratégie thérapeutique. L’arbre de décision de cet exercice, présenté ci-dessous, est un arbre binaire que l'on nommera `arb_decision`.

```
graph TD
   N0("Conjonctivie jaune") -->|non| N2("hypercaroténémie")
   N0 --> |oui| N1("bilirubine")
   N1 --> |non| N3("spiénomégalie")
   N1 --> |oui| N4("hépatomégalie")
   N3  -->|non| N5("maladie de Gilbert <br> par défaut <br> de glycuro-conjugaison")
   N3  --> |oui| N6("anémie <br> hémolytique")
   N4 --> |non| N7("hépatie")
   N4 --> |oui| N8("douleur <br> et fièvre")
   N8 --> |non| N9("cancer de la tête <br> du pancréas")
   N8 --> |oui| N10("lithiase <br> du cholédoque")
```

Arbre de décision en présence d’une jaunisse (peau anormalement jaune) chez un patient.

Rappels :

- Un arbre binaire est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément, appelé nœud, porte une étiquette.
- Le nœud initial est appelé racine.
- Chaque nœud d’un arbre binaire possède au plus deux sous-arbres.
- Chacun de ces sous-arbres est un arbre binaire, appelés sous-arbre gauche et sous-arbre droit.
- Un nœud dont les sous-arbres sont vides est appelé une feuille.
- Dans cet exercice, on utilisera la convention suivante : la hauteur d’un arbre binaire ne comportant qu’un nœud est égale à 1.

Dans l’arbre de décision en présence d’une jaunisse chez un patient,

- un nœud représente un symptôme dont le médecin doit étudier la présence ou l’absence ; la réponse ne peut être que oui ou non ;
- le sous-arbre gauche d’un nœud donné décrit la démarche à adopter si le symptôme est absent ;
- le sous-arbre droit d’un nœud donné décrit la démarche à adopter si le symptôme est présent ;
- l'étiquette d'une feuille est la maladie induite par le chemin parcouru.

Questions :

1. Déterminer la taille et la hauteur de l’arbre donné en exemple en introduction (arbre de décision en présence d’une jaunisse).

Réponse

- Taille = 11
- hauteur = 5

2. On choisit d'implémenter un arbre binaire à l'aide d'un dictionnaire.

🐍 **Script Python**

```
arbre_vide = {}
arbre = {'etiquette': 'valeur' ,
        'sag': sous_arbre_gauche ,
        'sad': sous_arbre_droit }
```

Le code ci-dessous représente un arbre selon le modèle précédent.

🐍 **Script Python**

```
{'etiquette' :'a',
   'sag':{'etiquette' :'b',
      'sag':{},
      'sad' : {'etiquette' : 'd',
         'sag' : {},
         'sad' : {}}},
   'sad': {'etiquette' :'f',
      'sag' : {'etiquette' : 'g',
         'sag' : {},
         'sad' : {}},
      'sad' : {} }}
```

a. À quelle représentation graphique correspond la structure implémentée ci-dessus ?

- arbre 1 :

```
graph TD
   A("a") --> B("b")
   B --> d("d")
   B --> d1(" ")
   A --> F("f")
   F --> G("g")
   F --> G1(" ")
   linkStyle 2 stroke-width:0px;
   style d1 opacity:0;
   linkStyle 5 stroke-width:0px;
   style G1 opacity:0;
```

- arbre 2 :

```
graph TD
   A("a") --> B("b")
   B --> d(" ")
   B --> d1("d")
   A --> F("f")
```

```
    F --> G("g")
    F --> G1(" ")
    linkStyle 1 stroke-width:0px;
    style d opacity:0;
    linkStyle 5 stroke-width:0px;
    style G1 opacity:0;
```

- arbre 3 :

```
graph TD
    A("a") --> B("b")
    B --> d(" ")
    B --> d1("d")
    A --> F("f")
    F --> G(" ")
    F --> G1("g")
    linkStyle 1 stroke-width:0px;
    style d opacity:0;
    linkStyle 4 stroke-width:0px;
    style G opacity:0;
```

Réponse

Arbre 2

b. Représenter graphiquement l’arbre correspondant au code ci-dessous.

🐍 **Script Python**

```
{'etiquette' :'H',
'sag':{ 'etiquette' :'G',
'sag': {'etiquette' : 'E',
'sag' : {},
'sad' : {}},
'sad' : {'etiquette' : 'D',
'sag' : {},
'sad' : {'etiquette' : 'B',
'sag' : {},
'sad' : {} }}},
'sad': {'etiquette' :'F',
'sag' : {'etiquette' : 'C',
'sag' : {},
'sad' : {'etiquette' : 'A',
'sag' : {} ,
'sad' : {}}},
'sad' : {} }}
```

Réponse

```
graph TD
    H("H") --> G("G")
    G --> E("E")
    G --> D("D")
    H --> F("F")
    F --> C("C")
    F --> F1(" ")
    D --> D1(" ")
    D --> B("B")
    C --> C1(" ")
    C --> A("A")
    linkStyle 5 stroke-width:0px;
    style F1 opacity:0;
    linkStyle 6 stroke-width:0px;
    style D1 opacity:0;
    linkStyle 8 stroke-width:0px;
    style C1 opacity:0;
```

3. La fonction parcours(arb) ci-dessous permet de réaliser le parcours des nœuds d'un arbre binaire arb donné en argument.

🐍 **Script Python**

```python
def parcours(arb):
    if arb == {}:
        return None
    parcours(arb['sag'])
    parcours(arb['sad'])
    print(arb['etiquette'])
```

a. Donner l'affichage après l'appel de la fonction parcours avec l'arbre dont une représentation graphique est ci-dessous.

```
graph TD
    A("a") --> B("b")
    B --> d("d")
    B --> d1(" ")
    A --> F("f")
    F --> G("g")
    F --> G1(" ")
    linkStyle 2 stroke-width:0px;
    style d1 opacity:0;
    linkStyle 5 stroke-width:0px;
    style G1 opacity:0;
```

Réponse

Parcours suffixe : d-b-g-f-a

b. Écrire une fonction `parcours_maladies(arb)` qui n'affiche que les feuilles de l'arbre binaire non vide arb passé en argument, ce qui correspond aux maladies possiblement induites par l'arbre de décision.

Réponse

🐍 **Script Python**

```python
def parcours_maladies(arb):
    if arb=={}:
        return None
    parcours_maladies (arb['sag'])
    parcours_maladies (arb['sad'])
    if arb['sag'] == {} and arb['sad'] == {}:
        print(arb['etiquette'])
```

4. On souhaite maintenant afficher l'ensemble des symptômes relatifs à une maladie. On considère la fonction `symptomes(arbre, mal)` avec comme argument arbre un arbre de décision binaire et mal le nom d'une maladie.

L'appel de cette fonction sur l'arbre de décision `arb_decision` de l'introduction fournit les affichages suivants.

🐍 **Script Python**

```
>>> symptomes(arb_decision, "anémie hémolytique")
symptômes de anémie hémolytique
splénomégalie
pas de bilirubine
conjonctive jaune
```

Pour cela, on modifie la structure précédente en ajoutant une clé `surChemin` qui sera un booléen indiquant si le nœud est sur le chemin de la maladie.
La clé `surChemin` est initialisée à False pour tous les nœuds.

🐍 **Script Python**

```python
arbre = {'etiquette': 'valeur' ,
        'surChemin': False ,
        'sag': 'sous-arbre gauche' ,
        'sad': 'sous-arbre droit' }
```

Recopier et compléter les lignes 6, 8, 14 et 18 du code suivant sur votre copie.

```
1   def symptomes(arb, mal):
2       if arb['sag'] != {} :
3           symptomes(arb['sag'],mal)
4
5       if arb['sad'] != {} :
6           symptomes(...)
7
8       if ... ... ... ... :
9           arb['surChemin'] = True
10          print('symptômes de', arb['etiquette'],':')
11
12      else :
13          if arb['sad'] != {} and arb['sad']['surChemin'] :
14              print(...)
15              arb['surChemin'] = True
16
17          if arb['sag'] != {} and arb['sag']['surChemin'] :
18              print(...)
19              arb['surChemin'] = True
```

Réponse

```
def symptomes(arb, mal):
    if arb['sag'] != {}:
        symptomes(arb['sag'], mal)

    if arb['sad'] != {}:
        symptomes(arb['sad'], mal)

    if arb['etiquette'] == mal:
        arb['surChemin'] = True
        print('symptômes de', arb['etiquette'],':')

    else :
        if arb['sad'] != {} and arb['sad']['surChemin']:
            print(arb['etiquette'])
            arb['surChemin'] = True

        if arb['sag'] != {} and arb['sag']['surChemin']:
            print('pas de ',arb['etiquette'])
            arb['surChemin'] = True
```

C1 Le Modéle relationnel

Programme Terminale

Contenus	Capacités attendues	Commentaires
Modèle relationel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel	Identifier les concepts définissant le modèle relationnel	Ces concept permettent d'exprimer les contraites 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 :

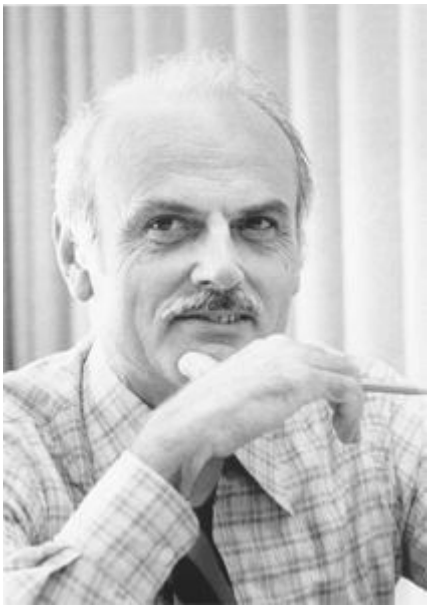
- les SGBD permettent de gérer la lecture, l'écriture ou la modification des informations contenues dans une base de données
- les SGBD permettent de gérer les autorisations d'accès à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur 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. 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**.

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
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
3	Fondation	Asimovann_publi	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	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 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

En-tête

Enregistrement,tuple, n-uplet

attribut

Vocabulaire

- **relation** , ou **table** : c'est l'endroit où sont rangées les données. L'ordre des lignes (que l'on appelera 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 «Fahreneit 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))`

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.

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)

5. Contrainte d’intégrité : clé étrangère

Ajoutons maintenant les relations ci-dessous :

Relation «Emprunts»

id_emprunteur	date	Nom	Prénom	titre	auteur	code
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»

id_emprunteur	Nom	Prénom
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**.

Clé 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.

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»

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

7. Résumé : Contraintes d'intégrité

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.

7.2. Contrainte de relation

Clé 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).

7.3. Contrainte de référence

Clé 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»

id_emprunteur	date	code
845	12/10/2020	3
125	13/10/2020	9
125	13/10/2020	**27**

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.

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 Boulle)

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'écrira 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

dossard	nomCoureur	prénomCoureur	codeEquipe
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`)

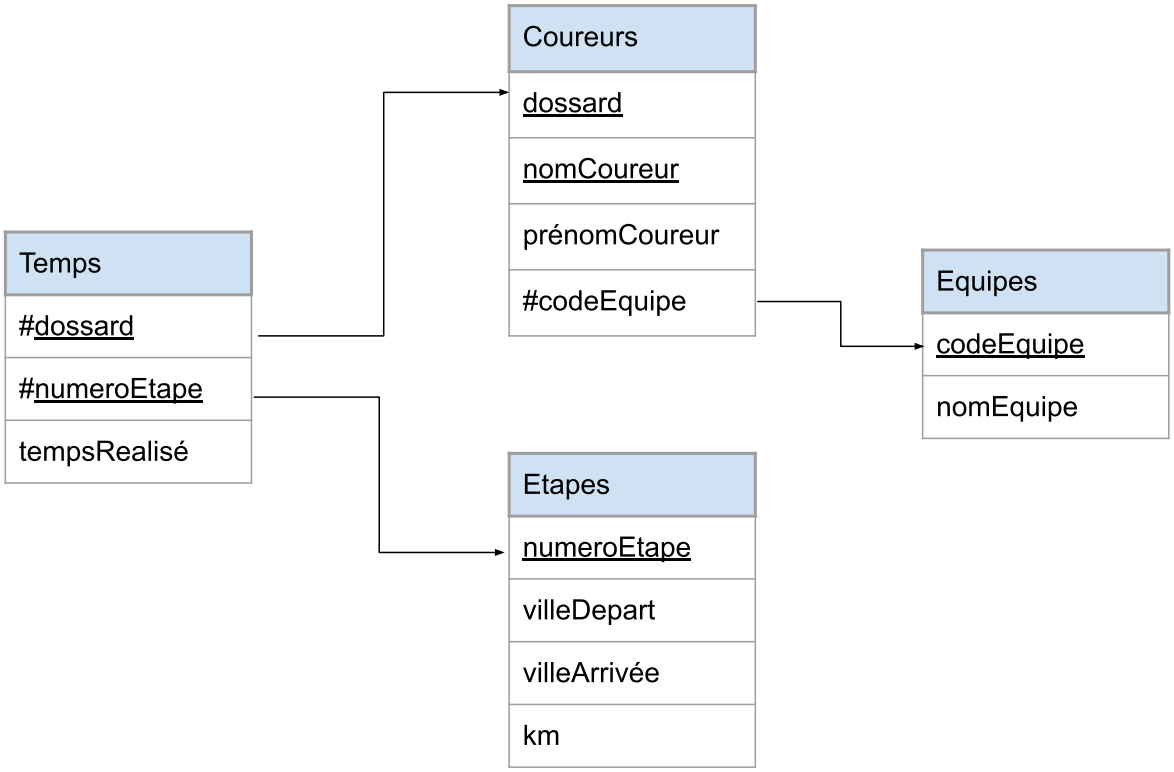
relation Temps

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


Coureurs
dossard
nomCoureur
prénomCoureur
#codeEquipe
Temps
#dossard
#numeroEtape
tempsRealisé
Equipes
codeEquipe
nomEquipe
Etapes
numeroEtape
villeDepart
villeArrivée
km

Langage SQL : Résumé

Thème 2 : Base de données

02 Résumé cours : Langage SQ

Programme Terminale

Contenus	Capacités attendues	Com
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 p ou le utilis

Voir TP sur Capytal

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 garan 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 (pou *structurées*).

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

Mais de plus en plus de SGBD **non-relationnels** sont utilisés, spécialement adaptés à des données plus diver retrouve sous l'appelation **NoSQL** (pour *Not only SQL*). Citons parmi eux MongoDB, Cassandra (Facebook), B

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
entièrement représentée dans le logiciel utilisant SQLite.

2. Création 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));
```

2.1. Création 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 Ā','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. 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');
```

3. Sélection de données

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 Ā	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

titre	auteur	note
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 Ā	Van Vogt	7
La Fin de l'éternité	Asimov	8
De la Terre à la Lune	Verne	10

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

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 résulats et dans la suite de la requête (important), la colonne "ann_publi" est renom

🗂️ **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 Ā	Van Vogt	1945
La Fin de l’éternité	Asimov	1955

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 pre

🗂️ **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 pa
  ASC est le paramètre par défaut).

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**

```sql
SELECT DISTINCT auteur
FROM LIVRES
```

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

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**

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

permet d'obtenir les titres de livres commençant par F

🗂️ **Requête SQL**

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

permet d'obtenir les titres de livres finissant par s

🗂️ **Requête SQL**

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

titre
Fondation
Fahrenheit 451

🗂️ **Requête SQL**

```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

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 effectuée.

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

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

7. 📌 La clause COUNT

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

🗂️ **Requête SQL**

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

total
2

8. 📌 La clause : SUM - Additionner

- **Commande :**

  🗂️ **Requête SQL**

  ```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*

somme
3904

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 s
nommée «moyenne».

🗂️ **Requête SQL**

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

moyenne
7.5

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	7

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**

```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 Ā',9,1945,7),
(15,'La Fin de l'éternité',3,1955,8),
(16,'De la Terre à la Lune',10,1865,10);
```

10.1. 🔗 Jointures simples

En général, les jointures consistent à associer des lignes de 2 tables. Elles permettent d'établir un lien entre 2

- **Commande :**

🗂️ **Requête SQL**

```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 tab

code	titre	id_auteur	ann_publi	note	id	nom	prenom	ann_naissance	langue_ecrit
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 Ā	9	1945	7	9	Van Vogt	Alfred Elton	1912	anglais

code	titre	id_auteur	ann_publi	note	id	nom	prenom	ann_naissance	langue_ecrit
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,
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 Ā	7	1912	1945	anglais
Verne	De la Terre à la Lune	10	1828	1865	français

11. Modifications d'une base

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)
```

11.2. 🔸 UPDATE

🗂️ **Requête SQL**

```
UPDATE LIVRES
SET note=7
WHERE titre = 'Hypérion'
```

11.3. 🔸 DELETE

🗂️ **Requête SQL**

```
DELETE FROM LIVRES
WHERE titre='Hypérion'
```

SQL : Exercices BAC

Thème 2 : Base de données

BAC

Langage SQL

1. Exercice n°1 : Métropole J1 : Base de données cinématographique

SQL

- 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, U**

Nous allons étudier une base de données traitant du cinéma dont voici le schéma relationnel qui comporte 3 r

- 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**

id_ind	nom	prenom	naissance
105	'Hulka'	'Daniel'	'01-06-1968'
403	'Travis'	'Daniel'	'10-03-1968'

id_ind	nom	prenom	naissance
688	'Crog'	'Daniel'	'07-07-1968'
695	'Pollock'	'Daniel'	'24-08-1968'

-Extrait de **realisation**

id_rea	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**

```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 d
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
réponse en expliquant pourquoi la requête refusée ne pourra pas fonctionner.

🗂️ **Requête SQL 1**

```sql
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

🗂️ **Requête SQL 2**

```sql
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 pr
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 p
relation emploi les rôles de Daniel Crog en tant que James Bond dans le film nommé `'Casino Impérial'` puis dan

🗂️ **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
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
le titre du film et l'année de sortie du film, à partir de tous les enregistrements de la relation **emploi** pour les
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 (De
est son nom).
On veillera à n'afficher que la description des emplois et non les films associés à ces emplois.

2. Exercice n°2 : D'après 2022, Métropole, J2

SQL

- 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`, `UPDA`

La clause `ORDER BY` suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut. L'instru
nombre de lignes d'une requête.

Un musicien souhaite créer une base de données relationnelle contenant ses morceaux et interprètes préféré
SQL.

Il crée une table **`morceaux`** qui contient entre autres attributs les titres des morceaux et leur année de sortie

- Table **`morceaux`**

id_morceau	titre	annee	id_interprete
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`**

id_interprete	nom	pays
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 **`interprete`**

1.a. Écrire le résultat de la requête suivante :

🗂️ **Requête SQL**

```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**

```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**

```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 « Imagin

3.b. Écrire une requête SQL permettant d'ajouter l'interprète « The Who » venant d'Angleterre à la table **inte** `id_interprete` égal à 6.

3.c. Écrire une requête SQL permettant d'ajouter le titre « My Generation » de « The Who » à la table **morcea** 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. 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 donné
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
qui fait référence à la clé primaire `idEleve` de la relation `Eleves` . De même l'attribut `isbn` de la relation `Emprun`
référence à la clé primaire `isbn` de la relationcompléter `Livres` .

1. Expliquer pourquoi le code `SQL` ci-dessous provoque une erreur.

🗂️ **Requête SQL**

```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 p
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**

```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**

```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 me
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 « *Le*

Sujet BAC 3 : Langage SQL

1. Métropole J1 : Base de données cinématographique

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

id_ind	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`

id_rea	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**

```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**

```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**

```sql
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

🗂️ **Requête SQL 2**

```sql
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**

```sql
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.

Réponse

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**

```sql
INSERT INTO emploi
VALUES (5400, 'Acteur(James Bond)', ... , ... );

INSERT INTO emploi
VALUES (5401, 'Acteur(James Bond)', ... , ...);
```

Réponse

🗂️ **Requête SQL**

```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`**.

Réponse

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**

```sql
SELECT ...
FROM emploi
JOIN individu ON ...
JOIN realisation ON ...
WHERE emploi.description = 'Acteur(James Bond)';
```

Réponse

🗂️ **Requête SQL**

```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**

```sql
SELECT description
FROM emploi
JOIN individu ON emploi.id_ind = individu.id_ind
WHERE prenom = 'Denis' AND nom = 'Johnson';
```

D'après 2022, Métropole, J2

!!! exo D'après 2022, Métropole, J2"

📋 **Texte**

- 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`

id_morceau	titre	annee	id_interprete
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`

id_interprete	nom	pays
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**

```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**

```sql
SELECT nom
FROM interpretes
WHERE pays = 'Angleterre';
```

1.c. Écrire le résultat de la requête suivante :

🗂️ **Requête SQL**

```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**

```sql
SELECT COUNT(*)
FROM morceaux;
```

1.e. Écrire une requête affichant les titres des morceaux par ordre alphabétique.

Réponse

🗂️ **Requête SQL**

```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`**.

Réponse

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**

```sql
INSERT INTO interpretes
VALUES (1, 'Trust', 'France');
```

Réponse

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.

Réponse

On utilise la clé primaire du morceau afin d'éviter toute méprise :

🗂️ **Requête SQL**

```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**

```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.

Réponse

🗂️ **Requête SQL**

```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.

Réponse

🗂️ **Requête SQL**

```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.

Réponse

On utilise une jointure :

🗂️ **Requête SQL**

```
SELECT titre
FROM morceaux
JOIN interpretes ON interpretes.id_interprete = morceaux.id_interprete
WHERE interpretes.pays = 'États-Unis';
```

2. Métropole, Candidats libres, J2 2021

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 relationcompléter `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

```sql
SELECT titre
FROM Livres
WHERE auteur = 'Molière'
```

4. Décrire le résultat renvoyé par la requête ci-dessous.

Requête SQL

```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

```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**

```sql
UPDATE Emprunts SET ..................... WHERE ..................... ;
```

Réponse

🗂️ **Requête SQL**

```sql
UPDATE Emprunts
SET dateRetour = '2020-09-30'
WHERE idEmprunt = 640
```

6. Décrire le résultat renvoyé par la requête ci-dessous.

🗂️ **Requête SQL**

```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**

```sql
SELECT nom, prenom
FROM Eleves
JOIN Emprunts ON Eleves.idEleves = Emprunts.idEleves
WHERE Emprunts.isbn = 192
```

Sans l'ISBN :

🗂️ **Requête SQL**

```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'
```

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.	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) Le lien avec les algorihtmes de recherche de chemion sur un graphe est mis en évidence.

1. Résumé des épisodes précédents

1.1. ■ Activité 1 : *Adresse IP, masque*

Rappel

- 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 `11111111.11111111.111111110.00000000`. 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.

Rappel

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.

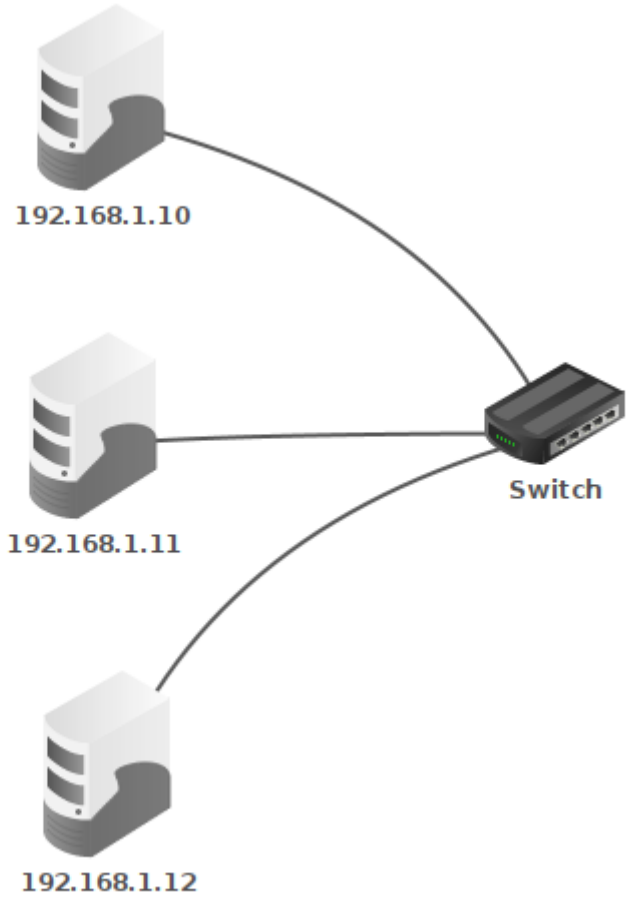
1. Lancer Filius, un outil de simulation de réseaux, et créer un simple réseau constitué de deux ordinateurs. Dans chacun des cas suivants, prévoir si les deux ordinateurs peuvent communiquer et le vérifier à l'aide d'une commande `ping`

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

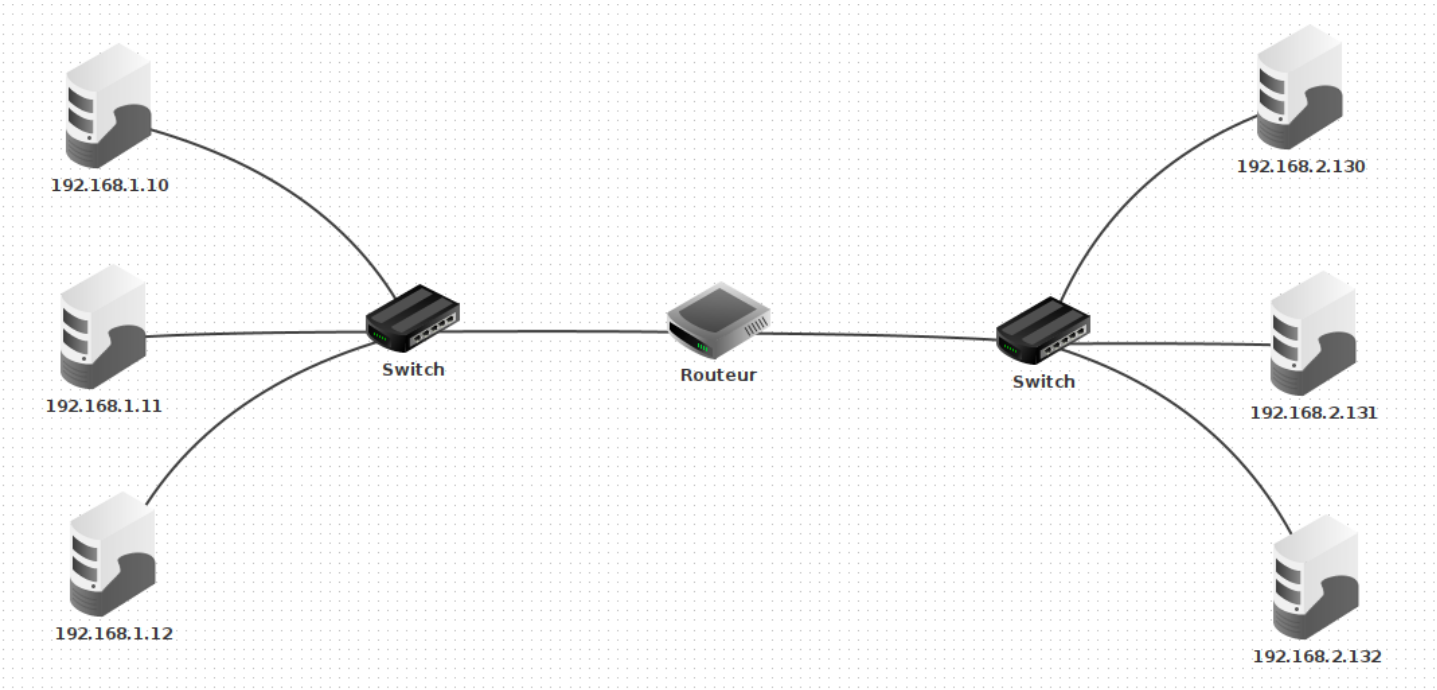
Aide

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.



Aide

En cas de difficultés, se référer à la vidéo suivante :

2. Protocoles de routage

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 : **R**outing **I**nformation **P**rotocol

   a. Sachant que dans ce protocole, on tente de minimiser le nombre de routeurs traversés, quel serait les chemins empruntés pour :

      i. relier `A` et `E` ?

      ii. relier `A` et `C` ?

      iii. 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

Destination	Distance
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 liason entre `A` et `E` (poids 1) est bien meilleure que celle entre `A` et `B` (poids 5)

```
graph TD
A(("A"))
B(("B"))
C(("C"))
D(("D"))
E(("E"))
A-- 5 ---B
A-- 1 --- E
B-- 2 --- E
B-- 2 --- D
C-- 1 --- E
C-- 4 --- D
```

   a. Sachant que dans ce protocole, on tente de minimiser le poids du chemin parcouru, quel serait les chemins empruntés pour :
      i. relier `A` et `E` ?
      ii. relier `A` et `C` ?
      iii. relier `E` et `D` ?

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'*arpanet*, ancêtre d'internet.
- 1973 : définition des protocoles *tcp* (Transmission Control Protol) 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 connectés.

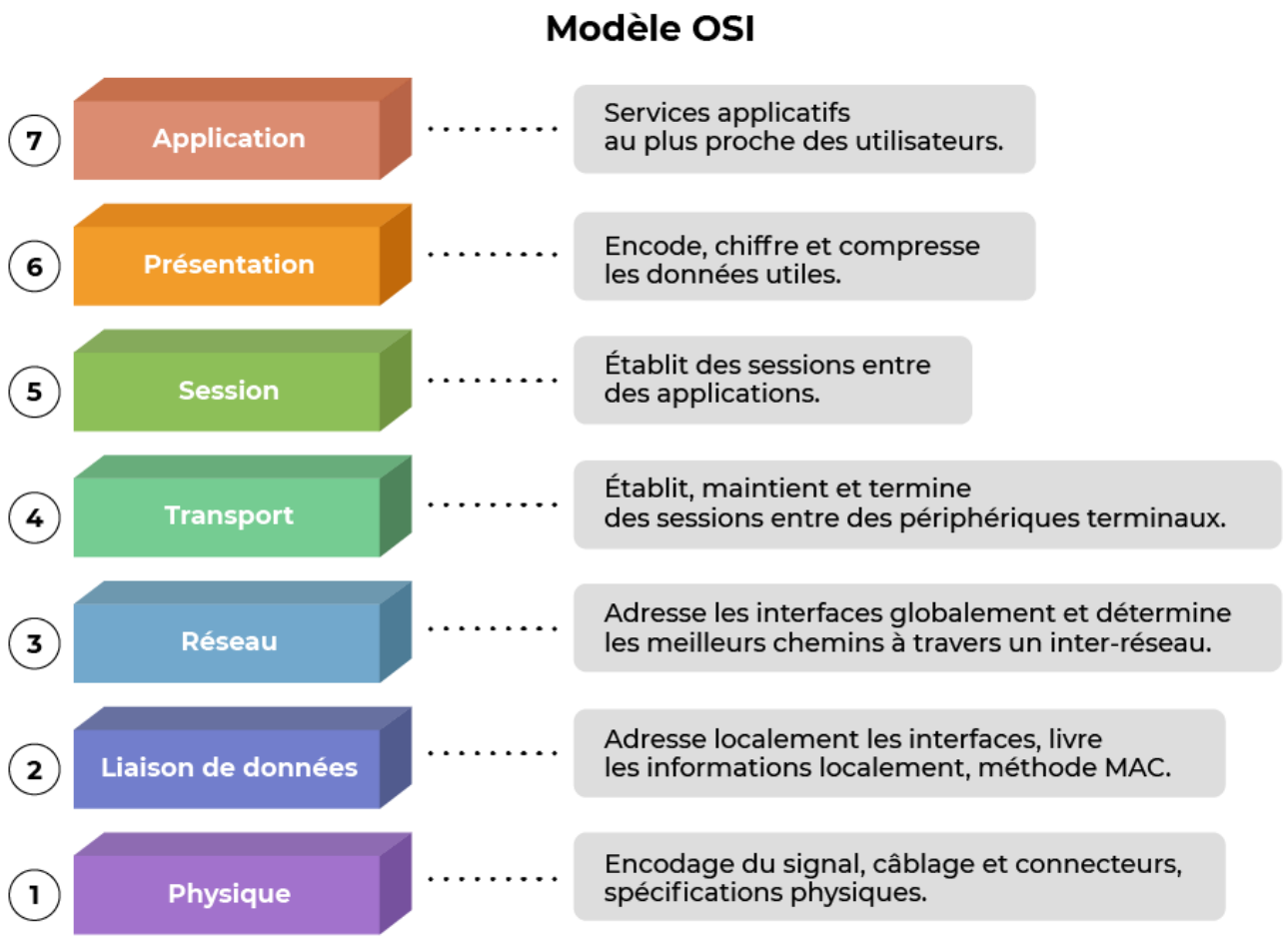
3.1. Protocole TCP/IP

Protocoles TCP/IP

Internet fonctionne suivant une architecture réseau en 4 couches :

Modéle 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**.



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. Elle sont composées de six nombres en hexadécimal séparé 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 octects = 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 : \((240)_{10} = (11110000)_2\), 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 \((130)_{10} = (100000010)_2\)
- B : `192.168.155.100`, or \((155)_{10} = (10011011)_2\)
- C : `192.168.144.203`, or \((144)_{10} = (10010000)_2\)

B et C peuvent communiquer entre elles mais pas avec A.

- Confirmer éventuellement par un test dans Filius

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 stockés 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.

3.4. Protocoles de routages

3.4.1. 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

4. Le protocole RIP

Le 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).

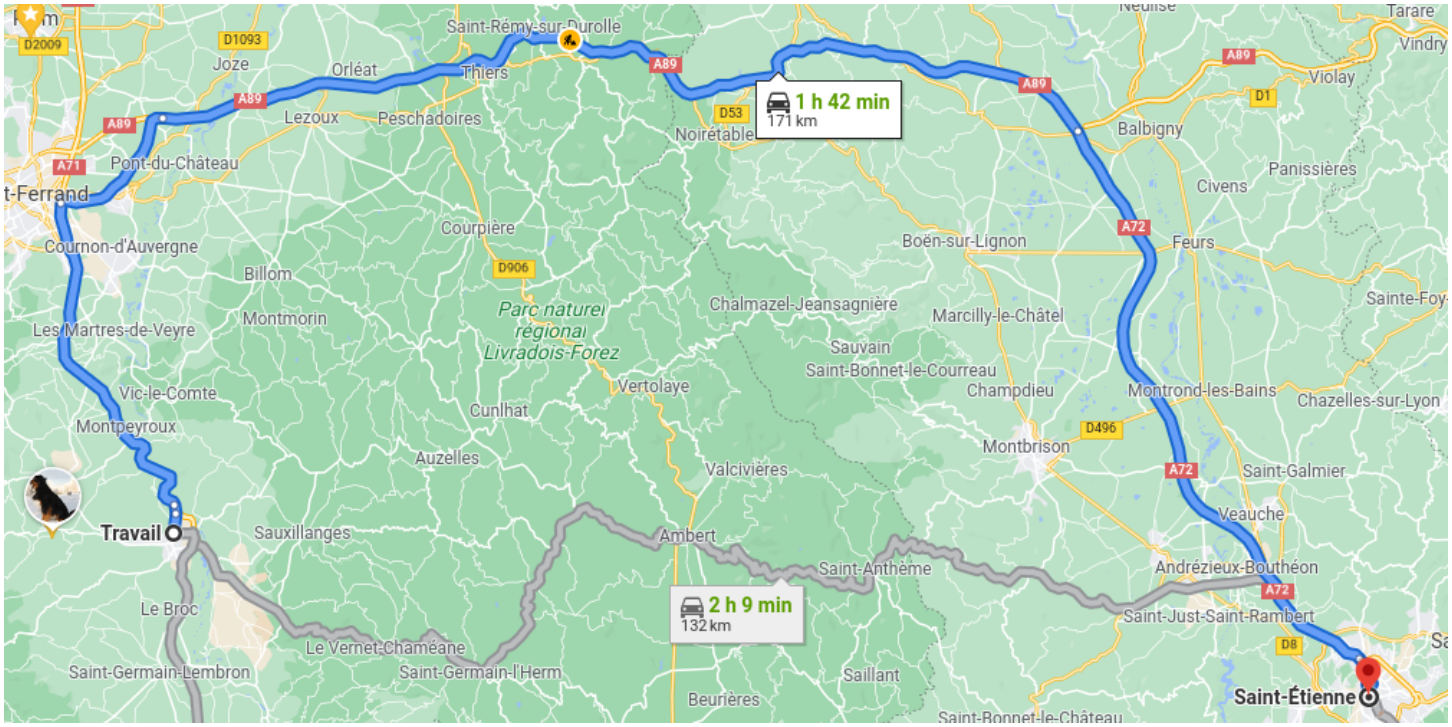
Remarques et incovénients :

- 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 un 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 Gbits/s	10 Mbit/s ~ 10 Gbits/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.

5. Le protocole OSPF

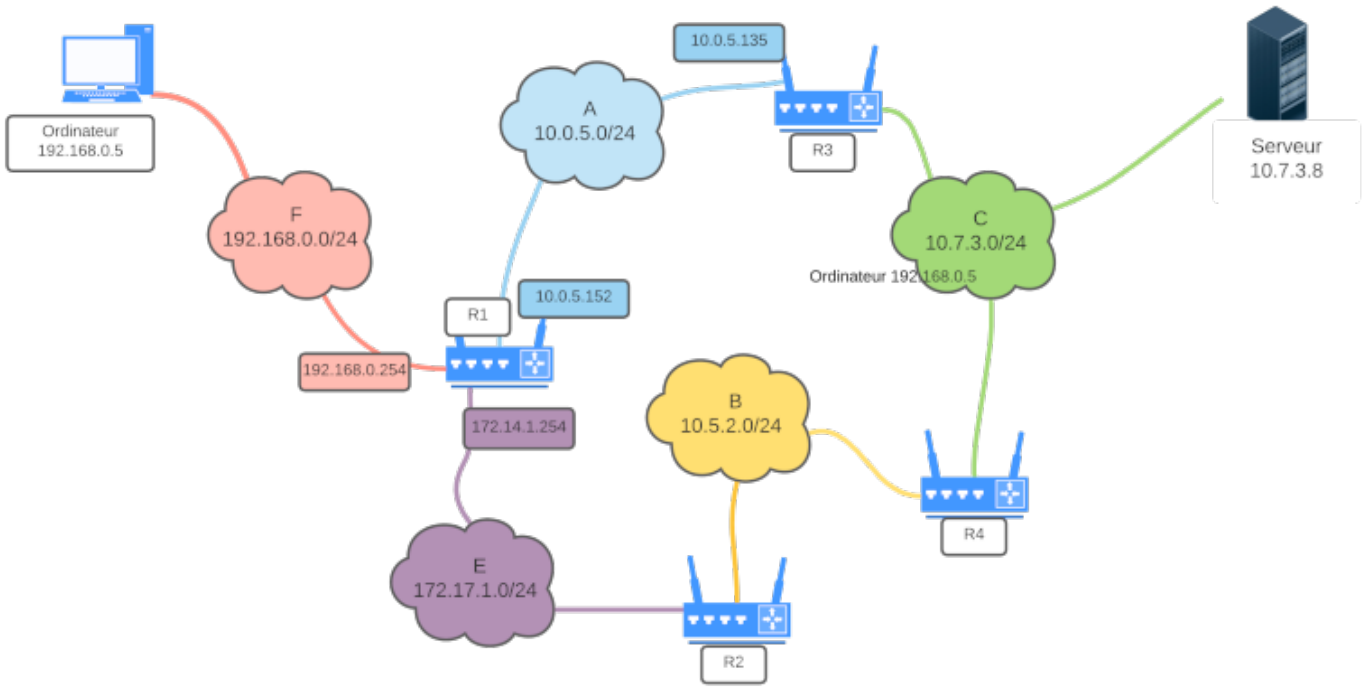
OSPF : Open Shortest Path First

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 :
- \(c = \dfrac{10^8}{d}\). Mais elle peut-être différente et sera donné 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).

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"))
O-- 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 **graphe**. 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.

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 :

Algorithme de Dijstra

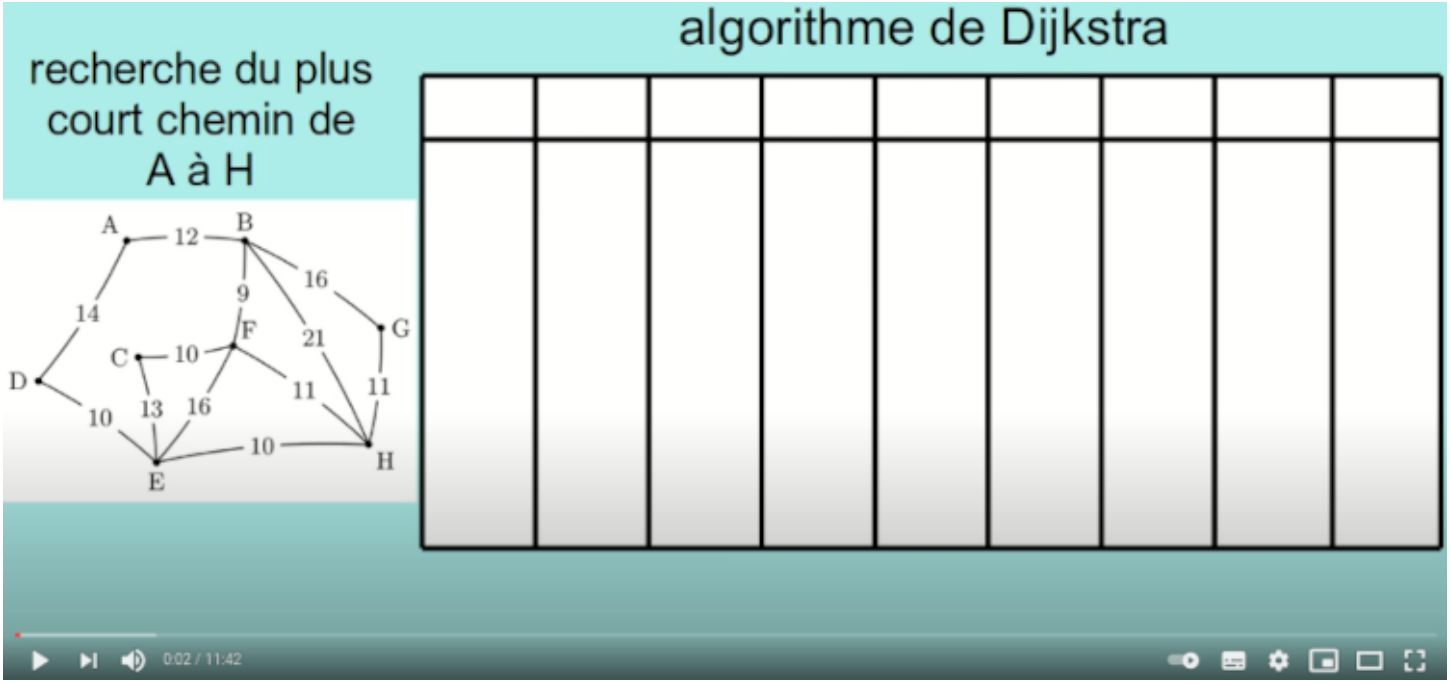
Cet algorithme, ici exécuté de manière manuelle, est bien sûr programmable. Et c'est donc grâce à lui que chaque routeur calcule la route la plus rapide pour acheminer les données qu'il reçoit.

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
```

Réponse

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



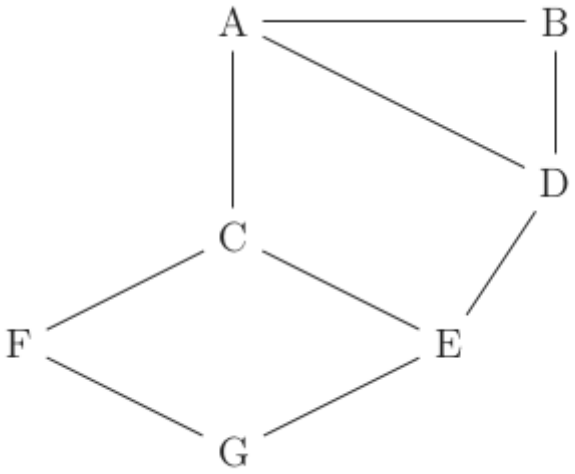
Donner le plus court chemin pour aller de A à H.

Sujet BAC 6 : Protocole de routage

1. Sujet n°1 : sujet zéro

Sujet zéro

On considère un réseau composé de plusieurs routeurs reliés de la façon suivante :



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

1. Le routeur A doit transmettre un message au routeur G, en effectuant un nombre minimal de sauts. Déterminer le trajet parcouru.
2. 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.
2.

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

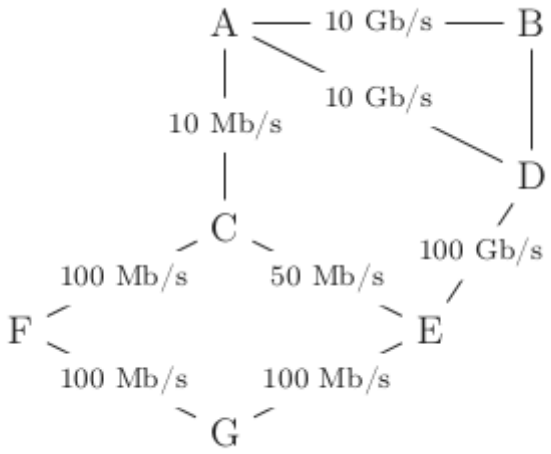
1.2. ➡ Le protocole OSPF

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 :

$coût = \dfrac{10^8}{d}$

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 -> B : 10 Gb/s soit un coût : $\dfrac{10^8}{10 \times 10^9}=0.01$

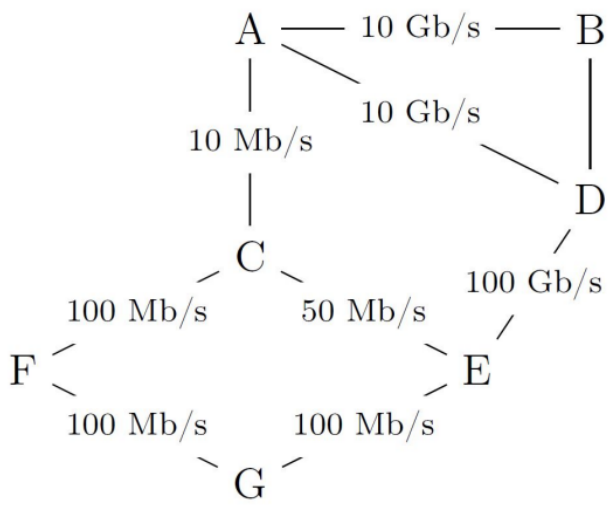
3.2
$\dfrac{10^8}{d}=5 \rightarrow d=\dfrac{10^8}{5} = 2 \times 10^7$ b/s soit 20 Mb/s

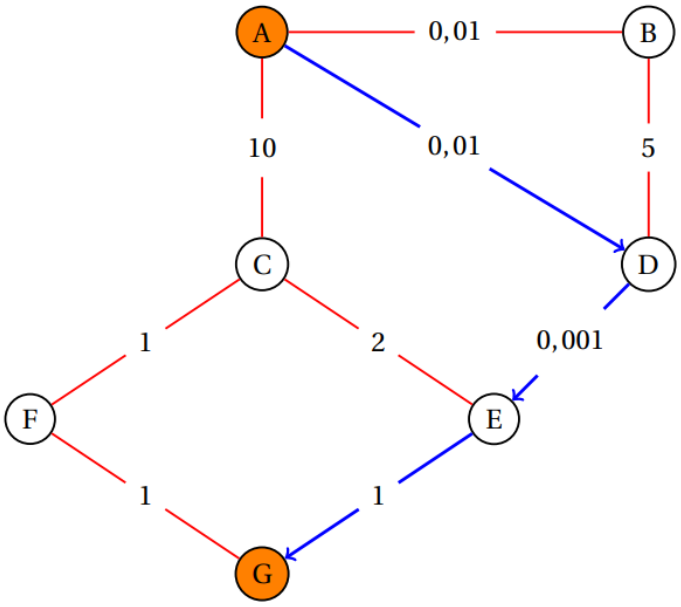
Question 4

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.

Réponse

Débit d bits/s	Coût $= \dfrac{10^8}{d}$
$d = 10Gb/s = 10^{10} bits/s$	coût $= 10^8 \times 10^{-10} = 10^{-2} = 0,01$
$d = 100Gb/s = 10^{11} bits/s$	coût $= 10^8 \times 10^{-11} = 10^{-3} = 0,001$
$d = 100Mb/s = 10^8 bits/s$	coût $= 10^8 \times 10^{-8} = 1$
$d = 50Mb/s = 5 \times 10^7 bits/s$	coût $= 0,2 \times 10^8 \times 10^{-7} = 2$
$d = 10Mb/s = 10^7 bits/s$	coût $= 10^8 \times 10^{-7} = 10$





Départ	A	B	C	D	E	F	G	Sommet choisi
	0	∞	∞	∞	∞	∞	∞	A
A(0)		0,01 (A)	10 (A)	0,01(A)	∞	∞	∞	B (A)
B(0,01)			10 (A)	~~5,01(B)~~ 0,01(A)	∞	∞	∞	D (A)
D(0,01)			10 (A)		0,011(D)	∞	∞	E(D)
E(0,011)			~~10 (A)~~ 2,011(E)			∞	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.

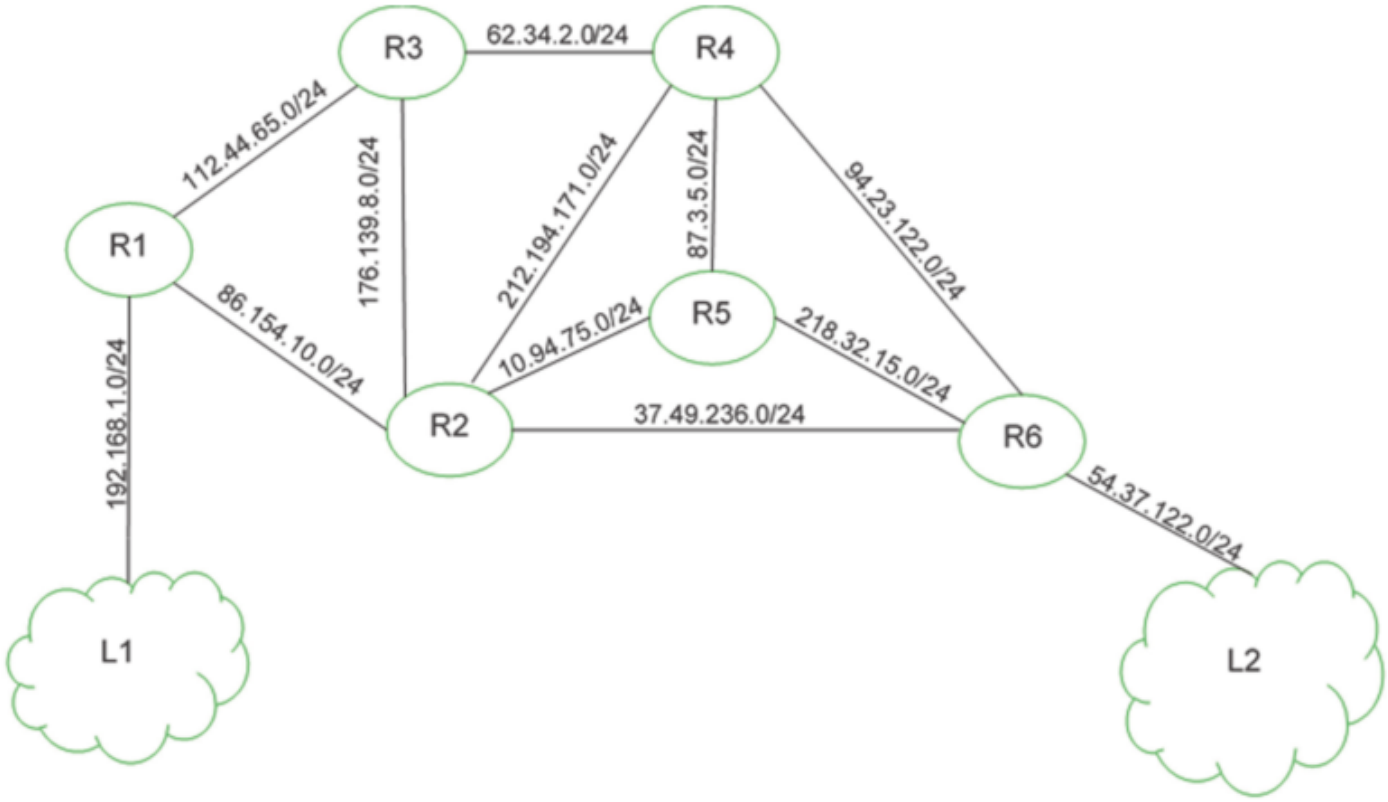
Correction du tableau de l'algorithme de Dijkstra

2. Sujet n°2

Exo

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 :

\(C=\dfrac{10^9}{BP}\)

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

\(C = \dfrac{10^9}{10^7} = 100\)

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

3. Sujet n°3 : 2022, Métropole, J2

D'après 2022, Métropole, J2, Ex. 3

Rappels :

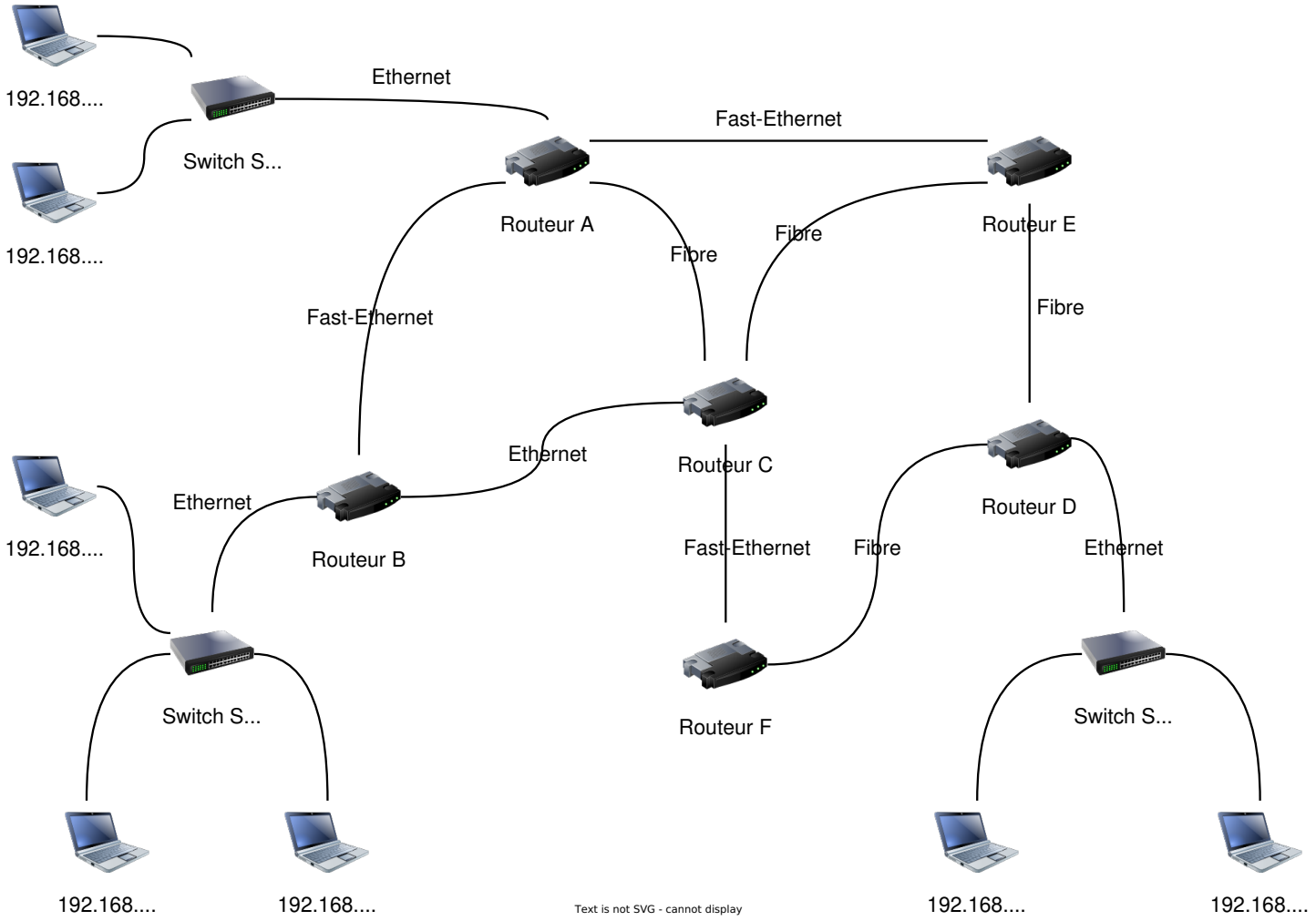
Une adresse IPv4 est composée de 4 octets, soit 32 bits. Elle est notée `a.b.c.d`, où `a`, `b`, `c` et `d` sont les valeurs des 4 octets.

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

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

Destination	passe par
B	...
C	...
D	E
E	...
F	C

Destination	passe par
A	A
C	C
D	C
E	C
F	C

Destination	passe par
A	A
B	B
D	E
E	E
F	F

Destination	passe par
A	E
B	F
C	F
E	E
F	F

Destination	passe par
A	A
B	C
C	C
D	D
F	C

Destination	passe par
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

Destination	Passe par
B	B
C	C
D	C
E	C
F	C

Routeur B

Destination	Passe par
A	A
C	A
D	A
E	A
F	A

Routeur C

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 $\mathrm{coût} = \frac{10^8}{d}$ où d représente le débit en bit/s et $\mathrm{coût}$ est sans unité. Le schéma du réseau reste celui du début de l'exercice.

Question 4.a

Déterminer le coût des liaisons Ethernet ($d = 10^7\,\mathrm{bit}/\mathrm{s}$), Fast-Ethernet ($d = 10^8\,\mathrm{bit}/\mathrm{s}$) et Fibre ($d = 10^9\,\mathrm{bit}/\mathrm{s}$).

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;
```

Réponse

Sans arrondi **Avec arrondi**

On conserve ici la valeur de \(0,1\) pour le coût de la liaison Fibre.

```
graph LR

A[(A)] --- |1| B[(B)]
    A --- |0,1| C[(C)]
    A --- |1| E[(E)]
    B --- |10| C
    C --- |0,1| E
    E --- |0,1| D[(D)]
    D --- |0,1| F[(F)]
    C --- |1| F
```

On arrondit le coût

de la liaison Fibre à \(1\).

```
graph LR

A[(A)] --- |1| B[(B)]
    A --- |1| C[(C)]
    A --- |1| E[(E)]
    B --- |10| C
    C --- |1| E
    E --- |1| D[(D)]
    D --- |1| F[(F)]
    C --- |1| F
```

Question 4.c

t IPv4 doit être acheminé d'une machine ayant pour adresse IPv4 `192.168.2.1` à ine ayant pour adresse IPv4 `192.168.4.1`.

routes possibles, c'est à dire la liste des routeurs traversés, et le coût de de ces routes, chaque routeur ne pouvant être traversé qu'une seule fois.

Réponse

Sans arrondi **Sans arrondi**

On conserve la valeur de \(0,1\) 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 \(1\). 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

iant, la route qui sera empruntée par un paquet IPv4 pour aller d'une r adresse IPv4 `192.168.2.1` à une machine ayant pour adresse IPv4

Réponse

n de coût minimal :

vé un coût de \(0,1\) pour la liaison Fibre, on obtient le chemin B → A → C → E → D pour un coût de \(1,3\) ;

- si l'on arrondit ce coût à \(1\), on obtient le chemin B → A → E → D pour un coût de \(3\).

4. Sujet 4 : 2022, Métropole, J1

: D'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

Destination	Métrique
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

```
flowchart 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 : $\text{métrique} = \dfrac{10^8}{\text{débit}}$ dans laquelle le débit est exprimé en bit par seconde (bps).

On rappelle qu'un kbps est égal à $10^3~\text{bps}$ et qu'un 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	$100~\text{kbps}$	$500~\text{kbps}$	$10~\text{Mbps}$	$100~\text{Mbps}$
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	Métrique
F	0
G	8
H	5
I	**13**
J	**6**
K	**8**
L	**11**

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.

C13 Gestion Processus

1. Introduction

Dans les années 1970 les ordinateurs personnels étaient incapables d'exécuter plusieurs tâches à la fois : il fallait attendre qu'un programme lancé se termine pour en exécuter un autre.

Les systèmes d'exploitations récents (GNU/Linux, macOS, iOS, Android, Windows...) permettent d'exécuter des tâches "simultanément". En effet, la plupart du temps, lorsque l'on utilise un ordinateur, plusieurs programmes sont exécutés "en même temps" : par exemple, on peut très bien ouvrir simultanément un navigateur Web, un traitement de texte, un IDE Python, un logiciel de musique (sans parler de tous les programmes exécutés en arrière-plan) ...

Ces programmes en cours d'exécution s'appellent des **processus**. Une des tâches du système d'exploitation est d'allouer à chacun des processus les ressources dont il a besoin en termes de mémoire, entrées-sorties ou temps d'accès au processeur, et de s'assurer que les processus ne se gênent pas les uns les autres.

Pourtant, on rappelle qu'un programme n'est qu'une suite d'instructions machine exécutées l'une après l'autre par le processeur et qu'un processeur n'est capable d'exécuter qu'une seule instruction à la fois.

Pour rappel, voici les étapes d'exécution d'une instruction.

- l'instruction pointée par le pointeur d'instruction est chargée en mémoire
- Le pointeur d'instruction est incrémenté vers l'adresse suivante
- l'instruction est décodée
- l'instruction est exécutée

Comment est-il alors possible que plusieurs programmes soient exécutés en même temps ?

2. Processus

2.1. Qu'est-ce qu'un *processus* ?

Il ne faut pas confondre programme et processus :

- Un **programme** est un fichier binaire (on dit aussi un *exécutable*) contenant des instructions machines que seul le processeur peut comprendre.
- Un **processus** est un *programme en cours d'exécution*, autrement dit le phénomène dynamique lié à l'exécution d'un programme par l'ordinateur.

Ainsi, lorsque nous cliquons sur l'icône d'un programme (ou lorsque nous exécutons une instruction dans la console pour lancer un programme), nous provoquons la naissance d'un ou plusieurs processus liés au programme que nous lançons.

Un processus est donc une *instance d'un programme* auquel est associé :

- du code
- des données/variables manipulées
- des ressources : processeur, mémoire, périphériques d'entrée/sortie (voir paragraphe suivant)

Il n'est d'ailleurs pas rare qu'un même programme soit exécuté plusieurs fois sur une machine au même moment en occupant des espaces mémoires différents : par exemple deux documents ouverts avec un traitement de texte, ou trois consoles distinctes... qui correspondent à autant d'instances du même programme et donc à des processus différents.

3. Activités

3.1. ■ Activité 1 : *Observer les processus*

Il est très facile de voir les différents processus s'exécutant sur une machine.

Sous GNU/Linux, on peut utiliser la commande `ps` (comme **p**roces**s**, la traduction anglaise de *processus*) pour afficher les informations sur les processus (sous windows : talklist). En passant des options à cette commande on peut obtenir des choses intéressantes.

Par exemple, en exécutant dans un terminal la commande `ps -aef`, on peut visualiser tous les processus en cours sur notre ordinateur :

3.2. Création d'un processus

Un processus peut être créé :

- au démarrage du système
- par un autre processus
- par une action d'un utilisateur (lancement d'un programme)

Sous GNU/Linux, un tout premier processus est créé au démarrage (c'est le processus 0 ou encore *Swapper*). Ce processus crée un processus souvent appelé *init* qui est le fils du processus 0. Ensuite, à partir de *init*, les autres processus nécessaires au fonctionnement du système sont créés. Ces processus créent ensuite eux-mêmes d'autres processus, etc.

Un processus peut créer un ou plusieurs processus, ce qui aboutit à une structure arborescente comme nous allons le voir maintenant.

3.3. PID et PPID

La commande précédente permet de voir que chaque processus est identifié par un numéro : son **PID** (pour *Process Identifier*). Ce numéro est donné à chaque processus par le système d'exploitation.

On constate également que chaque processus possède un **PPID** (pour *Parent Process Identifier*), il s'agit du PID du processus parent, c'est-à-dire celui qui a déclenché la création du processus. En effet, un processus peut créer lui même un ou plusieurs autres processus, appelés *processus fils*.

PID et PPID

- `PID` (Process Identifier) : le numéro unique d'identification, affecté chronologiquement par le système d'exploitation. Le processus de PID égal à 1 est `systemd`, qui est le tout premier programme lancé par le noyau Linux au démarrage.
- `PPID` (Parent PID) : certains processus vont *eux-mêmes* lancer plusieurs processus-fils, qui porteront le même nom. C'est ainsi qu'on peut retrouver de multiples processus s'appelant `chrome` :

3.4. Observation des processus⚓

Sous Linux, on peut observer les processus et leur état en ligne de commande.

Pour tester, cela on peut utiliser : terminal linux en ligne

Voici quelques commandes utiles pour observer les processus :

- `ps` : elle permet d'afficher la liste des processus en cours. (Pour avoir des informations sur les options taper man ps) Pour lister tous les processus : `ps - ef`

- `top` : Pour observer en temps réels les différents processus.(penser à utiliser man top)
  la commande `f` permet de gérer les colonnes affichées.
- `kill` : elle permet de tuer un processus en lui envoyant un signal de fin.
  - 15 pour arrêter le processus proprement
  - 9 pour arrêter immédiatement le processus.

  exemple d'utilisation : `kill -15 7654` pour tuer proprement le processus de `PID 7564`

Exo

Dans un nouvel onglet ouvrir : terminal linux en ligne

- créer un premier terminal :
- utiliser les commandes suivante : `ls, cd, touch, cat`
  - pour déterminer le nom d'utilisateur : `whoami`
  - créer un fichier vide `test.py` : `touch test.py`
  - éditer le fichier `test.py` avec la commande : `nano test.py`
    - y écrire le code suivant :

      🐍 **Script Python**

      ```python
      for a in range(100000):
      print(a)
      ```

    - pour sortir de l'éditeur : Ctrl+X, puis Y, puis Enter pour confirmer le nom
  - lancer le programme avec : `python3 test.py`
  - créer un second fichier `p2.py` avec le code suivant :
    `python`
    `while True:`
    `pass`
  - le lancer
  - Il tourne sans fin. Pour l'arrêter : Ctrl+C
- vérifier la présence dans le dossier des fichiers créés, avec `ls`
- ouvrir un second terminal puis :
  - dans ce second terminal lancer python3 sans nom de fichier
  - dans le premier terminal taper `ps -ef`
  - repérer le PID du processus python3 et le tuer avec la commande kill -9 (voir syntaxe au dessus)
- ouvrir un troisième terminal
  - dans ce troisième terminal lancer la commande `top`
  - modifier l'affichage pour faire apparaître le PPID (taper f, puis sélectionner/déplacer avec les touches curseur. Revenir à l'affichage avec Esc)

- Enfin ouvrir des terminaux supplémentaires pour en avoir au moins 5 et lancer dans les terminaux :
  - 1 aucun processus
  - 1 avec nano
  - 1 avec python3
  - 1 avec python3 lançant p2.py
  - 1 avec top
- observer les processus et essayer de les tuer avec la commande kill à partir du premier terminal
- recommencer en relançant les processus et tuer les processus avec le terminal lançant top(puis commande k)

Jeretiens

- La commande `pstree`
  À noter que la commande `pstree` permet d'afficher les processus sous forme d'arborescence.
- La commande `top` La commande `top` permet de connaître en temps réel la liste des processus, classés par ordre décroissant de consommation de CPU.
  On ferme `top` par la combinaison de touches `Ctrl-C`. Si on repère alors un processus qui consomme beaucoup trop de ressources, on peut utiliser...
- La commande `kill`
  La commande `kill` permet de fermer un processus, en donnant son `PID` en argument.

3.5. ■ Activité 2 : *Exécution concurrente*

Les systèmes d'exploitation modernes sont capable d'exécuter plusieurs processus "en même temps". En réalité ces processus ne sont pas toujours exécutés "en même temps" mais plutôt "à tour de rôle". On parle d'exécution *concurrente* car les processus sont en concurrence pour obtenir l'accès au processeur chargé de les exécuter.

> **Remarque** : Sur un système multiprocesseur, il est possible d'exécuter de manière parallèle plusieurs processus, autant qu'il y a de processeurs. Mais sur un même processeur, un seul processus ne peut être exécuté à la fois.

On peut voir assez facilement cette exécution concurrente. Considérons les deux programmes Python suivants :

progA.py

🐍 **Script Python**

```python
import time

for i in range(100):
    print("programme A en cours, itération", i)
    time.sleep(0.01)  # pour simuler un traitement avec des calculs
```

progB.py

🐍 **Script Python**

```python
import time

for i in range(100):
    print("programme B en cours, itération", i)
    time.sleep(0.01)  # pour simuler un traitement avec des calculs
```

En ouvrant un Terminal, on peut lancer simultanément ces deux programmes avec la commande

Bash

```bash
$ python progA.py & python progB.py &
```

> Le caractère `&` qui suit une commande permet de lancer l'exécution en arrière plan et de rendre la main au terminal.

Le shell indique alors dans la console les *PID* des processus correspondant à l'exécution de ces deux programmes puis on constate grâce aux affichages que le système d'exploitation alloue le processeur aux deux programmes *à tour de rôle* :

```
~$ ps -aef
UID          PID    PPID  C STIME TTY          TIME CMD
user           1       0  0 19:50 ?        00:00:00 /cocalc/bin/tini -v -g -- sh -c env -i /cocalc/init/init.sh $COC
user           7       1  0 19:50 ?        00:00:00 sh -c env -i /cocalc/init/init.sh $COCALC_PROJECT_ID $KUCALC_IMA
user           8       7  4 19:50 ?        00:00:15 node --optimize-for-size --gc-interval=100000 --always-compact /
user          20       8  0 19:50 ?        00:00:00 sshd: /usr/sbin/sshd -D -p 2222 -h /tmp/.cocalc/ssh_host_rsa_key
user         867       8  0 19:51 pts/0    00:00:00 /bin/bash
user         871       8  0 19:51 pts/1    00:00:00 /bin/bash
user        1199     867  0 19:57 pts/0    00:00:00 ps -aef
~$
```

3.6. Accès concurrents aux ressources

Une **ressource** est une entité dont a besoin un processus pour s'exécuter. Les ressources peuvent être matérielles (processeur, mémoire, périphériques d'entrée/sortie, ...) mais aussi logicielles (variables).

Les différents processus se partagent les ressources, on parle alors d'*accès concurrents aux ressources*. Par exemple,

- les processus se partagent tous l'accès à la ressource "processeur"
- un traitement de texte et un IDE Python se partagent la ressource "clavier" ou encore la ressource "disque dur" (si on enregistre les fichiers), ...

- un navigateur et un logiciel de musique se partagent la ressource "carte son", ...

C'est le système d'exploitation qui est chargé de gérer les processus et les ressources qui leur sont nécessaires, en partageant leur accès au processeur. Nous allons voir comment tout de suite !

3.7. États d'un processus

Au cours de son existence, un processus peut se retrouver dans trois états :

- état **élu** : lorsqu'il est en cours d'exécution, c'est-à-dire qu'il obtient l'accès au processeur
- état **prêt** : lorsqu'il attend de pouvoir accéder au processeur
- état **en bloqué** : lorsque le processus est interrompu car il a besoin d'attendre une ressource quelconque (entrée/sortie, allocation mémoire, etc.)

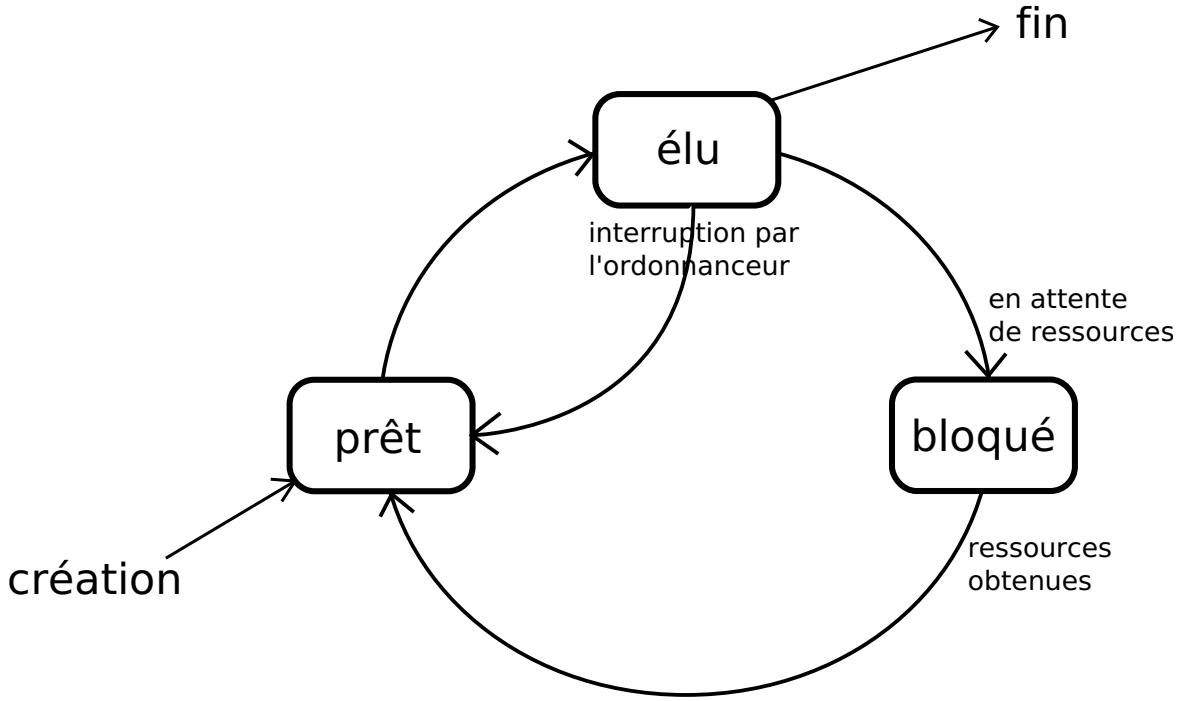
Il est important de comprendre que le processeur ne peut gérer qu'un seul processus à la fois : le processus *élu*.

En pratique, lorsqu'un processus est créé il est dans l'état *prêt* et attend de pouvoir accéder au processeur (d'être *élu*). Lorsqu'il est élu, le processus est exécuté par le processeur mais cette exécution peut être interrompue :

- soit pour laisser la main à un autre processus (qui a été élu) : dans ce cas, le processus de départ repasse dans l'état *prêt* et doit attendre d'être élu pour reprendre son exécution
- soit parce que le processus en cours a besoin d'attendre une ressource : dans ce cas, le processus passe dans l'état *bloqué*.

Lorsque le processus bloqué finit par obtenir la ressource attendue, il peut théoriquement reprendre son exécution mais probablement qu'un autre processus a pris sa place et est passé dans l'état élu. Auquel cas, le processus qui vient d'être "débloqué" repasse dans l'état *prêt* en attendant d'être à nouveau élu.

Ainsi, l'état d'un processus au cours de sa vie varie entre les états *prêt*, *élu* et *bloqué* comme le résume le schéma suivant :

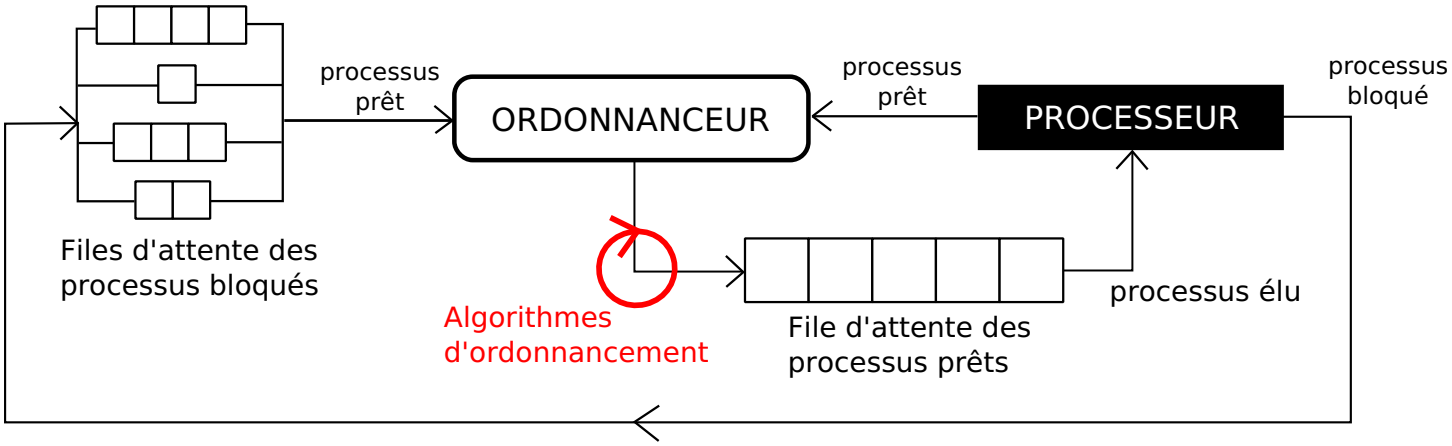


Lorsqu'un processus est interrompu, il doit pouvoir reprendre à l'endroit même où il a été interrompu. Pour cela, le système d'exploitation conserve pour chaque processus créé une zone mémoire (appelée PCB, pour *Process Control Bloc*, ou bloc de contrôle du processus) dans laquelle sont stockées les informations sur le processus : son PIB, son état, la valeur des registres lors de sa dernière interruption, la zone mémoire allouée par le processus lors de son exécution, les ressources utilisées par le processus (fichiers ouverts, connexions réseaux en cours d'utilisation, etc.).

3.8. Ordonnancement

C'est le système d'exploitation qui attribue aux processus leurs états *élu*, *prêt* et *bloqué*. Plus précisément, c'est l'**ordonnanceur** (un des composants du système d'exploitation) qui réalise cette tâche appelée *ordonnancement des processus*.

L'objectif de l'ordonnanceur est de choisir le processus à exécuter à l'instant \(t\) (le processus *élu*) et déterminer le temps durant lequel le processeur lui sera alloué.



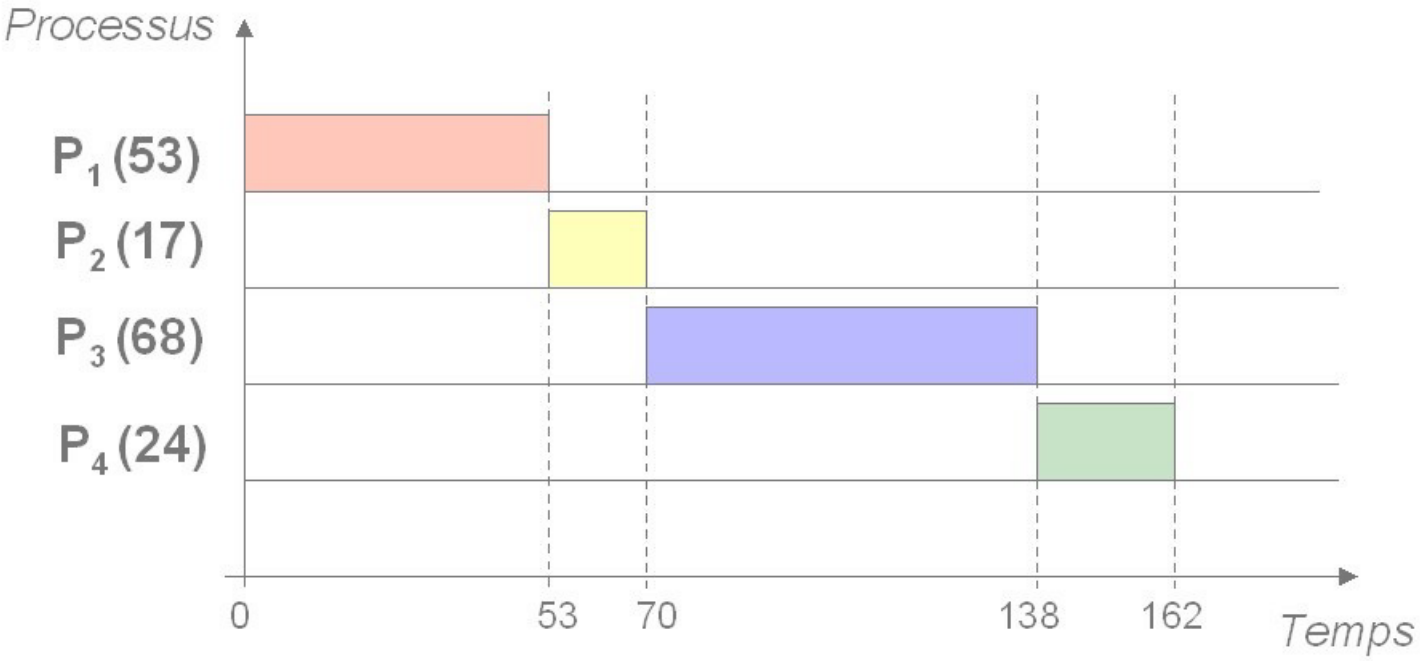
Ce choix est à faire parmi tous les processus qui sont dans l'état *prêt*, mais lequel sera élu ? et pour combien de temps ? Des algorithmes d'ordonnancement sont utilisés et il en existe plusieurs selon la stratégie utilisée. On en présente quelques-uns ci-dessous.

3.8.1. Ordonnancement First Come First Served (FCFS)

Principe : Les processus sont ordonnancés selon leur ordre d'arrivée ("premier arrivé, premier servi" en français)

Exemple : Les processus $P_1(53)$, $P_2(17)$, $P_3(68)$ et $P_4(24)$ arrivent dans cet ordre à $t=0$:

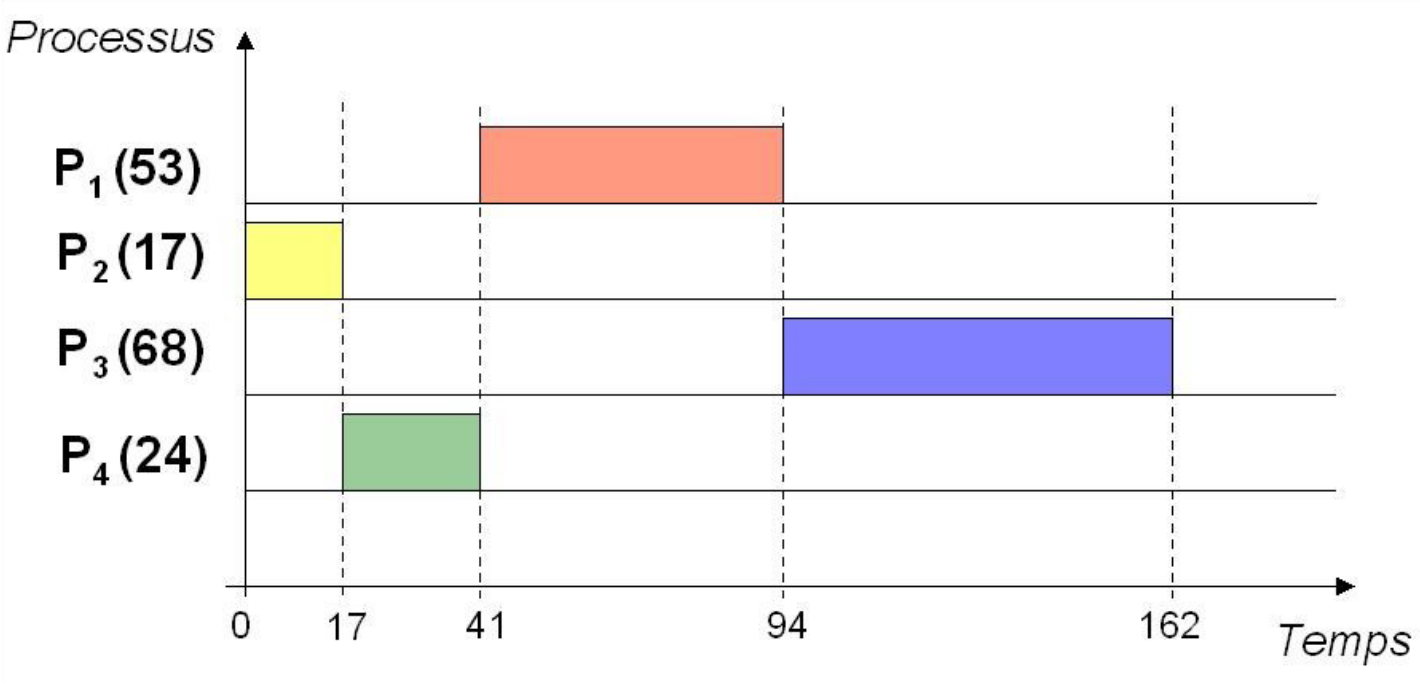
> Cela signifie que P_1, P_2, P_3 et P_4 ont besoin de respectivement 53, 17, 68 et 24 unités de temps pour s'exécuter.



3.8.2. Ordonnancement Shortest Job First (SJF)

Principe : Le processus dont le temps d'exécution est le plus court est ordonnancé en premier.

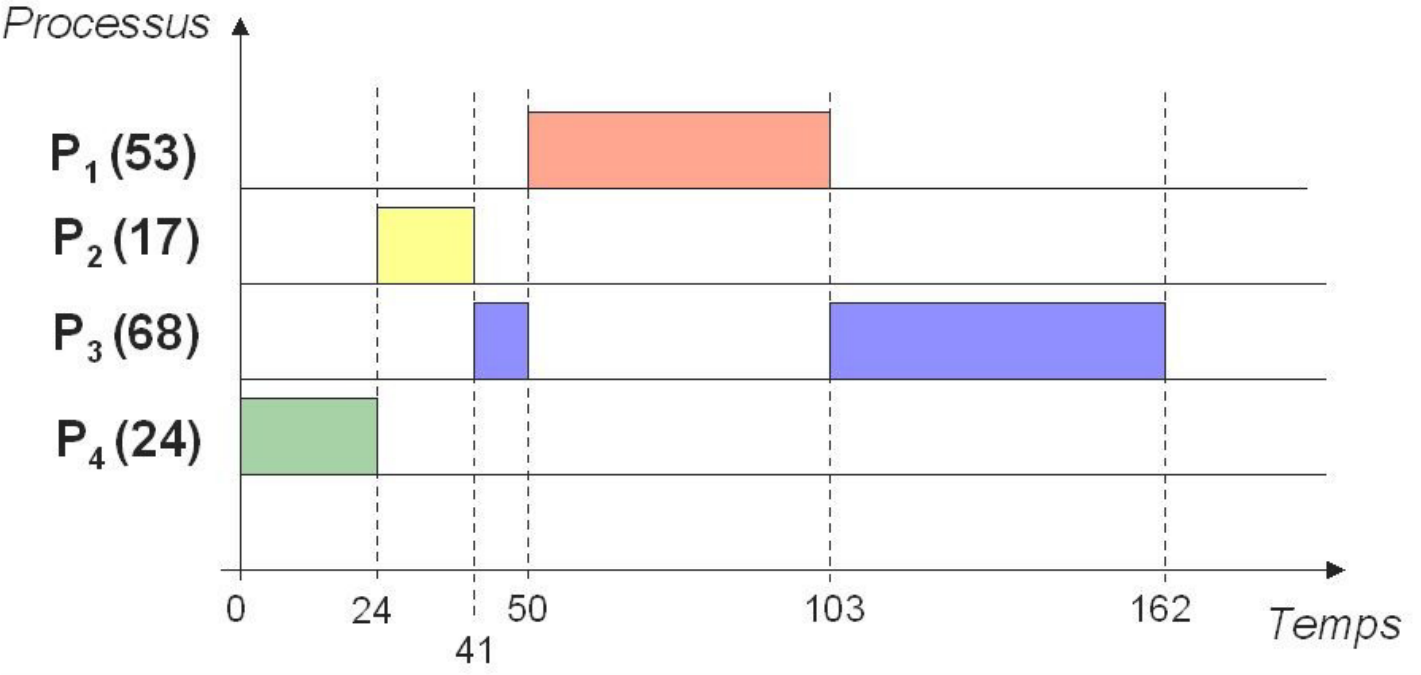
Exemple : P_1, P_2, P_3 et P_4 arrivent à $t=0$:



3.8.3. Ordonnancement Shortest Remaining Time (SRT)

Principe : Le processus dont le temps d'exécution restant est le plus court parmi ceux qui restent à exécuter est ordonnancé en premier.

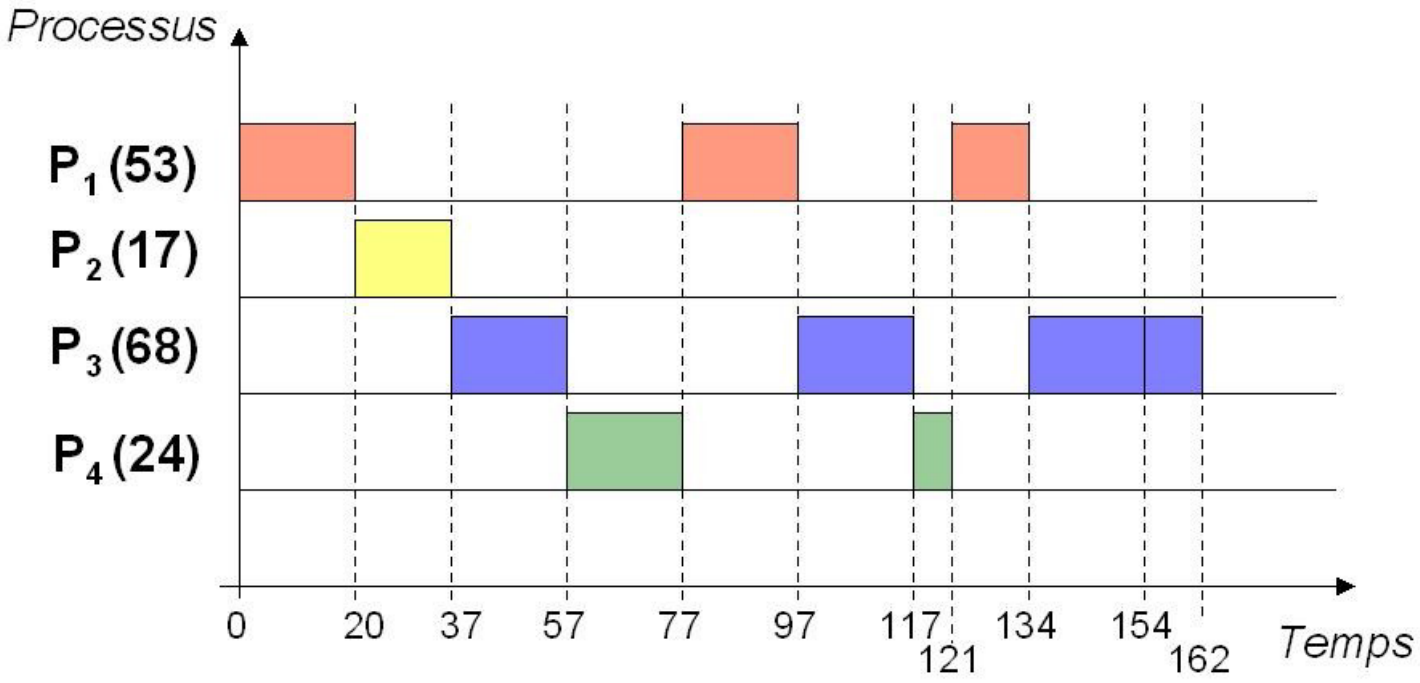
Exemple : \(P_3\) et \(P_4\) arrivent à \(t=0\) ; \(P_2\) à \(t=20\) ; \(P_1\) à \(t=50\) :



3.8.4. Ordonnancement temps-partagé (Round-Robin)

Principe : C'est la politique du tourniquet : allocation du processeur par tranche (= quantum \(q\)) de temps.

Exemple : quantum \(q = 20\) et \(n = 4\) processus

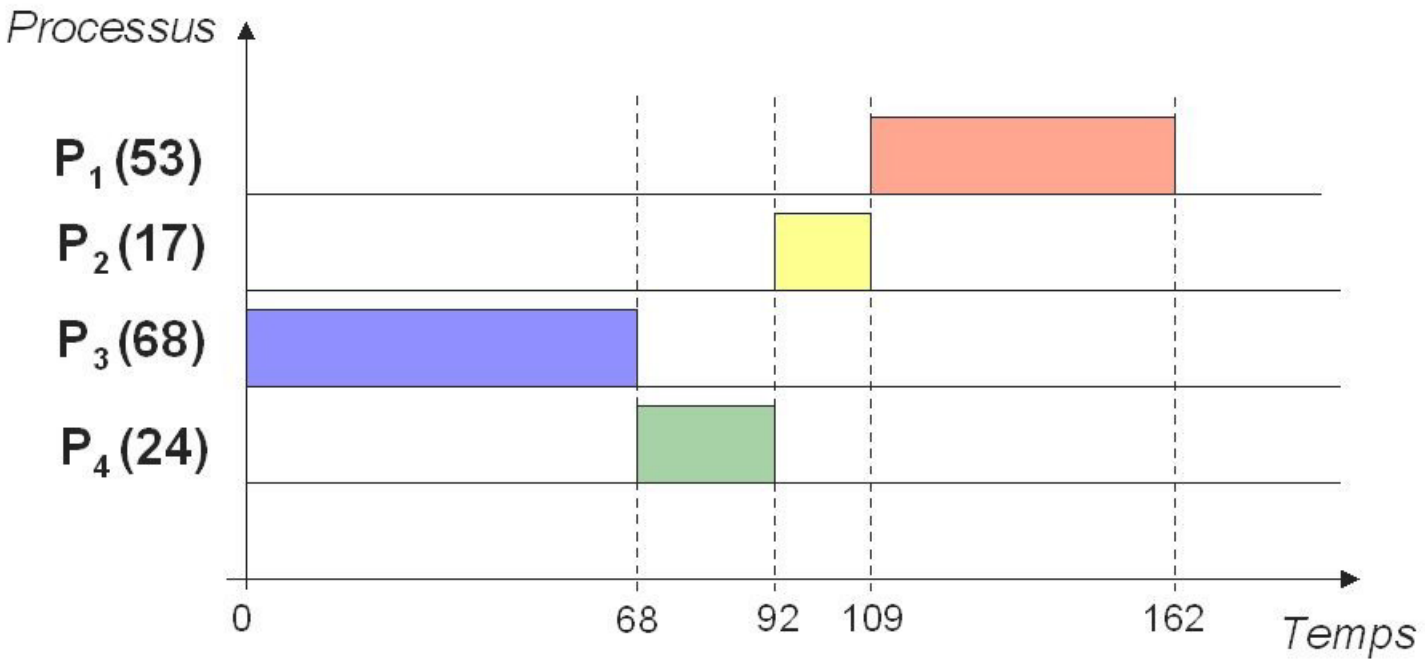


Dans ce cas, s'il y a \(n\) processus, chacun d'eux obtient le processeur au bout de \((n-1)\times q\) unités de temps au plus

3.8.5. Ordonnancement à priorités statiques

Principe : Allocation du processeur selon des priorités *statiques* (= numéros affectés aux processus pour toute la vie de l'application)

Exemple : priorités \((P_1, P_2, P_3, P_4) = (3, 2, 0, 1)\) où la priorité la plus forte est 0 (attention, dans certains systèmes c'est l'inverse : 0 est alors la priorité la plus faible)



4. Problèmes liés à l'accès concurrent aux ressources

Les processus se partagent souvent une ou plusieurs ressources, et cela peut poser des problèmes.

Pour info : Problèmes de synchronisation : illustration avec Python

4.1. Problèmes de synchronisation : illustration avec Python

4.1.1. Exemple d'une variable partagée

Prenons l'exemple d'une variable (= ressource logicielle) partagée entre plusieurs processus. Plus précisément, considérons un programme de jeu multi-joueur dans lequel une variable `nb_pions` représente le nombre de pions disponibles pour tous les joueurs.

Une fonction `prendre_un_pion()` permet de prendre un pion dans le tas commun de pions disponibles, s'il reste au moins un pion évidemment.

On va se mettre dans la situation où il ne reste plus qu'un pion dans le tas commun et on suppose que deux joueurs utilisent la fonction `prendre_un_pion()`, ce qui conduit à la création de deux processus `p1` et `p2`, chacun correspondant à un joueur.

Avec Python, on peut utiliser le module `multiprocessing` pour créer des processus. Le programme Python `pions.py` suivant permet de réaliser la situation de jeu décrite :

🐍 **Script Python**

```python
from multiprocessing import Process, Value
import time

def prendre_un_pion(nombre):
    if nombre.value >= 1:
        time.sleep(0.001)  # pour simuler un traitement avec des calculs
        temp = nombre.value
        nombre.value = temp - 1  # on décrémente le nombre de pions

if __name__ == '__main__':
    # création de la variable partagée initialisée à 1
    nb_pions = Value('i', 1)
    # on crée deux processus
    p1 = Process(target=prendre_un_pion, args=[nb_pions])
    p2 = Process(target=prendre_un_pion, args=[nb_pions])
    # on démarre les deux processus
    p1.start()
    p2.start()
    # on attend la fin des deux processus
    p1.join()
    p2.join()
    print("nombre final de pions :", nb_pions.value)
```

Explications :

- Le `if __name__ = '__main'__:` permet de ne créer qu'une seule fois les processus `p1` et `p2` qui suivent (c'est nécessaire sous Windows, pas sous GNU/Linux car la création des processus ne se fait pas de la même manière, mais cela reste conseillé ne serait-ce que pour des raisons de compatibilité), on n'en dira pas davantage ici car cela dépasse le niveau de ce cours.
- On a utilisé la classe `Process` du module `multiprocessing` pour instancier deux processus `p1` et `p2`.
  - L'argument `target` est le nom de la fonction qui sera exécutée par le processus : ici les deux processus doivent exécuter la même fonction `prendre_un_pion()`
  - L'argument `args` est une liste des arguments passés à la fonction cible : ici il s'agit de la variable `nb_pions` qui est partagée par les deux processus.
- Par défaut, deux processus ne partagent pas de données en mémoire : on ne peut pas donc pas utiliser `nb_pions` comme une variable globale. Il faut utiliser la classe `Value` du module `multiprocessing` pour créer `nb_pions` dans une mémoire partagée entre les processus. L'argument `'i'` indique que `nb_pions` est un entier (signé) et le deuxième argument est la valeur initiale de la variable, ici 1.
- La fonction `prendre_un_pion()` prend un nombre en paramètre et décrémente sa valeur d'une unité si le nombre est au moins égal à 1.
  - Lors de l'exécution de la fonction par les deux processus, l'argument en question sera l'objet `nb_pions` de la classe `Value` et on accède à sa valeur avec l'attribut `value`.
  - On a ajouté une temporisation permettant de simuler d'autres calculs qui pourraient avoir lieu (par exemple, des instructions de mise à jour du nombre de pions des joueurs)
- Les dernières lignes permettent de démarrer les deux processus et attendre qu'ils soient terminés pour afficher la valeur finale de `nb_pions`.

Si on exécute ce programme, les deux processus `p1` et `p2` sont exécutés et on s'attend au comportement suivant (en supposant qu'il ne reste qu'un seule pion dans le tas commun) :

- l'un des deux est élu en premier, par exemple `p1`, et exécute la fonction `prendre_un_pion()`, le nombre de pions est égal à 1 donc `nb_pions` est décrémenté d'une unité et prend donc la valeur 0, le processus `p1` est terminé ;
- le processus `p2`, qui était en attente, est ensuite élu, et comme le nombre de pions est désormais égal à 0 rien ne se passe et `p2` termine.

Ainsi, le premier joueur a pu prendre le pion restant et le second s'est retrouvé coincé, et la valeur finale de `nb_pions` vaut 0.

Et pourtant, il est tout à fait possible que les choses ne se passent pas ainsi ! En effet, en exécutant plusieurs fois le programme `pions.py` dans un terminal, on obtient parfois une valeur finale égale à 0 et parfois égale à -1 :

```
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : 0
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : 0
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions.py
nombre final de pions : -1
```

C'est un résultat très perturbant non ? Expliquons pourquoi !

Pour cela, on peut ajouter quelques instructions d'affichage pour suivre ce qu'il se passe. On obtient le script `pions_v2.py` suivant :

🐍 **Script Python**

```python
from multiprocessing import Process, Value
import time

def prendre_un_pion(nombre, numero_processus):
    print(f"début du processus {numero_processus}")
    if nombre.value >= 1:
        print(f"processus {numero_processus} : étape A")
        time.sleep(0.001)  # pour simuler un traitement avec des calculs
        print(f"processus {numero_processus} : étape B")
        temp = nombre.value
        nombre.value = temp - 1  # on décrémente le nombre de pions
    print(f"nombre de pions restants à la fin du processus {numero_processus} : {nombre.value}")

if __name__ == '__main__':
    # création de la variable partagée initialisée à 1
    nb_pions = Value('i', 1)
    # on crée deux processus
    p1 = Process(target=prendre_un_pion, args=[nb_pions, 1])
    p2 = Process(target=prendre_un_pion, args=[nb_pions, 2])
    # on démarre les deux processus
    p1.start()
```

```
    p2.start()
    # on attend la fin des deux processus
    p1.join()
    p2.join()
    print("nombre final de pions :", nb_pions.value)
```

Explications :

- Lors de la création des processus, on passe un deuxième argument à la fonction `prendre_un_pion()`, le numéro du processus : 1 pour `p1` et 2 pour `p2`.
- Cela permet d'afficher dans cette fonction le numéro du processus à des endroits stratégiques : au début, à l'entrée dans le `if`, juste avant de décrémenter le nombre de pions et à la fin du processus.

En exécutant `pions_v2.py` dans un terminal, on obtient ce genre de choses :

```
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions_v2.py
début du processus 1
début du processus 2
processus 1 : étape A
processus 2 : étape A
processus 1 : étape B
processus 2 : étape B
nombre de pions restants à la fin du processus 2 : -1
nombre de pions restants à la fin du processus 1 : 0
nombre final de pions : -1
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions_v2.py
début du processus 1
processus 1 : étape A
processus 1 : étape B
début du processus 2
nombre de pions restants à la fin du processus 1 : 0
nombre de pions restants à la fin du processus 2 : 0
nombre final de pions : 0
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions_v2.py
début du processus 1
processus 1 : étape A
début du processus 2
processus 1 : étape B
processus 2 : étape A
nombre de pions restants à la fin du processus 1 : 0
processus 2 : étape B
nombre de pions restants à la fin du processus 2 : -1
nombre final de pions : -1
```

Analysons la première exécution du programme.

- le processus `p1` est élu en premier (affichage de "début du processus 1") mais est de suite interrompu par l'ordonnanceur qui élit `p2` (affichage de "début du processus 2")

- puis c'est de nouveau le processus `p1` qui a la main et il rentre dans le `if` (affichage de "processus 1 : étape A") mais est interrompu à nouveau et l'ordonnanceur donne la main à `p2` qui rentre aussi dans le `if` (affichage de "processus 2 : étape A") : en effet, à ce stade le nombre de pions n'a pas encore été décrémenté par `p1` car il a été interrompu avant l'étape B, et donc `p2` a pu entrer dans le `if` puisque la condition `nombre.value >= 1` est toujours vraie à ce moment là !!
- ensuite le processeur est alloué alternativement à `p1` et `p2` (voir les affichages restants) mais le mal est fait puisque les deux processus sont désormais chacun entrés dans le `if`, ils vont chacun décrémenter le nombre de pions d'une unité et chacun des deux joueurs aura pioché un pion alors qu'il n'y en avait qu'un seul au départ !

Vous remarquerez que la troisième exécution du programme met en évidence le même problème car les deux processus ont chacun pu entrer dans le `if`, même si l'ordre des instructions exécutées après n'est pas tout à fait le même.

Si on analyse la seconde exécution du programme qui donne le comportement souhaité, on constate que `p1` a eu suffisamment de temps pour décrémenter le nombre de pions (qui vaut désormais 0) *avant* que `p2` ne fasse le test `nombre.value >= 1` et se rende compte que cette condition est fausse. Dans ce cas, seul le premier joueur a pas pu piocher un pion.

Heureusement, on peut éviter le problème mis en évidence dans l'exemple précédent.

4.1.2. Comment éviter les problèmes de synchronisation ?

On va utiliser ce qu'on appelle un **verrou** : un verrou est objet partagé entre plusieurs processus mais qui garantit qu'un seul processus accède à une ressource à un instant donné.

Concrètement, un verrou peut être acquis par les différents processus, et le premier à faire la demande acquiert le verrou. Si le verrou est détenu par un autre processus, alors tout autre processus souhaitant l'obtenir est bloqué jusqu'à ce qu'il soit libéré.

Le module `multiprocessing` de Python propose un objet `Lock()` correpondant à un verrou. Deux méthodes sont utilisées :

- la méthode `.acquire()` permet de demander le verrou (le processus faisant la demande est bloqué tant qu'il ne l'a pas obtenu)
- la méthode `.release()` permet de libérer le verrou (il pourra alors être obtenu par un autre processus qui en fait la demande)

On peut alors régler le problème de l'exemple précédent avec le script `pions_v3.py` suivant dans lequel on a laissé les affichages pour bien suivre :

🐍 **Script Python**

```python
from multiprocessing import Process, Value, Lock
import time


def prendre_un_pion(v, nombre, numero_processus):
    print(f"début du processus {numero_processus}")
    v.acquire()  # acquisition du verrou
    if nombre.value >= 1:
        print(f"processus {numero_processus} : étape A")
        time.sleep(0.001)
        print(f"processus {numero_processus} : étape B")
        temp = nombre.value
        nombre.value = temp - 1
    v.release()  # verrou libéré
    print(f"nombre de pions restants à la fin du processus {numero_processus} : {nombre.value}")

if __name__ == '__main__':
    # création de la variable partagée initialisée à 1
    nb_pions = Value('i', 1)
    # verrou partagé par les deux processus
    verrou = Lock()
    # on crée deux processus
    p1 = Process(target=prendre_un_pion, args=[verrou, nb_pions, 1])
    p2 = Process(target=prendre_un_pion, args=[verrou, nb_pions, 2])
    # on démarre les deux processus
    p1.start()
    p2.start()
    # on attend la fin des deux processus
    p1.join()
    p2.join()
    print("nombre final de pions :", nb_pions.value)
```

En exécutant (plusieurs fois) ce script dans un terminal on constate que le nombre final de pions est toujours égal à 0.

```
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python pions_v3.py
début du processus 1
processus 1 : étape A
début du processus 2
processus 1 : étape B
nombre de pions restants à la fin du processus 1 : 0
nombre de pions restants à la fin du processus 2 : 0
nombre final de pions : 0
```

Avant de faire le test du `if`, le processus essaye d'acquérir le verrou avec `v.acquire()`. Dès qu'il est acquis, le processus a la garantie qu'il est le seul à pouvoir exécuter le code jusqu'à l'instruction `v.release()`. Cette portion de code protégée s'appelle une *section critique*. Cela ne veut pas dire que le processus détenant le verrou ne peut pas être

interrompu, mais il ne le sera pas par un processus qui est essaie d'acquérir le même verrou.

🐍 **Script Python**

```
def prendre_un_pion(v, nombre, numero_processus):
    v.acquire()
    # début section critique
    if nombre.value >= 1:
        time.sleep(0.001)
        temp = nombre.value
        nombre.value = temp - 1
    # fin de la section critique
    v.release()
```

Si vous analysez l'affichage précédent dans le terminal, on voit d'ailleurs que `p1` est entré en section critique (affichage "processus 1 : étape A") mais est interrompu, puis c'est `p2` qui a la main (affichage "début processus 2") mais il va se retrouver bloquer à l'instruction `v.acquire()` puisque c'est `p1` qui détient le verrou. Lorsque `p1` reprendra la main, il pourra exécuter ses instructions jusqu'à `v.release()` sans être interrompu par `p2` (alors `nb_pions` sera décrémenté d'une unité). Lorsque `p1` libère le verrou, `p2` pourra alors l'obtenir, exécuter sa section critique et constater que la condition `nombre.value >= 1` est fausse : le deuxième joueur ne pourra alors pas prendre de pion.

Nous terminons en voyant que l'utilisation de verrous n'est pas sans risque car elle peut engendrer des problèmes d'interblocage.

4.2. Risque d'interblocage

Les interblocages (*deadlock* en anglais) sont des situations de la vie quotidienne. L'exemple classique est celui du carrefour avec priorité à droite où chaque véhicule est bloqué car il doit laisser le passage au véhicule à sa droite.

En informatique l'**interblocage** peut également se produire lorsque plusieurs processus concurrents s'attendent mutuellement. Ce scénario peut se produire lorsque plusieurs ressources sont partagées par plusieurs processus et l'un d'entre eux possède indéfiniment une ressource nécessaire pour un autre.

Ce phénomène d'*attente circulaire*, où chaque processus attend une ressource détenue par un autre processus, peut être provoquée par l'utilisation de *plusieurs* verrous.

pour info : illustration

Considérons le script `interblocage.py` suivant dans lequel on a créé deux verrous `v1` et `v2` utilisés par deux fonctions `f1` et `f2` exécutées respectivement par deux processus `p1` et `p2`. Le processus `p1` essaie d'acquérir d'abord `v1` puis `v2` tandis que le processus `p2` essaie de les acquérir dans l'ordre inverse.

🐍 **Script Python**

```python
from multiprocessing import Process, Lock
import time
import os

def f1(v1, v2):
    print("PID du processus 1:", os.getpid())
    for i in range(100):
        time.sleep(0.001)
        v1.acquire()
        v2.acquire()
        print("processus 1 en cours, itération ", i)
        v2.release()
        v1.release()

def f2(v1, v2):
    print("PID du processus 2:", os.getpid())
    for i in range(100):
        time.sleep(0.001)
        v2.acquire()
        v1.acquire()
        print("processus 2 en cours, itération ", i)
        v1.release()
        v2.release()

if __name__ == '__main__':
    # création de deux verrous
    v1 = Lock()
    v2 = Lock()
    # création de deux processus
    p1 = Process(target=f1, args=[v1, v2])
    p2 = Process(target=f2, args=[v1, v2])
    # on démarre les deux processus
    p1.start()
    p2.start()
    # on attend la fin des deux processus
    p1.join()
    p2.join()
```

Si on exécute ce programme, il y a de grandes chances de se retrouver bloqué. Par exemple, dans le cas de l'exécution suivante :

- le processus `p1` est élu : il s'exécute jusqu'à l'acquisition de `v1` mais avant la tentative d'acquisition de `v2`, puis est interrompu
- le processus `p2` est à son tour élu : il s'exécute et acquiert `v2` qui est toujours libre, puis bloque sur l'acquisition de `v1` (qui est détenu par `p1`).
- le processus `p1` reprend la main et bloque sur l'acquisition de `v2` (détenu par `p2`).

Chaque processus détient un verrou et attend l'autre : ils sont en interblocage et l'attente est infinie.

On peut lancer (plusieurs fois si nécessaire) le script à partir du terminal et constater que l'interblocage a lieu très souvent.

```
(base) terminale@mounier-01:~/Documents/TNSI/processus$ python interblocage.py
PID du processus 1: 8099
PID du processus 2: 8100
processus 1 en cours, itération  0
processus 2 en cours, itération  0
processus 1 en cours, itération  1
processus 2 en cours, itération  1
processus 1 en cours, itération  2
processus 2 en cours, itération  2
processus 1 en cours, itération  3
processus 2 en cours, itération  3
processus 1 en cours, itération  4
processus 2 en cours, itération  4
processus 1 en cours, itération  5
processus 2 en cours, itération  5
processus 1 en cours, itération  6
processus 2 en cours, itération  6
processus 1 en cours, itération  7
processus 2 en cours, itération  7
processus 1 en cours, itération  8
processus 2 en cours, itération  8
processus 1 en cours, itération  9
processus 2 en cours, itération  9
processus 1 en cours, itération  10
processus 2 en cours, itération  10
processus 1 en cours, itération  11
```

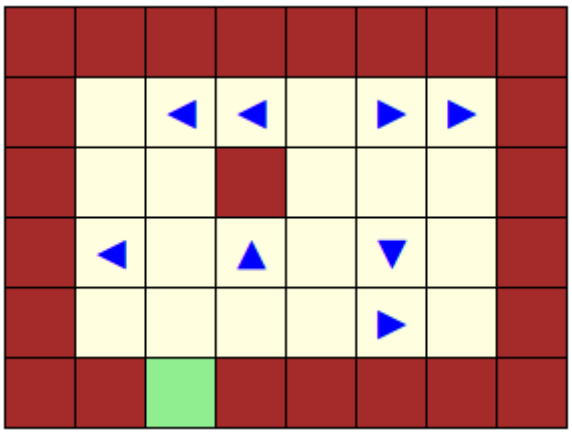
Il n'y a alors pas d'autres choix que d'interrompre les processus en interblocage, par exemple avec la commande `kill`.

Cependant, ce problème a lieu ici car les deux processus essaie d'acquérir les verrous dans l'ordre contraire. Si l'ordre d'acquisition est le même pour les processus, le problème n'a plus lieu (n'hésitez pas à tester !).

De manière générale, dans des problèmes complexes les situations d'interblocage sont difficiles à détecter et il se peut très bien que le programme se comporte bien pendant toute une phase de tests mais bloque lors d'une exécution ultérieure puisque l'on ne peut pas prévoir l'ordonnancement des processus.

4.3. Simulation d'interblocage⚓

Robosomes créé par Alain BUSSER , Sébastien HOARAU (Voir ici : Robosomes - IREM de la réunion



Le jeu robosomes se joue à un seul joueur sur une grille rectangulaire. Chaque case peut être

- soit vide
- soit couverte par un obstacle fixe (en noir comme aux mots croisés)
- soit couverte d'un pion pouvant bouger, appelé robot

Chaque robot peut être tourné dans l'une des quatre directions cardinales ◀▲▶▼. Les robots peuvent bouger tous en même temps de l'une des façons suivantes :

- G : tous les robots tournent vers leur gauche (de 90°) en même temps
- D : tous les robots tournent vers leur droite (de 90°) en même temps
- A : les robots qui peuvent avancer d'une case, le font. Un robot peut avancer d'une case s'il n'y a pas d'obstacle sur cette case et si aucun robot ne s'apprête à aller sur cette case.

Les cases du bord de la grille sont toutes couvertes d'obstacles fixes, à l'exception de l'une d'entre elles appelée « sortie ». Lorsqu'un robot est sur la case de sortie, tourné vers l'extérieur de la grille, il quitte le jeu et n'est plus soumis aux ordres donnés. Le but du jeu est de faire sortir tous les robots de la grille, en écrivant un mot dans l'alphabet A,G,D, appelé programme et que les robots interpréteront comme décrit ci-dessus.

Voici quelques exemples :

Un premier exemple pour se mettre en route.

Interblocage

Interblocage 1

Interblocage 2

Interblocage 3

Interblocage 4

Interblocage perpétuel ou pas

Interblocage 8

5. Et pour les systèmes multiprocesseurs ?

Les ordinateurs actuels possèdent généralement plusieurs processeurs, ce qui permet à plusieurs processus d'être exécutés parallèlement : un par processeur. Ce parallélisme permet bien évidemment une plus grande puissance de calcul.

Pour répartir les différents processus entre les différents processeurs, on distingue deux approches :

- l'approche *partitionnée* : chaque processeur possède un ordonnanceur particulier et les processus sont répartis entre les différents ordonnanceurs
- l'approche *globale* : un ordonnanceur global est chargé de déterminer la répartition des processus entre les différents processeurs

L'ordonnancement des processus des systèmes d'exploitation actuels est bien plus complexe que les quelques algorithmes évoqués dans ce cours, et cela dépasse largement le cadre du programme de NSI. Si vous souhaitez en savoir plus, voici néanmoins une vidéo intéressante (en français) sur l'ordonnancement du noyau Linux : https://www.youtube.com/watch?v=uCGe5WWd1OI&t=195s&ab_channel=Vitonimal.

6. Bilan

Jeretiens

- Un programme en cours d'exécution s'appelle un *processus*. Les systèmes d'exploitation récents permettent d'exécuter plusieurs processus simultanément.
- En réalité, ces processus sont exécutés *à tour de rôle* par le système d'exploitation qui est chargé d'allouer à chacun d'eux les ressources dont il a besoin en termes de mémoire, entrées-sorties ou temps d'accès au processeur, et de s'assurer que les processus ne se gênent pas les uns les autres.
- Au cours de leur vie, les processus varient entre trois états : *élu* si le processus est exécuté par le processeur, *prêt* si le processus est prêt à être exécuté, et *bloqué* si le processus est en attente d'une ressource.
- C'est l'*ordonnanceur* qui est chargé de définir l'ordre dans lequel les processus doivent être exécutés par le processeur. Ce choix se fait grâce à des algorithmes d'ordonnancement.
- Les processus se partagent les différentes ressources, on parle d'*accès concurrent* aux ressources. Ce partage des ressources n'est pas sans risque et peut conduire à des problèmes de synchronisation. Ces problèmes peuvent être évités en utilisant un *verrou*, qui permet à un processus de ne pas être interrompu dans sa section critique par un autre processus demandant le même verrou.
- L'utilisation de plusieurs verrous peut entraîner des *interblocages*, c'est-à-dire des situations où chaque processus attend une ressource détenue par un autre, conduisant à une attente cyclique infinie. L'ordre d'acquisition des verrous est important mais pas toujours évident à écrire dans le cas de problèmes complexes.

7. Exercice 3 : Algorithmes d'ordonnancement

Soient les différents processus suivants :

Processus	Date d'arrivée	Durée de traitement
\(P_1\)	0	3
\(P_2\)	1	6
\(P_3\)	3	4

Processus	Date d'arrivée	Durée de traitement
P_4	6	5
P_5	8	2

7.1. Application de plusieurs algorithmes

Q1 : Donnez le diagramme de Gantt pour l'exécution de ces différents processus en utilisant successivement les algorithmes FCFS, RR (quantum = 2 unités de temps) et SRT.

7.2. Performances des algorithmes d'ordonnancement

On définit les métriques suivantes :

- le **temps de séjour (ou d'exécution)** (ou de rotation) d'un processus : c'est la différence entre la date de fin d'exécution et la date d'arrivée : $T_{\text{sej}} = \text{date fin d'exécution} - \text{date d'arrivée}$
- le **temps d'attente** d'un processus : c'est la différence entre le temps de séjour et la durée du processus : $T_{\text{att}} = T_{\text{sej}} - \text{durée du processus}$
- le **rendement** d'un processus : c'est le quotient entre la durée du processus et le temps de séjour : $\text{rendement} = \dfrac{\text{durée du processus}}{T_{\text{sej}}}$

Q2 : Pour chacun des trois algorithmes, calculez le temps de séjour, le temps d'attente et le rendement de chaque processus.

Q3 : Quel vous semble être le *meilleur* des trois algorithmes dans notre exemple ? Expliquer.

Sujet BAC 13 : Gestion Processus

1. 2021, France sujet 1

Exo

Partie A

Cette partie est un questionnaire à choix multiples (QCM). Pour chacune des questions, une seule des quatre réponses est exacte. Le candidat indiquera sur sa copie le numéro de la question et la lettre correspondant à la réponse exacte. Aucune justification n'est demandée. Une réponse fausse ou une absence de réponse n'enlève aucun point.

Question 1

Parmi les commandes ci-dessous, laquelle permet d'afficher les processus en cours d'exécution ?

📋 **Texte**

a. dir
b. ps
c. man
d. ls

Question 2

Quelle abréviation désigne l'identifiant d'un processus dans un système d'exploitation de type UNIX ?

📋 **Texte**

a. PIX
b. SIG
c. PID
d. SID

Question 3

Comment s’appelle la gestion du partage du processeur entre différents processus ?

📋 **Texte**

a. L’interblocage
b. L’ordonnancement
c. La planification
d. La priorisation

Question 4

Quelle commande permet d’interrompre un processus dans un système d’exploitation de type UNIX ?

📋 **Texte**

a. stop
b. interrupt
c. end
d. kill

Correction

1.b
2.c
3.b
4.d

Partie B

Question 1

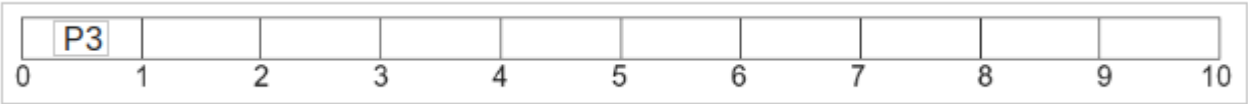
Un processeur choisit à chaque cycle d'exécution le processus qui doit être exécuté. Le tableau ci-dessous donne pour trois processus P1, P2, P3 :

- la durée d'exécution (en nombre de cycles),
- l'instant d'arrivée sur le processeur (exprimé en nombre de cycles à partir de 0),
- le numéro de priorité.

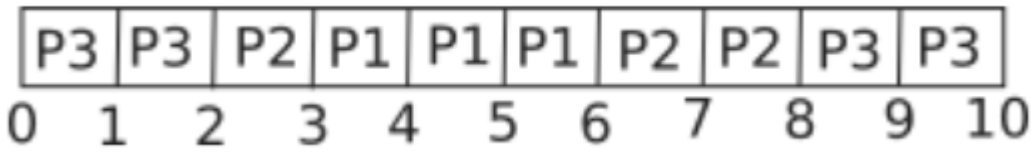
Le numéro de priorité est d'autant plus petit que la priorité est grande. On suppose qu'à chaque instant, c'est le processus qui a le plus petit numéro de priorité qui est exécuté, ce qui peut provoquer la suspension d'un autre processus, lequel reprendra lorsqu'il sera le plus prioritaire.

Processus	Durée d'exécution	Instant d'arrivée	Numéro de priorité
P1	3	3	1
P2	3	2	2
P3	4	0	3

Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle.



Correction



Question 1

On suppose maintenant que les trois processus précédents s'exécutent et utilisent une ou plusieurs ressources parmi R1, R2 et R3. Parmi les scénarios suivants, lequel provoque un interblocage ? Justifier.

Scénario 1
P1 acquiert R1
P2 acquiert R2
P3 attend R1
P2 libère R2
P2 attend R1
P1 libère R1

Scénario 2
P1 acquiert R1
P2 acquiert R3
P3 acquiert R2
P1 attend R2
P2 libère R3
P3 attend R1

Scénario 3
P1 acquiert R1
P2 acquiert R2
P3 attend R2
P1 attend R2
P2 libère R2
P3 acquiert R2

Correction

Il s'agit du scénario 2 car nous nous retrouvons dans la situation où P1 possède R1 et attend R2 avant de pouvoir continuer et P3 possède R2 et attend R1 avant de pouvoir continuer.

Partie C

Dans cette partie, pour une meilleure lisibilité, des espaces sont placées dans les écritures binaires des nombres. Il ne faut pas les prendre en compte dans les calculs.
Pour chiffrer un message, une méthode, dite du masque jetable, consiste à le combiner avec une chaîne de caractères de longueur comparable.
Une implémentation possible utilise l'opérateur XOR (ou exclusif) dont voici la table de vérité :

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Dans la suite, les nombres écrits en binaire seront précédés du préfixe 0b.

Question 1

Pour chiffrer un message, on convertit chacun de ses caractères en binaire (à l'aide du format Unicode), et on réalise l'opération XOR bit à bit avec la clé.
Après conversion en binaire, et avant que l'opération XOR bit à bit avec la clé n'ait été effectuée, Alice obtient le message suivant :

`m = 0b 0110 0011 0100 0110`

a. Le message m correspond à deux caractères codés chacun sur 8 bits : déterminer quels sont ces caractères. On fournit pour cela la table ci-dessous qui associe à l'écriture hexadécimale d'un octet le caractère correspondant (figure 2). Exemple de lecture : le caractère correspondant à l'octet codé 4A en hexadécimal est la lettre J.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Correction

0100 0110 => 46 en hexa => caractère F
0110 0011 => 63 en hexa => caractère c

donc cF

Question 1.b

Pour chiffrer le message d'Alice, on réalise l'opération XOR bit à bit avec la clé suivante : `k = 0b 1110 1110 1111 0000`

Donner l'écriture binaire du message obtenu.

Correction

`0b 1000 1101 1011 0110`

Question 2.a

Dresser la table de vérité de l'expression booléenne suivante :

`(a XOR b) XOR b`

Correction

a	b	(a XOR b) XOR b
0	0	0
0	1	0
1	0	1
1	1	1

Question 2.b

Bob connaît la chaîne de caractères utilisée par Alice pour chiffrer le message. Quelle opération doit-il réaliser pour déchiffrer son message ?

Correction

On peut remarquer que (a xor b) xor b permet de retrouver a, donc si a correspond au message non chiffré et a xor b correspond au message chiffré, un (a xor b) xor b permet donc de retrouver le message non chiffré. Si on appelle m le message non chiffré, m’ le message chiffré et k la clé de chiffrement, un m’ xor k permettra de retrouver m.

2. 2021, France sujet annulé

d'après le sujet du bac NSI 2021

Cet exercice porte sur les systèmes d’exploitation : gestion des processus et des ressources.
Les parties A et B peuvent être traitées indépendamment

Partie A

Question 1

La commande ps suivie éventuellement de diverses options permet de lister les processus actifs ou en attente sur une machine. Sur une machine équipée du système d’exploitation GNU/Linux, la commande “ps -aef” permet d’obtenir la sortie suivante (extrait) :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	10:01	?	00:00:02	/sbin/init splash
root	4	2	0	10:01	?	00:00:00	[kworker/0:0H]
root	6	2	0	10:01	?	00:00:00	[mm_percpu_wq]
....	....	.....	....	.....	.....	.....	.....
bob	3383	1	0	10:25	?	00:00:00	sh -c /usr/bin/google-chrome-sta

a) Quelle est la particularité de l’utilisateur “root” ?

b) Quel est le processus parent du processus ayant pour PID 3383

Correction

Dans un bureau d’architectes, on dispose de certaines ressources qui ne peuvent être utilisées simultanément par plus d’un processus, comme l’imprimante, la table traçante, le modem. Chaque programme, lorsqu’il s’exécute, demande l’allocation des ressources qui lui sont nécessaires. Lorsqu’il a fini de s’exécuter, il libère ses ressources.

Programme 1	Programme 2	Programme 3
demander(table traçante) demander (modem) exécution libérer (modem) libérer (table traçante)	demander (modem) demander (imprimante) exécution libérer (imprimante) libérer (modem)	demander (imprimante) demander (table traçante) exécution libérer (table traçante) libérer (imprimante)

Question 2

On appelle p1, p2 et p3 les processus associés respectivement aux programmes 1, 2 et 3.

a) Justifier qu'une situation d'interblocage peut se produire.

b) Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire.

Correction

2.a Pour passer à l’exécution, le Programme 1 a besoin du modem, mais le modem est déjà utilisé par le Programme 2. Le processus p1 est donc bloqué. Pour passer à l’exécution, le Programme 2 a besoin de l’imprimante, mais l’imprimante est déjà utilisée par le Programme 3. Le processus p2 est donc bloqué. Pour passer à l’exécution, le Programme 3 a besoin de la table traçante, mais la table traçante est déjà utilisée par le Programme 1. Le processus p3 est donc bloqué. Les 3 processus sont bloqués et ne pourront pas libérer les ressources attendues par les autres processus (la libération des ressources se faisant après l’exécution), nous sommes bien en situation d’interblocage.

2.b

Programme 3
demander (table traçante)
demander (imprimante)
exécution
libérer (imprimante)
libérer (table traçante)

Question 3

Supposons que le processus p1 demande la table traçante alors qu'elle est en cours d'utilisation par le processus p3. Parmi les états suivants, quel sera l'état du processus p1 tant que la table traçante n'est pas disponible :

a) élu
b) bloqué
c) prêt
d) terminé

Correction

bloqué

Partie B

Avec une ligne de commande dans un terminal sous Linux, on obtient l'affichage suivant :

```
UID        PID  PPID  C STIME TTY          TIME CMD
...
pi        6211   831  8 09:07 ?        00:01:16 /usr/lib/chromium-browser/chromium-browser-v7 --disable-quic --enable-tcp-fast-open --p
pi        6252  6211  0 09:07 ?        00:00:00 /usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi        6254  6252  0 09:07 ?        00:00:00 /usr/lib/chromium-browser/chromium-browser-v7 --type=zygote --ppapi-flash-path=/usr/lib
pi        6294  6211  4 09:07 ?        00:00:40 /usr/lib/chromium-browser/chromium-browser-v7 --type=gpu-process --field-trial-handle=1
pi        6300  6211  1 09:07 ?        00:00:16 /usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi        6467  6254  1 09:07 ?        00:00:11 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       11267  6254  2 09:12 ?        00:00:15 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       12035   836  0 09:13 ?        00:00:00 /usr/lib/libreoffice/program/oosplash --writer file:///home/pi/Desktop/mon_fichier.odt
pi       12073 12035  2 09:13 ?        00:00:15 /usr/lib/libreoffice/program/soffice.bin --writer file:///home/pi/Desktop/mon_fichier.c
pi       12253   831  1 09:13 ?        00:00:07 /usr/bin/python3 /usr/bin/sense_emu_gui
pi       20010  6211  1 09:21 ?        00:00:00 /usr/lib/chromium-browser/chromium-browser-v7 --type=utility --field-trial-handle=10758
pi       20029  6254 56 09:21 ?        00:00:28 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       20339  6254  4 09:21 ?        00:00:01 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       20343  6254  2 09:21 ?        00:00:00 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       20464  6211 17 09:22 ?        00:00:00 /proc/self/exe --type=utility --field-trial-handle=1075863133478894917,6306120996223181
pi       20488  6254 14 09:22 ?        00:00:00 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=1075
pi       20519   676  0 09:22 pts/0    00:00:00 ps -ef
```

La documentation Linux donne la signification des différents champs :

- UID : identifiant utilisateur effectif ;
- PID : identifiant de processus ;
- PPID : PID du processus parent ;
- C : partie entière du pourcentage d'utilisation du processeur par rapport au temps de vie des processus ;
- STIME : l'heure de lancement du processus ;
- TTY : terminal de contrôle
- TIME : temps d'exécution
- CMD : nom de la commande du processus

Question 1

Parmi les quatre commandes suivantes, laquelle a permis cet affichage ?

a) ls -l
b) ps –ef
c) cd ..
d) chmod 741 processus.txt

Correction

ps -ef

Question 2

Quel est l'identifiant du processus parent à l'origine de tous les processus concernant le navigateur Web (chromium-browser) ?

Correction

L'identifiant du processus parent à l'origine de tous les processus concernant le navigateur Web est 831

Question 3

Quel est l'identifiant du processus dont le temps d'exécution est le plus long ?

Correction

L'identifiant du processus dont le temps d'exécution est le plus long est 6211 (00:01:16)

3. 2021, France candidat libre

Extrait sujet BAC 2021

Cet exercice porte sur la gestion des processus par un système d'exploitation.

Partie A : Processus

La commande UNIX ps présente un cliché instantané des processus en cours d'exécution.
Avec l'option −eo pid,ppid,stat,command, cette commande affiche dans l'ordre l'identifiant du processus PID (process identifier), le PPID (parent process identifier), l'état STAT et le nom de la commande à l'origine du processus.

Les valeurs du champ STAT indique l’état des processus :

- R : processus en cours d’exécution
- S : processus endormi

Sur un ordinateur, on exécute la commande ps −eo pid,ppid,stat,command et on obtient un affichage dont on donne ci-dessous un extrait.

```
$ ps -eo pid,ppid,stat,command

      PID  PPID STAT COMMAND
      1    0    Ss   /sbin/init
      .... .... ..   ...
      1912 1908 Ss   Bash
      2014 1912 Ss   Bash
      1920 1747 Sl   Gedit
      2013 1912 Ss   Bash
      2091 1593 Sl   /usr/lib/firefox/firefox
      5437 1912 Sl   python programme1.py
      5440 2013 R    python programme2.py
      5450 1912 R+   ps -eo pid,ppid,stat,command
```

À l'aide de cet affichage, répondre aux questions ci-dessous.

Question 1

Quel est le nom de la première commande exécutée par le système d'exploitation lors du démarrage ?

Correction

La première commande exécutée par le système d’exploitation lors du démarrage est la commande init

Question 2

Quels sont les identifiants des processus actifs sur cet ordinateur au moment de l’appel de la commande ps ? Justifier la réponse.

Correction

Les processus actifs sont les processus ayant pour PID 5440 et 5450 (présence de l’indicateur R dans la colonne STAT pour ces 2 processus).

Question 3

Depuis quelle application a-t-on exécuté la commande ps ?
Donner les autres commandes qui ont été exécutées à partir de cette application.

Correction

La commande ps a été exécutée depuis l’application Bash (car le processus ps a pour PPID 1912 qui correspond au PID de Bash).
Deux autres processus Bash (PID 2014 et PID 2013) et un processus python programme1.py (PID 5437) ont été lancés depuis le processus Bash de PID 1912.

Question 4

Expliquer l'ordre dans lequel les deux commandes python programme1.py et python programme2.py ont été exécutées.

Correction

Le processus python programme1.py a un PID de 5437 alors que le processus python programme2.py a un PID de 5440. python programme1.py a été exécuté avant python programme2.py.

Question 5

Peut-on prédire que l'une des deux commandes python programme1.py et python programme2.py finira avant l’autre ?

Correction

Non, aucune prédiction n’est possible.

4. 2023, Polynésie, J1

Ordonnanceur avec liste d'attente

Un système est composé de 4 périphériques, numérotés de 0 à 3, et d'une mémoire, reliés entre eux par un bus auquel est également connecté un dispositif ordonnanceur. À l'aide d'un signal spécifique envoyé sur le bus, l'ordonnanceur sollicite à tour de rôle les périphériques pour qu'ils indiquent le type d'opération (lecture ou écriture) qu'ils souhaitent effectuer, et l'adresse mémoire concernée.

Un tour a lieu quand les 4 périphériques ont été sollicités. **Au début d'un nouveau tour, on considère que toutes les adresses sont disponibles en lecture et écriture.**

Si un périphérique demande l'écriture à une adresse mémoire **à laquelle on n'a pas encore accédé** pendant le tour, l'ordonnanceur répond `"OK"` et l'écriture a lieu. Si on a déjà demandé la lecture ou l'écriture à cette adresse, l'ordonnanceur répond `"ATT"` et l'opération n'a pas lieu.

Si un périphérique demande la lecture à une adresse à laquelle on n'a pas encore accédé en écriture pendant le tour, l'ordonnanceur répond `"OK"` et la lecture a lieu. Plusieurs lectures peuvent avoir donc lieu pendant le même tour à la même adresse.

Si un périphérique demande la lecture à une adresse à laquelle on a déjà accédé en écriture, l'ordonnanceur répond `"ATT"` et la lecture n'a pas lieu.

Ainsi, pendant un tour, une adresse peut être utilisée soit une seule fois en écriture, soit autant de fois qu'on veut en lecture, soit pas utilisée.

Si un périphérique ne peut pas effectuer une opération à une adresse, il demande la même opération à la même adresse au tour suivant.

1. Le tableau donné en annexe 1 indique, sur chaque ligne, le périphérique sélectionné, l'adresse à laquelle il souhaite accéder et l'opération à effectuer sur cette adresse. Compléter dans la dernière colonne de cette annexe, à rendre avec la copie, la réponse donnée par l'ordonnanceur pour chaque opération.

Annexe 1

N° périphérique	Adresse	Opération	Réponse de l'ordonnanceur
0	`10`	écriture	`"OK"`
1	`11`	lecture	`"OK"`
2	`10`	lecture	`"ATT"`
3	`10`	écriture	`"ATT"`
0	`12`	lecture	
1	`10`	lecture	
2	`10`	lecture	
3	`10`	écriture	

Réponse

N° périphérique	Adresse	Opération	Réponse de l'ordonnanceur
0	`10`	écriture	`"OK"`
1	`11`	lecture	`"OK"`
2	`10`	lecture	`"ATT"`
3	`10`	écriture	`"ATT"`
0	`12`	lecture	`"OK"`
1	`10`	lecture	`"OK"`
2	`10`	lecture	`"OK"`
3	`10`	écriture	`"ATT"`

Il s'agit d'un nouveau tour, les lectures sont possibles, la première écriture ne l'est pas, on a déjà accédé en lecture pendant le tour à l'adresse demandée.

On suppose dans toute la suite que :

- le périphérique 0 écrit systématiquement à l'adresse `10` ;
- le périphérique 1 lit systématiquement à l'adresse `10` ;
- le périphérique 2 écrit alternativement aux adresses `11` et `12` ;
- le périphérique 3 lit alternativement aux adresses `11` et `12` ;

Pour les périphériques 2 et 3, le changement d'adresse n'est effectif que lorsque l'opération est réalisée.

2. On suppose que les périphériques sont sélectionnés à chaque tour dans l'ordre 0 ; 1 ; 2 ; 3. Expliquer ce qu'il se passe pour le périphérique 1.

Réponse

- À chaque début de tour, le périphérique 0 demande à écrire à l'adresse `10` ; c'est accepté.
- Juste après, le périphérique 1 demande à lire à l'adresse `10` ; c'est refusé.

Le périphérique 1 ne pourra jamais lire l'adresse `10`.

Les périphériques sont sollicités de la manière suivante lors de quatre tours successifs :

- au premier tour, ils sont sollicités dans l'ordre 0 ; 1 ; 2 ; 3 ;
- au deuxième tour, dans l'ordre 1 ; 2 ; 3 ; 0 ;
- au troisième tour, 2 ; 3 ; 0 ; 1 ;
- puis 3 ; 0 ; 1 ; 2 au dernier tour.
- Et on recommence...

3.a. Préciser pour chacun de ces tours si le périphérique 0 peut écrire et si le périphérique 1 peut lire.

Réponse

- **Tour 1** : 0 ; 1 ; 2 ; 3
  - 0 peut écrire, puis
  - 1 ne peut pas lire
- **Tour 2** : 1 ; 2 ; 3 ; 0
  - 1 peut lire, puis
  - 0 ne peut pas écrire
- **Tour 3** : 2 ; 3 ; 0 ; 1
  - 0 peut écrire, puis
  - 1 ne peut pas lire
- **Tour 4** : 3 ; 0 ; 1 ; 2
  - 0 peut écrire, puis
  - 1 ne peut pas lire

3.b. En déduire la proportion des valeurs écrites par le périphérique 0 qui sont effectivement lues par le périphérique 1.

Réponse

- Au tour 1, la valeur écrite par le périphérique 0 sera lue par le périphérique 1 au tour suivant.
- Au tour 2, rien n'est écrit par le périphérique 0.
- Au tour 3, la valeur écrite par le périphérique 0 **ne sera jamais** lue par le périphérique 1 ; en effet, une autre écriture intervient avant la prochaine lecture.
- Au tour 4, la valeur écrite par le périphérique 0 **ne sera jamais** lue par le périphérique 1 ; en effet, une autre écriture intervient avant la prochaine lecture.

Ainsi, une seule valeur sur trois sera effectivement lue. La proportion est \(\frac13\).

On change la méthode d'ordonnancement : on détermine l'ordre des périphériques au cours d'un tour à l'aide de deux listes d'attente `ATT_L` et `ATT_E` établies au tour précédent.

Au cours d'un tour, on place dans la liste `ATT_L` toutes les opérations de lecture mises en attente, et dans la liste d'attente `ATT_E` toutes les opérations d'écriture mises en attente.

Au début du tour suivant, on établit l'ordre d'interrogation des périphériques en procédant ainsi :

- on interroge ceux présents dans la liste `ATT_L`, par ordre croissant d'adresse,
- on interroge ensuite ceux présents dans la liste `ATT_E`, par ordre croissant d'adresse,
- puis on interroge les périphériques restants, par ordre croissant d'adresse.

4. Compléter et rendre avec la copie le tableau fourni en annexe 2, en utilisant l'ordonnancement décrit ci-dessus, sur 3 tours.

Annexe 2

Tour	N° périphérique	Adresse	Opération	Réponse ordonnanceur	ATT_L	ATT_E
1	0	`10`	écriture	`"OK"`	vide	vide
1	1	`10`	lecture	`"ATT"`	`(1, 10)`	vide
1	2	`11`	écriture			
1	3	`11`	lecture			
2	1	`10`	lecture			vide
2						
2						
2						
3	0	`10`	écriture		vide	vide
3	1	`10`	lecture			vide
3	2	`11`	écriture	`"OK"`	`(1, 10)`	vide
3	3	`12`	lecture			

Réponse

Tour	N° périphérique	Adresse	Opération	Réponse ordonnanceur	ATT_L	ATT_E
1	0	10	écriture	"OK"	vide	vide
1	1	10	lecture	"ATT"	(1, 10)	vide
1	2	11	écriture	"OK"	(1, 10)	vide
1	3	11	lecture	"ATT"	(1, 10), (3, 11)	vide
2	1	10	lecture	"OK"	(3, 11)	vide
2	3	11	lecture	"OK"	vide	vide
2	0	10	écriture	"ATT"	vide	(0, 10)
2	2	12	écriture	"OK"	vide	(0, 10)
3	0	10	écriture	"OK"	vide	vide
3	1	10	lecture	"ATT"	(1, 10)	vide
3	2	11	écriture	"OK"	(1, 10)	vide
3	3	12	lecture	"OK"	(1, 10)	vide

Les colonnes **e0** et **e1** du tableau suivant recensent les deux chiffres de l'écriture binaire de l'entier **n** de la première colonne.

nombre n	écriture binaire de n sur deux bits	e1	e0
0	00	0	0
1	01	0	1
2	10	1	0
3	11	1	1

L'ordonnanceur attribue à deux signaux sur le bus de données les valeurs de **e0** et **e1** associées au numéro du circuit qu'il veut sélectionner. On souhaite construire à l'aide des portes ET, OU et NON un circuit pour chaque périphérique.
Chacun des quatre circuits à construire prend en entrée deux signaux **e0** et **e1**, le signal de sortie **s** valant 1 uniquement lorsque les niveaux de **e0** et **e1** correspondent aux bits de l'écriture en binaire du numéro du périphérique correspondant.

Par exemple, le circuit ci-dessous réalise la sélection du périphérique 3. En effet, le signal **s** vaut 1 si et seulement si **e0** et **e1** valent tous les deux 1.

```
flowchart LR
    e0---ET(ET)
    e1---ET
    ET---s
    style e0 stroke-width:0px,opacity:0
    style e1 stroke-width:0px,opacity:0
    style s stroke-width:0px,opacity:0
```

5.a. Recopier sur la copie et indiquer dans le circuit ci-dessous les entrées **e0** et **e1** de façon que ce circuit sélectionne le périphérique 1.

```
flowchart LR
    ea( )---NON(NON)---ET(ET)
    eb( )----ET
    ET---s
    style ea stroke-width:0px,opacity:0
    style eb stroke-width:0px,opacity:0
    style s stroke-width:0px,opacity:0
```

Réponse

```
flowchart LR
    ea(e1)---NON(NON)---ET(ET)
    eb(e0)----ET
    ET---s
    style ea stroke-width:0px,opacity:0
    style eb stroke-width:0px,opacity:0
    style s stroke-width:0px,opacity:0
```

5.b. Dessiner un circuit constitué d'une porte ET et d'une porte NON, qui sélectionne le périphérique 2.

Réponse

```
flowchart LR
    ea(e1)----ET(ET)
    eb(e0)---NON(NON)---ET
    ET---s
    style ea stroke-width:0px,opacity:0
    style eb stroke-width:0px,opacity:0
    style s stroke-width:0px,opacity:0
```

5.c. Dessiner un circuit permettant de sélectionner le périphérique 0.

Réponse

```
flowchart LR
    ea(e0)---NON1(NON)---ET(ET)
    eb(e1)---NON2(NON)---ET
    ET---s
    style ea stroke-width:0px,opacity:0
    style eb stroke-width:0px,opacity:0
    style s stroke-width:0px,opacity:0
```

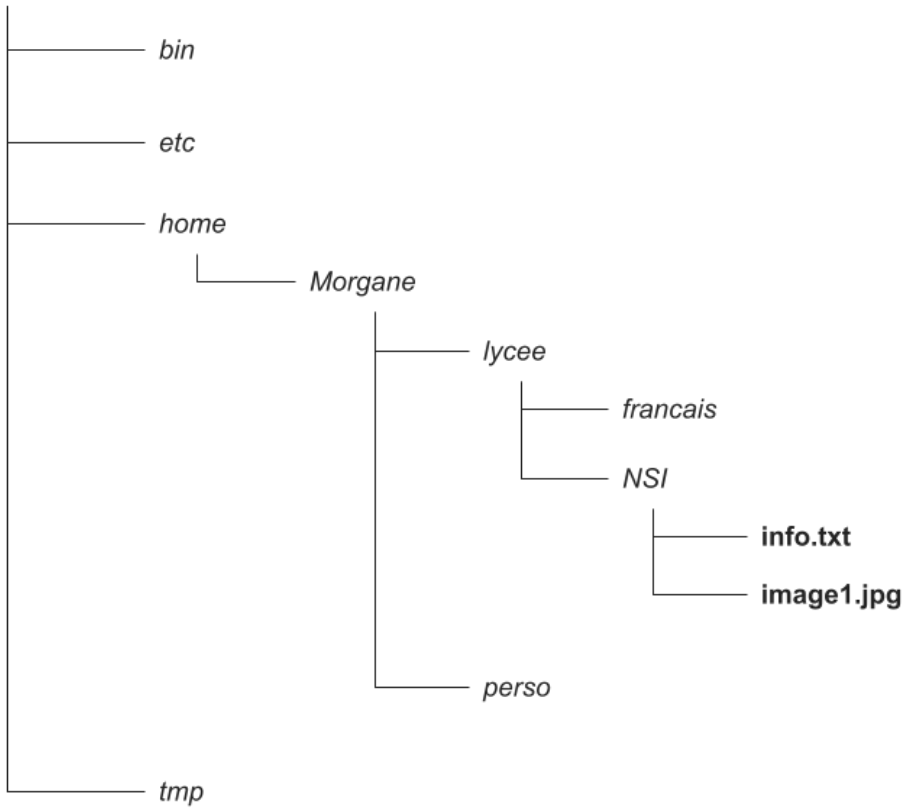
5. 2022, Amérique du Nord, J2

Exo

Cet exercice pourra utiliser des commandes de système d'exploitation de type UNIX telles que `cd, ls, mkdir, rm, rmd, mv, cat` .

Question 1

Dans un système d'exploitation de type UNIX, on considère l'arborescence des fichiers suivante dans laquelle les noms de dossiers sont en italique et ceux des fichiers qui sont en gras :



On souhaite, grâce à l'utilisation du terminal de commande, explorer et modifier les répertoires et fichiers présents.
On suppose qu'on se trouve actuellement à l'emplacement */home/morgane*

(a) Parmi les quatre propositions suivantes, donner celle correspondant à l'affichage obtenu lors se l'utilisation de la commande `ls`.

- **Proposition 1 :** lycee francais NSI info.txt image1.jpg perso
- **Proposition 2 :** lycee perso
- **Proposition 3 :** morgane
- **Proposition 4 :** bin etc home tmp

(b) Ecrire la commande qui permet, à partir de cet emplacement, d'attendre le répertoire **lycee**.
On suppose maintenant qu'on se trouve dans le repertoire `/home/morgane/lycee/NSI`

(c) Ecrire la commande qui permet de créer à cet emplacement un répertoire nommé `algorithmique`.

(d) Ecrire la commande qui permet, à partir de cet emplacement, de supprimer le fichier `image1.jpg`.

Quesiton 2

On rappelle qu'un processus est une instance d'application. Un processus peut être démarré par l'utilisateur, par un périphérique ou par un autre processus appelé parent. La commande UNIX `pa` présente un cliché instantané des procesus en cours d'exécution. On a exécuté la commande `ps` (avec quelques options qu'il n'est pas nécessaire de connaître pour la réussite de cet exercice). Un extrait du résultat de la commande est présenté ci-dessous :

```
UID        PID  PPID  C STIME TTY          TIME CMD
test       900   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfs-udisks2-vol
test       907   838  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-trash --sp
test       913   739  0 10:51 ?        00:00:00 /usr/lib/gvfs/gvfsd-metadata
test       918   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test       919   823  0 10:51 ?        00:00:00 /usr/lib/x86_64-linux-gnu/xfce
test       923     1  0 10:51 ?        00:00:02 xfce4-terminal
test       927   923  0 10:51 pts/0    00:00:00 bash
test      1036     1  0 11:18 ?        00:00:02 mousepad /home/test/Documents/
test      1058   923  0 11:22 pts/1    00:00:00 bash
root      1132     2  0 11:37 ?        00:00:00 [kworker/0:0-ata_sff]
root      1134     2  0 11:43 ?        00:00:00 [kworker/0:2-ata_sff]
test      1140   739  0 11:43 ?        00:00:00 /usr/lib/x86_64-linux-gnu/tumb
root      1149     2  0 11:43 ?        00:00:00 [kworker/u2:0-events_unbound]
test      1153   927  0 11:44 pts/0    00:00:00 vi
test      1154   927  0 11:44 pts/0    00:00:00 python3 prog.py
test      1155  1058  0 11:44 pts/1    00:00:00 ps -aef
```

On rappelle que :

- l'UID est l'identifiant de l'utilisateur propriétaire du processus;
- le PID est l'identifiant du processus parent;
- le PPID est l'identifiant du provessus parent;
- C indique l'utilisation processeur;
- STIME est l'heure de démarrage du processus;
- TTy est le nom du terminal de commande auquel le processus est attaché;
- TIME est la durée d'utilisation du processus par le processeur;
- CMD le nom de commande utilisé pour démarrer le processus.

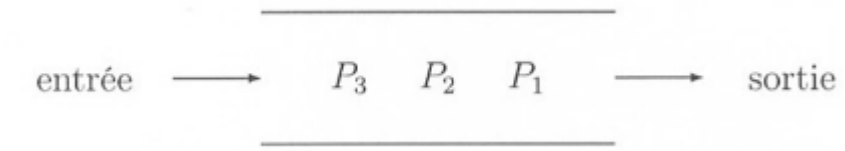
(a) Donner le PID du parent du processus démarré par la commande `vi`.
(b) Donner le PID d'un processus enfant du processus démarré par la commande `xfce4-terminal`.
(c) Citer le PID de deux processus qui ont le même parent.
(d) Parmi tous les processus affichés, citer le PID des deux qui ont consommé le plus de temps.

Question 3

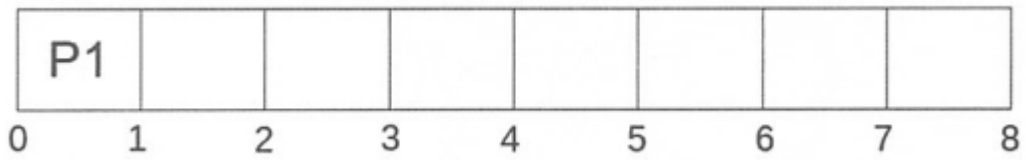
On considère les trois processus P1, P2, P3, tous soumis à l'instant 0 dans l'ordre 1, 2, 3 :

Nom du processus	Durée d'exécution en unité de temps	Ordre de soumission
P1	3	1
P2	1	2
P3	4	3

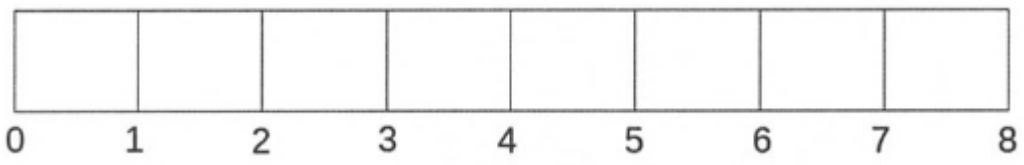
(a) Dans cette question, on considère que les processus sont exécutés de manière concurrente selon la politique du tourniquet : le temps est découpé en tranches nommées *quantums de temps*.
Les processus prêts à être exécutés sont placés dans une file d'attente selon leur ordre de soumission.
Lorsqu'un processus est élu, il s'exécute au plus durant un quantum de temps. Si le processus n'a pas terminé son exécution à l'issue du quantum de temps, il réintègre la file des processus prêts (coté entrée). Un autre processus, désormais en tête de la file (coté sortie) des processus prêts, est alors à son tour élu pour une durée égale à un quantum de temps maximum.



Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus exécuté à chaque cycle. Le quantum correspond à une unité de temps.



(b) Dans cette question, on considère que les processus sont exécutés en appliquant la politique du plus court d'abord : les processus sont exécutés complétement dans l'ordre croissant de leurs temps d'exécution, le plus court étant exécuté en premier.
Reproduire le tableau ci-dessous sur la copie et indiquer dans chacune des cases le processus excécuté à chaque cycle.



Question 4

On considère trois ressources R1, R2 et R3 et trois processus P1,P2 et P3 dont les files d'exécution des instructions élèmentaires sont indiquées ci-dessous :

Processus P_1
Demande R_1
Demande R_2
Libère R_1
Libère R_2

Processus P_2
Demande R_2
Demande R_3
Libère R_2
Libère R_3

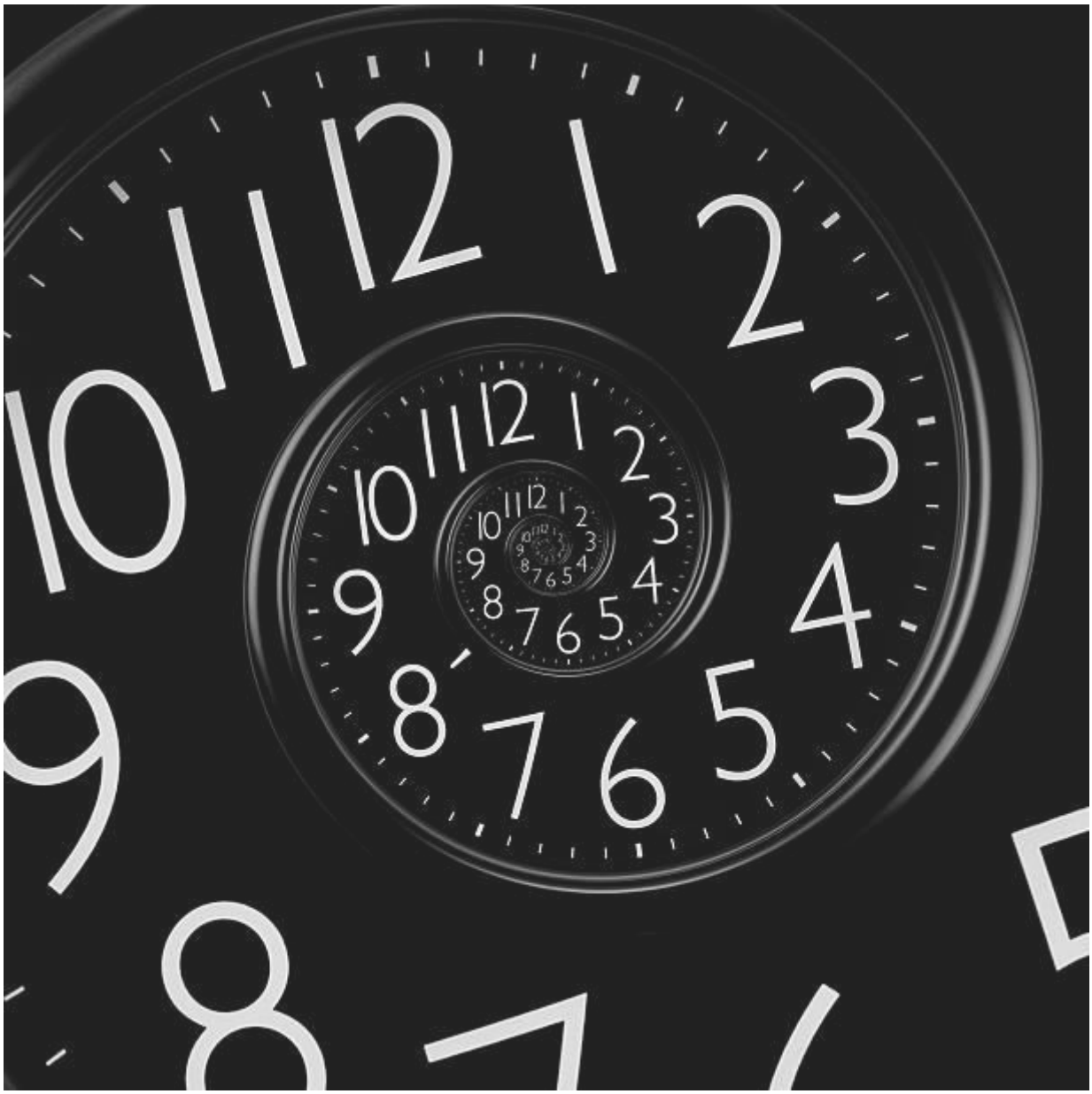
Processus P_3
Demande R_3
Demande R_1
Libère R_3
Libère R_1

(a) Rappeler les différens états d'un processus et expliquer pourquoi il y a ici risque d'interblocage, en proposant un ordre d'exécution des instructions élèmentaires le provoquant.
(b) Proposer un ordre d'exécution des instructions élèmentaires sans interblocage.

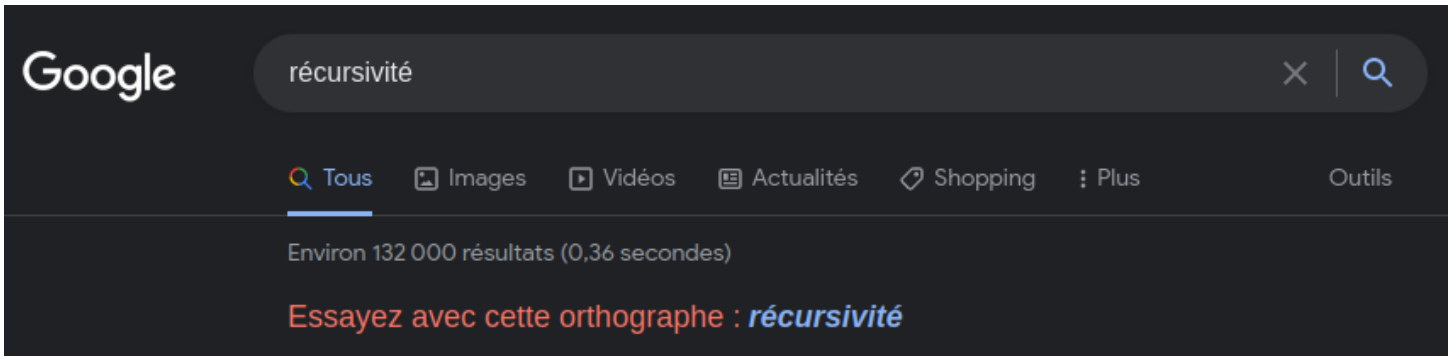
C3 Récursivité

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. Première approche



Faite l'essai

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 bancaire 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.

1.2. 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**

```python
def f(n):
    print(n)
    f(n-1)
    print("Hello world!")
```

- **Fonction 2**

🐍 **Script Python**

```python
def f(n):
    if n == 0:
        print("Hello world!")
    else:
        print(n)
        f(n-1)
```

- **Fonction 3**

🐍 **Script Python**

```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 blaques 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.
LA RÉCURSIVITÉ : MODE D'EMPLOI
G.P
«je vous l'achète le double de ce que vous donnerait le voisin du dessous»
ÉTAGE 4
«je vous l'achèterais le double de ce que vous donnerait le voisin du dessous»
ÉTAGE 3
«je vous l'achèterais le double de ce que vous donnerait le voisin du dessous»
ÉTAGE 2
«je vous l'achèterais le double de ce que vous donnerait le voisin du dessous»
ÉTAGE 1
«je vous en donnerais 3 euros»
ÉTAGE 0
«ok donc je vous en donnerais 6 euros»
ÉTAGE 1
«ok donc je vous en donnerais 12 euros»
ÉTAGE 2
«ok donc je vous en donnerais 24 euros»
ÉTAGE 3
LA RÉCURSIVITÉ : MODE D'EMPLOI
G.P
«ok, je vous l'achète pour 48 euros»
ÉTAGE 4

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'occurence 0)
- ligne 5 : l'appel récursif
- ligne 5 : la décrémentation du paramètre d'appel

🐍 **Script Python**

```
mystere(5)
15
```

🐍 **Script Python**

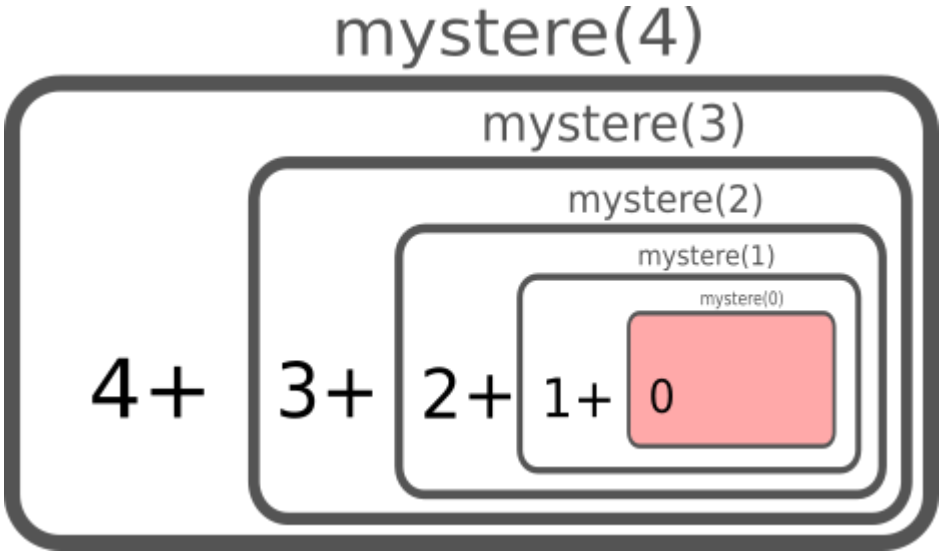
```
mystere(8)
36
```

Analyse avec Python tutor

Que se passe-t-il lorsqu'on appelle `mystere(4)` ?

\[\begin{align} \rm{mystere(4)} &= 4+ \rm{mystere(3)}\\ &= 4+ (3+\rm{mystere(2)}) \\ &= 4+ (3+(2+\rm{mystere(1)})) \\ &= 4+ (3+(2+(1+\rm{mystere(0)}))) \\ &= 4+ (3+(2+(1+0))) \\ \end{align}\]

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 : $1+2+3+\dots+n=\frac{n(n+1)}{2}$

Ici, $1+2+3+\dots+100=\dfrac{(100\times 101)}{2}=50 \times 101=5050$

A 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**

```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 $S = n + (n-1) + \dots + 3 + 2 + 1 + 0$ 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) :

Exercice 1 ❤️

🐍 **Script Python**

```
def somme_rec(n):
    if n == 0:
        return 0
    else:
        return n + somme(n-1)
somme_rec(100)
5050
```

nction `factorielle(n)` (notée \(n!\) en mathématiques), qui calcule le r \(n\) par les entiers positifs qui lui sont inférieurs: $$ n! = n \times times \dots \times 3 \times 2 \times 1$$ Exemple : \(5! times2\times1=120\)

ctions `somme` précédentes pour écrire deux fonctions `facto_iter` _rec (récursive) renvoyant la factorielle d'un nombre entier `n` .

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.

Reprenons l'exemple de la fonction récursive `somme`. Si on appelle cette fonction:

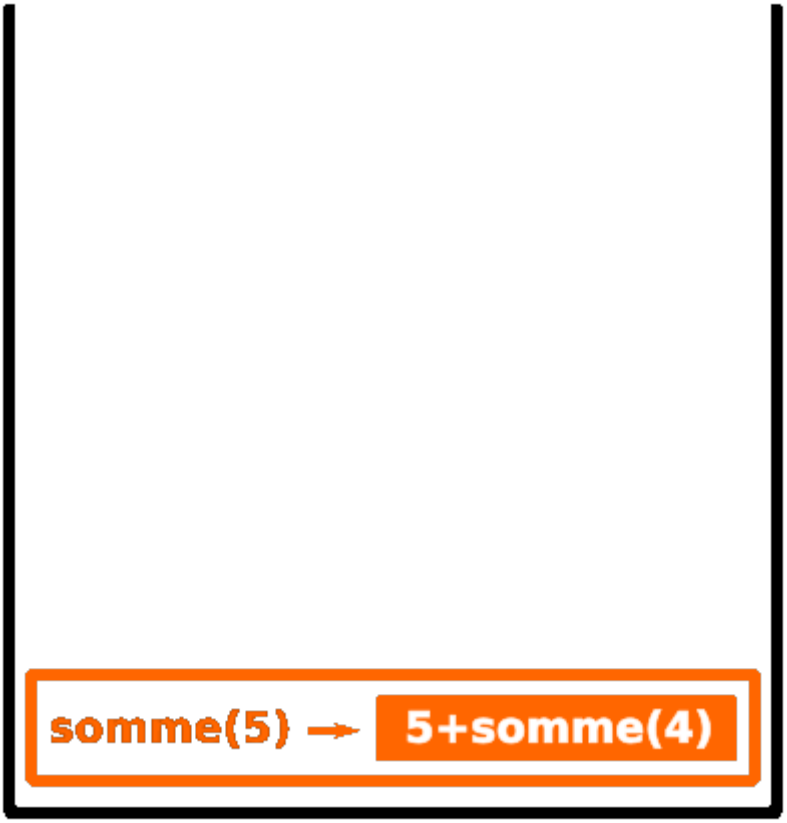
🐍 **Script Python**

```
>>> 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»).)



1.4. 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*).

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 mileu 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.

3. Exemples de récursivité double

Les expressions qui définissent une fonction peuvent aussi dépendre de plusieurs appels à la fonction en cours de définition.

3.1. La suite de Fibonnaci

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 \(F_2=0+1=1, F_3=F_2+F_1=1+1=2, F_4=F_3+F_2=2+1=3, F_5=F_4+F_3=3+2=5\) ...

Exercice 5

Implémenter de façon récursive la suite de Fibonnaci.

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**

```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**

```python
from rcviz import viz
```

🐍 **Script Python**

```python
@viz
def fibo(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    else:
        return fibo(n-1)+fibo(n-2)
```

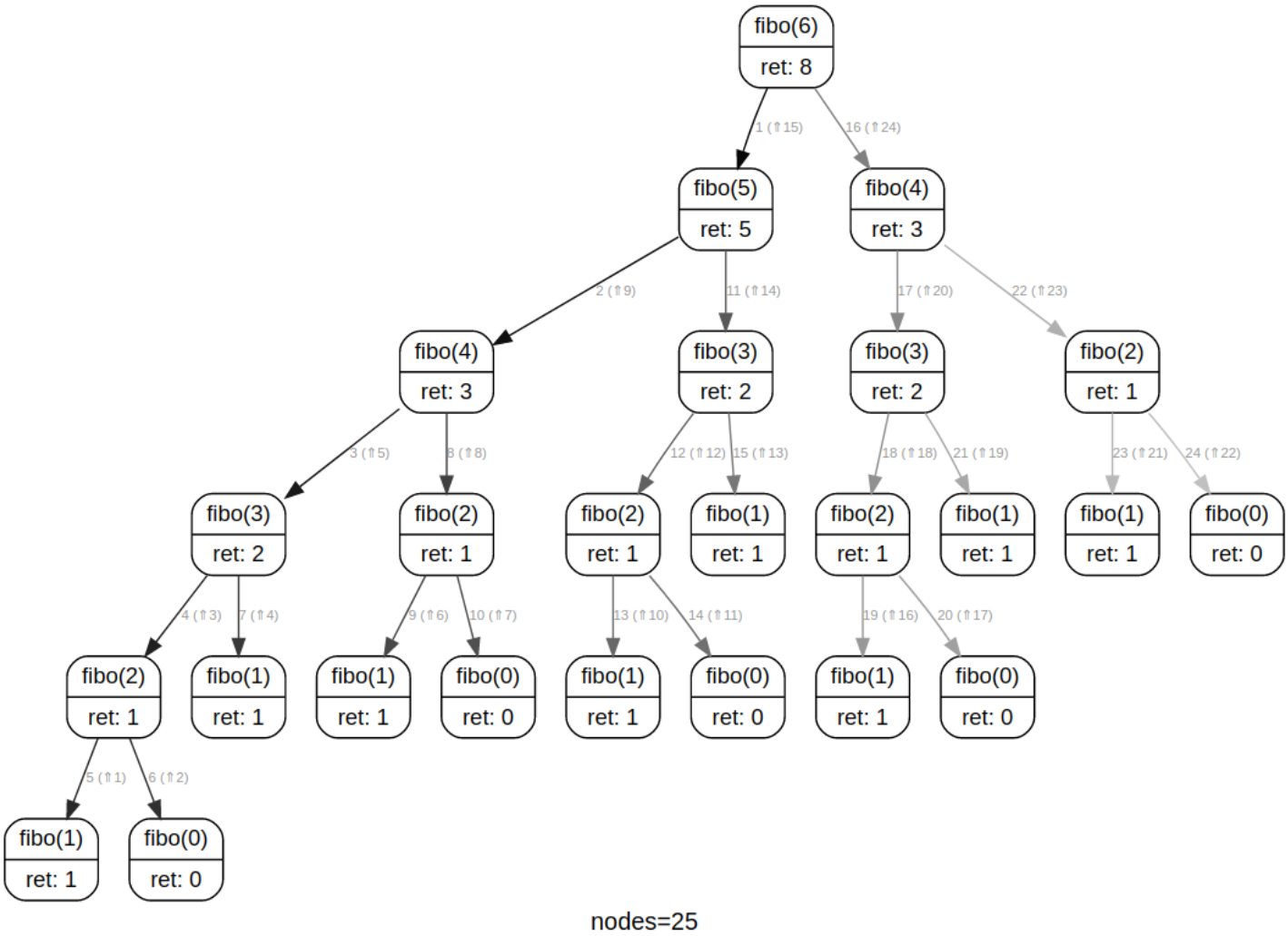
🐍 **Script Python**

```python
fibo(6)
```

```
8
```

📋 **Texte**

fibo.callgraph()



On y remarque (par exemple) que `fibo(2)` est calculé 5 fois…

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.

1. Ecrire la fonction correspondante 2 Testez votre fonction obtenue.
   Vous pouvez utiliser l'image ci-dessus pour connaître quelques valeurs à obtenir.

🐍 **Script Python**

```python
def nb_cartes(n):
    pass

nb_cartes(15)
```
```
345
```

Exercice 7

Soit (u_n) la suite d'entiers définie par :

$u_{n+1}= \left\{\begin{array}{ll} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair}\\ 3 \times u_n +1 & \text{sinon}\\ \end{array} \right.$

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**

```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 : $u_{n}= \left\{\begin{array}{ll} a \in R & \text{si } n =0\\ b \in R & \text{si } n=1\\ 3u_{n-1}+2u_{n-2}+5 & \forall n \geq 2 \end{array} \right.$

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.

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).

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**

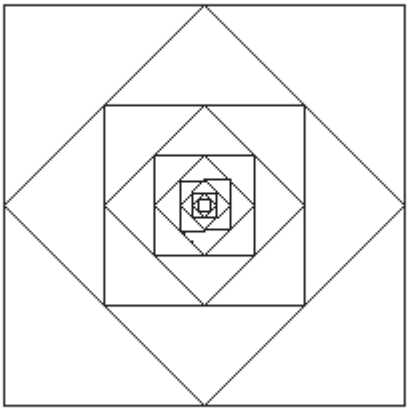
```
from turtle import *

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.



C7 Algorithmes de tri

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.

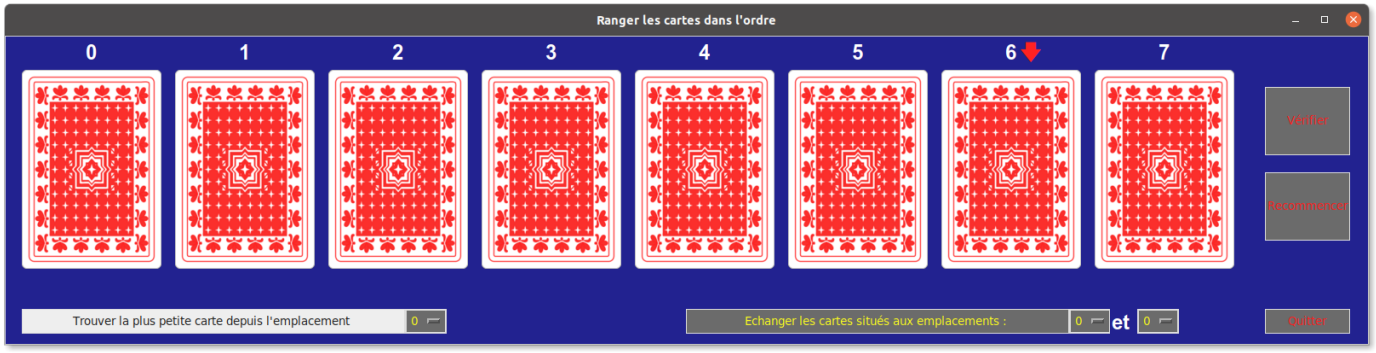
2. Activités

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.

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.

Aide

Bien évidemment, des boucles et des tests seront aussi nécessaires

4. Proposer une implémentation en Python de cet algorithme

Aide

On pourra utiliser la fonction `echange` définie dans l'activité précédente.

5. Tester cette fonction

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.

4. Cours : tri par insertion

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.

4.2. Codage de l'algorithme

Algorithme :

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]
```

A vous

Réaliser le tri par insertion de la liste suivante : `[27,10,12,8,11]`
Ecrire toutes les étapes.

A vous

Réaliser le tri par insertion de la liste suivante : `[9,6,1,4,8]`
Ecrire toutes les étapes.

4.3. Complexité de l'algorithme

🐍 **Script Python**

```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 «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.

4.3.1. Démonstration

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

- boucle `for` : elle s'exécute $n-1$ fois.
- boucle `while` : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$.
  Or

$$1+2+3+\dots+n-1=\dfrac{n \times (n-1)}{2}$$

Le terme de plus haut degré de l'expression $\dfrac{n \times (n-1)}{2}$ 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 $O(n^2)$.

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 $n-1$, ce qui caractérise une complexité linéaire.

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**

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 avonc 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.

4.5.1. 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.»

Aide

On appelle cette propriété un **invariant de boucle**.
Invariant siginifie qu'elle reste vraie pour chaque boucle.

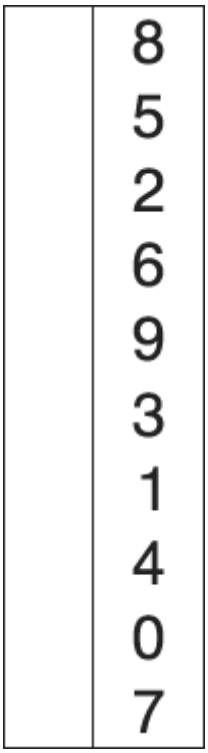
- quand k vaut 0, on place le minimum de la liste en l[0], la sous-liste l[0] est donc triée.

- 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.

5. Cours : tri par sélection

5.1. Animation

Considérons la liste `[8,5,2,6,9,3,1,4,8,7]`
Voici le fonctionnement de l'algorithme :



Vidéo Tri par sélection

Tri par sélection

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 élémént plus petit que notre mininum 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.

5.3. Implémentation de l'algorithme

Tri par sélection ❤️

🐍 **Script Python**

```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**

```python
>>> ma_liste = [7, 5, 2, 8, 1, 4]
>>> tri_selection(ma_liste)
>>> ma_liste
[1, 2, 4, 5, 7, 8]
```

5.4. Complexité de l'algorithme

5.4.1. 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 $1+2+3+\dots+n-1=\dfrac{n \times (n-1)}{2}$

Ceci est bien un polynôme du second degré, ce qui confirme que la complexité de ce tri est quadratique.

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

3. 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~~

4. Un programme de tri par insertion prend environ 1 seconde pour trier une liste de \(10\,000\) éléments, combien de temps prendra-t-il environ pour trier une liste de \(100\,000\) 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~~

5. 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`

7. Exercices

Fonction `echange(liste,i,j)`

🐍 **Script Python**

```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', 'M', 'P', 'O', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
```

Fonctionnement du tri par insertion

Enoncé **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**

```
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'E']
['P', 'O', 'R', 'G', 'R', 'A', 'M', 'M', 'E']
['O', 'P', 'R', 'G', 'R', 'A', 'M', 'M', 'E']
['O', 'P', 'G', 'R', 'R', 'A', 'M', 'M', 'E']
['O', 'G', 'P', 'R', 'R', 'A', 'M', 'M', 'E']
['G', 'O', 'P', 'R', 'R', 'A', 'M', 'M', 'E']
['G', 'O', 'P', 'R', 'A', 'R', 'M', 'M', 'E']
['G', 'O', 'P', 'A', 'R', 'R', 'M', 'M', 'E']
['G', 'O', 'A', 'P', 'R', 'R', 'M', 'M', 'E']
['G', 'A', 'O', 'P', 'R', 'R', 'M', 'M', 'E']
['A', 'G', 'O', 'P', 'R', 'R', 'M', 'M', 'E']
['A', 'G', 'O', 'P', 'R', 'M', 'R', 'M', 'E']
['A', 'G', 'O', 'P', 'M', 'R', 'R', 'M', 'E']
['A', 'G', 'O', 'M', 'P', 'R', 'R', 'M', 'E']
['A', 'G', 'M', 'O', 'P', 'R', 'R', 'M', 'E']
['A', 'G', 'M', 'O', 'P', 'R', 'M', 'R', 'E']
['A', 'G', 'M', 'O', 'P', 'M', 'R', 'R', 'E']
['A', 'G', 'M', 'O', 'M', 'P', 'R', 'R', 'E']
['A', 'G', 'M', 'M', 'O', 'P', 'R', 'R', 'E']
['A', 'G', 'M', 'M', 'O', 'P', 'R', 'E', 'R']
```

```
['A', 'G', 'M', 'M', 'O', 'P', 'E', 'R', 'R']
['A', 'G', 'M', 'M', 'O', 'E', 'P', 'R', 'R']
['A', 'G', 'M', 'M', 'E', 'O', 'P', 'R', 'R']
['A', 'G', 'M', 'E', 'M', 'O', 'P', 'R', 'R']
['A', 'G', 'E', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
```

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]
```

```python
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

Aide

Comme une *nouvelle liste* est crée, 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.

```
def tri_iteratif(tab):
    for k in range(..., 0 ,-1):
        imax = ...
        for i in range(0, ...):
            if tab[i] > ... :
                imax = i
        if tab[max] > ... :
            ..., tab[imax] = tab[imax], ...
    return tab
```

Com
donn

🐍 **Script Python**

```
>>> tri_iteratif([41, 55, 21, 18, 12, 6, 25])
[6, 12, 18, 21, 25, 41, 55]
```

On rappelle que l'instruction `a, b = b, a` échange les contenus de `a` et `b`.

🐍 **Script Python**

```
def echange(tab, i, j):
    tab[i], tab[j] = tab[j],
tab[i]

def tri_iteratif(tab):
    for k in range(len(tab)-1,
0, -1):
        indice_max = k
        for i in range(0, k):
            if tab[i] >
tab[indice_max]:
                indice_max = i
        echange(tab, k,
indice_max)
    return tab
```

C8 Diviser pour régner

1. Activités

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.

A compléter - Dichotomie version impérative ❤️

```python
def recherche_dichotomique(tab, val) :
    '''
    renvoie True ou False suivant la présence de la valeur val dans le tableau trié tab.
    '''
    i_debut = 0
    i_fin = len(tab) - 1
    while i_debut <= i_fin :
        i_centre = (... + ...) // 2     # (1)
        val_centrale = tab[...]         # (2)
        if val_centrale == val:         # (3)
            return True
        if val_centrale < val:          # (4)
            i_debut = ....              # (5)
        else :
            i_fin = ...
    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**

```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 k tel que $n \leq 2^k$, 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 $\dfrac{2^k}{2} = 2^{k-1}$ éléments,
- un quart aura au plus 2^{k-2}
- et au bout de i itérations, la taille de ce qui reste à étudier est de taille au plus 2^{k-i}.
- En particulier, si l'on fait k itérations, il reste au plus $2^{k-k} = 1$ valeur du tableau à examiner. On est sûr de s'arrêter cette fois-ci

On a donc montré que si l'entier k vérifie $n \leq 2^k$, alors l'algorithme va effectuer au plus n itérations. La plus petite valeur est obtenue pour $\log(n) = \log_2 k$.
Ainsi, la complexité de la fonction est de l'ordre du logarithme de la longueur de la liste ($O(log_2(n))$).

Donc l'algorithme du tri fusion a une complexité de l'ordre de $n \times log_2(n)$.

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 ❤️

```python
def dicho_rec_2(tab, val, i=0, j=None): # (1)
    if j is None:                       # (2)
        j = len(tab)-1
    if i > j :
        return False
    m = (i + j) // 2
    if tab[m] < val :
        return dicho_rec_2(tab, val, m + 1, j)
    elif tab[m] > val :
        return dicho_rec_2(tab, val, i, m - 1 )
    else :
        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 intercepter.

Exemple d'utilisation :

🐍 **Script Python**

```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).

1.2. ■ Activité 2 : *Tri fusion*

1. Algorithmes de tri vus en première et revus cette année :
   a. 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]`
   b. 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]`
   c. Quelle est la complexité de ces deux algorithmes ?
2. L'algorithme du **tri fusion** consiste à :
   1️⃣ partager la liste en deux moitiés (à une unité près),
   2️⃣ trier chacune des deux moitiés,
   3️⃣ 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 :

```
graph TD
subgraph Partager en deux
S["[10,6,3,9,7,5]"] --> S1["[10,6,3]"]
S --> S2["[9,7,5]"]
end
subgraph Fusionner
S1 -.Trier.-> T1["[3,6,10]"]
S2 -.Trier.-> T2["[5,7,9]"]
T1 --> T["[3,5,6,7,9,10]"]
T2 --> T
end
```

   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.

```
graph TD
   subgraph Partager en deux
   S["[10,6,3,9,7,5]"] --> S1["[10,6,3]"]
   S --> S2["[9,7,5]"]
   S1 --> S11["[10]"]
   S1 --> S12["[6,3]"]
   S2 --> S21["[9]"]
   S2 --> S22["[...,...]"]
   S12 --> S121["[6]"]
   S12 --> S122["[3]"]
   S22 --> S221["[...]"]
   S22 --> S222["[...]"]
end
```

```
    subgraph Fusionner
    S121 --> T21["[...,...]"]
    S122 --> T21
    S221 --> T22["[5,7]"]
    S222 --> T22["[5,7]"]
    S11 --> T1["[...,...,...]"]
    T21 --> T1
    S21 --> T2["[...,...,...]"]
    T22 --> T2
    T1 --> T["[3,5,6,7,9,10]"]
    T2 --> T
end
```

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]`.

> **Aide**
>
> - 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` :

```python
def fusion(l1,l2):
    ind1=0
    ind2=0
    l = []
    while ind1<len(l1) and ind2<len(l2):
        if l1[ind1]<l2[ind2]:
            l.append(...)
            ind1+=1
        else:
            l.append(...)
            ind2+=1
    if ind1==len(l1):
        for k in range(ind2,len(l2)):
            l.append(...)
    else:
        for k in range(ind1,len(l1)):
            l.append(...)
    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)
```

Important

On montre que l'algorithme du tri fusion a une complexité en \(O(n\log(n))\), 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 \(O(n^2)\).

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 `l1` et `l2`
2. Chercher les maximums `m1` de `l1` et `m2` de `l2`
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

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.

```
def dichotomie(tab, x):
    """
        tab : tableau trié
dans l’ordre croissant
        x : nombre entier
        La fonction
renvoie True si tab
contient x et False
sinon
    """
    # cas du tableau vide
    if ...:
        return False,1
    # cas où x n'est pas
compris entre les
valeurs extrêmes
    if (x < tab[0]) or ...:
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
```

```
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    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

```python
def fusion(L1,L2):
    n1 = len(L1)
    n2 = len(L2)
    L12 = [0]*(n1+n2)
    i1 = 0
    i2 = 0
    i = 0
    while i1 < n1 and ... :
        if L1[i1] < L2[i2]:
            L12[i] = ...
            i1 = ...
        else:
            L12[i] = L2[i2]
            i2 = ...
        i += 1
    while i1 < n1:
        L12[i] = ...
        i1 = i1 + 1
        i = ...
    while i2 < n2:
        L12[i] = ...
        i2 = i2 + 1
        i = ...
    return L12
```

Compléter le code.

Exemple :

🐍 **Script Python**

```
>>> fusion([1,6,10],[0,7,8,9])
[0, 1, 6, 7, 8, 9, 10]
```

Soit `T` un tableau non vide d'entiers triés dans l'ordre croissant et `n` un entier.
La fonction `chercher`, donnée à la page suivante, doit renvoyer un indice où la valeur `n` apparaît éventuellement dans `T`, et `None` sinon.

Les paramètres de la fonction sont :

- `T`, le tableau dans lequel s'effectue la recherche ;
- `n`, l'entier à chercher dans le tableau ;
- `i`, l'indice de début de la partie du tableau où s'effectue la recherche ;
- `j`, l'indice de fin de la partie du tableau où s'effectue la recherche.

La fonction `chercher` est une fonction récursive basée sur le principe « diviser pour régner ».

Le code de la fonction commence par vérifier si `0 <= i` et `j < len(T)`.
Si cette condition n'est pas vérifiée, elle affiche `"Erreur"` puis renvoie `None`.

Recopier et compléter le code de la fonction `chercher` proposée ci-dessous :

```
def chercher(T, n, i, j):
    if i < 0 or ??? :
        print("Erreur")
        return None
    if i > j :
        return None
    m = (i + j) // ???
    if T[m] < ??? :
        return chercher(T, n, ??? , ???)
    elif ??? :
        return chercher(T, n, ??? , ??? )
    else :
        return ???
```

L'exécution du code doit donner :

🐍 **Script Python**

```
>>> chercher([1,5,6,6,9,12],
7,0,10)
Erreur
>>> chercher([1,5,6,6,9,12],
7,0,5)
>>> chercher([1,5,6,6,9,12],
9,0,5)
4
>>> chercher([1,5,6,6,9,12],
6,0,5)
2
```

Sujet BAC 8 : Diviser pour régner

1. France 2021

Sujet 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

a. Quel est l’ordre de grandeur du coût, en nombre de comparaisons, de l’algorithme de tri fusion pour une liste de longueur ?
b. 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.

a. \(O(nlog_2(n))\)
b. L’algorithme de tri par

insertion a une complexité en temps dans le pire des cas en \(O(n^2)\). L'algorithme du tri par insertion est moins efficace que l'algorithme

L'algorithme de tri fusion utilise deux fonctions moitie_gauche et moitie_droite qui prennent en argument une liste L et renvoient respectivement :

- 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])
```

Voici l'affichage obtenu :

🐍 **Script Python**

```
[7, 4, 2, 1, 8, 5, 6, 3]
[7, 4, 2, 1]
[7,4]
[2, 1]
[8, 5, 6, 3]
[8, 5]
[6, 3]
```

Or ormais à différentes fonctions appelées par tri_fusion, à savoir moitie_droite et fu

Question 3

Enoncé **Solution**

Ecrire la fonction moitie_droite. résultat renvoyé par la

🐍 **Script Python**

```
def moitie_droite(L):
    n = len(L)
    deb = n//2
    tab = []
    for i in range(deb,n):
        tab.append(L[i])
    return tab
```

Question 4

Enoncé Solution

On donne ci-dessous une version incomplète de la fonction fusion.

```
def fusion(L1, L2):
    L = []
    n1 = len(L1)
    n2 = len(L2)
    i1 = 0
    i2 = 0
    while i1 < n1 or i2 < n2 :
        if i1 >= n1:
            L.append(L2[i2])
            i2 = i2 + 1
        elif i2 >= n2:
            L.append(L1[i1])
            i1 = i1 + 1
        else:
            e1 = L1[i1]
            e2 = L2[i2]

    return L
```

Dans
entie
repré
respe
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.

🐍 **Script Python**

```
def fusion(L1, L2):
    L=[]
    n1 = len(L1)
    n2 = len(L2)
    i1 = 0
    i2 = 0
    while i1<n1 or i2<n2:
        if i1>=n1:
            L.append(L2[i2])
            i2 = i2+1
        elif i2>=n2:
            L.append(L1[i1])
            i1=i1+1
        else :
            e1 = L1[i1]
            e2 = L2[i2]
            if e1 > e2:
                L.append(e2)
                i2 = i2 + 1
            else :
                L.append(e1)
                i1 = i1 + 1
    return L
```

2

Sujet n°2 : BAC Polynésie 2021

Ce ment du thème « algorithmique, langages et programmation ». Le but est de on (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 »).

2.1. Partie A : Manipulation d’une liste en Python

Question A.1

Enoncée **Solution**

Donner les affichages obtenus après l’exécution du code Python suivant.

🐍 **Script Python**

```python
notes = [8, 7, 18, 14, 12, 9, 17, 3]
notes[3] = 16
print(len(notes))
print(notes)
```

`! stion A.2" === "Enoncé" Écrire un code Python permettant d'afficher les éléments d'indice 2 à 4 de la liste notes.

📋 **Texte**

=== "Solution"

```python
for i in range(2,5):
    print(notes[i])
```

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
2  tri_insertion(liste):
3      """ trie par
4  insertion la liste
5  en paramètre """
6      for
7  indice_courant in
8  range(1,len(liste)):
9
   element_a_inserer
   =
```

🐍 **Script Python**

```
def tri_insertion(liste):
    for indice_courant in
range(1, len(liste)):
        element_a_inserer
= liste[indice_courant]
        i = indice_courant
```

O s instructions suivantes :

🐍 **Script Python**

```
notes = [8, 7, 18, 14, 12, 9, 17, 3]
tri_insertion(notes)
print(notes)
```

```
liste[i]
            i=i-1
        liste[i + 1] =
```

O nt :

🐍 **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(notes).

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.

Script Python

```
>>> [7, 8, 18, 14, 12, 9, 17, 3]
```

2. [illegible] 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 [illegible] parer le tableau en deux parties à peu près égales.

(3 [illegible] eux parties avec l'algorithme de tri fusion.

(4 [illegible] les deux tableaux triés en un seul tableau.

so [illegible] pedia

Script Python

```
>>> [7, 8, 14, 18, 12, 9, 17, 3]
```

Question C.1

Enoncé Solution

Cet algorithme est-il itératif ou récursif ? Justifier en une phrase.

récursif : à l'étape (3) l'algorithme de tri fusion s'appelle lui-même.

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.

1. Comparer les cartes du haut des 2 tas.
2. Placer la carte de valeur plus faible dans la main.
3. Recommencer l’étape 1 jusqu’à épuisement des 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 surlignées (uniquement celles-ci).

```
1  from math import
2  floor
3
4  def tri_fusion
5  (liste, i_debut,
6  i_fin):
7      if i_debut <
8  i_fin:
        i_partage =
   floor((i_debut +
   i_fin) / 2)

   tri_fusion(liste,
   i_debut, ..................

   tri_fusion(liste, .....
```

Rem
floor
entiè
pass

🐍 **Script Python**

```
from math import floor

def tri_fusion (liste, i_debut, i_fin):
    if i_debut < i_fin:
        i_partage = floor((i_debut + i_fin) / 2) # milieu pour diviser le tableau en deux moitiés
        tri_fusion(liste, i_debut, i_partage) # Appel tri_fusion pour 1ère moitié du tableau
        tri_fusion(liste, i_partage + 1, i_fin) # Appel tri_fusion pour 2ème moitié du tableau
        fusionner(liste,
```

```
i_debut, i_fin,
i_partage) # Fusion
des deux moitiés triées
```

Question C.4

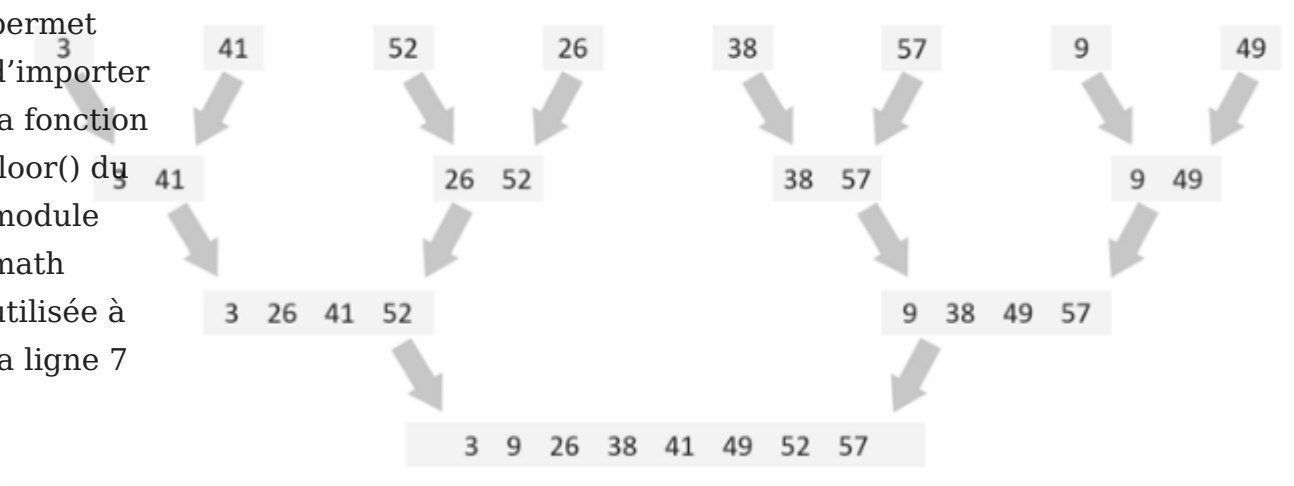
Enoncé **Solution**

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

2.4. 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.

- tri par insertion : \(O(n^2)\)
- tri par fusion : \(O(n.log_2 n)\)

tri par fusion : à chaque étape, le tri se fait par fusion de 2 tas deja tries

Question D.3

Enoncé **Solution**

Justifier brièvement ces deux complexités.

- Le tri par insertion utilise 2 boucles imbriquées, soit dans le pire des cas \(\frac{n(n-1)}{2}\) imes n(n+1)\) opérations soir \ (O(n^2)\).
- Le tri par fusion, divise la liste à trier par 2 pour chaque itérations, soit ~ \ (log_2 n\) opérations d'où une complexité global \ (O(n.log_2 n)\).

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.

2.5 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.

A l'indice 1 du tableau on trouve 8 ; à l'indice 3 on trouve 7. Nous avons 1 < 3 alors que 8 > 7, nous avons donc bien une inversion

A l'indice 2 du tableau on trouve 3 ; à l'indice 3 on trouve 7. Nous avons 2 < 3 et 3 < 7, donc nous n'avons donc pas d'inversion

Question A.1

Enoncé **Solution**

On donne la fonction suivante.

🐍 **Script Python**

```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
```

a. 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.

b. Expliquer ce que permet de déterminer cette fonction.

a.

- cas n°1 : 0
- cas n°2 : 1
- cas n°3 : 2

b.

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
```

🐍 **Script Python**

```python
def nombre_inversions(tab):
    nb_inv = 0
    n = len(tab)
    for i in range(n-1):
        nb_inv = nb_inv + fonction1(tab, i)
    return nb_inv
```

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.

L’ordre de grandeur de la complexité en temps de l’algorithme est \(O(n^2)\)

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.

Le tri fusion a une complexité en \(O(n.log_2(n))\)

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]
```

🐍 **Script Python**

```python
def moitie_gauche(tab):
    n = len(tab)
    nvx_tab = []
    if n==0:
        return []
    mil = n//2
    if n%2 == 0:
        lim = mil
    else :
        lim =mil+1
    for i in range(lim):
        nvx_tab.append(tab[i])
    return nvx_tab
```

une autre possibilité un peu plus concise :

🐍 **Script Python**

```python
def moitie_gauche(tab):
    return [tab[i] for i in range(len(tab)//2+len(tab)%2)]
```

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.

🐍 **Script Python**

```
def nb_inversions_rec(tab):
    if len(tab) > 1:
        tab_g =
moitie_gauche(tab)
        tab_d =
moitie_droite(tab)
        return
nb_inv_tab(tri(tab_g),
tri(tab_d))
        nb_inversions_rec(tab_g)
+ nb_inversions_rec(tab_d)
    else:
        return 0
```

ou

🐍 **Script Python**

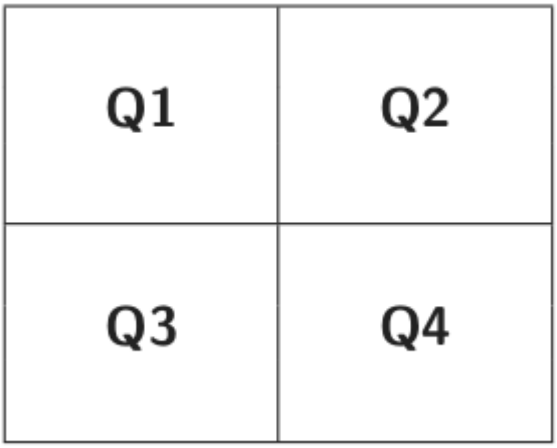
```
def nb_inversions_rec(tab :
list, n : int = 0) -> int:
    if len(tab) <= 1:
        return 0
    else:
        #Séparer le tableau en
deux tableaux de tailles
égales (à une unité près).
        gauche =
moitie_gauche(tab)
        droite =
moitie_droite(tab)
        #Compter le nombre
d’inversions dans chacun des
```

```
deux tableaux.
        n =
nb_inv_tab(sorted(gauche),
sorted(droite))

#Appeler récursivement la
fonction nb_inversions_rec
        return n +
nb_inversions_rec(gauche, n)
+ nb_inversions_rec(droite,
n)
```

Quart de tour d'une image

1. Pour faire tourner une image carré de côté \(2^n\) pixels d'un quart de tour à gauche, on propose la méthode suivante :

   - Diviser l'image en quatre quarts Q1,Q2,Q3,Q4

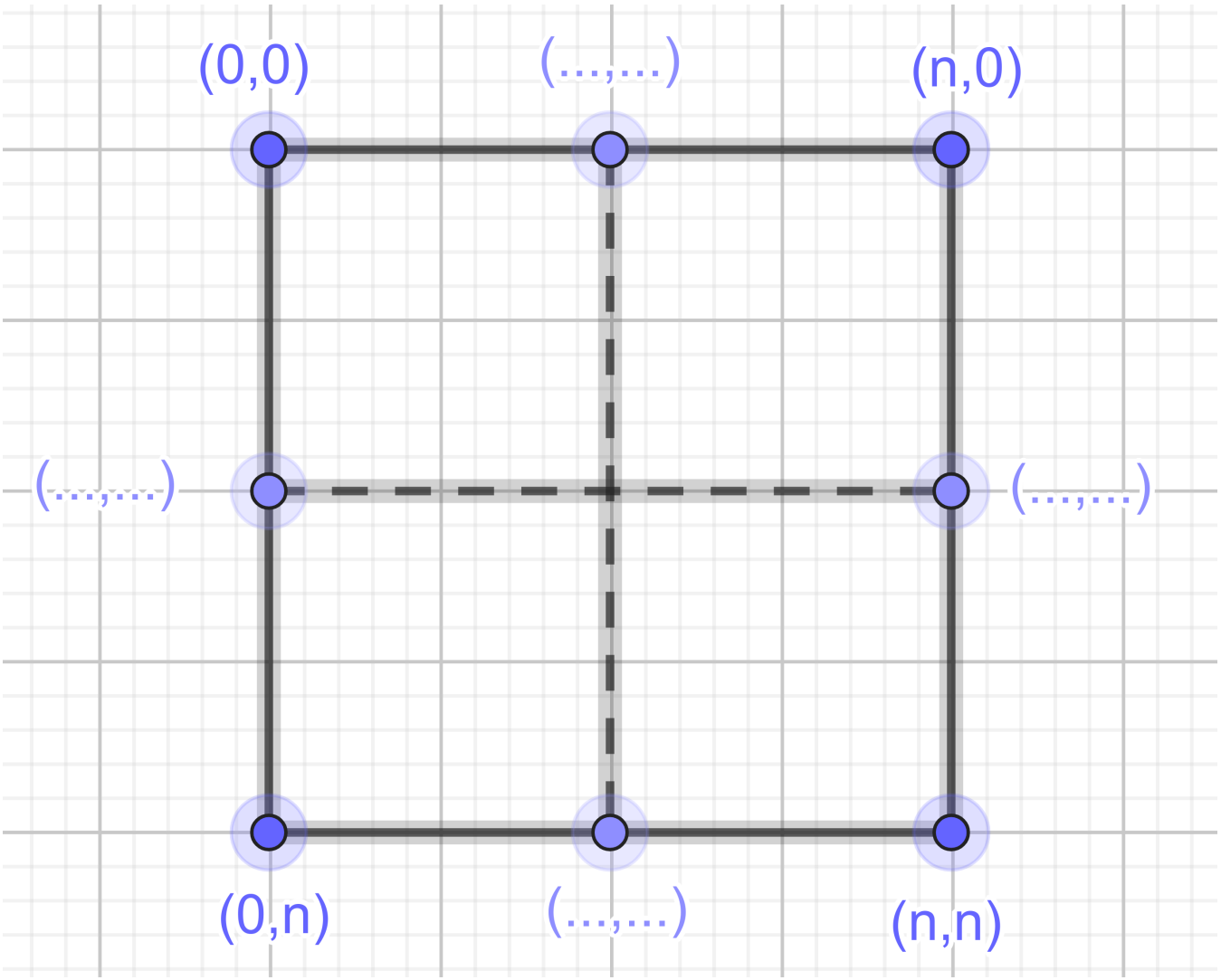


   - Faire tourner chacun des quarts d'un quart de tour à gauche

- Permuter chaque quart afin de le placer correctement

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 quarts 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 quarts 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((...,...,...,...))
        q3 = image.crop((...,...,...,...))
        q4 = image.crop((...,...,...,...))
        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 quarts
    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
```

C11 Algorithmes sur les arbres

1. Implémentation d'un arbre : Version 1

1.1. Avec la Programmation Orientée Objet⚓

Le but est d'obtenir l'interface ci-dessous.

Il est à remarquer que ce que nous allons appeler «Arbre» est en fait un nœud et ses deux fils gauche et droit.

🐍 **Script Python**

```
>>> a = Arbre(4) # pour créer l'arbre dont le nœud a pour valeur 4,
          # et dont les sous-arbres gauche et droit sont None
>>> a.gauche = Arbre(3) # pour donner la valeur 3 au nœud du sous-arbre gauche de a
>>> a.droit = Arbre(1) # pour donner la valeur 1 au nœud du sous-arbre droit de a
>>> a.droit.etiquette # pour accéder à la valeur du fils droit de a
```

Dessinez l'arbre créé par les instructions suivantes :

🐍 **Script Python**

```
>>> a = Arbre(4)
>>> a.gauche = Arbre(3)
>>> a.droit = Arbre(1)
>>> a.droit.gauche = Arbre(2)
>>> a.droit.droit = Arbre(7)
>>> a.gauche.gauche = Arbre(6)
>>> a.droit.droit.gauche = Arbre(9)
```

Implémentation :

▶ **Principe** : nous allons créer une classe `Arbre`, qui contiendra 3 attributs :

- `etiquette` : la valeur du nœud (de type `Int`)
- `gauche` : le sous-arbre gauche (de type `Arbre`)
- `droit` : le sous-arbre droit (de type `Arbre`).

Par défaut, les attributs `gauche` et `droit` seront à `None`, qui représentera l'arbre vide (ce qui n'est pas très rigoureux, car `None` n'est pas de type `Arbre`...).

▶ **Encapsulation ou pas ???** :

Afin de respecter le paradigme de la Programmation Orientée Objet, nous devrions jouer totalement le jeu de l'**encapsulation** en nous refusant d'accéder directement aux attributs.

Pour cela il faut construire des méthodes permettant d'accéder à ces attributs (avec des **getters**, ou **accesseurs** en français) ou de les modifier (avec des **setters**, ou **mutateurs** en français) .

1.2. Classe Arbre avec encapsulation ❤

🐍 **Script Python**

```
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, sousarbre): # mutateur
        self.gauche = sousarbre

    def ajout_droit(self, sousarbre): # mutateur
        self.droit = sousarbre

    def get_gauche(self): # accesseur
        return self.gauche

    def get_droit(self): # accesseur
        return self.droit

    def get_etiquette(self): # accesseur
        return self.etiquette

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.etiquette,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

Utilisation

🐍 **Script Python**

```
a = Arbre(4)
a.ajout_gauche(Arbre(3))
a.ajout_droit(Arbre(1))
a.get_droit().ajout_gauche(Arbre(2))
a.get_droit().ajout_droit(Arbre(7))
a.get_gauche().ajout_gauche(Arbre(6))
a.get_droit().get_droit().ajout_gauche(Arbre(9))
a.affiche()
```

📋 **Texte**

```
[4, [3, [6, None, None], None], [1, [2, None, None], [7, [9, None, None], None]]]
```

Accés à l'étiquette :

🐍 **Script Python**

```
a.get_droit().get_gauche().get_etiquette()
```

📋 **Texte**

```
2
```

Autre possibilité

📋 **Texte**

```
class Arbre:
    def __init__(self,valeur):
        """Initialisation de l'arbre racine + sous-arbre gauche et sous-arbre droit"""
        self.v=valeur
        self.gauche=None
        self.droit=None

    def ajout_gauche(self,val):
        """On ajoute valeur dans le sous-arbre gauche sous la forme [val,None,None]"""
        self.gauche=Arbre(val)

    def ajout_droit(self,val):
        """ On ajoute valeur dans le sous-arbre droit sous la forme [val,None,None]"""
        self.droit=Arbre(val)

    def get_gauche(self):
        return self.gauche

    def get_droit(self):
        return self.droit


    def get_valeur(self):
        if self==None:
            return None
        else:
            return self.v

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.v,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

📋 **Texte**

```
a = Arbre(4)
a.ajout_gauche(3)
a.ajout_droit(1)
a.droit.ajout_gauche(2)
a.droit.ajout_droit(7)
a.gauche.ajout_gauche(6)
a.droit.droit.ajout_gauche(9)
```

```
print(a.affiche())
a.get_droit().affiche()
```

📋 **Texte**

```
[4, [3, [6, None, None], None], [1, [2, None, None], [7, [9, None, None], None]]]
```

2. Implémentation d'un arbre : version 2

2.1. Classe Arbre sans encapsulation ❤

🐍 **Script Python**

```python
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None
```

C'est déjà fini !

Exo

Représenter l'arbre défini ci-dessous :

🐍 **Script Python**

```python
a = Arbre(4)
a.gauche = Arbre(3)
a.droit = Arbre(1)
a.droit.gauche = Arbre(2)
a.droit.droit = Arbre(7)
a.gauche.gauche = Arbre(6)
a.droit.droit.gauche = Arbre(9)
```

Exo

Ajouter à classe noeud ci-dessus une méthode `est_feuille` qui renvoie `True` si le noeud est une feuille et `False` sinon.

A rajouter dans la class pour visualier sous forme de liste

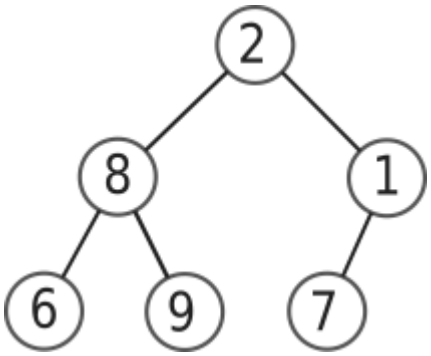
🐍 **Script Python**

```
    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.etiquette,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

3. Implémentation à partir de tuples imbriqués⚓

Considérons qu'un arbre peut se représenter par le tuple (valeur, sous-arbre gauche, sous-arbre droit).

L'arbre ci-dessous :



peut alors être représenté par le tuple :

📋 **Texte**

```
a = (2, (8, (6,(),()), (9,(),())), (1, (7, (),()), ()))
```

Le sous-arbre gauche est alors a[1] et le sous-arbre droit est a[2].

📋 **Texte**

```
a[1]
```

📋 **Texte**

```
(8, (6, (), ()), (9, (), ()))
```

📋 **Texte**

```
a[2]
```

📋 **Texte**

```
(1, (7, (), ()), ())
```

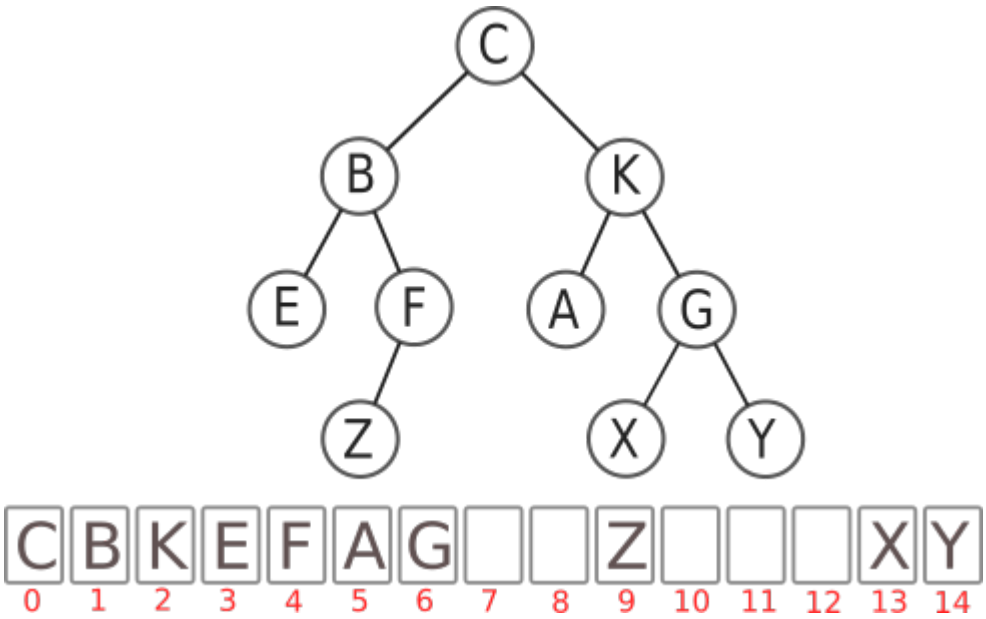
4. Implémentation à partir d'une «simple» liste⚓

De manière plus surprenante, il existe une méthode pour implémenter un arbre binaire (qui est une structure hiérarchique) avec une liste (qui est une structure linéaire). Ceci peut se faire par le biais d'une astuce sur les indices :

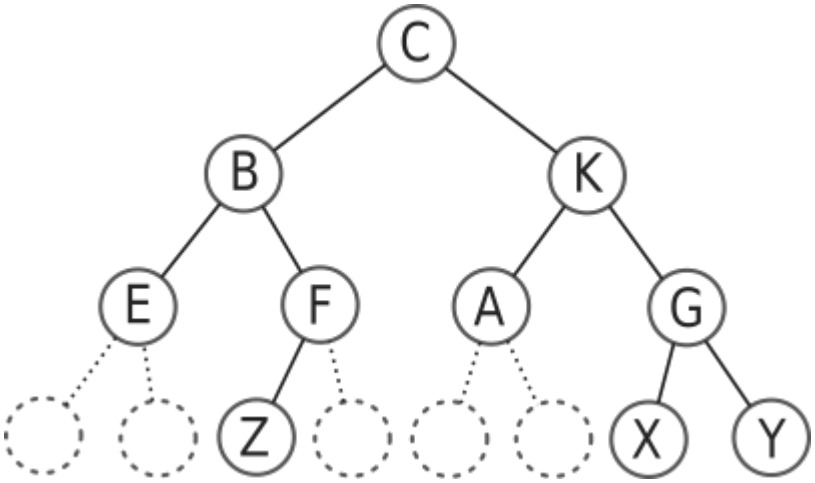
Les fils du nœud d'indice i sont placés aux indice 2i+1 et 2i+2.

Cette méthode est connue sous le nom de «méthode d'Eytzinger», et utilisée notamment en généalogie pour numéroter facilement les individus d'un arbre généalogique.

Exemple :



Pour comprendre facilement la numérotation, il suffit de s'imaginer l'arbre complet (en rajoutant les fils vides) et de faire une numérotation en largeur, niveau par niveau :



Remarque : parfois (comme dans le sujet 0...) la racine de l'arbre est placée à l'indice 1. Dans ce cas, les fils du nœud d'indice i sont placés aux indice 2i et 2i+1.

5. Utilisation de l'implémentation : parcours, taille...⚓

Dans toute la suite, sauf mention contraire, on utilisera l'implémentation en Programmation Orientée Objet, en version sans encapsulation (la plus simple). Nous allons créer des fonctions renvoyant les différents parcours d'un arbre, ou encore sa taille, sa hauteur, son nombre de feuilles... Toutes ses fonctions exploiteront la structure récursive d'un arbre.

Rappel de l'implémentation :

🐍 **Script Python**

```python
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None
```

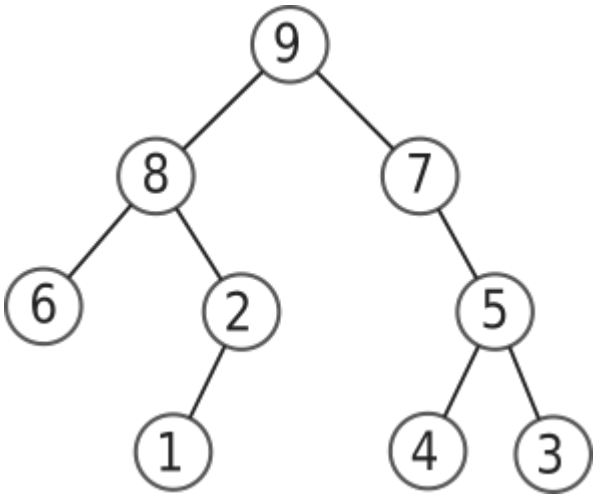
5.1. Parcours préfixe, infixe, postfixe⚓

5.1.1. Parcours préfixe ❤

🐍 **Script Python**

```python
def prefixe(arbre):
    if arbre is None :
        return None
    print(arbre.data, end = '-')
    prefixe(arbre.left)
    prefixe(arbre.right)
```

Exemple avec l'arbre



Construire l'arbre `a` à l'aide de la class précédente puis tester `prefixe(a)`

5.1.2. Parcours infixe ❤

🐍 **Script Python**

```
def infixe(arbre):
    if arbre is None :
        return None
    infixe(arbre.left)
    print(arbre.data, end = '-')
    infixe(arbre.right)
```

5.1.3. Parcours postfixe ou suffixe ❤

🐍 **Script Python**

```
def postfixe(arbre):
    if arbre is None :
        return None
    postfixe(arbre.left)
    postfixe(arbre.right)
    print(arbre.data, end = '-')
```

5.1.4. Pause vidéo

Regardez et appréciez cette vidéo

Video

À l'aide de la vidéo, codez le parcours infixe en itératif

5.2. Calcul de la taille d'un arbre⚓

Rappel : la taille d'un arbre est le nombre de ses nœuds.

🐍 **Script Python**

```
def taille(arbre):
    if arbre is None:
        return ...
    else:
        return ... + taille( ...) + taille( ...)
```

5.3. Calcul de la hauteur d'un arbre⚓

Rappel : on prendra comme convention que l'arbre vide a pour hauteur 0.

🐍 **Script Python**

```
def hauteur(arbre):
    if arbre is None:
        return ...
```

```
    else :
        return ... + max(..., ...)
```

5.4. Calcul du nombre de feuilles d'un arbre⚓

Rappel : une feuille est un nœud d'arité 0, autrement dit sans fils gauche ni fils droit.

🐍 **Script Python**

```
def nbfeuilles(arbre):
    if arbre is None:
        return 0
    if ... and ...:
        return 1
    else :
        return ... +  ...
```

5.5. Recherche d'une valeur dans un arbre⚓

On renverra True ou False en fonction de la présence ou non de la valeur dans l'arbre.

🐍 **Script Python**

```
def recherche(arbre, valeur):
    if arbre is None:
        return ...
    if ...:
        return True
    else :
        return ... or ...
```

Sujet BAC 11 : Algorihtmes Arbres Binaires

1. Amérique du Sud 2022 J2

Exo

Dans un arbre binaire, chaque noeud admet au plus deux enfants, appelés sous-arbre gauche et sous-arbre droit. On considère dans cet exercice des arbres binaires étiquetés avec des nombres entiers. On définit un chemin racine-feuille dans un tel arbre comme une liste ordonnée de noeuds telle que :

- le premier noeud est la racine
- chaque noeud suivant est enfant du précédent;
- le dernier noeud est une feuille.

On appellera somme d'un chemin racine-feuille la somme des étiquettes des noeuds du chemin. Enfin, la plus graande somme racine-feuille d'un arbre est la plus grande somme qu'il est possible d'obtenir en considérant tous les chemins racine-feuille de l'arbre.

Question 1

Déterminer la plus grande somme racine-feuille de l'arbre représenté ci-dessous

```
graph TD
    A("2") --> B("7")
    B --> D("4")
    B --> E("1")
    A --> C("5")
    C --> C1("3")
    C --> C2("8")
    D --> D1("2")
    D --> D2("3")
    E --> F(" ")
    E --> E1("5")
    linkStyle 8 stroke-width:0px;
    style F opacity:0;
    C1 --> C3(" ")
    C1 --> C4("1")
    C2 --> C5(" ")
    C2 --> C6(" ")
    linkStyle 10 stroke-width:0px;
    style C3 opacity:0;
    linkStyle 12 stroke-width:0px;
    style C5 opacity:0;
    linkStyle 13 stroke-width:0px;
    style C6 opacity:0;
```

Réponse

La plus grand somme racine-feuille de cette arbre est **16**, elle est obtenu pour la branche en rouge dans le schéma suivant :

```
graph TD
A["2"] --> B["7"]
A --> C["5"]
B --> D["4"]
B --> E["1"]
C --> F["3"]
C --> G["8"]
D --> H["2"]
D --> I["3"]
E --> V1[" "]
E --> J["5"]
F --> V2[" "]
F --> K["1"]
style V1 opacity:0;
style V2 opacity:0;
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
linkStyle 0 stroke:#FF0000,stroke-width:2px
linkStyle 2 stroke:#FF0000,stroke-width:2px
linkStyle 7 stroke:#FF0000,stroke-width:2px
```

Question 2

La class `Noeud` ci-dessous implémente le type abstrait d'arbre binaire.

🐍 **Script Python**

```python
class Noeud:
    def __init__(self,v):
        self.etiquette = v
        self.sag = None
        self.sad = None

    def niveau(self):
        if self.sag != None and self.sad != None:
            hg = self.sag.niveau()
            hd = self.sad.niveau()
            return 1 + max(hg,hd)
        if self.sag != None:
            return self.sag.niveau() + 1
        if self.sad != None:
            return self.sad.niveau() + 1
        return 0

    def modifier_sag(self,nsag):
        self.sag = nsag

    def modifier_sad(self,nsad):
        self.sad = nsad
```

a. Ecrire une suite d'instructions utilisant la class `Noeud` permettant de représenter l'arbre ci-dessous :

```
graph TD
    A("2") --> B("7")
    B --> D("4")
    B --> E("1")
    A --> C("5")
    C --> C1(" ")
    C --> C2("8")
    D --> D2(" ")
    linkStyle 6 stroke-width:0px;
    style D2 opacity:0;
    linkStyle 4 stroke-width:0px;
    style C1 opacity:0;
```

b. Que renvoie l'appel de la méthode `niveau` sur l'arbre ci-dessus ?

Réponse

a. On peut écrire la suite d'instructions suivante :

🐍 **Script Python**

```
s2 = Noeud(2)
s7 = Noeud(7)
s5 = Noeud(5)
s2.modifier_sag(s7)
s2.modifier_sad(s5)
s4 = Noeud(4)
s1 = Noeud(1)
s7.modifier_sag(s4)
s7.modifier_sad(s1)
s8 = Noeud(8)
s5.modifier_sad(s8)
```

b. L'appel à niveau sur cet arbre renvoie 3.

Question 3

S'inspirer du code de la méthode `niveau` pour écrire une méthode récursive `pgde_somme` qui renvoie la plus grande somme racine-feuille d'un arbre.

Réponse

🐍 **Script Python**

```
def pgde_somme(self):
    if self.sag != None and self.sad!=None:
        pgde_gauche = self.sag.pgde_somme()
        pgde_droite = self.sad.pgde_somme()
        return self.etiquette + max(pgde_gauche,pgde_droite)
    if self.sag != None:
        return self.etiquette + self.sag.pgde_somme()
    if self.sad != None:
        return self.etiquette + self.sad.pgde_somme()
    return self.etiquette
```

Question 4

On appelle arbre magique un arbre binaire dont toutes les sommes des chemins racine-feuille sont égales.

a. Recopier et compléter l'arbre ci-dessous pour qu'il soit magique.

```
graph TD
    A("5") --> B("...")
    B --> D("4")
    B --> E("...")
    A --> C("5")
    C --> C1("3")
    C --> C2("...")
    D --> D1("2")
    D --> D2("...")
    E --> F("...")
    E --> E1("3")
    linkStyle 8 stroke-width:0px;
    style F opacity:0;
    C1 --> C3("...")
    C1 --> C4("1")
    C2 --> C5("...")
    C2 --> C6("...")
    linkStyle 10 stroke-width:0px;
    style C3 opacity:0;
    linkStyle 12 stroke-width:0px;
    style C5 opacity:0;
    linkStyle 13 stroke-width:0px;
    style C6 opacity:0;
```

b. Un arbre est magique si ses sous-arbres sont magiques et qu'ils ont de plus la même plus grande somme racine-feuille. Ecrire une méthode récursive `est_magique` quie renvoie `True` si l'arbre est magique et `False` sinon.

Réponse

a. Arbre complété :

```
graph TD
A["2"] --> B["3"]
A --> C["5"]
B --> D["4"]
B --> E["3"]
C --> F["3"]
C --> G["4"]
D --> H["2"]
D --> I["2"]
E --> V1[" "]
E --> J["5"]
F --> V2[" "]
F --> K["1"]
style V1 opacity:0;
style V2 opacity:0;
style B stroke:#FF0000
style E stroke:#FF0000
style G stroke:#FF0000
style I stroke:#FF0000
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
```

b.

🐍 **Script Python**

```python
def est_magique(self):
    if self.sad is not None and self.sag is not None:
        return self.sad.est_magique() and self.sag.est_magique() and self.sag.pgde_somme() == self.sad.pgde_somme()
    elif self.sad is not None:
        return self.sad.est_magique()
    elif self.sag is not None:
        return self.sag.est_magique()
    else:
        return True
```

2. 2022 Mayotte J1

Exo

On s’intéresse dans cet exercice à l’étude d’un arbre généalogique.
Voici un extrait de l’arbre généalogique fictif d’une personne nommée Albert Normand.

L’arbre généalogique est présenté avec les parents vers le bas et les enfants vers le haut.

Albert Normand est considéré comme la génération 0. On considère ses parents comme la génération 1, ses grands-parents comme la génération 2 et ainsi de suite pour les générations précédentes.

```
graph TD
   A("Albert <br> Normand") --> B("Jules <br> Normand <br> Père")
   B --> D("Michel <br> Normand <br> (Père)")
   B --> E("Hélène <br> Breton <br> (Mère)")
   A --> C("Marie <br> Comtois <br> (Mère)")
   C --> C1("Thibaut <br> Comtois <br> (Père)")
   C --> C2("Gabrielle <br> Savoyard <br> (Mère)")
   D --> D1("Jules <br> Normand <br> (Père)")
   D --> D2("Odile <br> Picard <br> (Mère)")
   E --> F("Evariste <br> Breton <br> (Père)")
   E --> E1("Camèlia <br> Charentais <br> (Mère)")
   C1 --> C3("Léo <br> Comtois <br> (Père)")
   C1 --> C4("Eulalie <br> Lorrain <br> (Mère)")
   C2 --> C5("Guillaume <br> Savoyard <br> (Père)")
   C2 --> C6("Janet <br> Chesterfield <br> (Mère)")
```

MODELISATION DE L’ARBRE
L’arbre généalogique d’un individu est modélisé par un arbre :

- chaque nœud de l’arbre représente un individu ;
- le premier nœud du sous-arbre gauche d’un individu est associé à son père ;
- le premier nœud du sous-arbre droit est associé à sa mère.

IMPLEMENTATION DE L’ARBRE
Pour implémenter l’arbre, on utilise des notions de programmation orientée objet. Chaque nœud de l’arbre est représenté par un objet qui est l’instance d’une classe Noeud ayant trois attributs. Ainsi l’objet N de type Noeud aura les attributs suivants :

- `N.identite` de type tuple : `(prenom,nom)` de l’individu référencé par l’objet N ;
- `N.gauche` de type arbre binaire : le sous-arbre gauche de l’objet N ;
- `N.droit` de type arbre binaire : le sous-arbre droit de l’objet N.

Pour un individu, référencé par l’objet N de type Noeud, dont on ne connait pas les parents, on considèrera que `N.gauche` et `N.droit` ont la valeur `None`.

Question 1

a. Expliquer en quoi cet arbre généalogique est un arbre binaire.
b. Pourquoi un arbre généalogique n'est pas un arbre binaire de recherche (ABR) ?

Réponse

a. Un arbre binaire est un arbre d'arité 2, c'est à dire un arbre dans lequel chaque noeud possède au plus deux fils. C'est bien le cas ici, une personne ayant au maximum deux parents connus.
b. Dans un arbre binaire de recherche, on dispose d'une relation d'ordre entre les clés associées à chaque noeud et pour tout noeud, sa clé est supérieure aux clés du sous arbre gauche et inférieure aux clés du sous arbre droit. Ici les clés sont des personnes sur lesquelles on n'a pas de relation d'ordre.

Question 2

On souhaite obtenir la liste de tous les ascendants (ancêtres) d'Albert Normand. Pour cela, on utilise un parcours en profondeur de l'arbre.
a. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur préfixe.
b. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur infixe.
On donne ci-dessous le code incomplet de la fonction d'un parcours en profondeur de l'arbre, dans lequel il manque la ligne correspondant à l'instruction d'affichage du prénom et du nom de l'individu :

🐍 **Script Python**

```python
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
        noeud_actuel = racine_de_l_arbre
        parcours(noeud_actuel.gauche)
        parcours(noeud_actuel.droite)
```

c. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur **préfixe**.
d. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur **infixe**.

Réponse

a. On rappelle que dans un parcours en *profondeur préfixe*, on liste en premier la racine puis récursivement les clés du sous arbre gauche et du sous arbre droit. Ce qui donne ici : Albert Normand

—

Jules Normand

—

Michel Normand

—

Jules Normand

—

Odile Picard

—

Hélène Breton

—

Evariste Breton

b. Dans le parcours en *profondeur infixe*, on liste récursivement les clés du sag puis la racine puis les clés du sad. Ce qui donne ici : Jules Normand

—

Michel Normand

—

Odile Picard

—

Jules Normand

—

Evariste Breton

—

Hélène Breton

—

Camélia Charentais

c. En parcours **prefixe** on insère l'affichage du tuple `(prenom,nom)` **avant** de relancer les parcours récursifs sur les deux sous arbres.

🐍 **Script Python**

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
    noeud_actuel = racine_de_l_arbre
    print(noeud_actuel.identite)
```

```
    parcours(noeud_actuel.gauche)
    parcours(noeud_actuel.droite)
```

d. En parcours **infixe** on insère l'affichage du tuple `(prenom,nom)` **entre** les parcours récursifs sur les deux sous arbres.

🐍 **Script Python**

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
    noeud_actuel = racine_de_l_arbre
    parcours(noeud_actuel.gauche)
    print(noeud_actuel.identite)
    parcours(noeud_actuel.droite)
```

Question 3

On souhaite maintenant préciser la génération d'un individu dans l'implémentation de l'arbre généalogique. Lors de la création de l'instance, on donnera la valeur 0 par défaut.
a. Recopier et compléter la définition de la classe `Noeud` pour ajouter un attribut generation qui indique la génération d'un individu.

🐍 **Script Python**

```
class Noeud():
    def __init__(prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
```

b. Ecrire la fonction récursive `numerotation` qui parcourt l'arbre et modifie l'attribut generation de tous les ancêtres d'Albert Normand, de sorte que les parents d'Albert Normand soient la génération 1 etc...
Cette fonction prend en paramètres `racine_de_l_arbre` de type `Noeud` et `num_gen` de type entier.

🐍 **Script Python**

```
def numerotation(racine_de_l_arbre, num_gen=0) :
    ...
```

Réponse

a.

🐍 **Script Python**

```python
class Noeud() :
    def __init__(self, prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
        self.generation = 0
```

Bug

Dans l'énoncé, `self` ne figure pas dans les paramètres de `__init__` (ajouté dans cette correction)

b.

🐍 **Script Python**

```python
def numerotation(racine_de_l_arbre, num_gen=0) :
    if racine_de_l_arbre != None:
        racine_de_l_arbre.generation = num_gen
        numerotation(racine_de_l_arbre.gauche,num_gen+1)
        numerotation(racine_de_l_arbre.droit,num_gen+1)
```

Question 4

On donne la fonction suivante qui prend en paramètres l'objet N de type `Noeud` et la variable affiche de type booléen :

🐍 **Script Python**

```python
def mystere(N,affiche) :
    if N != None :
        if affiche :
            print( N.identite[0])
        mystere(N.gauche,False)
        mystere(N.droite,True)
```

Ecrire, dans l'ordre d'affichage, le résultat de l'exécution de `mystere(racine_de_l_arbre,False)` où `racine_de_l_arbre` est le `nœud` qui référence Albert Normand

Réponse

Cette fonction parcourt l'arbre en préfixe mais affiche seulement les noeuds droit, ce qui donne :
Odile Picard
—
Hélène Breton
—
Camélia Charentais
—
Marie Comtois
—
Eulalie Lorrain
—
Gabrielle Savoyard
—
Janet Chesterfield

3. 2021 France J2

Exo

Une agence immobilière développe un programme pour gérer les biens immobiliers qu'elle propose à la vente.
Dans ce programme, pour modéliser les données de biens immobiliers, on définit une classe Bim avec les attributs suivants :

- `nt` de type `str` représente la nature du bien (appartement, maison, bureau, commerces,...);
- `sf` de type `float` est la surface du bien ;
- `pm` de type `float` est le prix moyen par m² du bien qui dépend de son emplacement.

La classe `Bim` possède une méthode `estim_prix` qui renvoie une estimation du prix du bien. Le code (incomplet) de la classe `Bim` est donné ci-dessous :

🐍 **Script Python**

```python
class Bim:
    def __init__(self, nature, surface, prix_moy):
        ...

    def estim_prix(self):
        return self.sf * self.pm
```

Question 1

Recopier et compléter le code du constructeur de la classe `Bim`.

Réponse

🐍 **Script Python**

```python
def __init__(self, nature, surface, prix_moy):
    self.nt = nature
    self.sf = surface
    self.pm = prix_moy
```

Question 2

On exécute l'instruction suivante :

🐍 **Script Python**

```python
b1 = Bim('maison', 70.0, 2000.0)
```

Que renvoie l'instruction `b1.estim_prix()` ?
Préciser le type de la valeur renvoyée.

Réponse

L'instruction `b1.estim_prix()` renvoie 140000.0 (estimation du prix de b1) ; la valeur renvoyée est de type flottant, car `self.sf` et `self.pm` sont de type flottant.

Question 3

On souhaite affiner l'estimation du prix d'un bien en prenant en compte sa nature :

- pour un bien dont l'attribut `nt` est 'maison' la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 1,1 ;
- pour un bien dont l'attribut `nt` est 'bureau' la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 0,8 ;
- pour les biens d'autres natures, l'estimation du prix ne change pas.

Modifier le code de la méthode `estim_prix` afin de prendre en compte ce changement de calcul.

Réponse

🐍 **Script Python**

```
def estim_prix(self):
    prix_brut = self.sf * self.pm
    if self.nt=='maison':
        return prix_brut*1.1
    elif self.nt=='bureau':
        return prix_brut*0.8
    else :
        return prix_brut
```

Question 4

Écrire le code Python d'une fonction `nb_maison(lst)` qui prend en argument une liste Python de biens immobiliers de type `Bim` et qui renvoie le nombre d'objets de nature 'maison' contenus dans la liste `lst`.

Réponse

🐍 **Script Python**

```
def nb_maison(lst):
    compteur = 0
    for b in lst :
        if b.nt=='maison':
            compteur = compteur + 1
    return compteur
```

Question 5

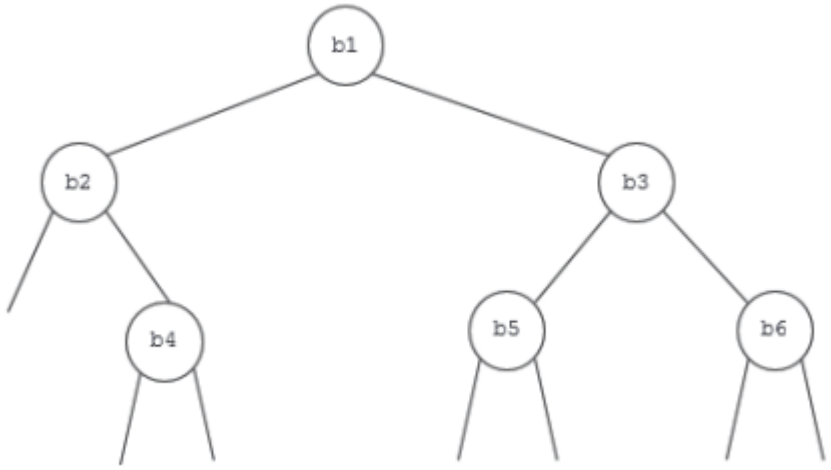
Pour une recherche efficace des biens immobiliers selon le critère de leur surface, on stocke les objets de type `Bim` dans un arbre binaire de recherche, nommé `abr`. Pour tout nœud de cet arbre :

- tous les objets de son sous-arbre gauche ont une surface inférieure ou égale à la surface de l'objet contenue dans ce nœud ;
- tous les objets de son sous-arbre droit ont une surface strictement supérieure à la surface de l'objet contenue dans ce nœud.

L'objet abr dispose des méthodes suivantes :

- `abr.est_vide()` : renvoie True si abr est vide et False sinon.
- `abr.get_v()` : renvoie l'élément (de type `Bim`) situé à la racine de `abr` si `abr` n'est pas vide et `None` sinon.
- `abr.get_g()` : renvoie le sous-arbre gauche de `abr` si `abr` n'est pas vide et `None` sinon.
- `abr.get_d()` : renvoie le sous-arbre droit de abr si `abr` n'est pas vide et `None` sinon.

a. Dans cette question, on suppose que l'arbre binaire abr a la forme ci-dessous :



Donner la liste les biens b1, b2, b3, b4, b5, b6 triée dans l'ordre croissant de leur surface.
b. Recopier et compléter le code de la fonction récursive contient donnée ci-dessous, qui prend en arguments un nombre surface de type float et un arbre binaire de recherche abr contenant des éléments de type Bim ordonnés selon leur attribut de surface sf. La fonction `contient(surface, abr)` renvoie `True` s'il existe un bien dans abr d'une surface supérieure ou égale à surface et `False` sinon.

🐍 **Script Python**

```python
def contient(surface, abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= ...:
        return True
    else:
        return contient( surface,... )
```

Réponse

a. on effectue un parcours infixe : b2 - b4 - b1 - b5 - b3 - b6

b.

🐍 **Script Python**

```python
def contient(surface,abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= surface:
        return True
    else :
        return contient(surface, abr.get_d())
```

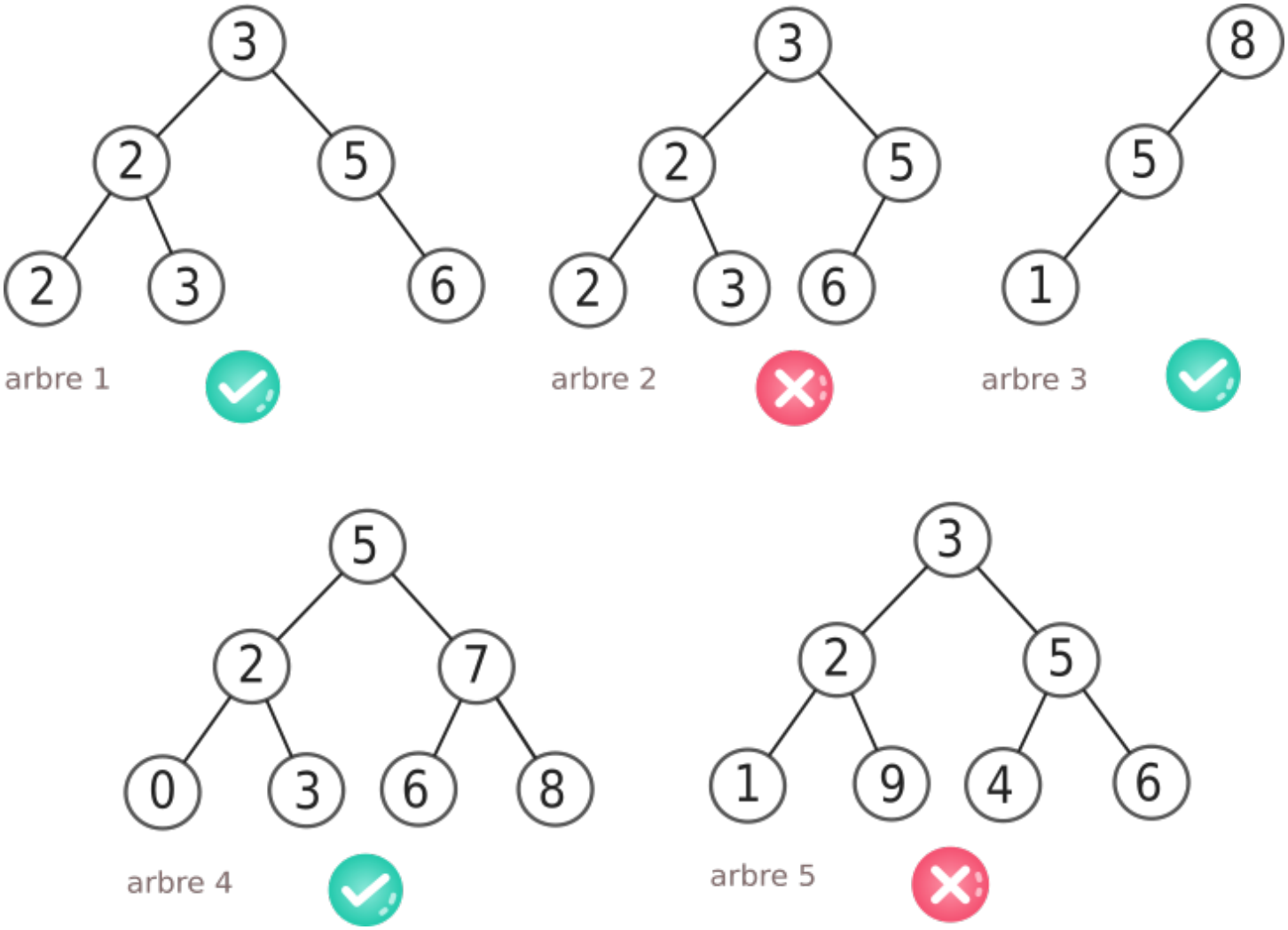
C12 arbres binaires de recherche

1. Arbres binaires de recherche (ABR)

Définition d'un ABR

Un **arbre binaire de recherche** est un arbre binaire dont les valeurs des nœuds (valeurs qu'on appelle étiquettes, ou clés) vérifient la propriété suivante :

- l'étiquette d'un nœud est **supérieure ou égale** à celle de **chaque** nœud de son **sous-arbre gauche**.
- l'étiquette d'un nœud est **strictement inférieure** à celle du **chaque** nœud de son **sous-arbre droit**.



À noter que l'arbre 3 (qui est bien un ABR) est appelé **arbre filiforme**.

L'arbre 5 n'est pas un ABR à cause de la feuille 9, qui fait partie du sous-arbre gauche de 3 sans lui être inférieure.

Remarque : on pourrait aussi définir un ABR comme un arbre dont le parcours infixe est une suite croissante.

1.1. Déterminer si un arbre est un ABR

Employer une méthode récursive imposerait de garder en mémoire dans l'exploration des sous-arbres la valeur maximale ou minimale. Nous allons plutôt utiliser la remarque précédente, et nous servir du parcours infixe.

Method

récupérer le parcours infixe dans une liste, et faire un test sur cette liste.

Être ou ne pas être un ABR ❤️

```python
def infixe(arbre, s = None):
    if s is None:
        s = []
    if arbre is None :
        return None
    infixe(arbre.left, s)
    s.append(arbre.data)
    infixe(arbre.right, s)
    return s


def est_ABR(arbre):
    '''renvoie un booléen indiquant si arbre est un ABR'''
    parcours = infixe(arbre)
    return parcours == sorted(parcours) # on regarde si le parcours est égal au parcours trié
```

🐍 **Script Python**

```python
# arbres-tests

#arbre n°4
a = Arbre(5)
a.left = Arbre(2)
a.right = Arbre(7)
a.left.left = Arbre(0)
a.left.right = Arbre(3)
a.right.left = Arbre(6)
a.right.right = Arbre(8)
```

```
#arbre n°5
b = Arbre(3)
b.left = Arbre(2)
b.right = Arbre(5)
b.left.left = Arbre(1)
b.left.right = Arbre(9)
b.right.left = Arbre(4)
b.right.right = Arbre(6)
```

🐍 **Script Python**

```
>>> est_ABR(a)
True
>>> est_ABR(b)
False
```

1.2. Rechercher une clé dans un ABR

Un arbre binaire de taille \(n\) contient \(n\) clés (pas forcément différentes). Pour savoir si une valeur particulière fait partie des clés, on peut parcourir tous les nœuds de l'arbre, jusqu'à trouver (ou pas) cette valeur dans l'arbre. Dans le pire des cas, il faut donc faire \(n\) comparaisons.

Mais si l'arbre est un ABR, le fait que les valeurs soient «rangées» va considérablement améliorer la vitesse de recherche de cette clé, puisque la moitié de l'arbre restant sera écartée après chaque comparaison.

Recherche d'une clé dans un ABR ❤️

🐍 **Script Python**

```python
def contient_valeur(arbre, valeur):
    if arbre is None :
        return False
    if arbre.data == valeur :
        return True
    if valeur < arbre.data :
        return contient_valeur(arbre.left, valeur)
    else:
        return contient_valeur(arbre.right, valeur)
```

Exemple

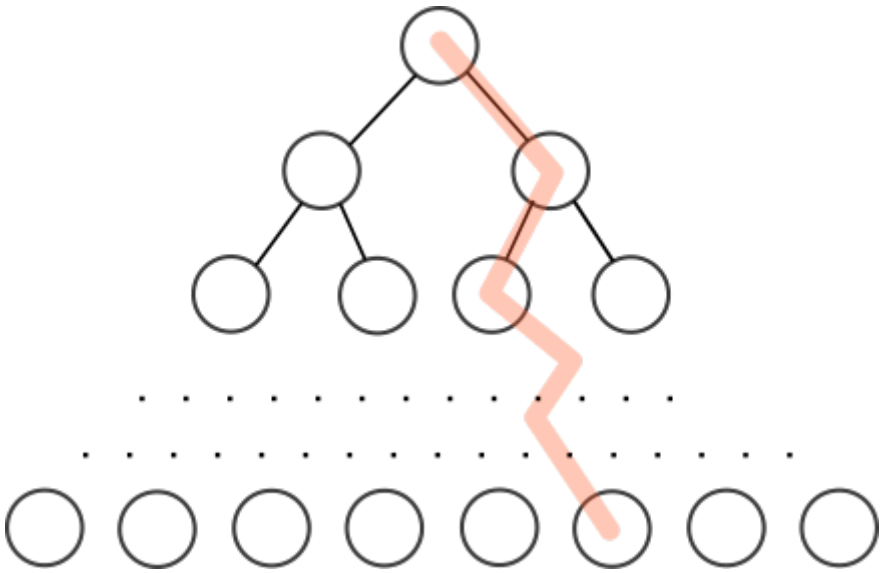
L'arbre `a` contient la valeur 8, mais l'arbre `b` ne la contient pas :

🐍 **Script Python**

```
>>> contient_valeur(a,8)
True
```

```
>>> contient_valeur(b,8)
False
```

1.3. Coût de la recherche dans un ABR équilibré



Imaginons un arbre équilibré de taille \(n\). Combien d'étapes faudra-t-il, dans le pire des cas, pour trouver (ou pas) une clé particulière dans cet arbre ?

Après chaque nœud, le nombre de nœuds restant à explorer est divisé par 2. On retrouve là le principe de recherche dichotomique.

S'il faut parcourir tous les étages de l'arbre avant de trouver (ou pas) la clé recherchée, le nombre de nœuds parcourus est donc égal à la hauteur \(h\) de l'arbre.

Pour un arbre complet, cette hauteur vérifie la relation \(2^h -1= n\). et donc \(2^h = n+1\).

\(h\) est donc le «nombre de puissance de 2» que l'on peut mettre dans \(n+1\). Cette notion s'appelle le logarithme de base 2 et se note \(\log_2\).

Par exemple, \(\log_2(64)=6\) car \(2^6=64\).

Le nombre maximal de nœuds à parcourir pour rechercher une clé dans un ABR équilibré de taille \(n\) est donc de l'ordre de \(\log_2(n)\), ce qui est très performant !

Pour arbre contenant 1000 valeurs, 10 étapes suffisent.

Cette **complexité logarithmique** est un atout essentiel de la structure d'arbre binaire de recherche.

1.4. Insertion dans un ABR

L'insertion d'une clé va se faire au niveau d'une feuille, donc au bas de l'arbre. Dans la version récursive de l'algorithme d'insertion, que nous allons implémenter, il n'est pourtant pas nécessaire de descendre manuellement dans l'arbre jusqu'au bon endroit : il suffit de distinguer dans lequel des deux

sous-arbres gauche et droit doit se trouver la future clé, et d'appeler récursivement la fonction d'insertion dans le sous-arbre en question.

Algorithme :

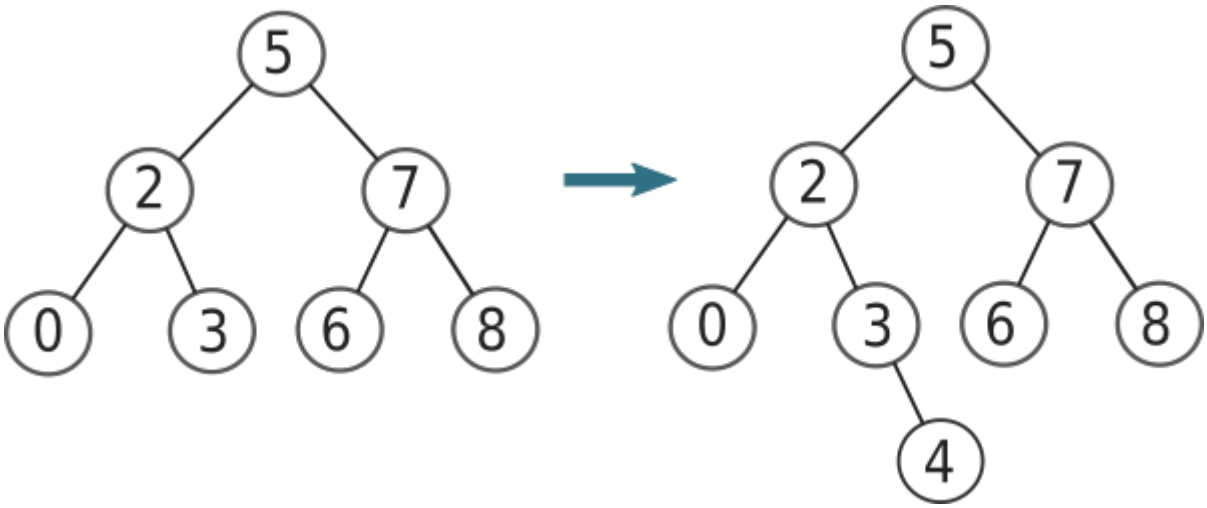
- Si l'arbre est vide, on renvoie un nouvel objet Arbre contenant la clé.
- Sinon, on compare la clé à la valeur du nœud sur lequel on est positionné :
  - Si la clé est inférieure à cette valeur, on va modifier le sous-arbre gauche en le faisant pointer vers ce même sous-arbre une fois que la clé y aura été injecté, par un appel récursif.
  - Si la clé est supérieure, on fait la même chose avec l'arbre de droite.
  - on renvoie le nouvel arbre ainsi créé.

Insertion dans un ABR ❤️

🐍 **Script Python**

```python
def insertion(arbre, valeur):
    if arbre is None :
        return Arbre(valeur)
    else :
        v = arbre.data
        if valeur <= v :
            arbre.left = insertion(arbre.left, valeur)
        else:
            arbre.right = insertion(arbre.right, valeur)
        return arbre
```

Exemple : Nous allons insérer la valeur 4 dans l'arbre `a` et vérifier par un parcours infixe (avant et après l'insertion) que la valeur 4 a bien été insérée au bon endroit.



🐍 **Script Python**

```
a = Arbre(5)
a.left = Arbre(2)
a.right = Arbre(7)
a.left.left = Arbre(0)
a.left.right = Arbre(3)
a.right.left = Arbre(6)
a.right.right = Arbre(8)
```

🐍 **Script Python**

```
>>> infixe(a)
0-2-3-5-6-7-8-
>>> insertion(a,4)
<__main__.Arbre at 0x7f46f0507e80>
>>> infixe(a)
0-2-3-4-5-6-7-8-
```

La valeur 4 a donc bien été insérée au bon endroit.

2. Exercices :

Exo

1. Dessiner un arbre binaire de recherche *complet*, contenant les valeurs suivantes : 2,7,10,15,19,24,25.
2. Même question lorsque l'arbre à la structure suivante :

```
graph TD
A[" "] --> B[" "]
A --> C[" "]
B --> D[" "]
B --> E[" "]
C --> V1[" "]
C --> F[" "]
D --> V2[" "]
D --> G[" "]
style V1 fill:#FFFFFF, stroke:#FFFFFF
linkStyle 4 stroke:#FFFFFF,stroke-width:0px
style V2 fill:#FFFFFF, stroke:#FFFFFF
linkStyle 6 stroke:#FFFFFF,stroke-width:0px
```

Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.

3. Bibliographie

- Numérique et Sciences Informatiques, Terminale, T. BALABONSKI, S. CONCHON, J.-C. FILLIATRE, K. NGUYEN, éditions ELLIPSES.

Sujet BAC 12 : Arbres Binaires de Recherche

1. 2020, sujet 0

2020, sujet 0

Question 1

Déterminer la taille et la hauteur de l'arbre binaire suivant :

```
graph TD
    A("A") --> B("B")
    B --> C("C")
    B --> D("D")
    A --> E("E")
    C --> C1(" ")
    C --> C2(" ")
    D --> G("G")
    D --> D2(" ")
    E --> F("F")
    E --> E2(" ")
    F --> H("H")
    F --> I("I")
    E2 --> E3(" ")
    E2 --> E4(" ")
    linkStyle 4 stroke-width:0px;
    style C1 opacity:0;
    linkStyle 5 stroke-width:0px;
    style C2 opacity:0;
    linkStyle 7 stroke-width:0px;
    style D2 opacity:0;
    linkStyle 9 stroke-width:0px;
    style E2 opacity:0;
    linkStyle 12 stroke-width:0px;
    style E3 opacity:0;
    linkStyle 13 stroke-width:0px;
    style E4 opacity:0;
```

corrigé

La taille est 9, la hauteur est 4.

Question 2

On décide de numéroter en binaire les nœuds d'un arbre binaire de la façon suivante :

- la racine correspond à 1 ;
- la numérotation pour un fils gauche s'obtient en ajoutant le chiffre 0 à droite au numéro de son père ;
- la numérotation pour un fils droit s'obtient en ajoutant le chiffre 1 à droite au numéro de son père ;

Par exemple, dans l'arbre ci-dessous, on a utilisé ce procédé pour numéroter les nœuds A, B, C, E et F .

```
graph TD
   A("A : 1") --> B("B : 10")
   B --> C("C : 100")
   B --> D("D : ?")
   A --> E("E : 11")
   C --> C1(" ")
   C --> C2(" ")
   D --> G("G: ?")
   D --> D2(" ")
   E --> F("F : 110")
   E --> E2(" ")
   F --> H("H : ?")
   F --> I("I : ?")
   E2 --> E3(" ")
   E2 --> E4(" ")
   linkStyle 4 stroke-width:0px;
   style C1 opacity:0;
   linkStyle 5 stroke-width:0px;
   style C2 opacity:0;
   linkStyle 7 stroke-width:0px;
   style D2 opacity:0;
   linkStyle 9 stroke-width:0px;
   style E2 opacity:0;
   linkStyle 12 stroke-width:0px;
   style E3 opacity:0;
   linkStyle 13 stroke-width:0px;
   style E4 opacity:0;
```

1. Dans l'exemple précédent, quel est le numéro en binaire associé au nœud G ?
2. Quel est le nœud dont le numéro en binaire vaut 13 en décimal ?
3. En notant \(h\) la hauteur de l'arbre, sur combien de bits seront numérotés les nœuds les plus en bas ?

4. Justifier que pour tout arbre de hauteur h et de taille $n \geqslant 2$, on a : $h \leqslant n \leqslant 2^h-1$

corrigé

1. G est associé à 1010.
2. 13 s'écrit 1101 en binaire, c'est donc le nœud I.
3. Les nœuds les plus en bas sont notés sur h bits.
4. L'arbre de hauteur h de taille minimale est l'arbre filiforme, qui est de taille h. L'arbre de hauteur h de taille maximale est l'arbre complet, qui est de taille 2^h-1. Si n est la taille d'un arbre quelconque de taille h, on a donc bien $$ h \leqslant n \leqslant 2^h-1 $$.

Question 3

Un arbre binaire est dit complet si tous les niveaux de l'arbre sont remplis.

```
graph TD
    A("A") --> B("B")
    B --> D("D")
    B --> E("E")
    A --> C("C")
    C --> F("F")
    C --> G("G")
    D --> H("H")
    D --> I("I")
    E --> J("J")
    E --> K("K")
    F --> L("L")
    F --> M("M")
    G --> N("N")
    G --> O("O")
```

On décide de représenter un arbre binaire complet par un tableau de taille n + 1, où n est la taille de l'arbre, de la façon suivante :

- La racine a pour indice 1 ;
- Le fils gauche du nœud d'indice i a pour indice $2 \times i$;
- Le fils droit du nœud d'indice i a pour indice $2 \times i + 1$;
- On place la taille n de l'arbre dans la case d'indice 0.

Répondre aux questions suivantes :

1. Déterminer le tableau qui représente l'arbre binaire complet de l'exemple précédent.
2. On considère le père du nœud d'indice i avec $i \geqslant 2$. Quel est son indice dans le tableau ?

corrigé

1. Tableau : `[15, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O]` .
2. Le père du nœud d'indice `i` a pour indice `i//2` .

Question 4

On se place dans le cas particulier d'un arbre binaire de recherche complet où les nœuds contiennent des entiers et pour lequel la valeur de chaque noeud est supérieure à celles des noeuds de son fils gauche, et inférieure à celles des noeuds de son fils droit.

Écrire une fonction `recherche` ayant pour paramètres un arbre `arbre` et un élément `element`. Cette fonction renvoie `True` si `element` est dans l'arbre et `False` sinon. L'arbre sera représenté par un tableau comme dans la question précédente.

corrigé

🐍 **Script Python**

```python
def recherche(arbre, element):
    i = 1
    while i < len(arbre):
        if arbre[i] == element:
            return True
        if element < arbre[i]:
            i = 2*i # on se place sur le fils gauche
        else:
            i = 2*i +  1 # on se place sur le fils droit
    return False
```

2. 2021, Métropole sujet 1

2021, Métropole sujet 1

Dans cet exercice, les arbres binaires de recherche ne peuvent pas comporter plusieurs fois la même clé. De plus, un arbre binaire de recherche limité à un nœud a une hauteur de 1. On considère l'arbre binaire de recherche représenté ci-dessous (figure 1), où `val` représente un entier :

```
graph TD
    A(18) --> B(15)
    B --> D(13)
    B --> E(val)
    A --> C(23)
    C --> F(19)
    C --> G(32)
    D --> H(12)
    D --> D1(" ")
    E --> E1(" ")
    E --> E2(" ")
```

```
F --> F1(" ")
F --> M(21)
G --> G1(" ")
G --> G2(" ")
linkStyle 7 stroke-width:0px;
style D1 opacity:0;
linkStyle 8 stroke-width:0px;
style E1 opacity:0;
linkStyle 9 stroke-width:0px;
style E2 opacity:0;
linkStyle 10 stroke-width:0px;
style F1 opacity:0;
linkStyle 12 stroke-width:0px;
style G1 opacity:0;
linkStyle 13 stroke-width:0px;
style G2 opacity:0;
```

Question 1

a Donner le nombre de feuilles de cet arbre et préciser leur valeur (étiquette).
b Donner le sous arbre-gauche du nœud 23.
c Donner la hauteur et la taille de l'arbre.
d Donner les valeurs entières possibles de `val` pour cet arbre binaire de recherche.

corrigé

1.a. Il y a 4 feuilles, d'étiquette 12, `val`, 21 et 32.
1.b. Le sous-arbre gauche du nœud 23 est 19-21.
1.c. La hauteur de l'arbre est 4. Sa taille est 9.
1.d. Les valeurs possibles de `val` sont 16 et 17.

On suppose, pour la suite de cet exercice, que `val` est égal à 16.

Question 2

On rappelle qu'un parcours infixe depuis un nœud consiste, dans l'ordre, à faire un parcours infixe sur le sous arbre-gauche, afficher le nœud puis faire un parcours infixe sur le sous-arbre droit.
Dans le cas d'un parcours suffixe, on fait un parcours suffixe sur le sous-arbre gauche puis un parcours suffixe sur le sous-arbre droit, avant d'afficher le nœud.

a. Donner les valeurs d'affichage des nœuds dans le cas du parcours infixe de l'arbre.
b. Donner les valeurs d'affichage des nœuds dans le cas du parcours suffixe de l'arbre.

corrigé

2.a. Parcours infixe : 12-13-15-16-18-19-21-23-32

2.b. Parcours suffixe : 12-13-16-15-21-19-32-23-18

Question 3

On considère la classe `Noeud` définie de la façon suivante en Python :

🐍 **Script Python**

```
class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere :
            if v == n.v:
                est_insere = True            | Bloc 1
            elif v < n.v:
                if n.ag != None:          ---
                    n = n.ag                |
                else:                     |  Bloc 2
                    n.ag = Noeud(v)         |
                    est_insere = True      ---
            else:
                if n.ad != None:          ---
                    n = n.ad                |
                else:                     | Bloc 3
                    n.ad = Noeud(v)         |
                    est_insere = True      ---

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)
```

a. Représenter l'arbre construit suite à l'exécution de l'instruction suivante :

🐍 **Script Python**

```
racine = Noeud(18)
racine.insere_tout([12, 13, 15, 16, 19, 21, 32, 23])
```

b. Écrire les deux instructions permettant de construire l'arbre de la figure 1. On rappelle que le nombre `val` est égal à 16.

c. On considère l'arbre tel qu'il est présenté sur la figure 1. Déterminer l'ordre d'exécution des blocs (repérés de 1 à 3) suite à l'application de la méthode `insere(19)` au nœud racine de cet arbre.

corrigé

a.

```
graph TD
    A(18) --> B(12)
    B --> D(" ")
    B --> E(13)
    A --> C(19)
    C --> F(" ")
    C --> G(21)
    D --> H(" ")
    D --> D1(" " )
    E --> E1(" ")
    E --> E2(15)
    F --> F1(" ")
    F --> M(" ")
    M --> G1(" ")
    M --> G2(" ")
    E2 --> I(" ")
    E2 --> J(16)
    G --> K(" ")
    G --> L(32)
    L --> N(23)
    L --> O(" ")
    linkStyle 1 stroke-width:0px;
    style D opacity:0;
    linkStyle 4 stroke-width:0px;
    style F opacity:0;
    linkStyle 6 stroke-width:0px;
    style H opacity:0;
    linkStyle 7 stroke-width:0px;
    style D1 opacity:0;
    linkStyle 8 stroke-width:0px;
    style E1 opacity:0;
    linkStyle 10 stroke-width:0px;
    style F1 opacity:0;
    linkStyle 11 stroke-width:0px;
    style M opacity:0;
    linkStyle 12 stroke-width:0px;
    style G1 opacity:0;
    linkStyle 13 stroke-width:0px;
    style G2 opacity:0;
    linkStyle 14 stroke-width:0px;
    style I opacity:0;
    linkStyle 16 stroke-width:0px;
    style K opacity:0;
    linkStyle 19 stroke-width:0px;
    style O opacity:0;
```

b.

🐍 **Script Python**

```python
racine = Noeud(18)
racine.insere([15, 13, 12, 16, 23, 32, 19, 21])
```

(d'autres solutions sont possibles)

3.c. Bloc 3 - Bloc 2 - Bloc 1

Question 4

Écrire une méthode `recherche(self, v)` qui prend en argument un entier `v` et renvoie la valeur `True` si cet entier est une étiquette de l'arbre, `False` sinon.

corrigé

```python
class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere:
            if v == n.v:
                est_insere = True
            elif v < n.v:
                if n.ag != None:
                    n = n.ag
                else:
                    n.ag = Noeud(v)
                    est_insere = True
            else:
                if n.ad != None:
                    n = n.ad
                else:
                    n.ad = Noeud(v)
                    est_insere = True

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)

    def recherche(self, v):
        arbre = self
        while not arbre is None:
            if arbre.v == v:
                return True
            if v < arbre.v:
                arbre = arbre.ag
            else:
                arbre = arbre.ad
        return False


    # version récursive (non demandée)

    def recherche_rec(self, v):
        if self is None:
            return False
        if self.v == v:
            return True
        if v < self.v:
            if self.ag is not None:
                return self.ag.recherche_rec(v)
            else:
                return False
        else:
```

```
        if self.ad is not None:
            return self.ad.recherche_rec(v)
        else:
            return False


racine = Noeud(18)
racine.insere_tout([12, 13, 15, 14, 19, 21, 32, 23])
print(racine.recherche(149))
print(racine.recherche(12))
```

3. 2021, Métropole Candidats Libres 2

2021, Métropole Candidats Libres 2

On rappelle qu’un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit. Un nœud sans sous-arbre est appelé feuille. La taille d’un arbre est le nombre de nœuds qu’il contient ; sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l’une des feuilles. Ainsi la hauteur d’un arbre réduit à un nœud, c’est-à-dire la racine, est 1.

Dans un arbre binaire de recherche, chaque nœud contient une clé, ici un nombre entier, qui est :

- strictement supérieure à toutes les clés des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les clés des nœuds du sous-arbre droit.

Un arbre binaire de recherche est dit « bien construit » s’il n’existe pas d’arbre de hauteur inférieure qui pourrait contenir tous ses nœuds.

On considère l’arbre binaire de recherche ci-dessous.

```
graph TD
   A(12) --> B(10)
   B --> D(5)
   B --> E(" ")
   A --> C(15)
   C --> F(" ")
   C --> G(20)
   D --> H(4)
   D --> D1(8)
   E --> E1(" ")
   E --> E2(" ")
   F --> F1(" ")
   F --> M(" ")
   G --> G1(" ")
   G --> G2(" ")
   linkStyle 2 stroke-width:0px;
   style E opacity:0;
   linkStyle 4 stroke-width:0px;
   style F opacity:0;
   linkStyle 2 stroke-width:0px;
   style E opacity:0;
   linkStyle 8 stroke-width:0px;
   style E1 opacity:0;
   linkStyle 9 stroke-width:0px;
   style E2 opacity:0;
   linkStyle 10 stroke-width:0px;
   style M opacity:0;
   linkStyle 11 stroke-width:0px;
   style F1 opacity:0;
   linkStyle 12 stroke-width:0px;
   style G1 opacity:0;
   linkStyle 13 stroke-width:0px;
   style G2 opacity:0;
```

Question 1

a. Quelle est la taille de l’arbre ci-dessus ?

b. Quelle est la hauteur de l’arbre ci-dessus ?

corrigé

1.a. La taille de l'arbre est 7.
1.b. La hauteur de l'arbre est 4.

Question 2

Cet arbre binaire de recherche n’est pas « bien construit ». Proposer un arbre binaire de recherche contenant les mêmes clés et dont la hauteur est plus petite que celle de l’arbre initial.

corrigé

```
graph TD
    A(10) --> B(5)
    B --> D(4)
    B --> E(8)
    A --> C(15)
    C --> F(12)
    C --> G(20)
```

Question 3

Les classes Noeud et Arbre ci-dessous permettent de mettre en œuvre en Python la structure d'arbre binaire de recherche. La méthode `insere` permet d'insérer récursivement une nouvelle clé.

```python
class Noeud :

    def __init__(self, cle):
        self.cle = cle
        self.gauche = None
        self.droit = None

    def insere(self, cle):
        if cle < self.cle :
            if self.gauche == None :
                self.gauche = Noeud(cle)
            else :
                self.gauche.insere(cle)
        elif cle > self.cle :
            if self.droit == None :
                self.droit = Noeud(cle)
            else :
                self.droit.insere(cle)

class Arbre :

    def __init__(self, cle):
        self.racine = Noeud(cle)

    def insere(self, cle):
        self.racine.insere(cle)
```

Donner la représentation de l'arbre codé par les instructions ci-dessous.

🐍 **Script Python**

```python
a = Arbre(10)
a.insere(20)
a.insere(15)
a.insere(12)
a.insere(8)
a.insere(4)
a.insere(5)
```

corrigé

```
graph TD
    A(10) --> B(8)
    B --> D(4)
    D --> D1(" ")
    D --> D2(5)
    B --> E(" " )
    E --> E1(" ")
    E --> E2(" ")
    A --> C(20)
    C --> F(15)
    C --> G(" ")
    F --> H(12)
    F --> J(" ")
    G --> G1(" ")
    G --> G2(" ")
    linkStyle 2 stroke-width:0px;
    style D1 opacity:0;
    linkStyle 4 stroke-width:0px;
    style E opacity:0;
    linkStyle 5 stroke-width:0px;
    style E opacity:0;
    linkStyle 6 stroke-width:0px;
    style E1 opacity:0;
    style E2 opacity:0;
    linkStyle 9 stroke-width:0px;
    style G opacity:0;
    linkStyle 11 stroke-width:0px;
    style J opacity:0;
    linkStyle 12 stroke-width:0px;
    style G1 opacity:0;
    linkStyle 13 stroke-width:0px;
    style G2 opacity:0;
```

Question 4

Pour calculer la hauteur d'un arbre non vide, on a écrit la méthode ci-dessous dans la classe Noeud.

🐍 **Script Python**

```python
def hauteur(self):
    if self.gauche == None and self.droit == None:
        return 1
    if self.gauche == None:
        return 1 + self.droit.hauteur()
    elif self.droit == None:
        return 1 + self.gauche.hauteur()
    else:
        hg = self.gauche.hauteur()
        hd = self.droit.hauteur()
        if hg > hd:
            return hg + 1
        else:
            return hd + 1
```

Écrire la méthode `hauteur` de la classe `Arbre` qui renvoie la hauteur de l'arbre.

corrigé

4.

```python
def hauteur(self):
    return self.racine.hauteur()
```

Question 5

Écrire les méthodes `taille` des classes `Noeud` et `Arbre` permettant de calculer la taille d'un arbre.

corrigé

5. Méthode `taille` de la classe `Noeud` :

```
def taille(self):
    if self.gauche is None and self.droit is None:
        return 1
    elif self.gauche is None:
        return 1 + self.droit.taille()
    elif self.droit is None:
        return 1 + self.gauche.taille()
    else:
        return 1 + self.gauche.taille() + self.droit.taille()
```

Méthode `taille` de la classe `Arbre` :

```
def taille(self):
    return self.racine.taille()
```

Question 6

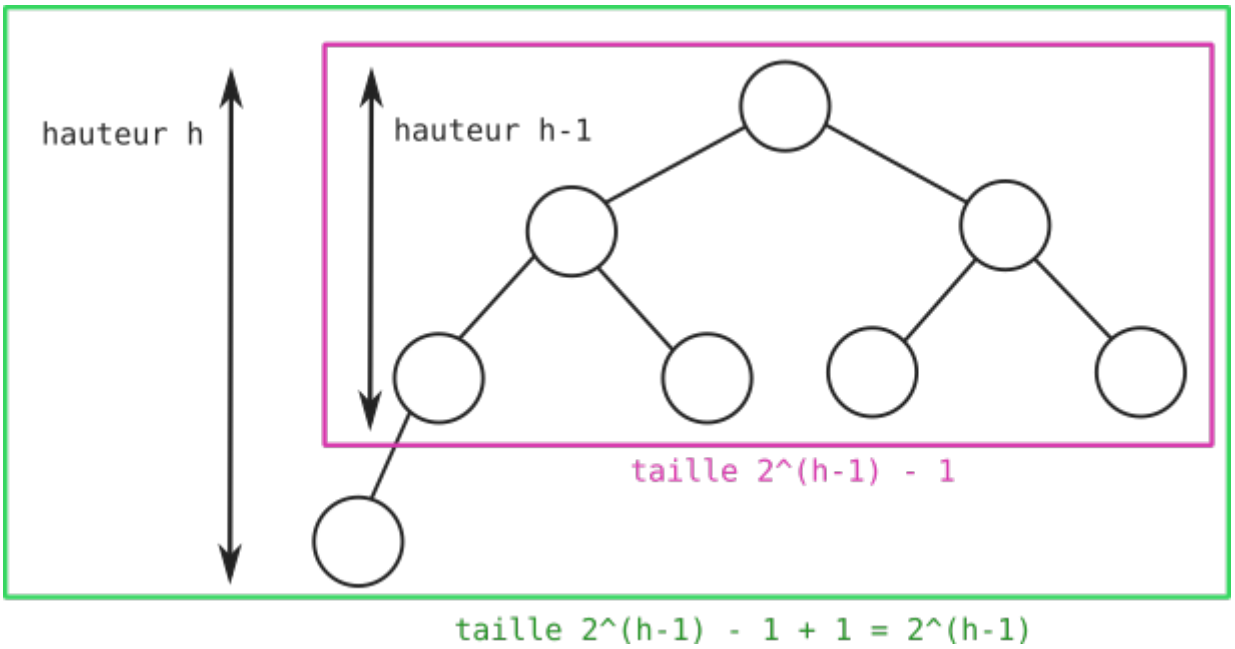
On souhaite écrire une méthode `bien_construit` de la classe `Arbre` qui renvoie la valeur `True` si l'arbre est « bien construit » et `False` sinon.

On rappelle que la taille maximale d'un arbre binaire de recherche de hauteur \(h\) est \(2^h - 1\).

a. Quelle est la taille minimale, notée `min` d'un arbre binaire de recherche « bien construit » de hauteur \(h\) ?
b. Écrire la méthode `bien_construit` demandée.

corrigé

6.a. La configuration minimale d'un arbre bien construit de hauteur \(h\) peut être :



La taille minimale `min` est donc égale à \(2^{h-1}\).

6.b. Intuitivement, un arbre est *mal construit* si sa hauteur est trop grande par rapport à sa taille (trop *étiré*).

Donc un arbre est *mal construit* si sa taille est trop petite par rapport à sa hauteur.

Donc un arbre de taille \(t\) et de hauteur \(h\) est *mal construit* si \(t < 2^{h-1}\), puisqu'on a démontré que \(2^{h-1}\) était la taille minimale.

Pour tester si un arbre est *bien construit*, on va donc juste vérifier que \(t \geqslant 2^{h-1}\) :

```
def bien_construit(self):
    h = self.taille()
    return self.taille() >= 2**(h-1)
```

4. 2021, Polynésie

2021, Polynésie

Cet exercice traite principalement du thème « algorithmique, langages et programmation » et en particulier les arbres binaires de recherche. La première partie aborde les arbres en mode débranché

via l'application d'un algorithme sur un exemple.
La suivante porte sur la programmation orientée objet. La dernière partie fait le lien avec les algorithmes de tri.

Partie A : Étude d'un exemple

Considérons l'arbre binaire de recherche ci-dessous :

```
graph TD
    A(5) --> B(2)
    B --> D(" ")
    B --> E(3)
    A --> C(7)
    C --> F(" ")
    C --> G(8)
    linkStyle 1 stroke-width:0px;
    style D opacity:0;
    linkStyle 4 stroke-width:0px;
    style F opacity:0;
```

Question A.1

Indiquer quelle valeur a le nœud racine et quels sont les fils de ce nœud.

Réponse

- Racine : 5
- Fils gauche : 2
- Fils droit : 7

Question A.2

Indiquer quels sont les nœuds de la branche qui se termine par la feuille qui a pour valeur 3.

Réponse

5, 2 et 3

Question A.3

Dessiner l’arbre obtenu après l’ajout de la valeur 6.

Réponse

```
graph TD
    A(5) --> B(2)
    B --> D(" ")
    B --> E(3)
    A --> C(7)
    C --> F(6)
    C --> G(8)
    linkStyle 1 stroke-width:0px;
    style D opacity:0;
```

Partie B : Implémentation en Python

Voici un extrait d’une implémentation en Python d'une classe modélisant un arbre binaire de recherche.

```python
class ABR:
    """Implémentation d’un arbre binaire de recherche (ABR)"""
    def __init__(self, valeur=None):
        self.valeur = valeur
        self.fg = None
        self.fd = None

    def estVide(self):
        return self.valeur == None

    def insererElement(self, e):
        if self.estVide():
            self.valeur = e
        else:
            if e < self.valeur:
                if self.fg:
                    self.fg.insererElement(e)
                else:
                    self.fg = ABR(e)
            if e > self.valeur:
                if self.fd:
                    self.fd.insererElement(e)
                else:
                    self.fd = ABR(e)
```

Question B.1

Expliquer le rôle de la fonction `__init__`.

Réponse

C'est un constructeur qui permet d'initialiser les attributs de l'instance.

Question B.2

Dans cette implémentation, expliquer ce qui se passe si on ajoute un élément déjà présent dans l'arbre.

Réponse

Il ne se passe rien.

Question B.3

Recopier et compléter les pointillés ci-dessous permettant de créer l'arbre de la partie A.

🐍 **Script Python**

```
arbre = ABR(..........)
arbre.insererElement(2)
arbre.insererElement(..........)
arbre.insererElement(7)
arbre.insererElement(..........)
```

Réponse

🐍 **Script Python**

```python
arbre = ABR(5)
arbre.insereElement(2)
arbre.insereElement(3)
arbre.insereElement(7)
arbre.insereElement(8)
```

Partie C : Tri par arbre binaire de recherche

On souhaite trier un ensemble de valeurs entières distinctes grâce à un arbre binaire de recherche. Pour cela, on ajoute un à un les éléments de l'ensemble dans un arbre initialement vide. Il ne reste plus qu'à parcourir l'arbre afin de lire et de stocker dans un tableau résultat les valeurs dans l'ordre croissant.

Question C.1

Donner le nom du parcours qui permet de visiter les valeurs d'un arbre binaire de recherche dans l'ordre croissant.

Réponse

Parcours infixe.

Question C.2

Comparer la complexité de cette méthode de tri avec celle du tri par insertion ou du tri par sélection.

Réponse

Elle est en $O(nlog(n))$ alors que les autres sont en $O(n^2)$. Elle est plus efficace.

5. 2021, Centres Étrangers, sujet 1

2021, Centres Etrangers, sujet 1

Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit).

```
graph TD
   X("X") --> G1("G1")
   G1 --> G2("G2")
   G1 --> G3("G3")
   X --> D1("D1")
   D1 --> D2("D2")
   D1 --> D3("D3")
```

- X est un nœud, sa valeur est X.valeur
- G1 est le fils gauche de X, noté X.fils_gauche
- D1 est le fils droit de X, noté X.fils_droit

Un arbre binaire de recherche est ordonné de la manière suivante :

Pour chaque nœud X,

- les valeurs de tous les nœuds du sous-arbre gauche sont **strictement inférieures** à la valeur du nœud X
- les valeurs de tous les nœuds du sous-arbre droit sont **supérieures ou égales** à la valeur du nœud X.

Ainsi, par exemple, toutes les valeurs des nœuds G1, G2 et G3 sont strictement inférieures à la valeur du nœud X et toutes les valeurs des nœuds D1, D2 et D3 sont supérieures ou égales à la valeur du nœud X.

Voici un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les valeurs : `[26, 3, 42, 15, 29, 19, 13, 1, 32, 37, 30]`

L'étiquette d'un nœud indique la valeur du nœud suivie du nom du nœud. Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous.

`'29, noeud04'` signifie que le nœud nommé `noeud04` possède la valeur 29.

```
graph TD
   A("26, noeud00") --> B("3, noeud01")
   B --> D("1, noeud07")
   B --> E("15, noeud03")
   A --> C("42, noeud02")
   C --> F("29, noeud04")
   C --> G(" ")
   D --> H(" ")
   D --> D1(" " )
```

```
E --> E1("13, noeud06")
E --> E2("19, noeud05")
F --> F1(" ")
F --> M("32, noeud08")
M --> G1("30, noeud10")
M --> G2("37, noeud09")
style G opacity:0;
linkStyle 6 stroke-width:0px;
style H opacity:0;
linkStyle 7 stroke-width:0px;
style D1 opacity:0;
style F1 opacity:0;
```

Question 1

On insère la valeur 25 dans l'arbre, dans un nouveau nœud nommé nœud11.

Recopier l'arbre binaire de recherche étudié et placer la valeur 25 sur cet arbre en coloriant en rouge le chemin parcouru.

Préciser sous quel nœud la valeur 25 sera insérée et si elle est insérée en fils gauche ou en fils droit, et expliquer toutes les étapes de la décision.

Réponse

```
graph TD
    A("26, noeud00") --> B("3, noeud01")
    B --> D("1, noeud07")
    B --> E("15, noeud03")
    A --> C("42, noeud02")
    C --> F("29, noeud04")
    C --> G(" ")
    D --> H(" ")
    D --> D1(" " )
    E --> E1("13, noeud06")
    E --> E2("19, noeud05")
    F --> F1(" ")
    F --> M("32, noeud08")
    M --> G1("30, noeud10")
    M --> G2("37, noeud09")
    E2 --> O(" ")
    E2 --> P("25, noeud11")
    style G opacity:0;
    linkStyle 6 stroke-width:0px;
    style H opacity:0;
    linkStyle 7 stroke-width:0px;
    style D1 opacity:0;
    style F1 opacity:0;
    linkStyle 14 stroke-width:0px;
    style O opacity:0;
    linkStyle 0 stroke:red;
    linkStyle 2 stroke:red;
    linkStyle 9 stroke:red;
    linkStyle 15 stroke:red;
```

On désire insérer le noeud11 (valeur 25). On part de la racine (noeud00 de valeur 26), 25 est plus petit que 26, on considère donc le sous-arbre gauche et on se retrouve au niveau du noeud01 (valeur 3). 25 est plus grand que 3, on considère donc le sous-arbre droit au noeud01 et on se retrouve au niveau du noeud03 (valeur 15). 25 est plus grand que 15, on considère donc le sous-arbre droit au noeud03 et on se retrouve au niveau du noeud05 (valeur 19). 25 est plus grand que 19, on considère donc le sous-arbre droit du noeud05, ce sous-arbre droit est vide et on insère donc le noeud11 à cet emplacement. Le noeud11 est donc inséré sous le noeud5 en fils droit

Question 2

Préciser toutes les valeurs entières que l’on peut stocker dans le nœud fils gauche du nœud04 (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche ?

Réponse

Question 3

Voici un algorithme récursif permettant de parcourir et d'afficher les valeurs de l'arbre :

🐍 **Script Python**

```
Parcours(A)  # A est un arbre binaire de recherche
    Afficher(A.valeur)
    Parcours(A.fils_gauche)
    Parcours(A.fils_droit)
```

Réponse

Il est possible de stocker toutes les valeurs comprises entre 26 et 29, c'est à dire : 26, 27 et 28 (on peut prendre 26 car il est précisé dans l'énoncé que "les valeurs de tous les nœuds du sous-arbre droit sont supérieures ou égales à la valeur du nœud X"

Question 3

a. Écrire la liste de toutes les valeurs dans l'ordre où elles seront affichées.
b. Choisir le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : Préfixe, Suffixe ou Infixe.

Réponse

a. 26 - 3 - 1 - 15 - 13 - 19 - 25 - 42 - 29 - 32 - 30 - 37
b. C'est un parcours préfixe

Question 4

En vous inspirant de l'algorithme précédent, écrire un algorithme Parcours2 permettant de parcourir et d'afficher les valeurs de l'arbre A dans l'ordre croissant.

Réponse

🐍 **Script Python**

```
Parcours2(A)
    Parcours2(A.fils_gauche)
    Afficher(A.valeur)
    Parcours2(A.fils_droit)
```

2023 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
23-NSIZERO-A	Sujet Zéro-A	x	23-NSIZERO-A	① ② ③
23-NSIZERO-B	Sujet Zéro-B	x	23-NSIZERO-B	① ② ③

1. Sujet Zéro-A - jour x : *23-NSIZERO-A*

1.1. Enoncé

23-NSIZERO-A

- **Exercice 1 [3 points]** : *bases de données et types construits de données*
- **Exercice 2 [3 points]** : *architecture matérielle, réseaux et systèmes d'exploitation*
- **Exercice 3 [6 points]** : *arbres binaires de recherche et programmation objet*

1.2. Correction

①

②

③

2. Sujet Zéro-B - jour x : *23-NSIZERO-B*

2.1. Enoncé

23-NSIZERO-B

- **Exercice 1 [4 points]** : *ligne de commande sous Linux, traitement de données en tables et bases de données*
- **Exercice 2 [4 points]** : *analyse et écriture de programmes récursifs*
- **Exercice 3 [4 points]** : *arbres binaires de recherche et notion d'objet*

2.2. Correction

①

②

③

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-NSIJ1AN1	Amérique du nord	1	22-NSIJ1AN1	① ② ③ ④ ⑤
22-NSIJ1AS1	Amérique du sud	1	22-NSIJ1AS1	① ② ③ ④ ⑤
22-NSIJ1G11	Etranger	1	22-NSIJ1G11	① ② ③ ④ ⑤
22-NSIJ1JA1	Asie-Pacifique	1	22-NSIJ1JA1	① ② ③ ④ ⑤
22-NSIJ1LR1	Mayotte et réseau AEFE	1	22-NSIJ1LR1	① ② ③ ④ ⑤

Repère	Centre	Jour	Téléchargement	Correction
22-NSIJ1ME1	Métropole	1	22-NSIJ1ME1	① ② ③ ④ ⑤
22-NSIJ1ME3	Métropole session de septembre	1	22-NSIJ1ME3	① ② ③ ④ ⑤
22-NSIJ1PO1	Polynésie	1	22-NSIJ1PO1	① ② ③ ④ ⑤
22-NSIJ2AN1	Amérique du nord	2	22-NSIJ2AN1	① ② ③ ④ ⑤
22-NSIJ2AS1	Amérique du sud	2	22-NSIJ2AS1	① ② ③ ④ ⑤
22-NSIJ2G11	Etranger	2	22-NSIJ2G11	① ② ③ ④ ⑤
22-NSIJ2JA1	Asie-Pacifique	2	22-NSIJ2JA1	① ② ③ ④ ⑤
22-NSIJ2LR1	Mayotte et réseau AEFE	2	22-NSIJ2LR1	① ②

Repère	Centre	Jour	Téléchargement	Correction
				③ ④ ⑤
22-NSIJ2ME1	Métropole	2	22-NSIJ2ME1	① ② ③ ④ ⑤

1. Amérique du nord - jour 1 : *22-NSIJ1AN1*

1.1. Enoncé

22-NSIJ1AN1

- **Exercice 1** : *bases de données relationnelles et langage SQL*
- **Exercice 2** : *réseaux et protocoles de routage*
- **Exercice 3** : *abres 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*

1.2. Correction

①

②

③

④

⑤

2. Amérique du sud - jour 1 : *22-NSIJ1AS1*

2.1. Enoncé

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. Correction

①

②

③

④

⑤

3. Etranger - jour 1 : *22-NSIJ1G11*

3.1. Enoncé

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*

3.2. Correction

①
②
③
④
⑤

4. Asie-Pacifique - jour 1 : *22-NSIJ1JA1*

4.1. Enoncé

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*

4.2. Correction

①
②
③
④
⑤

5. Mayotte et réseau AEFE - jour 1 : *22-NSIJ1LR1*

5.1. Enoncé

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 infixe)*
- **Exercice 5** : *réseau, protocoles de routage, langage et programmation*

5.2. Correction

①
②
③
④
⑤

6. Métropole - jour 1 : *22-NSIJ1ME1*

6.1. Enoncé

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*

6.2. Correction

①

②

③

④

⑤

7. Métropole session de septembre - jour 1 : *22-NSIJ1ME3*

7.1. Enoncé

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*

7.2. Correction

①

②

③

④

⑤

8. Polynésie - jour 1 : *22-NSIJ1PO1*

8.1. Enoncé

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 abres binaires*

8.2. Correction

①

②

③

④

⑤

9. Amérique du nord - jour 2 : *22-NSIJ2AN1*

9.1. Enoncé

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*

9.2. Correction

①

②

③

④

⑤

10. Amérique du sud - jour 2 : *22-NSIJ2AS1*

10.1. Enoncé

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*

10.2. Correction

①

②

③

④

⑤

11. Etranger - jour 2 : *22-NSIJ2G11*

11.1. Enoncé

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)*

11.2. Correction

①

②

③

④

⑤

12. Asie-Pacifique - jour 2 : *22-NSIJ2JA1*

12.1. Enoncé

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*

12.2. Correction

①

②

③

④

⑤

13. Mayotte et réseau AEFE - jour 2 : *22-NSIJ2LR1*

13.1. Enoncé

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*

13.2. Correction

①

②

③

④

⑤

14. Métropole - jour 2 : *22-NSIJ2ME1*

14.1. Enoncé

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*

14.2. Correction

①

②

③

④

⑤

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

Repère	Centre	Jour	Téléchargement	Correction
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. Amérique du nord - jour 1 : *21-NSIJ1AN1*

1.1. Enoncé

21-NSIJ1AN1

- **Exercice 1** : *bases de données relationnelles et langage SQL*
- **Exercice 2** : *routage, processsus 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. Correction

①
②
③
④
⑤

2. Etranger - jour 1 : *21-NSIJ1G11*

2.1. Enoncé

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)*

2.2. Correction

①
②
③
④
⑤

3. Métropole - jour 1 : *21-NSIJ1ME1*

3.1. Enoncé

21-NSIJ1ME1

- **Exercice 1** : *abres 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*

3.2. Correction

①
②
③
④
⑤

4. Métropole candidats libres - jour 1 : *21-NSIJ1ME2*

4.1. Enoncé

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"*

4.2. Correction

①

②

③

④

⑤

5. Métropole session de septembre - jour 1 : *21-NSIJ1ME3*

5.1. Enoncé

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)*

5.2. Correction

①

②

③

④

⑤

6. Etranger - jour 2 : *21-NSIJ2G11*

6.1. Enoncé

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*

6.2. Correction

①
②
③
④
⑤

7. Métropole - jour 2 : *21-NSIJ2ME1*

7.1. Enoncé

21-NSIJ2ME1

- **Exercice 1** : *abres 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*

7.2. Correction

①
②
③
④
⑤

8. Métropole candidats libres - jour 2 : *21-NSIJ2ME2*

8.1. Enoncé

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*

8.2. Correction

①

②

③

④

⑤

9. Métropole session de septembre - jour 2 : *21-NSIJ2ME3*

9.1. Enoncé

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*"

9.2. Correction

①

②

③

④

⑤

10. Polynésie - jour 2 : *21-NSIJ2PO1*

10.1. Enoncé

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*

10.2. Correction

①

②

③

④

⑤

Épreuve pratique

1. Modalités

Textes règlementaires

- 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.

2. Banque d'exercices

Textes règlementaires

- 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.

Epreuve Pratique - Consigne

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. Evaluation du candidat

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 prés. 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 obtenue 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 a minima 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.

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

Attent

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				2022-S08

Numéro	Lien de téléchargement	Thème exercice 1	Thème exercice 2	Code fourni	Correction
		Recherche de la première occurrence	Insertion dans une liste triée	Code	
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 correspodants 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

Numéro	Lien de téléchargement	Thème exercice 1	Thème exercice 2	Code fourni	Correction
19	Sujet N°19	Multiplications avec uniquement additions et soustractions	Recherche dichotomique	Code	2022-S19
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

Numéro	Lien de téléchargement	Thème exercice 1	Thème exercice 2	Code fourni	Correction
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
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			2022-S38

Numéro	Lien de téléchargement	Thème exercice 1	Thème exercice 2	Code fourni	Correction
			Jeu du nombre mystère	Code	
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

Index

1. Devoirs sur l'année

	Devoir	Le
	DS 1- Langage SQL	
	DS 3- Pile-File et Protocole de Routage	16/11/2023
	DS 4- Les tris - Diviser pour régner	02/12/2023
	DS 5- Les arbres binaires	16/12/2023

Devoir 1 : Langage SQL

1. D'après 2021, Métropole , 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 relationcompléter `Livres`.

1. Expliquer pourquoi le code `SQL` ci-dessous provoque une erreur.

🗂️ **Requête SQL**

```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**

```sql
SELECT titre
FROM Livres
WHERE auteur = 'Molière'
```

4. Décrire le résultat renvoyé par la requête ci-dessous.

🗂️ **Requête SQL**

```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**

```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**

```sql
UPDATE Emprunts SET ..................... WHERE ..................... ;
```

Réponse

🗂️ **Requête SQL**

```sql
UPDATE Emprunts
SET dateRetour = '2020-09-30'
WHERE idEmprunt = 640
```

6. Décrire le résultat renvoyé par la requête ci-dessous.

🗂️ **Requête SQL**

```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**

```sql
SELECT nom, prenom
FROM Eleves
JOIN Emprunts ON Eleves.idEleves = Emprunts.idEleves
WHERE Emprunts.isbn = 192
```

Sans l'ISBN :

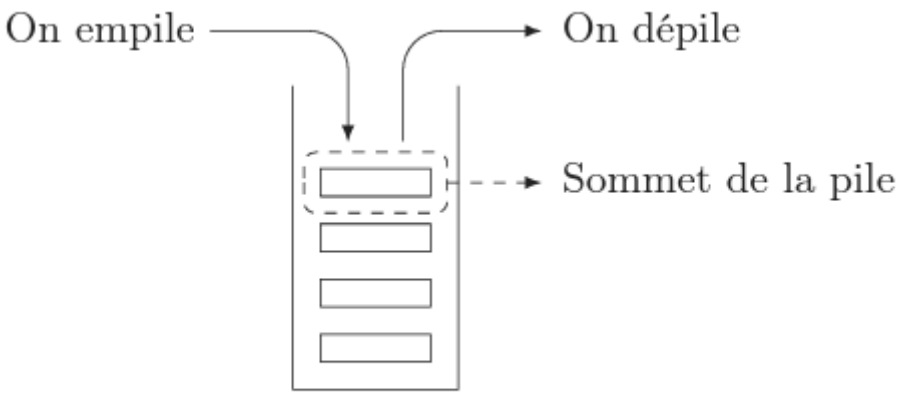
🗂️ **Requête SQL**

```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'
```

Devoir 3 : Pile-File et Protocole de Routage

Pile - File

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe «dernier arrivé, premier sorti» :



Une pile

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` : \(\varnothing\) \(\rightarrow\) Pile
  - `creer_pile_vide()` renvoie une file vide
- `est_vide` : Pile \(\rightarrow\) Booléen
  - `est_vide(pile)` renvoie True si pile est vide, False sinon
- `empiler` : Pile, Élément \(\rightarrow\) Rien
  - `empiler(pile, element)` ajoute element au sommet de la pile
- `depiler` : Pile \(\rightarrow\) Élément
  - `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 élèments étant empilés par le haut) :

Quel serait le contenu de la pile Q aprés l'exécution de la suite d'instruction suivante ?

🐍 **Script Python**

```
Q=creer_pile_vide()
while not est_vide(P):
    empiler(Q, depiler(P))
```

2. 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.

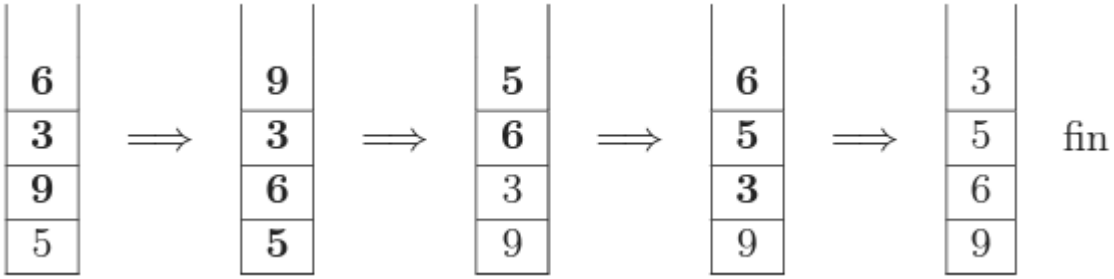
3. Créer une fonction `retourner` ayant pour paramètres une pile P et un entier j. Cette fonction inverse l'ordre des j derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

   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 :

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

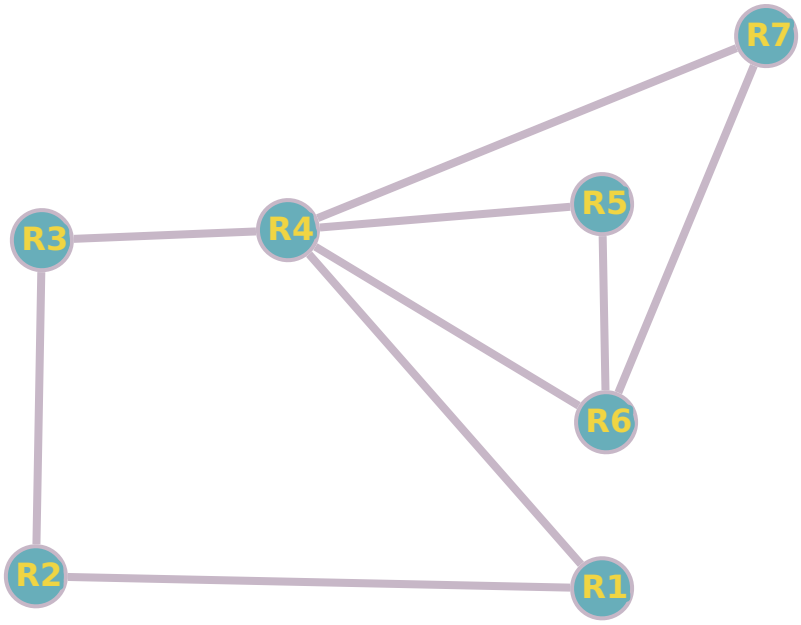
,après l'appel de `tri_crepes(P)`, la pile P devient

5
7
8
12
14

.

Routage

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 Protocole). Le protocole RIP permet de construire les tables de routage des différents

routeurs, en indiquant pour chaque routeur, la distance, en nombr de sauts, qui le sépare d'un autre routeur.

Voir ANNEXE

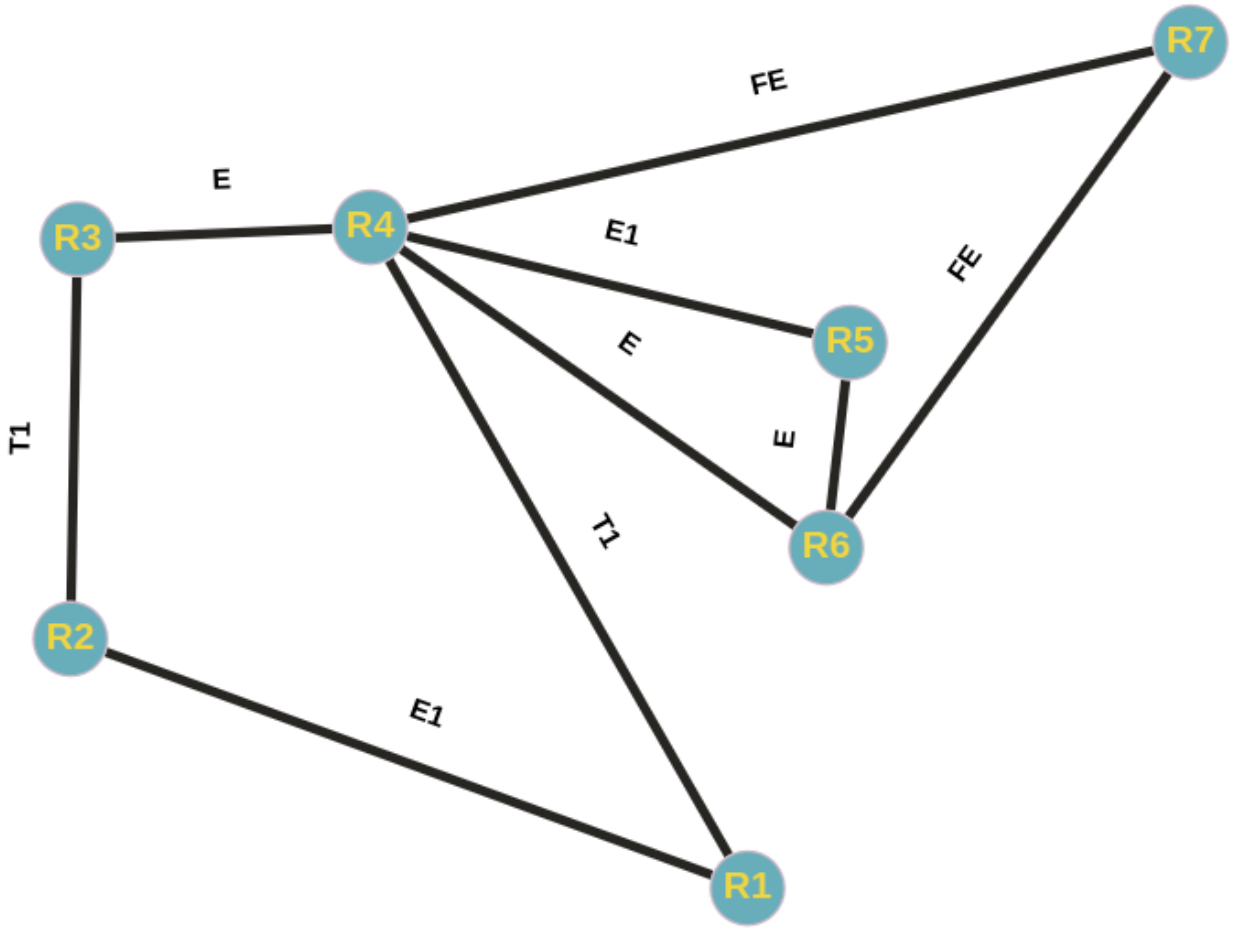
1. Le routeur R2 doit envoyer un paquet de données au routeur R7 qui accuse réception.
   Déterminer le chemin parcouru par le paquet de données ainsi que celui parcouru par l'éaccusé de réception.
2. a. Indiquer la faiblesse que représente ce réseau en cas de panne du routeur R4.
   b. Proposer une 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 pren 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 :
   \(coût=\dfrac{10^8\text{ bit/s}}{\text{ bande passante du lien en bit/s}}\)

   Voici le tableau référençant les coûts des liaisons en fonction du type de liaison entre deux routeurs :

Type de laiison	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 Mb/s = 1 000 kb/s = \(10^6\) bit/s.

   a. Déterminer la bande passante di 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 rencontrés. 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

Devoir 4 : Les tris - Diviser pour régner

1. Q.C.M.

Exo

1. On applique l'algorithme du tri par sélection à la liste `[8,12,6,19]`, après la première étape, le contenu de la liste sera :

a) `[12,8,6,19]`

b) `[6,12,8,19]`

c) `[19,12,6,9]`

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 ?

a) `[11,9,7,16]`

b) `[9,11,16,7]`

c) `[9,7,11,16]`

d) Aucune des propositions ci-dessus

3. L'algorithme du tri par sélection a une complexité :

a) logarithmique

b) linéaire

c) quadratique

d) exponentielle

4. Un programme de tri par insertion prend environ 1/10 seconde pour trier une liste de \(1\,000\) éléments, combien de temps prendra-t-il environ pour trier une liste de \(100\,000\) éléments ?

a) 1 seconde

b) 10 secondes

c) 100 secondes

d) 1000 secondes

2. D'aprés exercice BAC

Exo

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

Expliquer pourquoi le couple (2, 3) n'est pas une inversion dans le tableau [4, 8, 3, 7].

Question2

Enoncé

Justifier que le couple (2, 3) n'est pas une

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

On donne la fonction suivante.

🐍 **Script Python**

```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
```

a. 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.

b. Expliquer ce que permet de déterminer cette fonction.

Question A.2

Enoncé

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

Quelle est l'ordre de grandeur de la complexité en temps de l'algorithme obtenu ? Aucune justification n'est attendue.

2. [illegible]e récursive

Le[illegible]e concevoir une version récursive de la fonction nombre_inversion.
O[illegible]ctions auxiliaires.

Question B.1

Enoncé

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

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

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.

01 Epreuve Pratique

1. Exercice 1

Exo

Enoncé Solution

Écrire une fonction `tri_selection` qui prend en paramètre une liste `tab` de nombres entiers et qui trie ce tableau en place (c'est-à-dire que le tableau est modifié) par ordre croissant.

On utilisera l'algorithme suivant :

- on parcours le tableau de gauche à droite :
  - on recherche le minimum du tableau entre cette position courante et la fin du tableau.
  - on échange alors les 2 valeurs.

Exemple :

🐍 **Script Python**

```
>>> tab = [1, 52, 6, -9, 12]
>>> tri_selection(tab)
>>> tab
[-9, 1, 6, 12, 52]
>>> tab_vide = []
>>> tri_selection(tab_vide)
```

```
>>> tab_vide
[]
```

🐍 **Script Python**

```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
```

2. Exercice 2

Exo

Enoncé Solution

Chaque soir, les auditeurs d'une radio votent en ligne pour leur artiste favori. Ces votes sont stockés dans un tableau.

Exemple :

🐍 **Script Python**

```
Urne = ['Oreilles sales', 'Oreilles sales', 'Oreilles sales', 'Extra Vomit', 'Lady Baba', 'Extra Vomit', 'Lady Baba', 'Extra Vomit', 'Lady Baba', 'Extra Vomit']
```

La fonction depouille doit permettre de compter le nombre de votes exprimés pour chaque artiste. Elle prend en paramètre un tableau et renvoie le résultat dans un dictionnaire dont les clés sont les noms des artistes et les valeurs le nombre de votes en leur faveur.

La fonction vainqueur doit désigner le nom du ou des gagnants. Elle prend en paramètre un dictionnaire dont la structure est celle du dictionnaire renvoyé par la fonction depouille et renvoie un tableau. Ce tableau peut donc

contenir plusieurs éléments s’il y a des artistes ex- aequo.

Compléter les fonctions depouille et vainqueur ci-après pour qu’elles renvoient les résultats attendus.

🐍 **Script Python**

```python
urne = ['Oreilles sales', 'Oreilles sales', 'Oreilles sales',
    'Extra Vomit', 'Lady Baba', 'Extra Vomit', 'Lady Baba',
    'Extra Vomit', 'Lady Baba', 'Extra Vomit']

def depouille(urne):
    resultat = ...
    for bulletin in urne:
        if ...:
            resultat[bulletin] = ...
        else:
            ...
    return resultat

def vainqueur(election):
    nmax = 0
    for candidat in election:
        if ... > ... :
            nmax = ...
    liste_finale = [nom for nom in election if election[nom] == ...]
    return ...
```

```
def
depouille(urne):
    resultat = {}
    for bulletin in
urne:
        if bulletin in
resultat:

resultat[bulletin]
=
resultat[bulletin]
+ 1
        else:

resultat[bulletin]
= 1
    return resultat

def
vainqueur(election):
    nmax = 0
    for candidat in
election:
        if
election[candidat]
> nmax:
            nmax =
election[candidat]
    liste_finale =
[nom for nom in
election if
election[nom] ==
nmax]
    return
liste_finale
```

02 Epreuve Pratique

1. Exercice 1

Exo

Enoncé Solution 1 Solution 2 : avec la méthode `items`

Sur le réseau social TipTop, on s'intéresse au nombre de « like » des abonnés. Les données sont stockées dans des dictionnaires où les clés sont les pseudos et les valeurs correspondantes sont les nombres de « like » comme ci-dessous :

🐍 **Script Python**

```
{'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}
```

Écrire une fonction `top_like` qui :

- Prend en paramètre un dictionnaire likes non vide dont les clés sont des chaînes de caractères et les valeurs associées sont des entiers ;
- Renvoie un tuple dont :
  - La première valeur est la clé du dictionnaire associée à la valeur maximale ; en cas d'égalité sur plusieurs

clés, on choisira la plus petite suivant l'ordre alphabétique

- La seconde valeur est la valeur maximale présente dans le dictionnaire.

Exemples :

🐍 **Script Python**

```
>>> top_like({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50})
('Ada', 201)
>>> top_like({'Alan': 222, 'Ada': 201, 'Eve': 222, 'Tim': 50})
('Alan', 222)
```

🐍 **Script Python**

```
def top_like(likes):
    top_pseudo = None
    top_nb_likes = 0
    for pseudo in likes:
        nb_likes = likes[pseudo]
        if nb_likes > top_nb_likes or nb_likes == top_nb_likes and pseudo < top_pseudo:
            top_pseudo = pseudo
            top_nb_likes = nb_likes
    return top_pseudo, top_nb_likes
```

🐍 **Script Python**

```
def top_like(likes):
    top_pseudo = None
    top_nb_likes = 0
    for pseudo, nb_likes in likes.items():
        if nb_likes > top_nb_likes or nb_likes == top_nb_likes and pseudo < top_pseudo:
            top_pseudo = pseudo
            top_nb_likes = nb_likes
    return top_pseudo, top_nb_likes
```

2. Exercice 2

Exo

Enoncé **Solution**

Recopier et compléter sous Python la fonction suivante en respectant la spécification. On ne recopiera pas les commentaires.

```
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    debut = ...
    fin = ...
    while debut <= fin:

        milieu = ...
        if x == tab[milieu]:
```

```
return ...
        if x > ...:
            debut
= ...
        else:
            fin = ...
    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
```

```
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        milieu = (debut + fin) // 2
        if x == tab[milieu]:
            return True
        if x > tab[milieu]:
            debut = milieu + 1
        else:
            fin = milieu - 1
    return False
```

Eval EP3

03 Epreuve Pratique

1. Exercice 1

Exo

L'occurrence d'un caractère dans un phrase est le nombre de fois où ce caractère est présent.

Exemples :

- l'occurrence du caractère 'o' dans 'bonjour' est 2 ;
- l'occurrence du caractère 'b' dans 'Bébé' est 1 ;
- l'occurrence du caractère 'B' dans 'Bébé' est 1 ;
- l'occurrence du caractère ' ' dans 'Hello world !' est 2.

On cherche les occurrences des caractères dans une phrase. On souhaite stocker ces occurrences dans un dic
les caractères de la phrase et les valeurs l'occurrence de ces caractères.

Par exemple : avec la phrase 'Hello world !' le dictionnaire est le suivant :

`{'H': 1,'e': 1,'l': 3,'o': 2,' ': 2,'w': 1,'r': 1,'d': 1,'!': 1}`

Écrire une fonction `occurence_lettres` prenant comme paramètre une variable `phrase` de type `str`. Cette fonctio
de type constitué des occurrences des caractères présents dans la phrase.

Exo

La fonction `fusion` prend deux listes `L1`, `L2` d'entiers triées par ordre croissant et les fusionne en une liste tri

Le code Python de la fonction est

```python
def fusion(L1,L2):
    n1 = len(L1)
    n2 = len(L2)
    L12 = [0]*(n1+n2)
    i1 = 0
    i2 = 0
    i = 0
    while i1 < n1 and ... :
        if L1[i1] < L2[i2]:
            L12[i] = ...
            i1 = ...
        else:
            L12[i] = L2[i2]
            i2 = ...
        i += 1
    while i1 < n1:
        L12[i] = ...
        i1 = i1 + 1
        i = ...
    while i2 < n2:
        L12[i] = ...
        i2 = i2 + 1
        i = ...
    return L12
```

Devoir 5 : Les arbres binaires

1. Question de cours

Exo

Question de cours

On note h la hauteur d'un arbre binaire et n sa taille.
En utilisant vos connaissances de cours, prouver que $h \leqslant n \leqslant 2^h -1$.

Réponse

On étudie les deux cas de figure "extrème" :

- **Cas minimaliste** : arbre filiforme

```
graph TD
    A(" ") --> B(" ")
    B --> C(" ")
    B --> D(" ")
    C --> E(" ")
    C --> C1(" ")
    E --> E1(" ")
    E --> E2(" ")
    C1 --> C2(" ")
    C1 --> C3(" ")
    A --> F(" ")
    F --> J(" ")
    F --> K(" ")
    J --> L(" ")
    J --> N(" ")
    K --> M(" ")
    style F opacity:0;
    style D opacity:0;
    style C1 opacity:0;
    style C2 opacity:0;
    style C3 opacity:0;
    style E2 opacity:0;
    style J opacity:0;
    style K opacity:0;
    style L opacity:0;
    style N opacity:0;
    style M opacity:0;
    linkStyle 2 stroke-width:0px;
    linkStyle 4 stroke-width:0px;
    linkStyle 6 stroke-width:0px;
    linkStyle 7 stroke-width:0px;
    linkStyle 8 stroke-width:0px;
    linkStyle 9 stroke-width:0px;
    linkStyle 10 stroke-width:0px;
    linkStyle 11 stroke-width:0px;
    linkStyle 12 stroke-width:0px;
    linkStyle 13 stroke-width:0px;
    linkStyle 14 stroke-width:0px;
```

On a alors $h = n$, on en déduit l'inégalité : $h \leqslant n$

- **Cas maximaliste** : arbre complet

```
graph TD
    A(" ") --> B(" ")
```

```
B --> C(" ")
B --> D(" ")
C --> E(" ")
C --> C1(" ")
E --> E1(" ")
E --> E2(" ")
C1 --> C2(" ")
C1 --> C3(" ")
A --> F(" ")
F --> J(" ")
F --> K(" ")
J --> L(" ")
J --> N(" ")
D --> D1(" ")
D --> D2(" ")
D1 --> D3(" ")
D1 --> D4(" ")
D2 --> D5(" ")
D2 --> D6(" ")
L --> L3(" ")
L --> L4(" ")
N --> N1(" ")
N --> N2(" ")
K --> K1(" ")
K --> K2(" ")
K1 --> K3(" ")
K1 --> K4(" ")
K2 --> K5(" ")
K2 --> K6(" ")
```

Chaque niveau a 2^{h-1} noeuds donc le nombre total de noeuds est donnée par $1 + 2 + 2^2 + 2^3 + ... + 2^{h-1} = 1 \times \dfrac{1-2^h}{1-2}=2^h - 1$.

- On obtient alors les inégalités demandées : $h \leqslant n \leqslant 2^h -1$

2. D'aprés exercice BAC

Exo

Cet exercice porte sur le thème "Algorithmique", les arbes bianires de recherche et leurs parcours

Rappel : Un arbre binaire de recherche (ABR) est un arbre binaire étiqueté avec des clés tel que :

- Les clés du sous-arbre gauche sont inférieures ou égales à celle de la racine;
- Les clés du sous-arbre droit sont strictement supérieures à celle de la racine;
- Les deux sous-arbres sont eux-mêmes des arbres binaires de recherche.

2.1. Partie A : Préambule

Question A.1

Recopier sur votre copie le ou les numéro(s) correspondant aux arbres binaires de recherche parmi les arbres suivants :

Arbre 1

```
graph TD
    A(3) --> B(2)
    B --> D(1)
    B --> E(3)
    A --> C(4)
    C --> F(4)
    C --> G(5)
```

Arbre 2

```
graph TD
    A(4) --> B(2)
    B --> D(1)
    B --> E(3)
    A --> C(4)
    C --> F(4)
    C --> G(5)
```

Arbre 3

```
graph TD
    A(3) --> B(2)
    B --> D(1)
    B --> E(1)
    A --> C(4)
    C --> F(3)
    C --> G(5)
```

Réponse

Seul l'arbre 1 est un arbre binaire de recherche. En effet, dans l'arbre 2 la clé 4 apparaît dans le sous arbre droit de la racine de clé 4, or les clé du sous arbre droit doivent être *strictement* supérieurs à la racine. Pour la même raison, l'arbre 3 n'est pas non plus un arbre binaire de recherche, la valeur 1 figure dans le sous arbre droit du noeud d'étiquette 2.

2.2. Partie B : Analyse

On considère la structure de données abstraites ABR (Arbre Binaire de Recherche) que l'on munit des opérations suivantes :

Strucutres de données ABR

Utilise : Booleen, Element
Opérations :
- `creer_arbre()` : renvoie un arbre vide
- `est_vide(a)` : renvoie `True` si l'arbre a est vide et `False` sinon
- `racine(a)` : renvoie la clé de la racine de l'arbre non vide a.
- `sous_arbre_gauche(a)` : renvoie le sous-arbre gauche de l'arbre non vide a.
- `sous_arbre_droit(a)` : renvoie le sous-arbre droit de l'arbre non vie a.
- `ìnserer(a,e)` insère la clé e dans l'arbre a.

Question B.2.a

Dans un ABR, où se trouve le plus petit élément ? Justifier

Réponse

Le plus petit élément se trouve sur la feuille la plus à gauche de l'arbre. En effet en descendant récursivement à gauche, on trouve à chaque étape une valeur inférieure à toutes celles se trouvant à droite.

Attention

La question demande **le** plus petit élément, or les valeurs présentes dans un arbre binaire de recherche ne sont pas forcément uniques. A titre d'exemple, dans l'arbre binaire de recherche suivant, la plus petite valeur apparaît 2 fois :

```
graph TD
S5(5) --> S6(5)
S5 --> S8(8)
```

Pour rechercher une clé dans un ABR, il faut comparer la clé donnée avec la clé située à la racine. Si cette clé est ç la racine, la fonction renvoie vrai sinon il faut procéder récursivement sur les sous-arbres à gauche ou à droite.

Question B.2.b

En utilisant les fonctions ci-dessus, écrire une fonction récursive `RechercheValeur` prenant en arguments la clé recherchée et l'arbre ABR considéré. Cette fonction retourne un booléen (vrai ou faux) indiquant si la clé est présente dans l'arbre ou non..

Réponse

Si l'arbre est vide alors il ne contient pas la cle cherchée. Sinon trois cas se présente : la clé cherchée est la racine et on renvoie `True`, la clé cherchée est inférieure à la racine et on relance la recherche dans le sous arbre gauche (là ou se trouve les valeurs inférieures à la racine), la clé cherchée est supérieure à la racine et on relance la recherche dans le sous arbre droit.

🐍 **Script Python**

```python
def RechercheValeur(cle,abr):
    if est_vide(abr):
        return False
    if racine(abr)==cle:
        return True
    elif racine(abr)>cle:
        return RechercheValeur(cle,sous_arbre_gauche(a))
    else:
        return RechercheValeur(cle,sous_arbre_droit(a))
```

Question B.3

On considère l'arbre ABR suivant :

```
graph TD
    A(7) --> B(2)
    B --> D(1)
    D --> D1(" ")
    D --> D2(" ")
    B --> E(5)
    E --> H(3)
    E --> I(6)
    A --> C(10)
    C --> F(8)
    C --> G(" ")
    F --> F1(" ")
    F --> F2(9)
    G --> G1(" ")
    G --> G2(" ")
    linkStyle 2 stroke-width:0px;
    style D1 opacity:0;
    linkStyle 3 stroke-width:0px;
    style D2 opacity:0;
    linkStyle 9 stroke-width:0px;
    style G opacity:0;
    linkStyle 10 stroke-width:0px;
    style F1 opacity:0;
    linkStyle 12 stroke-width:0px;
    style G1 opacity:0;
    linkStyle 13 stroke-width:0px;
    style G2 opacity:0;
```

a. Donner le parcours infixe de cette arbre.

b. Donner le parcours suffixe de cette arbre.

c. Donner le parcours préfixe de cette arbre.

d. Donner le parcours en largeur d'abord de cette arbre.

Réponse

a. En parcours en profondeur infixe, on liste de façon récursive les clé du sous arbre gauche, puis la racine, puis les clé du sous arbre droit. On obtient donc : \(1 - 2 - 3 - 5 - 6 - 7 - 8 - 9 - 10\)

b. En parcours préfixe, on liste récursivement la racine puis les noeuds du sous arbre gauche puis ceux du sous arbre droit. On obtient donc : \(7 - 2 - 1 - 5 - 3 - 6 - 10 - 8 - 9\).

c. En parcours suffixe, on liste récursivement les noeuds du sous arbre gauche, ceux du sous arbre droit et enfin la racine. Ici, on obtient : \(1 - 3 - 6 - 5 - 2 - 9 - 8 - 10 - 7\).

d. Le parcours en largeur corresponda à l'ordre de la lecture (de gauche à droite et de haut en bas). Ici on obtient : \(7 - 2 - 10 - 1 - 5 - 8 - 3 - 5 - 9\).

3. D'aprés BAC

Club d'Informatique

D'après 2022, Nouvelle-Calédonie, J1, Ex. 4

Un club de passionné·e·s d'informatique fonctionne de la façon suivante : pour être membre du club, à l'exception du fondateur ou de la fondatrice, il faut être parrainé·e. De plus, chaque membre peut parrainer au maximum deux personnes.

Dans ce club, on distingue trois profils de membres :

- membre or : le membre a parrainé deux personnes ;
- membre argent : le membre a parrainé une seule personne ;
- membre bronze : le membre n'a parrainé personne.

On peut modéliser ce fonctionnement de parrainage à l'aide d'un arbre binaire dont les étiquettes sont les pseudonymes des membres du club. Lorsque deux personnes ont été parrainées, celle qui a été parrainée en premier apparait comme racine du sous-arbre à gauche tandis que l'autre est racine du sous-arbre à droite.

On donne ci-dessous l'arbre \(P\) représentant les membres du club issus des parrainages de AnneB, fondatrice du club. Par exemple, Sophia a parrainé Malik2 avant AstridM.

```
graph TD
   A("AnneB") --> B("Pedro")
   B --> D("FredB")
   B --> E(" ")
   A --> H("Sophia")
```

```
H --> I("Malik2")
H --> J("AstridM")
I --> K("Marc")
I --> L(" ")
J --> N("KevinH")
J --> M("Nico")
linkStyle 2 stroke-width:0px;
style E opacity:0;
linkStyle 7 stroke-width:0px;
style L opacity:0;
```

On munit la structure de données `ArbreBinaire` des opérations suivantes :

Interface de la structure `ArbreBinaire`

🐍 **Script Python**

```python
def est_vide(arbre: ArbreBinaire) -> bool:
    """renvoie True si arbre est vide, False sinon"""

def racine(arbre: ArbreBinaire) -> str:
    """renvoie l'étiquette du nœud racine de arbre"""

def gauche(arbre: ArbreBinaire) -> ArbreBinaire:
    """renvoie le sous-arbre à gauche de arbre"""

def droite(arbre: ArbreBinaire) -> ArbreBinaire:
    """renvoie le sous-arbre à droite de arbre"""
```

Question 1

On appelle feuille, un nœud qui ne possède pas de successeurs ou dit autrement dont l'arbre dont il est la racine possède deux sous-arbres vides. On définit la hauteur d'un arbre binaire non vide comme la longueur (en nombre de nœuds) du plus long chemin allant de la racine à une feuille. Un arbre vide a une hauteur égale à 0.

a) **Indiquer** la hauteur de l'arbre P
b) **Recopier** et **compléter** la définition de la fonction récursive `hauteur` qui prend un `arbre` binaire en paramètre et renvoie la hauteur de cet `arbre`. On pourra utiliser la fonction `max` renvoyant la valeur maximale entre deux valeurs.

```
def hauteur(arbre):
    if ...... :
        return 0
    else:
        hauteur_a_gauche = hauteur(gauche(arbre))
        hauteur_a_droite = ......
        return 1 + ......
```

c) **Indiquer** le type de la valeur renvoyée par la fonction `hauteur`

Réponse a

L'arbre binaire P a une hauteur de 4 (par exemple en considérant le chemin qui va de la racine `'AnneB'` jusqu'à la feuille `'Marc'`, on dénombre bien 4 nœuds, aucun autre chemin en dénombre plus).

Réponse b

```
def hauteur(arbre):
    if est_vide(arbre) :
        return 0
    else:
        hauteur_a_gauche = hauteur(gauche(arbre))
        hauteur_a_droite = hauteur(droite(arbre))
        return 1 + max(hauteur_a_gauche, hauteur_a_droite)
```

Réponse c

Il s'agit d'un entier (type `int`)

Question 2

La fonction `membres` ci-dessous prend un `arbre` binaire et une `liste_membres` en paramètres et ajoute, dans un certain ordre, les étiquettes de l'`arbre` à la `liste_membres`.

```
def membres(arbre, liste_membres):
    if not est_vide(arbre):
        liste_membres.append(racine(arbre))
        membres(gauche(arbre), liste_membres)
        membres(droite(arbre), liste_membres)
```

a) En supposant la liste `membres_p` initialement vide, **écrire** la valeur de cette liste après l'appel `membres(arbre_p, membres_p)` où `arbre_p` référence l'arbre \(P\).

b) **Indiquer** le nom du type de parcours d'arbre binaire réalisé par la fonction `membres`.

Réponse a

🐍 **Console Python**

```
>>> membres_p
['AnneB', 'Pedro', 'FredB', 'Sophia', 'Malik2', 'Marc', 'AstridM', 'KevinH', 'Nico']
```

Réponse b

Il s'agit d'un parcours **préfixe** (on traite la racine **avant** de traiter les sous-arbres à gauche et à droite).

Question 3

Dans cette question, on s'intéresse aux profils des membres (or, argent ou bronze).

a) **Indiquer** les étiquettes des feuilles de l'arbre \(P\).
b) À partir des propositions suivantes, **indiquer** le profil des membres dont les pseudonymes sont les étiquettes des feuilles.

- Réponse A : membre or
- Réponse B : membre argent
- Réponse C : membre bronze
- Réponse D : on ne peut pas savoir

c) **Écrire** la fonction `profil` qui prend un `arbre` binaire non vide en paramètre et renvoie le profil du membre dont le pseudonyme est l'étiquette de la racine de l'`arbre` sous la forme d'une chaine de caractères : `'or'`, `'argent'` ou `'bronze'`. Par exemple, l'appel `profil(arbre_p)` doit renvoyer `'or'` qui correspond au profil du membre `'AnneB'`, racine de \(P\).

Réponse a

Les feuilles sont : `'FredB'`, `'Marc'`, `'KevinH'` et `'Nico'`

Réponse b

Puisqu'il s'agit des feuilles, cela signifie que ces nœuds n'ont pas de successeurs et donc que les membres associés n'ont parrainé personne. La bonne réponse est donc la C : membre bronze.

Réponse c

```
def profile(arbre):
    if est_vide(gauche(arbre)) and est_vide(droite(arbre)):
        return 'bronze'
    elif est_vide(gauche(arbre)) or est_vide(droite(arbre)):
        return 'argent'
    else:
        return 'bronze'
```

Autre solution, consistant à compter le nombre de sous-arbres non vides qui correspond au nombre de personnes parrainées. On en déduit le profil :

```
def profile(arbre):
    les_profiles = ('bronze', 'argent', 'or')
    nb_parraines = 0
    if not est_vide(gauche(arbre)):
        nb_parraines += 1
    if not est_vide(droite(arbre)):
        nb_parraines += 1
    return les_profils[nb_parraines]
```

Question 4

Afin d'obtenir un tableau dont chaque élément est un *tuple* contenant le pseudonyme d'un membre et son profil, on propose la fonction `membres_profils` définie ci-dessous :

```
def membres_profils(arbre, liste_membres_profils):
    if not est_vide(arbre):
        liste_membres_profils.append((racine(arbre), profil(A))
        membres_profils(gauche(arbre), liste_membres_profils)
        membres_profils(droite(arbre), liste_membres_profils)
```

On appelle cette fonction sur un arbre `arbre_2` et on obtient ceci :

🐍 **Console Python**

```
>>> liste_2 = []
>>> membres_profils(arbre_2, liste_2)
>>> liste_2
[('LeaC', 'or'), ('Ali', 'bronze'), ('Tom45', 'argent'), ('Vero', 'bronze')]
```

Dessiner un arbre possible pouvant correspondre à l' `arbre_2` .

Réponse

Comme pour la question 2b, on a à faire à un parcours préfixe donc `'LeaC'` est la racine de l'`arbre_2`. Ensuite à gauche on a `'Ali'` qui est une feuille (membre bronze). À droite on tombe sur `'Tom45'` qui a parrainé un seul membre : `'Vero'`.

```
graph TD
    A("LeaC") --> B("Ali")
    B --> D(" ")
    B --> E(" ")
    A --> H("Tom45")
    H --> I("Vero")
    H --> J(" ")
    linkStyle 2 stroke-width:0px;
    style E opacity:0;
    linkStyle 1 stroke-width:0px;
    style D opacity:0;
    linkStyle 5 stroke-width:0px;
    style J opacity:0;
```

Question 5

Chaque année, les membres versent une cotisation en fonction de leur profil.

- membre or : cotisation de 20€
- membre argent : cotisation de 30€
- membre bronze : cotisation de 40€

Écrire une fonction `cotisation` qui prend un `arbre` binaire et renvoie le total des cotisations reçues par le club dont `arbre` modélise les relations de parrainage. On pourra utiliser la fonction `membres_profils` de la question précédente.

Réponse

On commence par définir un dictionnaire des tarifs de cotisations :

🐍 **Script Python**

```python
tarifs = {'or': 20, 'argent': 30, 'bronze': 40}
```

```python
def cotisations(arbre):
    total = 0
    liste_membres_profils = []
    membres_profils(arbre, liste_membres_profils)
    for _, profil in liste_membres_profils:
        total += tarifs[profil]
    return total
```

On peut aussi faire sans la fonction `membres_profils` :

```python
def cotisations(arbre):
    if est_vide(arbre):
        return 0
    else:
        a_gauche = cotisations(gauche(arbre))
        a_droite = cotisations(droite(arbre))
        return tarifs[profil(a)] + a_gauche + a_droite
```

4. D'aprés BAC

Insertion et valeurs supérieures dans un ABR

D'après 2022, Métropole, J2, Ex. 1

Dans cet exercice, la taille d'un arbre est le nombre de nœuds qu'il contient. Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles (nœuds sans sous-arbres). On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et la hauteur de l'arbre vide vaut 0.

On considère l'arbre binaire représenté ci-dessous :

```
graph
N0((15)) --- N1((13))
N0 --- N2((21))
N1 --- N3((11))
N1 --- N4((14))
N2 --- N5((18))
```

```
N2 --- N6((27))
N5 --- N7(( ))
linkStyle 6 stroke-width:0px;
style N7 opacity:0;
N5 --- N8((20))
```

Question 1

1.a. Donner la taille de cet arbre.
1.b. Donner la hauteur de cet arbre.
1.c. Représenter sur la copie le sous-arbre à droite du nœud de valeur 15.
1.d. Justifier que l'arbre de la figure 1 est un arbre binaire de recherche.
1.e. On insère la valeur 17 dans l'arbre de la figure 1 de telle sorte que 17 soit une nouvelle feuille de l'arbre et que le nouvel arbre obtenu soit encore un arbre binaire de recherche. Représenter sur la copie ce nouvel arbre.

Réponse 1.a

La taille vaut 8.

Réponse 1.b

La hauteur vaut 4.

Réponse 1.c

```
graph
N2((21)) --- N5((18))
N2 --- N6((27))
N5 --- N7(( ))
linkStyle 2 stroke-width:0px;
style N7 opacity:0;
N5 --- N8((20))
```

Réponse 1.d

Un arbre binaire de recherche est un arbre tel que, pour tout nœud \(n\), les valeurs associées à tous les nœuds du sous-arbre à gauche sont inférieures à la valeur de \(n\), et les valeurs associées à tous les nœuds du sous-arbre à droite sont supérieures à la valeur associée à \(n\).

On vérifie donc :

- 15 > 13
- 13 > 11
- 13 < 14
- 15 < 21
- 21 > 18
- 18 < 20
- 21 < 27

Cette propriété étant récursive, on doit aussi vérifier que l'on a :

- 15 > 14
- 15 < 18
- 15 < 17
- 21 > 20

Remarque : Il est inutile de vérifier que l'on a 15 > 11 car on a vérifié 15 > 13 et 13 > 11. De la même façon, la vérification 15 < 27 est inutile.

Réponse 1.e

```
graph
N0((15)) --- N1((13))
N0 --- N2((21))
N1 --- N3((11))
N1 --- N4((14))
N2 --- N5((18))
N2 --- N6((27))
N5 --- N7((17))
N5 --- N8((20))
```

On considère l'arbre la classe `Noeud` définie de la façon suivante en Python :

```python
class Noeud:
    def __init__(self, gauche, valeur, droite):
        self.gauche = gauche
        self.valeur = valeur
        self.droite = droite
```

Question 2

2.a. Parmi les trois instructions **(A)**, **(B)** et **(C)** suivantes, écrire sur la copie la lettre correspondant à celle qui construit et stocke dans la variable `abr` l'arbre représenté ci-dessous.

```
graph
N0((15)) --- N1((13))
N0 --- N2((21))
```

- **(A)** `abr = Noeud(Noeud(Noeud(None, 13, None), 15, None), 21, None)`
- **(B)** `abr = Noeud(None, 13, Noeud(Noeud(None, 15, None), 21, None))`
- **(C)** `abr = Noeud(Noeud(None, 13, None), 15, Noeud(None, 21, None))`

2.b. Recopier et compléter la ligne 7 du code de la fonction `inserer` ci-dessous qui prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et qui renvoie l'arbre obtenu suite à l'insertion de la valeur v dans l'arbre `abr`. Les lignes 8 et 9 permettent de ne pas insérer la valeur `v` si celle-ci est déjà présente dans `abr`.

```python
1  def inserer(v, abr):
2      if abr is None:
3          return Noeud(None, v, None)
4      if v > abr.valeur:
5          return Noeud(abr.gauche, abr.valeur, inserer(v, abr.droite))
6      elif v < abr.valeur:
7          return ...
8      else:
9          return abr
```

Réponse 2.a

On utilise l'instruction **(C)** :

🐍 **Script Python**

```
abr = Noeud(
        Noeud(None, 13, None),
        15,
        Noeud(None, 21, None)
)
```

Réponse 2.b

On propose :

```
1   def inserer(v, abr):
2       if abr is None:
3           return Noeud(None, v, None)
4       if v > abr.valeur:
5           return Noeud(abr.gauche, abr.valeur, ins(v, abr.droite))
6       elif v < abr.valeur:
7           return Noeud(ins(v, abr.gauche), abr.valeur, abr.droite)
8       else:
9           return abr
```

3. La fonction `nb_sup` prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et renvoie le nombre de valeurs supérieures ou égales à la valeur `v` dans l'arbre `abr`.

Le code de cette fonction `nb_sup` est donné ci-dessous :

```
1   def nb_sup(v, abr):
2       if abr is None:
3           return 0
4       elif abr.valeur >= v:
5           return 1+nb_sup(v, abr.gauche)+nb_sup(v, abr.droite)
6       else:
7           return nb_sup(v, abr.gauche)+nb_sup(v, abr.droite)
```

Question 3

3.a. On exécute l'instruction `nb_sup(16, abr)` dans laquelle `abr` est l'arbre initial de la question **1.**. Déterminer le nombre d'appels à la fonction `nb_sup`.
3.b. L'arbre passé en paramètre étant un arbre binaire de recherche, on peut améliorer la fonction `nb_sup` précédente afin de réduire ce nombre d'appels.
Écrire sur la copie le code modifié de cette fonction.

Réponse 3.a

La fonction va faire autant d'appel que l'arbre compte de sous-arbres **y compris les sous-arbres vides**. En comptant l'appel initial cela fait donc 17 appels.

Réponse 3.b

Dans la mesure où l'on souhaite compter les valeurs supérieures, il est inutile d'explorer le sous-arbre à gauche si la valeur d'un nœud est inférieure à la valeur passée en argument.

On peut donc faire :

🐍 **Script Python**

```python
def nb_sup(v, abr):
    if abr is None:
        return 0
    elif abr.valeur >= v:
        return nb_sup(v, abr.gauche) + 1 + nb_sup(v, abr.droite)
    else:
        return nb_sup(v, abr.droite)
```

Projet

Projet	Pygame : Initiation

1. Preambule

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`.

2. Apparition d'un personnage

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.

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.

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'occurence : 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()
```

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.

3.1. Comment les capturer ?

On utilise le module event de Pygame.

Voici ce que nous dit la documentation à propos de ce module :

Afficher/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 évènements des périphériques gérés par le système d'exploitation.
- Nous allons nous intéresser aux évènements de type `KEYDOWN` (touche de clavier appuyée) ou de type `MOUSEBUTTONDOWN` (boutons de souris appuyé).

3.2. Évènements clavier

3.2.1. 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**

```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**

```python
from pygame.locals import *
```

que la variable `K_RIGHT` (et toutes les autres) est reconnue.

3.2.2. 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**

```python
pygame.key.set_repeat(50)
```

3.3. Évènements souris

3.3.1. 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")
```

3.3.2. Récupération des coordonnées de la souris

Le tuple `(abscisse, ordonnée)` des coordonnées de la souris sera récupéré avec l'instruction `pygame.mouse.get_pos()` .

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.

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.

Exo

Coder un script pour déplacer Paragoomba à la souris (Paragoomba doit toujours suivre la souris) (`MOUSEMOTION`)

Correction

🐍 **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()
```

Exo

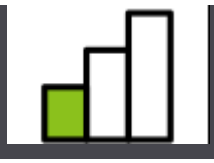
Coder un script pour dessiner un rectangle sur l’écran au relâchement d’un bouton de la souris.

Correction

🐍 **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.MOUSEBUTTONUP:
      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()
```

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...

4.1. Les packages utilisés

On commence par introduire les packages (bibliothèques) qui seront utilisées :

🐍 **Script Python**

```python
import pygame
from pygame.locals import *
from constantes import *
```

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**

```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)
```

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**

```python
import pygame
from pygame.locals import *
from constantes import *









# Gestion du rafraichissement 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:
  # recupère la liste des évènements du joueur
  event = pygame.event.Event(pygame.USEREVENT)

  # dessine le mur de droite

  # EVENEMENTS
  # détecte le clic sur le bouton close de la fenêtrepygame.Rect
  for event in pygame.event.get():
      if event.type == pygame.QUIT:
        Termine = True

  # récupère la liste des touches claviers appuyeées 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()
```

4.4. Le jeu Pong en lui-même

A vous

Faites votre propre jeu avec une deuxième raquette, meilleur gestion des rebonds, changement de vitesses....

5. SNAKE en Python, le plus simplement possible

5.1. Version 0

5.1.1. Pygame

On importe pygame avec :

🐍 **Script Python**

```python
import pygame
from pygame.locals import *
```

Le second import sert à quitter le jeu propremement.

5.1.2. Constantes

On crée quelques constantes :

🐍 **Script Python**

```python
HAUTEUR = 600  # hauteur de la fenetre
LARGEUR = 600  # largeur de la fenetre
BLOC = 20

# Les couleurs utilisées
NOIR = (..., ..., ...)  # fond
ROUGE = (..., ..., ...) # pomme
JAUNE = (..., ..., ...)  # tête
VERT = (..., ..., ...)  # corps
CYAN = (..., ..., ...)  # texte

FPS = 30
```

5.1.3. initialisation

On initialise le jeu :

🐍 **Script Python**

```python
pygame.init()
horloge = pygame.time.Clock()
fenetre = pygame.display.set_mode((LARGEUR, HAUTEUR))
pygame.display.set_caption('Snake')

pygame.display.update()
```

5.1.4. 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

5.1.5. Boucle infinie

🐍 **Script Python**

```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()
```

5.1.6. 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

5.2. Version 1

5.2.1. 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)
```

5.2.2. Taille de la police, valeur du score

🐍 **Script Python**

```
font = pygame.font.SysFont(None, 48)
```

et

🐍 **Script Python**

```
score = 0
```

5.2.3. 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.

5.2.4. 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.

5.2.5. Afficher le score

On utilise notre fonction crée plus tôt :

🐍 **Script Python**

```
drawText(str(score), font, fenetre, 0.2*LARGEUR, 0.2*HAUTEUR)
```

5.3. Version 2

On ne capturait que "Escape" et le clic sur la croix. On ajoute les flêches.

5.3.1. Capturer les touches du jeu

5.3.2. Diminuer la vitesse de rafaîchissement

🐍 **Script Python**

```
FPS = 5
```

5.3.3. Déplacer le serpent

On commence par créer une direction (= la vitesse) en dehors de la boucle infinie

🐍 **Script Python**

```
direction = (1, 0)
```

5.3.4. Déplacer le serpent

Chaque pression d'une flêche 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 = (..., ...)
```

5.3.5. 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]]
```

5.3.6. Déplacer le serpent

Le corps se déplace.

1. On ajoute la tête au début :

Correction

🐍 **Script Python**

```
snake.insert(0, head)
```

1. On perd un élément de fin :

Correction

🐍 **Script Python**

```
snake.pop(-1)
```

5.4. Version 4

5.4.1. La mort du serpent

Il meurt :

- s'il quitte l'écran
- si sa tête est dans son corps

5.4.2. La mort du serpent

Correction

🐍 **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()
```

5.5. Version 5

5.5.1. Fluidité

Le jeu n'est pas fluide.

On va mettre à jour les éléments du jeux 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

5.5.2. Fluidité

🐍 **Script Python**

```
FPS = 30
MAJ = 10

# ...

# juste avant la boucle infinie
compteur = 0
```

5.5.3. 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 snake

 # On augmente le compteur
 # tout à la fin de la boucle infinie
 compteur += 1
```

5.5.4. Nourriture

On crée d'abord une nouvelle liste :

🐍 **Script Python**

```
pomme = [8, 3]
```

5.5.5. Nourriture

On dessine la pomme comme la tête, mais en rouge

Correction

🐍 **Script Python**

```
pygame.draw.rect(fenetre, ROUGE,(pomme[0] * BLOC, pomme[1] * BLOC, BLOC, BLOC))
```

5.5.6. Nourriture

Puis on détecte la collision avec la pomme.

En cas de collision :

1. Le score augmente
2. Une nouvelle pomme est crée.

La boucle `while` empêche la pomme d'apparaître sur le serpent

5.5.7. Nourriture

Correction

🐍 **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)]
```

5.5.8. Nourriture

S'il n'y a pas de collision le serpent diminue, sinon il conserve sa taille

🐍 **Script Python**

```python
else:
  snake.pop(-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**

```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 rafaîchissement 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 cooordonnées [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
```

6. Mini-Tetris

Un peu d’histoire

Tetris est un jeu vidéo entre arcade et puzzle, conçu par Alekseï 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**

```python
import pygame
import copy
import random

# initialisation de l'écran de jeu
pygame.init()

# Definit 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 AfficheDecors():
    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 rafraichissement de l'écran
clock = pygame.time.Clock()
# Cache le pointeur de la souris
pygame.mouse.set_visible(0)


# variables d'état
# piece courante
idpiece = 1
px  = 6
py  = 0
```

```
rot = 0
#Touches
KEyDown  = 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:
  # recupère la liste des évènements 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 appuyeées 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
    AfficheDecors()
    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.

6.1. Présentation du code

Les constantes couleurs :

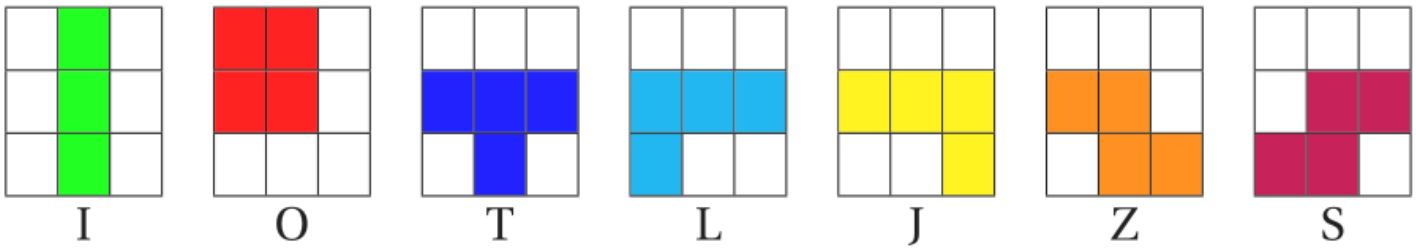
🐍 **Script Python**

```
# Definit 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, ROUGE, 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.

6.1.1. 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
# piece 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 variable `px`, `py` et `rot` indique la position en \(x\), et \(y\) da la pièce dans la grille, ainsi que sa

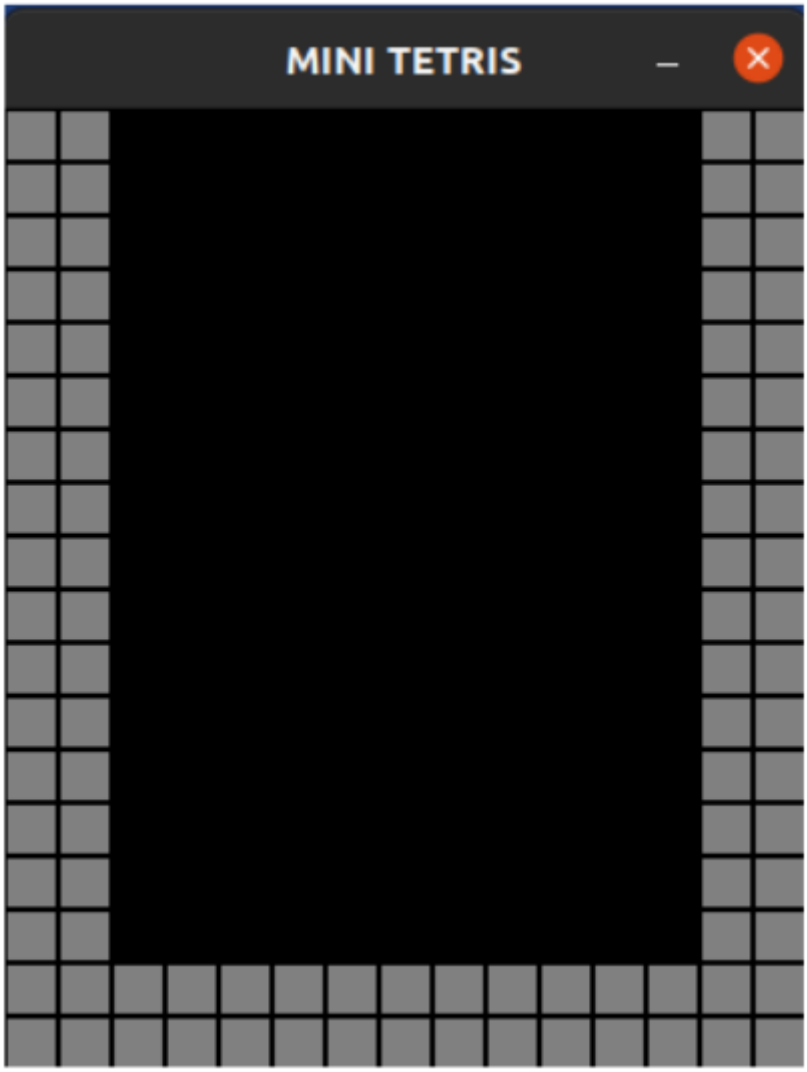
rotation : 0 pour \(0^{\circ}\) et 1 pour \(90^{\circ}\) .
Et quatre variables pour stocker l'état précédent des touches fléchées : enfoncé ou non. L'intérêt des de pouvoir détecter les appuis sur ces touches.

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 AfficheDecors():
    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 15 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 code couleur associé 1. Les 11 cases centrales vides sont remplies avec la valeur 0? Le décor est défini comme une liste de 18 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.

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. a 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**

```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**

```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.

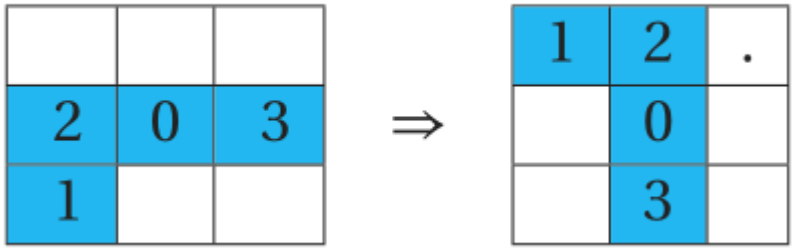
Gestion de la rotation

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 orignal, 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 \(90^{\circ}\). 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 \(90^{\circ}\). 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 calcul 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 appuyer sur la touche [Flèche Haut], vous devez voir la pièce tourner.

Déplacement latéraux

Question 1 Question 2

Écrivez une fonction `DetectColission()` 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.

Conseils de travail

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);

C'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).

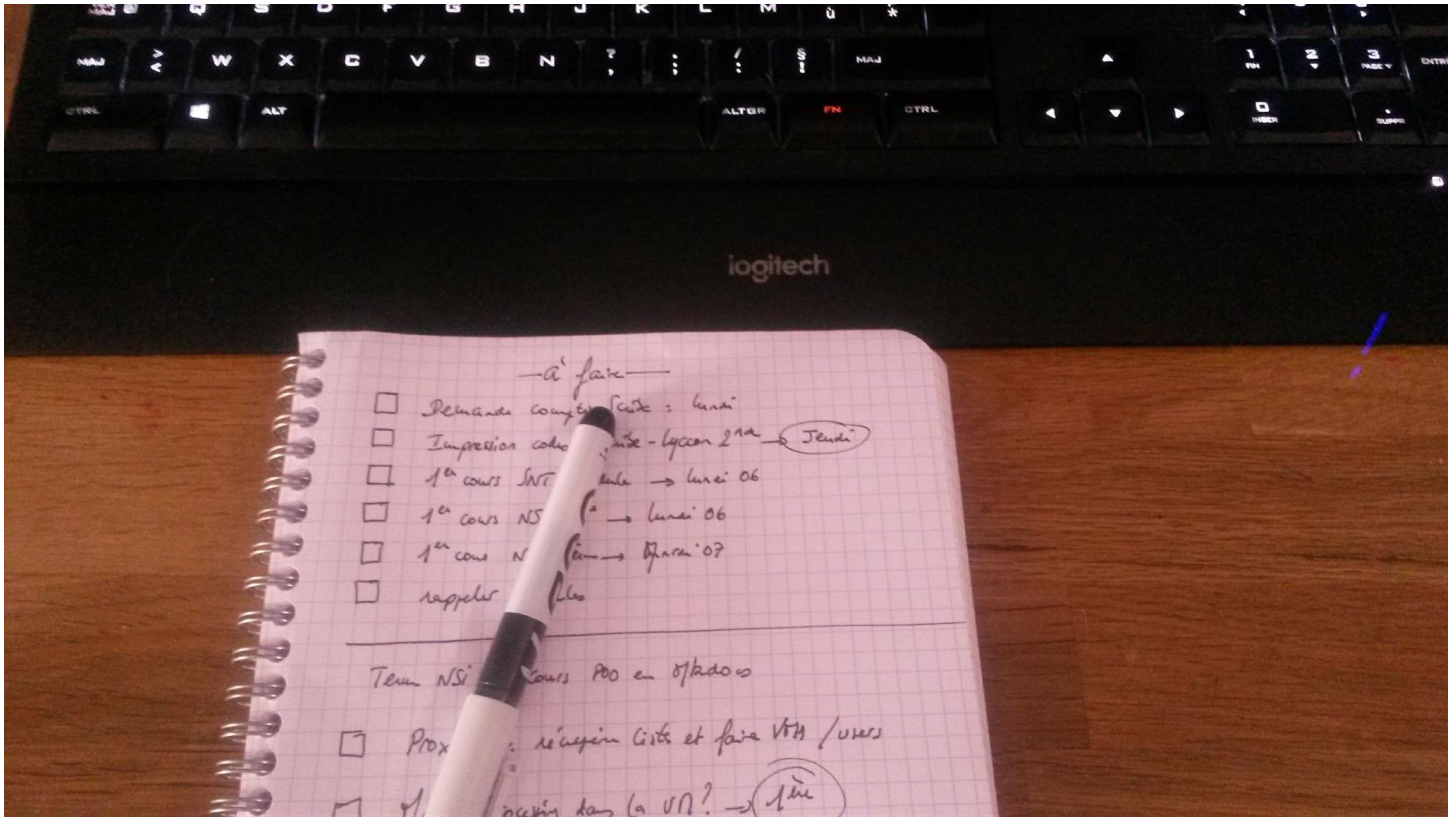
Cela permet de :

- 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. 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 numéroter : `NSI_projet_morpion_v1.py`, puis `NSI_projet_morpion_v2.py`, `NSI_projet_morpion_v3.py`, etc.

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 directionelles 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 triplette 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;
- site de Cédric Gouygou

Sources