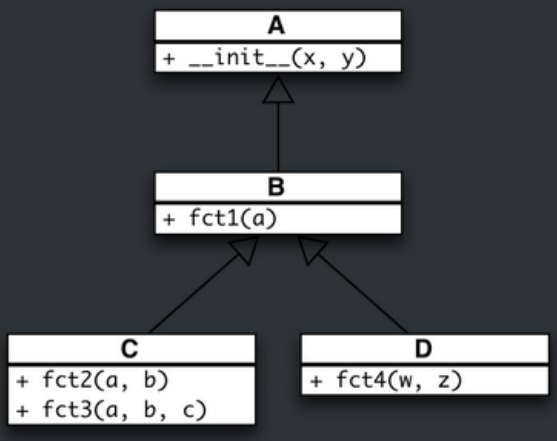
# Examen 2 A2019

## Diagramme UML

Soit le diagramme UML suivant :




Implantez les classes décrites dans ce diagramme en utilisant l'énoncé `pass` pour définir leurs méthodes.

**Bravo!**

Votre score est **100/100**.

Il vous reste **1 soumission** avec rétroaction.

# Machine électorale

Définissez une **classe** nommée `MachineElectorale` permettant d'**encapsuler** un processus de **vote** électronique. Votre classe doit supporter l'interface **publique** suivante:

1. Un **constructeur** qui accepte deux arguments : la **liste** (`list`) des noms des candidats à l'élection et l'**ensemble** (`set`) des identifiants uniques (p.ex. les numéros d'assurance sociale) associés aux électeurs qui ont droit de vote à l'élection.
2. Une **méthode** nommée `reinitialiser` qui permet de remettre à **zéro** la machine électorale, c'est-à-dire d'oublier tous les votes exprimés jusqu'à présent.
3. Une **méthode** nommée `voter` qui accepte **deux** arguments : l'**identifiant** d'un électeur et le **nom** d'un candidat ; et qui ne retourne **rien**, mais **soulève** une exception de type `ValueError` si l'identifiant ne correspond **pas** à un électeur admissible OU si l'électeur à déjà voté OU si le nom du candidat spécifié ne fait **pas** parti des candidatures de cette élection.
4. Une **méthode** nommée `participation` qui retourne le **taux** de participation actuel à l'élection. Lorsque tous les électeurs admissibles ont exercé leur droit de vote, le taux est de 1, sinon, il est compris entre 0 et 1.
5. Une **méthode** nommée `resultats` qui retourne dans l'ordre **décroissant** du nombre de votes, la liste des résultats **actuels** de l'élection, ou chaque résultat est un couple (`tuple`) qui réuni le nom du candidat et son score actuel.

Dans le cas où vous auriez besoin de **trier** une liste de **tuples** selon l'ordre établi par l'un des éléments de ces tuples, sachez que la **fonction** sorted permet de le faire grâce à son **argument** nommé `key`.

Prenez soin de **bien** tester votre classe dans votre **cellule de test** ci-dessous, **avant** de soumettre votre solution au correcteur automatique.

## Contexte de l'exercice

1 ▶

```
candidatures = ['Marc', 'Julien', 'Max', 'Renaud', 'Guillaume']
identifiants = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

### Bien que votre solution soit fonctionnelle, vous devriez la simplifier

- son nombre **d'énoncés** logiques est **9%** plus élevé qu'attendu (notez que *énoncé ≠ ligne*)
- assurez-vous de n'inclure **aucun** énoncé de **test** dans votre cellule de solution
- le cas échéant, déplacez-les dans la **cellule de test** prévue à cette fin

Votre score est **100/100**.

Il vous reste **1 soumission** avec rétroaction.

# Compter récursivement les objets

Écrivez le code source d'une **fonction** nommée `compter` qui accepte en argument un objet **itérable** dont les éléments sont potentiellement eux-mêmes itérables, **récursivement**, et qui **retourne** le nombre **total** d'objets. Votre fonction doit reconnaître **quatre** types particuliers d'objets :

1. la **liste** ;
2. le **tuple** ;
3. l'**ensemble** ;
4. et le **dictionnaire**.

Votre fonction doit **initialiser** une variable pour accumuler le compte, puis **itérer** sur les éléments de l'itérable. Lorsqu'un élément correspond à l'un ou l'autre des **types** ci-dessus, votre fonction doit **ajouter** au compte le résultat d'un traitement **particulier** sur cet élément. Autrement, pour **tout** autre élément, elle ajoute simplement la **valeur 1** au compte actuel.

Dans le cas d'une **liste** ou d'un **tuple**, le traitement consiste à faire un appel **récursif** directement sur l'objet. Dans le cas d'un **ensemble**, le traitement consiste à déterminer le nombre d'éléments de l'ensemble. Dans le cas d'un **dictionnaire**, le traitement consiste à faire un appel **récursif** sur les **valeurs** de l'objet.

À la fin de l'itération, la fonction **retourne** simplement la valeur du compte accumulé. Par exemple :

```
compter([1, [2, [3, 4, 5], {'a': 6, 'b': (7, 8, 9), 'c': {'x': 10}}], (11, 12)])
```

doit retourner **12**.

**Rappel**: la fonction isinstance permet de déterminer si un **objet** est oui on non une **instance** d'une classe.

### Bien que votre solution soit fonctionnelle, vous devriez la simplifier

- son nombre cyclomatique est **20%** plus élevé qu'attendu (trop de conditions et/ou de boucles)
- assurez-vous de n'inclure **aucun** énoncé de **test** dans votre cellule de solution
- le cas échéant, déplacez-les dans la **cellule de test** prévue à cette fin

Votre score est **99/100**.

Il vous reste **2 soumissions** avec rétroaction.

# Journal de bord

Concevez une **classe** qui a toutes les fonctionnalités d'une **liste** et qui encapsule les entrées d'un journal de bord. Votre classe doit définir les méthodes publiques suivantes :

1. Une **constructeur** qui accepte en argument le nom (`str`) du propriétaire du journal. Ce constructeur doit s'assurer que le nom reçu est bien sous la forme d'une instance de la classe `str`. Sinon, il doit soulever une exception de type `TypeError`.
2. Une méthode nommée `ajouter_entree` qui accepte en argument le texte (`str`) d'une entrée au journal de bord, ainsi qu'une date (`datetime`) optionnelle pour cette entrée. Si l'utilisateur omet de préciser la date, alors la méthode doit utiliser la date et l'heure courante. À cette fin, faites appel à la méthode `today` de la classe datetime. Cette méthode doit aussi s'assurer que ses arguments sont du bon type (respectivement `str` et `datetime`) et soulever une exception de type `TypeError` si ce n'est pas le cas.
3. Une méthode nommée `convertir_html` qui va produire en sortie une chaîne de caractères contenant une balise HTML nommée `journal`, elle-même contenant une série de balises nommées `entree`, une pour chacune des entrées du journal de bord (voir exemple ci-dessous). Les balises d'entrée du journal doivent être **triées** dans l'ordre **croissant** des dates. Cette méthode doit accepter deux arguments optionnels nommés `deb` et `fin`, tous les deux sous la forme d'une instance de `datetime`. Lorsque `deb` est spécifié, seules les entrées dont la date est **postérieures** ou égale à `deb` doivent apparaître dans les balises du journal. De façon similaire, lorsque `fin` est spécifié, seules les entrées dont la date est **antérieure** ou égale à `fin` doivent apparaître dans les balises du journal. Et tout comme les méthodes précédentes, on doit **valider** le type de ces deux arguments et, le cas échéant, soulever une exception de type `TypeError`.

Voici un exemple d'utilisation de la classe :

```
journal = JournalDeBord('Marc')
journal.ajouter_entree('Ceci est une 1re entrée sans date')
journal.ajouter_entree('Ceci est une 2e entrée sans date')
journal.ajouter_entree('Ceci est une 3e entrée avec date', datetime(year=2016, month=10, day=2))
print(journal.convertir_html())
print(journal.convertir_html(deb=datetime(year=2017, month=9, day=1)))
```

et les balises HTML produites :

```
<journal nom="Marc">
    <entree date="2016-10-02 00:00:00">Ceci est une 3e entrée avec date</entree>
    <entree date="2017-12-13 23:33:59.507464">Ceci est une 1re entrée sans date</entree>
    <entree date="2017-12-13 23:33:59.507529">Ceci est une 2e entrée sans date</entree>
</journal>
<journal nom="Marc">
    <entree date="2017-12-14 23:10:06.595285">1re entrée</entree>
    <entree date="2017-12-14 23:10:06.595357">2e entrée</entree>
</journal>
```

Notez bien que les entrées du journal sont classées dans l'ordre **croissant** de leur date et que ces dates sont formattées selon le format **par défaut** des objets de la classe `datetime`. Notez aussi que les balises des entrées du journal sont **indentées** de 4 espaces vers la droite. Finalement, notez que nous avons **déjà** importé la classe `datetime` dans la cellule de contexte ci-dessous et que vous ne devez **pas** modifier la valeur de l'identifieur **datetime**, car le correcteur en a besoin pour faire son travail.

## Contexte de l'exercice

1 ▶

```
1 from datetime import datetime
```

### Bien que votre solution soit fonctionnelle, vous devriez la simplifier

- son nombre **d'énoncés** logiques est **29%** plus élevé qu'attendu (notez que *énoncé ≠ ligne*)
- assurez-vous de n'inclure **aucun** énoncé de **test** dans votre cellule de solution
- le cas échéant, déplacez-les dans la **cellule de test** prévue à cette fin

Votre score est **99/100**.

Il vous reste **0 soumission** avec rétroaction.

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
class A:
    def __init__(self, x, y):
        pass


class B(A):
    def fct1(self, a):
        pass


class C(B):
    def fct2(self, a, b):
        pass

    def fct3(self, a, b, c):
        pass


class D(B):
    def fct4(self, w, z):
        pass
```

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
class MachineElectorale:
    """Classe qui encapsule une machine électorale"""
    def __init__(self, candidatures, electeurs):
        """Construire une machine électorale"""
        assert isinstance(candidatures, list) and isinstance(electeurs, set)
        self.candidatures = candidatures
        self.electeurs = electeurs
        self.reinitialiser()

    def participation(self):
        """Déterminer le taux de participation actuel"""
        return sum(1 if vote else 0 for vote in self.votes.values()) / len(self.votes)

    def reinitialiser(self):
        """Réinitialiser l'état de la machine électorale"""
        # le dictionnaire des votes indique pour chaque électeurs s'il a voté ou non
        self.votes = {id: False for id in self.electeurs}
        # le dictionnaire des scores indique le score actuel de chaque candidat
        self.scores = {nom: 0 for nom in self.candidatures}

    def resultats(self):
        """Produire les résultats de l'élection"""
        return sorted(self.scores.items(), key=lambda x: x[1], reverse=True)

    def voter(self, id, candidat):
        """Enregistrer le vote d'un électeur pour un candidat"""
        if id in self.electeurs and candidat in self.candidatures and not self.votes[id]:
            self.votes[id] = True
            self.scores[candidat] += 1
        else:
            raise ValueError('vote invalide')
```

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def compter(itérable):
    """Fonction qui compte récursivement les éléments d'un itérable"""
    res = 0
    for item in itérable:
        # traiter chaque élément de l'itérable
        if isinstance(item, (list, tuple)):
            # l'item est une list ou un tuple; on fait un appel récursif
            res += compter(item)
        elif isinstance(item, set):
            # l'item est un ensemble; on additionne son nombre d'éléments
            res += len(item)
        elif isinstance(item, dict):
            # l'item est un dictionnaire; on fait un appel récursif sur ses valeurs
            res += compter(item.values())
        else:
            # autrement, on additionne 1 au compte
            res += 1
    return res
```

## Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
class JournalDeBord(list):
    """Classe qui encapsule un journal de bord"""
    def __init__(self, nom):
        """Construire une instance d'un journal de bord"""
        if not isinstance(nom, str):
            raise TypeError(
                "Le nom doit être spécifié sous la forme d'une chaîne de caractères"
            )
        self.nom = nom

    def ajouter_entree(self, texte, date=None):
        """Ajouter une entrée au journal de bord"""
        if not isinstance(texte, str):
            raise TypeError(
                "Le texte doit être spécifié sous la forme d'une chaîne de caractères"
            )
        if not (date is None or isinstance(date, datetime)):
            raise TypeError(
                "La date de l'entrée doit être spécifiée sous la forme d'un objet de "
                "la classe datetime"
            )
        if date is None:
            date = datetime.today()
        self.append((date, texte))
        # garder les entrées triées en tout temps
        self.sort()

    def convertir_html(self, deb=None, fin=None):
        """convertir le journal de bord en HTML"""
        if not (deb is None or isinstance(deb, datetime)):
            raise TypeError(
                "La date de début doit être spécifiée sous la forme d'un objet de la "
                "classe datetime"
            )
        if not (fin is None or isinstance(fin, datetime)):
            raise TypeError(
                "La date de fin doit être spécifiée sous la forme d'un objet de la "
                "classe datetime"
            )
        chaîne = '<journal nom="{}">'.format(self.nom)
        for date, texte in self:
            if (not deb or date >= deb) and (not fin or date <= fin):
                chaîne += '\n    <entree date="{}">{}</entree>'.format(date, texte)
        return chaîne + "\n</journal>"
```