Numéro d'anonymat

Examen 2020 – 2021
Éléments de Programmation I – LU1IN001
Durée : 1h30

Documents autorisés : Aucun document ni machine électronique n'est autorisé à l'exception de la carte de référence de Python.

Le sujet comporte 21 pages. Ne pas désagrafer les feuilles.

Répondre directement sur le sujet, dans les boîtes appropriées. La taille des boîtes suggère le nombre de lignes de la réponse attendue. Le quadrillage permet d'aligner les indentations.

Le barème indiqué pour chaque question n'est donné qu'à titre indicatif. Le barème total est lui aussi donné à titre indicatif : 48 points auxquels peuvent s'ajouter des points bonus explicités dans l'énoncé des questions.

La clarté des réponses et la présentation des programmes seront appréciées. Les exercices peuvent être résolus dans un ordre quelconque. Pour répondre à une question, **il est possible, et souvent utile, d'utiliser les fonctions qui sont l'objet des questions précédentes, même si vous n'avez pas répondu à ces questions précédentes.**

Remarque : si nécessaire, on considère que la bibliothèque de fonctions mathématiques a été importée avant les fonctions à écrire. Sauf mention contraire explicite, seules les primitives Python présentes sur la carte de référence peuvent être utilisées.

Important: Sauf en cas d'exception (notamment dans les exercices d'analyse de code), pour les fonctions demandées, **il est nécessaire de donner une définition avec précondition(s) éventuelle(s)**. En revanche, la description textuelle et les jeux de tests ne sont *pas* demandés, contrairement aux exercices sur machine.

L'examen est composé d'exercices indépendants :

— Exercice 1 : Consonne ou voyelle – (p. 2)
— Exercice 2 : Informations nutritionnelles (Dictionnaires) – (p. 7)
— Exercice 3 : Analyse de code – (p. 12)
— Exercice 4 : Sorties de produits culturels (Listes de n-uplets) – (p. 14)

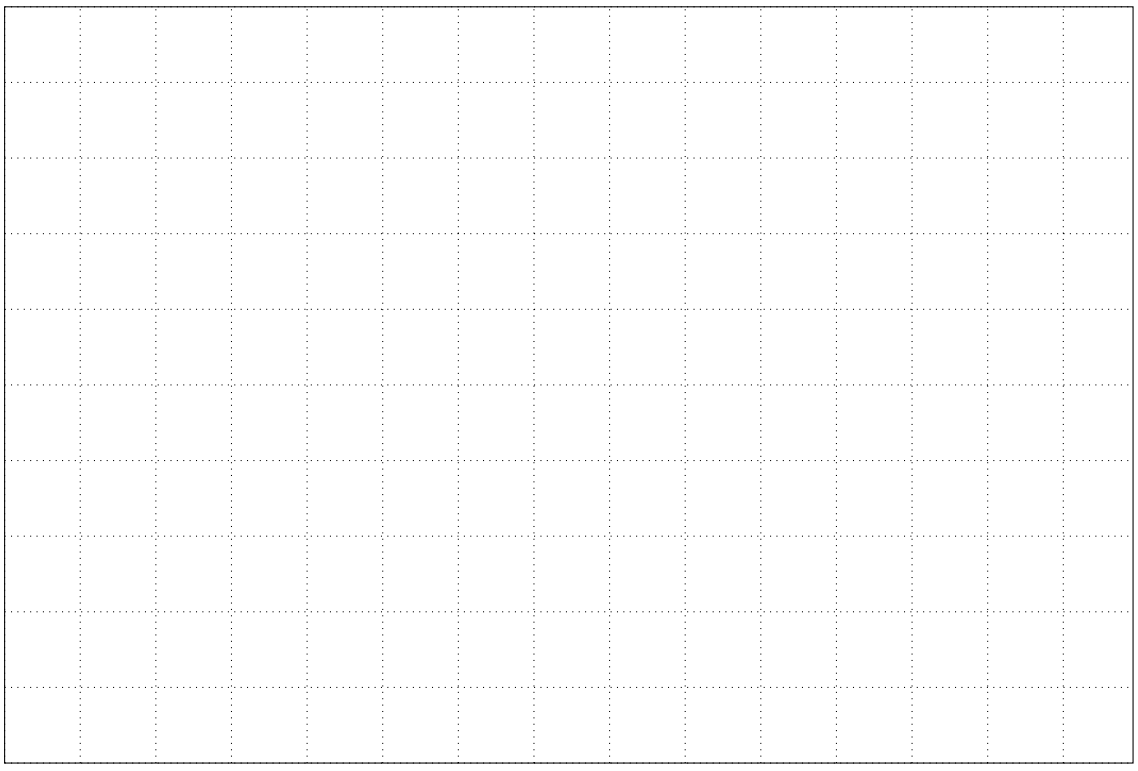
Exercice 1 : Consonne ou voyelle

Dans cet exercice, on considère des chaînes de caractères qui contiennent uniquement les 26 lettres de l'alphabet minuscules sans accent, mais il ne sera pas nécessaire de le préciser en préconditions des fonctions.

Question 1.1 : [2/48]

Donner une définition de la fonction `nb_occ` qui, étant données une chaîne de caractères `mot` et une chaîne de caractères `lettre` contenant un unique caractère, renvoie le nombre d'occurrences du caractère `lettre` dans la chaîne de caractères `mot`.

```
>>> nb_occ('','a')
0
>>> nb_occ('bbb','b')
3
>>> nb_occ('abc','c')
1
>>> nb_occ('abcdabcd','d')
2
```



Question 1.2 : [2/48]

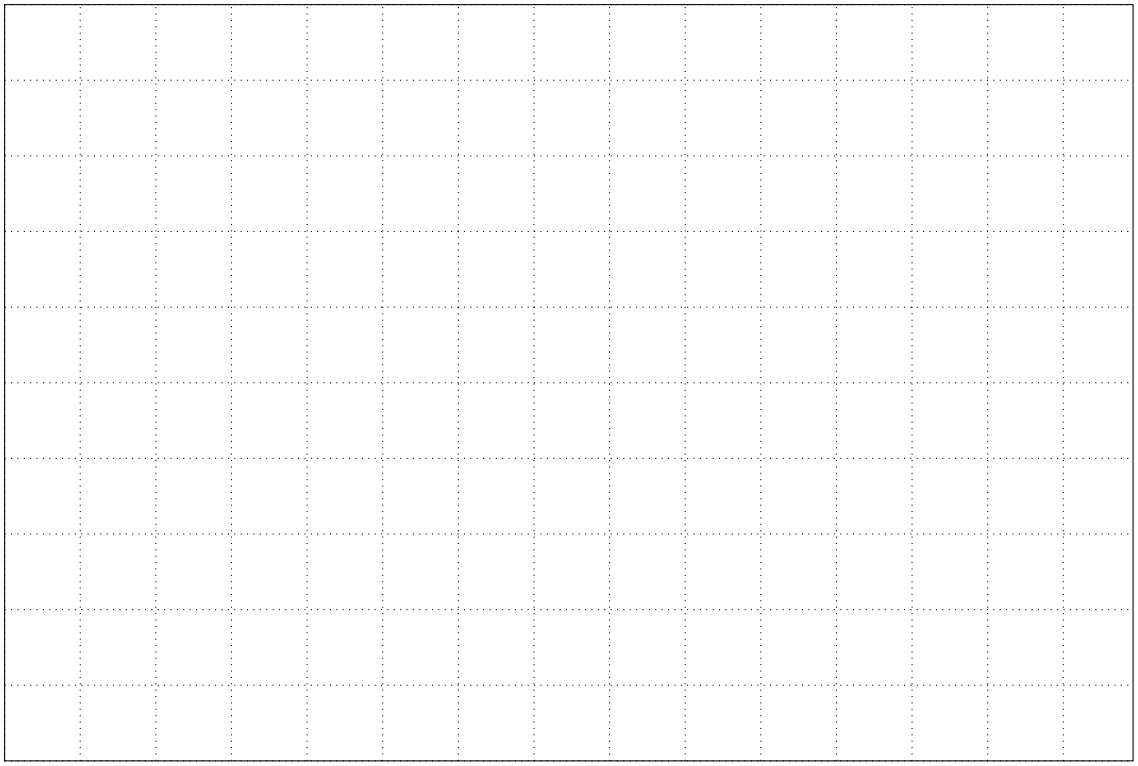
Donner une définition de la fonction `est_voyelle` qui, étant donnée une chaîne de caractères `lettre` contenant un unique caractère, renvoie vrai si le caractère est une voyelle et faux si le caractère est une consonne.

```
>>> est_voyelle('a')
True
>>> est_voyelle('b')
False
>>> est_voyelle('y')
True
>>> est_voyelle('z')
False
```

Question 1.3 : [2/48]

Donner une définition de la fonction `nb_voyelles` qui, étant donnée une chaîne de caractères `mot`, renvoie le nombre total de voyelles qui apparaissent dans `mot`.

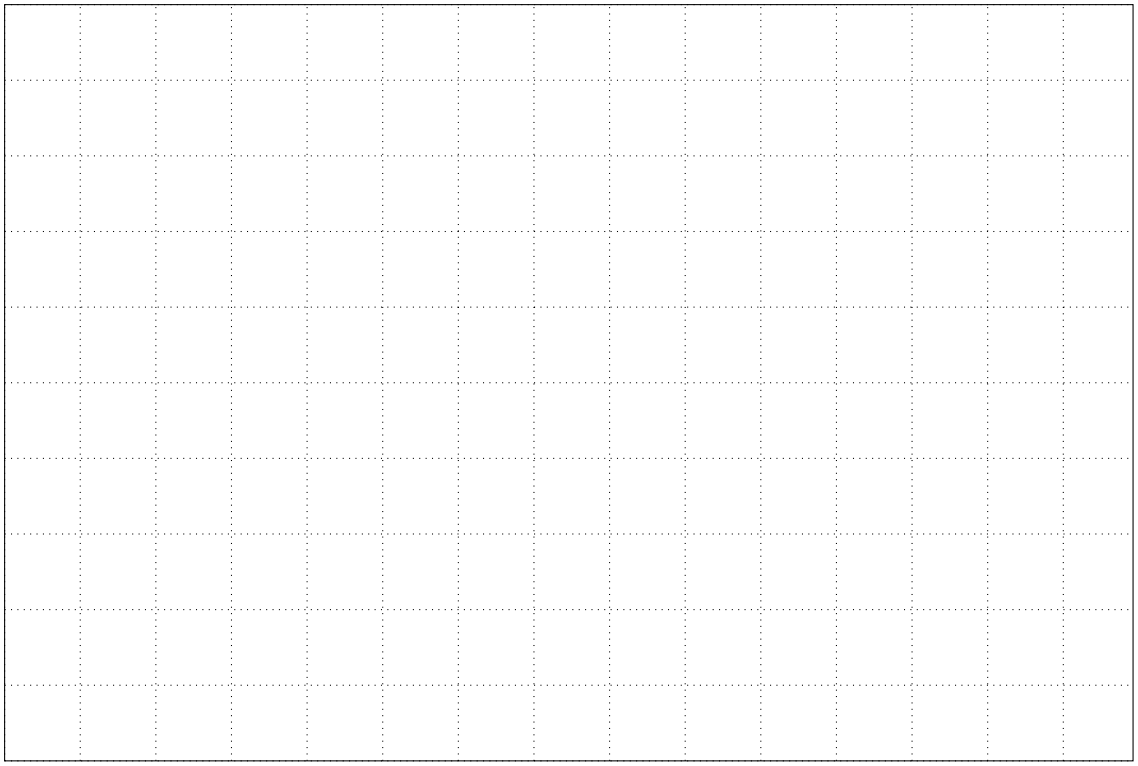
```
>>> nb_voyelles('')
0
>>> nb_voyelles('aeiouy')
6
>>> nb_voyelles('bonjour')
3
>>> nb_voyelles('zzzzzzz')
0
```



Question 1.4 : [2/48]

Donner une définition de la fonction `rm_voyelles` qui, étant donnée une chaîne de caractères `mot`, renvoie une chaîne de caractères égale à `mot` mais dont les voyelles ont été supprimées.

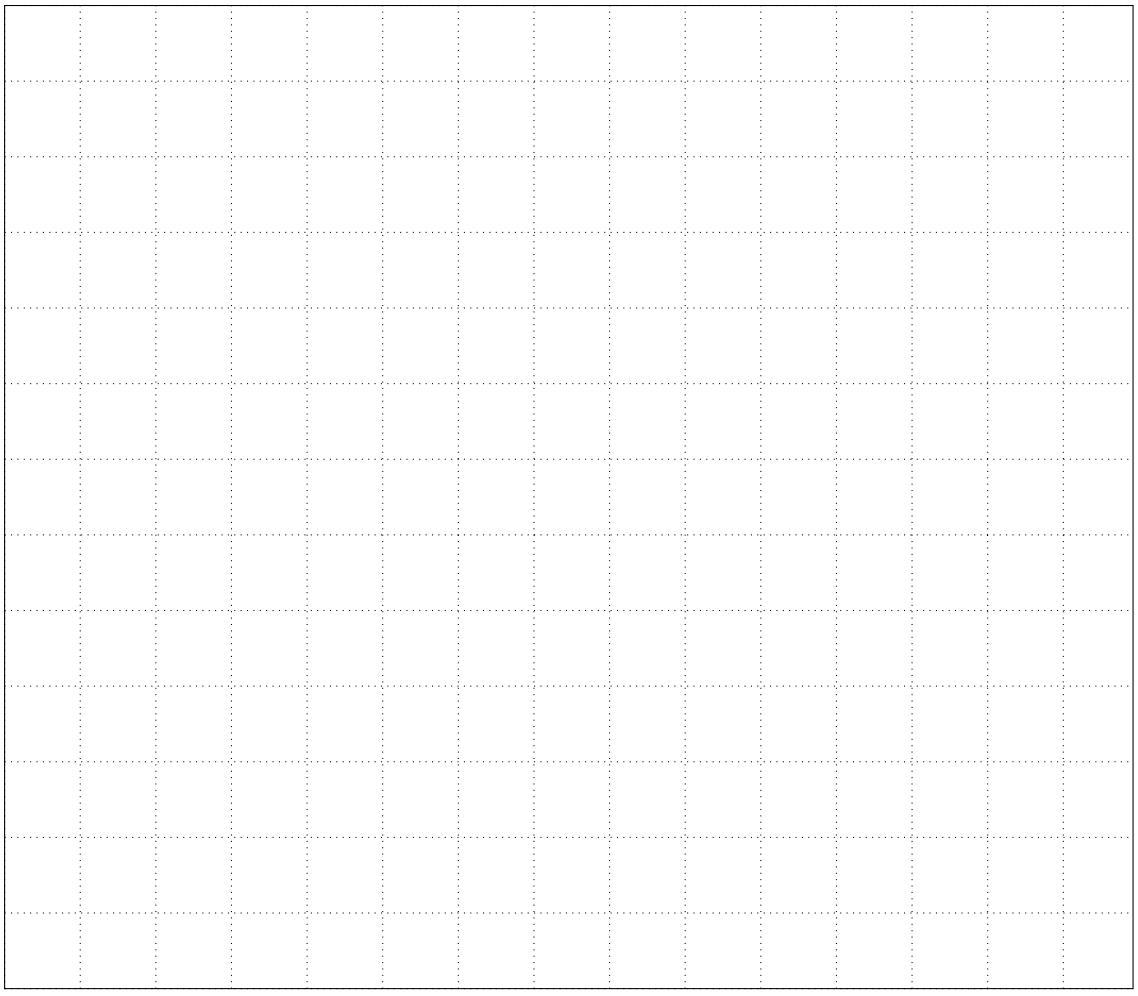
```
>>> rm_voyelles('')
''
>>> rm_voyelles('aeiouy')
''
>>> rm_voyelles('bonjour')
'bnjr'
>>> rm_voyelles('zzzzzzz')
'zzzzzzz'
```



Question 1.5 : [3/48]

Donner une définition de la fonction `consonne_ou_voyelle` qui, étant donnée une chaîne de caractères `mot`, renvoie une chaîne de caractères égale à `mot` mais où les consonnes ont été remplacées par le caractère 'c' et les voyelles par le caractère 'v'.

```
>>> consonne_ou_voyelle('')
''
>>> consonne_ou_voyelle('aeiouy')
'vvvvvv'
>>> consonne_ou_voyelle('bonjour')
'cvccvvc'
>>> consonne_ou_voyelle('zzzzzzz')
'ccccccc'
```



Exercice 2 : Informations nutritionnelles

Dans cet exercice, on se propose d'écrire des fonctions qui manipulent les informations nutritionnelles d'aliments. La composition nutritionnelle pour 100g d'un aliment correspond aux masses en grammes de glucides, de protides, de lipides, de fibres ou de tout autre nutriment.

Pour cela, on définit le type `Composition` qui est un dictionnaire qui associe au nom du nutriment (`'glucides'`, `'protides'`...), sa masse en grammes.

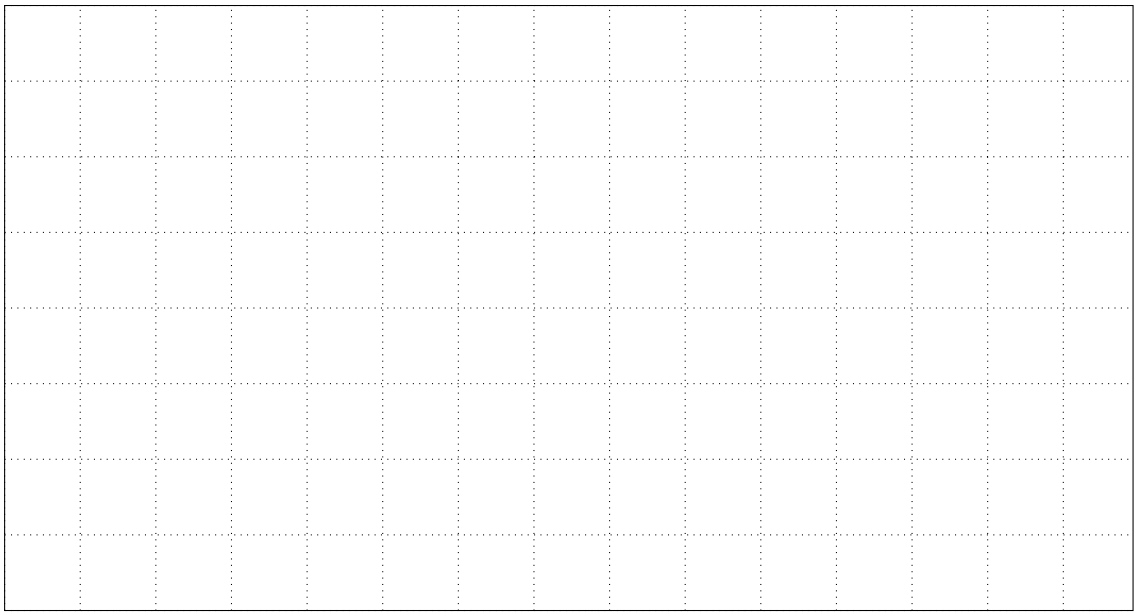
```
Composition = Dict[str, float]

comp_huile : Composition = { 'lipides': 100.0 }
comp_beurre : Composition = { 'glucides': 0.5, 'protides': 0.8,
                              'lipides': 84.0 }
comp_nectarine : Composition = { 'glucides': 11.2, 'protides': 0.8,
                                 'fibres': 2.1 }
```

Ainsi on peut lire que 100g de nectarines sont composés de 11,2g de glucides, 0,8g de protides et de 2,1g de fibres. On remarque que la somme des masses de nutriments n'est pas nécessairement égale à 100g.

Question 2.1 : [2/48]

Donner une définition de la fonction `quantite_dans` qui, étant donné la composition `c` d'un aliment et un nutriment `nut`, renvoie la masse en grammes de nutriment présente dans 100g de l'aliment. *Si le nutriment est absent de la composition, la fonction renvoie* `0.0`.

```
>>> quantite_dans(comp_beurre, 'glucides')
0.5
>>> quantite_dans(comp_huile, 'glucides')
0.0
>>> quantite_dans(comp_nectarine, 'fibres')
2.1
```



Question 2.2 : [2/48]

Des savants ont déterminé que 1g de glucides ou de protides libère 4 Calories alors que 1g de lipides libère 9 Calories. Les autres nutriments ne libèrent pas d'énergie.

Donner une définition de la fonction `energie` qui, étant donné la composition `c` de 100g d'un aliment, renvoie l'énergie en Calories que libèrent 100g de cet aliment.

```
>>> energie(comp_huile)
900.0
>>> energie(comp_beurre)
761.2
>>> energie(comp_nectarine)
48.0
```

Question 2.3 : [2/48]

Dans cet exercice, la composition nutritionnelle d'un aliment ne précise pas la quantité d'eau. On suppose que tout ce qui n'est pas indiqué comme nutriment est de l'eau.

Donner une définition de la fonction `quantite_eau` qui, étant donné la composition `c` de 100g d'un aliment, renvoie la masse d'eau en grammes qu'il contient.

```
>>> quantite_eau(comp_huile)
0.0
>>> quantite_eau(comp_nectarine)
85.9
```

Question 2.4 : [3/48]

On introduit une base de données d'aliments qui à chaque nom d'aliment associe sa composition nutritionnelle. Pour cela, on introduit le type `BaseAliments` ci-dessous :

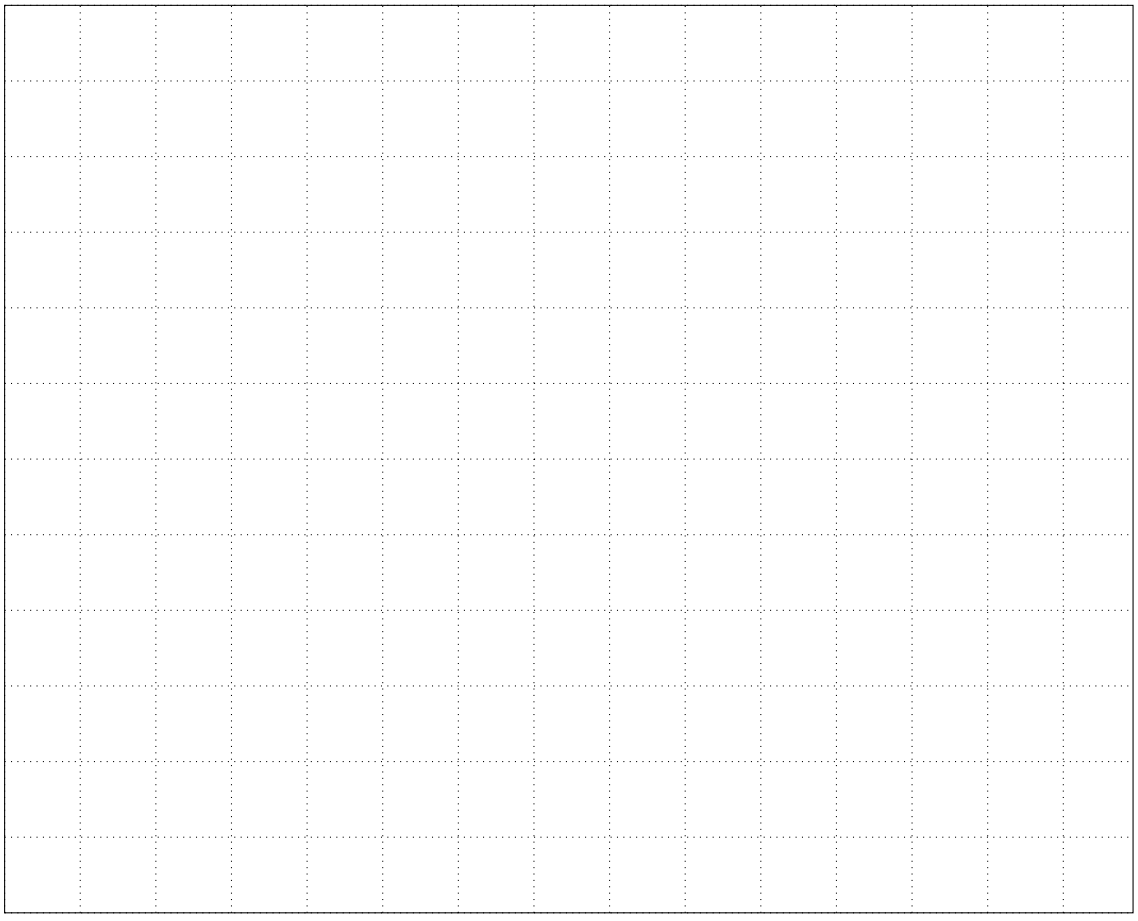
```
BaseAliments = Dict[str, Composition]
```

Un exemple de base d'aliments est défini ci-dessous :

```
crous : BaseAliments = {'huile': comp_huile, 'beurre': comp_beurre,
                        'nectarine': comp_nectarine,
                        "jaune d'oeuf": {'glucides': 3.6,
                                         'protides': 16.0, 'lipides': 27.0},
                        'moutarde': {'glucides': 5.0, 'protides': 4.4,
                                     'lipides': 4.0},
                        'sucre': {'glucides': 96.0}}
```

On souhaite connaître les aliments qui contiennent plus d'une certaine quantité de lipides pour 100g. Donner une définition de la fonction `plus_gras_que` qui, étant donné une base `b` d'aliments et une quantité `q` strictement positive de lipides, renvoie la liste des aliments dont la teneur en lipides pour 100g est supérieure ou égale à `q`.

```
>>> plus_gras_que(crous, 90.0)
['huile']
>>> plus_gras_que(crous, 20.0)
['huile', 'beurre', "jaune d'oeuf"]
```



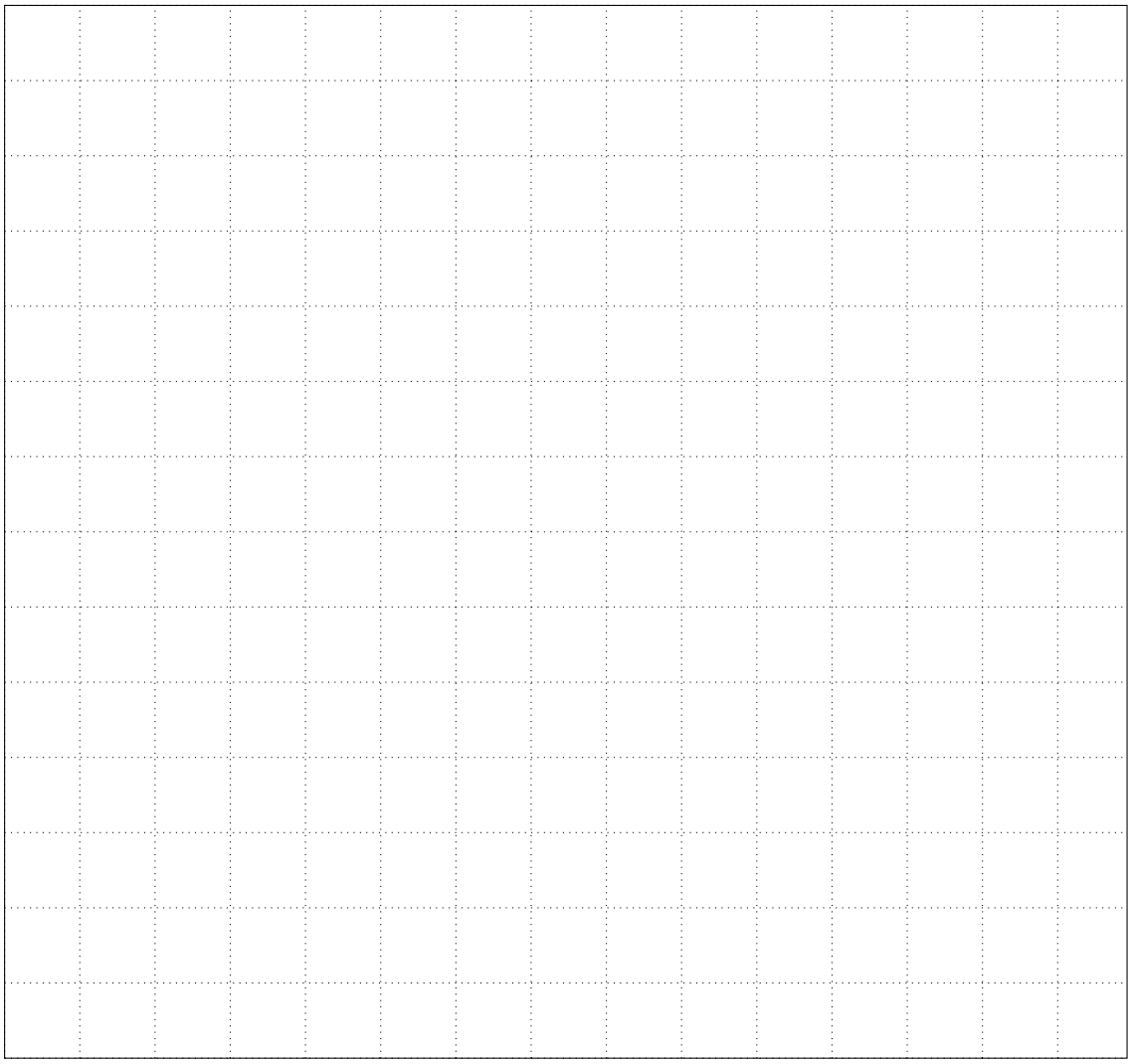
Question 2.5 : [5/48]

On mélange des aliments de base à partir d'une recette. On souhaite connaître les informations nutritionnelles de la recette avant une éventuelle cuisson. Pour représenter une recette, on utilise une liste de couples composés du nom de l'ingrédient (aliment) et de son pourcentage dans la recette. Par exemple, on donne ci-dessous la recette de la mayonnaise :

```
mayonnaise : List[Tuple[str, float]]
mayonnaise = [('huile', 70.0), ("jaune d'oeuf", 16.0), ('moutarde', 14.0)]
```

Donner une définition de la fonction `composition_recette` qui, étant donné une base `b` de compositions nutritionnelles d'aliments et une recette `rec`, renvoie la composition nutritionnelle de la recette pour 100g. *On supposera que les aliments de la liste recette existent dans la base.*

```
>>> composition_recette(crous, mayonnaise)
{'lipides': 74.88, 'glucides': 1.276, 'protides': 3.176}
>>> composition_recette(crous, [('nectarine', 50.0), ('sucre', 50.0)])
{'glucides': 53.6, 'protides': 0.4, 'fibres': 1.05}
>>> composition_recette(crous, [('huile', 100.0)])
{'lipides': 100.0}
```



Exercice 3 : Analyse de code

Question 3.1 : [1/48]

Complétez les informations de type et donnez, **si nécessaire**, la précondition dans la définition de `f` ci-dessous.

```
def f(b:          ) ->          :
    """ Précondition:                    """
    c :
    c = 0
    d :
    d = 0
    e :
    e = 1
    while (d < b):
        c = c + e
        e = e + 2
        d = d + 1
    return c
```

Question 3.2 : [5/48]

Choisissez laquelle de ces 4 **égalités mathématiques** est un invariant de la boucle.

1. $\texttt{e} = \texttt{d} + 2$
2. $\texttt{e} = 2\texttt{d}$
3. $\texttt{e} = 2\texttt{d} + 1$
4. $\texttt{e} = \texttt{e} + 2$

Vérifiez la validité de votre choix sur la table de simulation ci-dessous pour l'application `f(5)`.

Vous ferez figurer dans la dernière colonne de cette table

— sur la première ligne : l'égalité choisie

— sur les autres lignes : vous écrirez les instances correspondant à chaque valeur de `c`, `d` et `e` de cette égalité.

Tour	`c`	`d`	`e`	`e=`
entrée				
1er				

Question 3.3 : [2/48]

Exprimez par une phrase ou une formule ce que calcule la fonction `f` appliquée à un paramètre n respectant l'éventuelle précondition que vous avez posée.

Exercice 4 : Liste de Sorties

Une application manipule des dates écrite sous la forme `JJ-MM-AAAA` c'est-à-dire qu'une date (un jour particulier) est représentée par une chaine de caractères composée de deux chiffres représentant le jour du mois, deux chiffres représentant le mois de l'année et quatre chiffres représentant l'année (à partir de l'an 1). Ces trois groupes de chiffres sont séparés par des `-`.

Ainsi la chaîne de caractères `"15-01-2021"` représente la date du 15 janvier 2021.

Question 4.1 : [2/48]

Donner la définition d'une fonction `triplet_date` qui prend en entrée une chaîne de caractères `d` correspondant à une date sous la forme `JJ-MM-AAAA` et qui renvoie un triplet d'entiers `(j, m, a)` où `j` est le jour de `d`, `m` son mois et `a` son année.

```
>>> triplet_date("15-01-2021")
(15, 1, 2021)
>>> triplet_date("10-12-2020")
(10, 12, 2020)
```

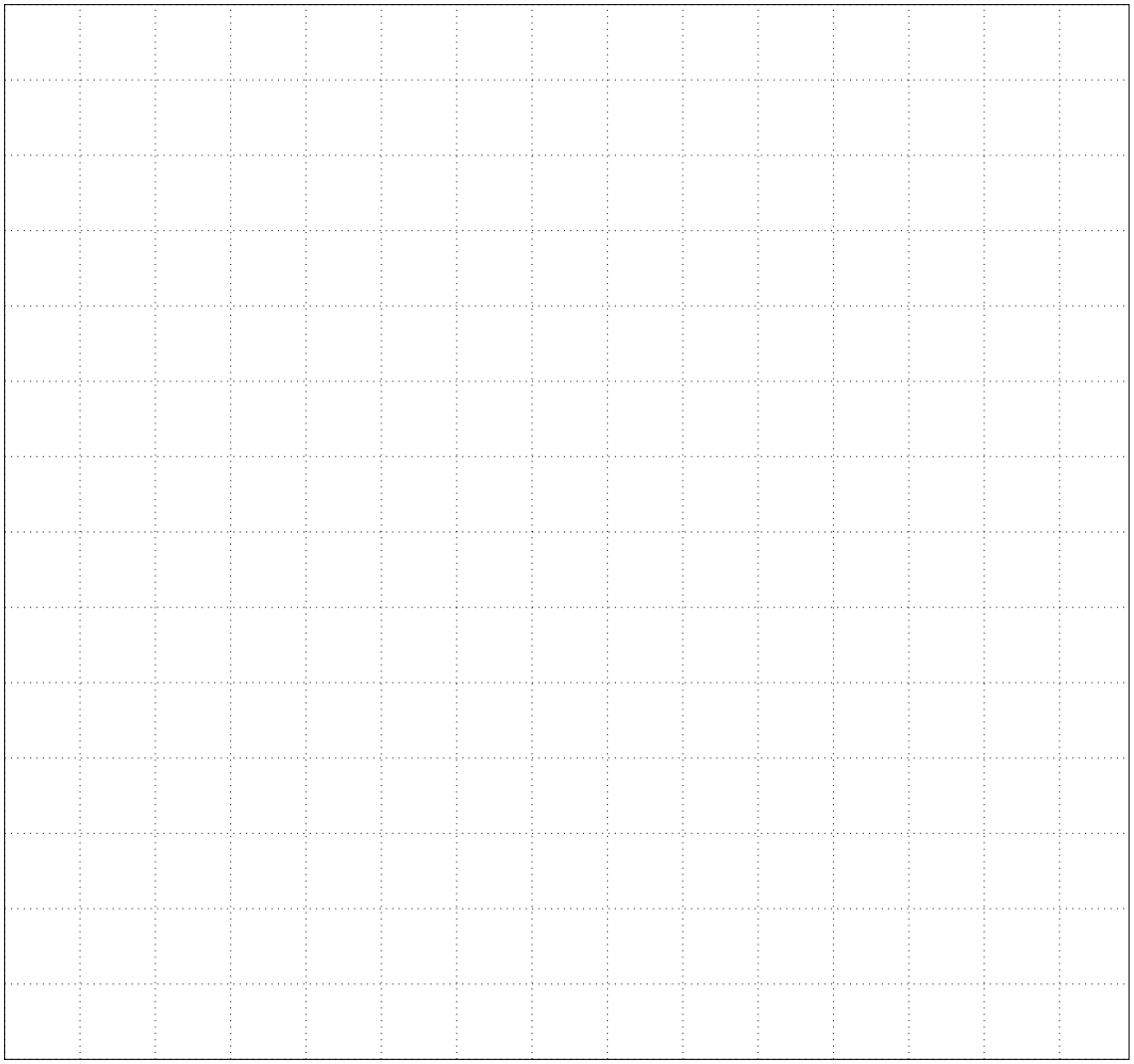
Indication : On utilisera la primitive `int` qui, entre autres, convertit une chaîne de caractères en entier quand c'est possible :

```
>>> int("15")
15
>>> int("1")
1
```

Question 4.2 : [2/48]

Donner une fonction `avant_ou_egale` qui prend en entrée deux chaînes de caractères `d1` et `d2` correspondant toutes les deux à des dates au format `JJ-MM-AAAA` et qui décide (c'est-à-dire qui renvoie `True` quand c'est vrai et `False` sinon) si la `d1` est une date antérieure ou égale à `d2` (c'est-à-dire, si `d1` est inférieure ou égale pour l'ordre chronologique à `d2`)

```
>>> avant_ou_egale("25-12-2020", "15-01-2021")
True
>>> avant_ou_egale("25-12-2020", "15-01-2020")
False
>>> avant_ou_egale("15-01-2021", "15-01-2021")
True
```



Une application propose à ses utilisateurs de mémoriser des listes de dates de sorties dans le commerce de produits culturels (films, albums, livres, ...). Chaque *sortie* est représentée par un triplet de chaînes de caractères : la date de sortie sous la forme `JJ-MM-AAAA`, le titre du produit et sa catégorie,
par exemple `("10-12-2020", "Minitelpunk 1977", "jeu video")`.

On définit le type `Sorties` pour des listes de sorties (les listes composées de tels triplets) **quand les triplets sont classés par ordre croissant de dates de sortie**.

```
Sorties = List[Tuple[str, str, str]]
# Le premier membre de chaque element d'une Sorties de sortie
#     est une date au format JJ-MM-AAAA
# Les elements de la listes sont classes dans l'ordre chronologique
#     de leur premier membre.
```

Quand on annote un paramètre d'une fonction avec le type `Sorties`, on n'a pas besoin de préciser en précondition que ses éléments sont triés dans l'ordre chronologique, ni que le premier membre de chaque élément est une date au format JJ-MM-AAAA.

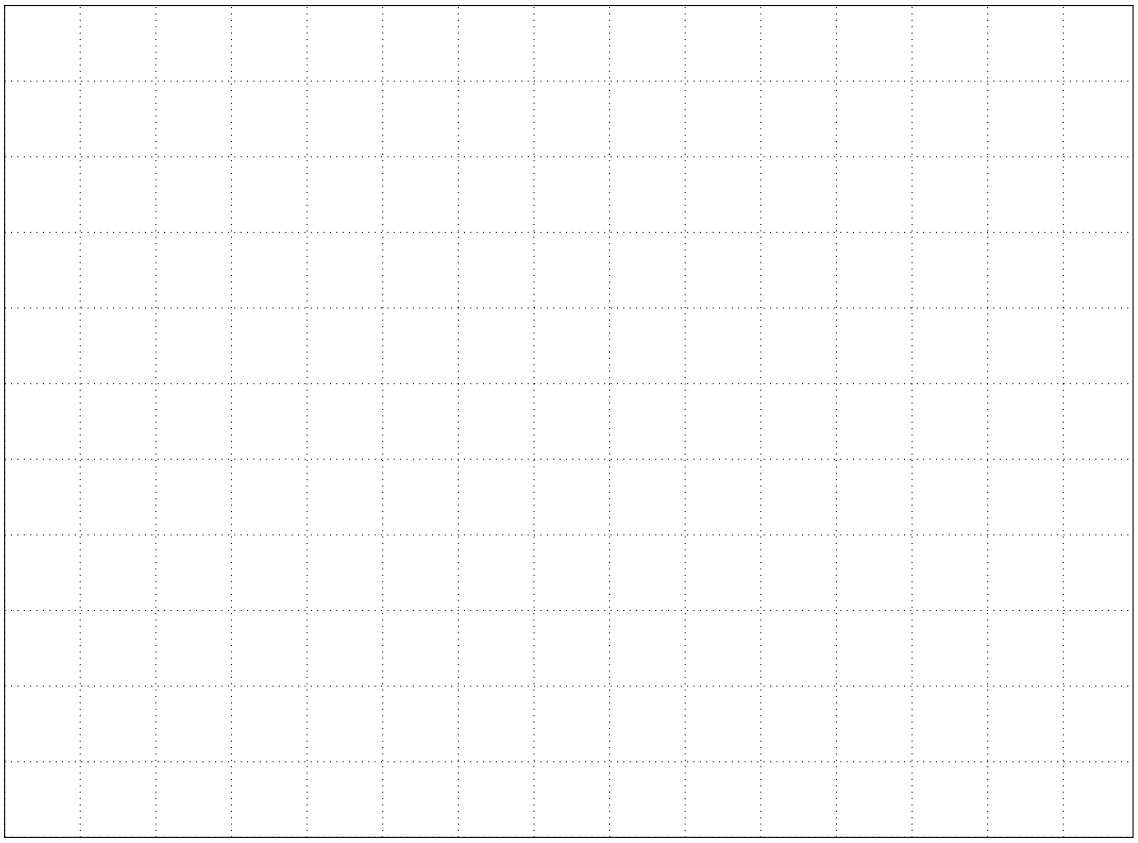
On utilisera cet exemple dans les énoncés suivants :

```
automne2020 : Sorties
automne2020 = [("27-08-2020", "Yoda", "livre"),
               ("02-10-2020", "Emily in Melun", "serie"),
               ("06-10-2020", "ITTF 21", "jeu video"),
               ("23-10-2020", "Le Jeu des Petits Chevaux", "serie"),
               ("10-11-2020", "AC: Nogent-le-Rotrou", "jeu video"),
               ("10-12-2020", "Minitelpunk 1977", "jeu video")]
```

Question 4.3 : [3/48]

Donner la définition d'une fonction `sorties_categorie` qui prend en entrée une liste de sorties `li` (de type `Sorties`), une chaîne de caractères `cat` et qui renvoie une liste de sorties correspondant à toutes les sorties de `li` dont la catégorie est `cat`.

```
>>> sorties_categorie(automne2020, "jeu video")
[("06-10-2020", "ITTF 21", "jeu video"),
 ("10-11-2020", "AC: Nogent-le-Rotrou", "jeu video"),
 ("10-12-2020", "Minitelpunk 1977", "jeu video")]
>>> sorties_categorie(automne2020, "livre")
[("27-08-2020", "Yoda", "livre")]
>>> sorties_categorie(automne2020, "cinema")
[]
```

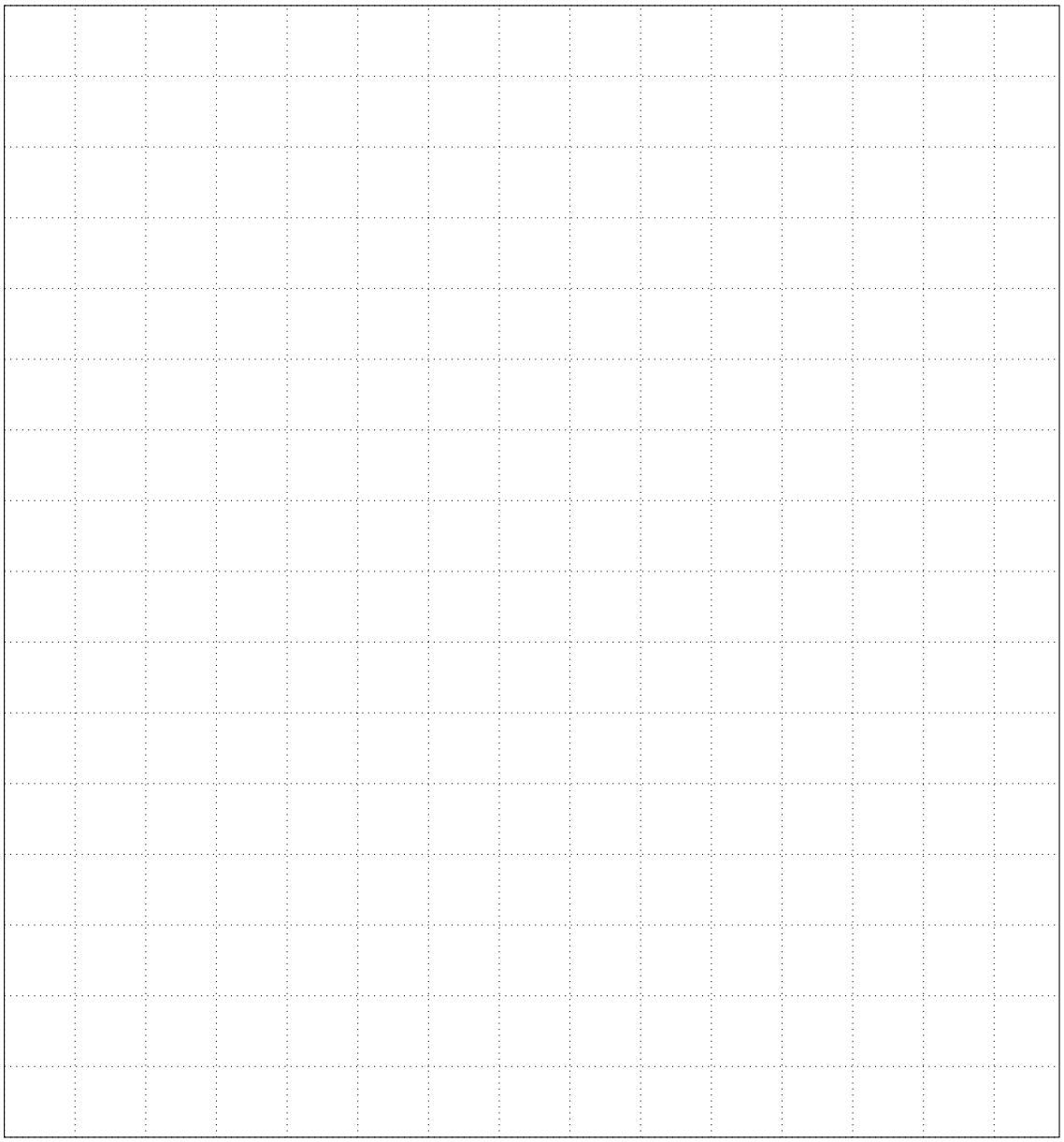


Question 4.4 : [4/48]

Donner une fonction `sorties_ajout` qui prend en entrée une liste de sorties `li` (de type `Sorties`), un triplet de chaînes de caractères `sort` correspondant à une sortie (son premier membre doit être une date au format `JJ-MM-AAAA`) et qui renvoie une liste de sorties correspondant à `li` dans laquelle `sort` a été insérée **au bon endroit** (c'est à dire que l'ordre chronologique doit être respecté).

```
>>> sorties_ajout(automne2020, ("01-09-2020", "Elements de programmation", "livre"))
[("27-08-2020", "Yoda", "livre"),
 ("01-09-2020", "Elements de programmation", "livre"),
 ("02-10-2020", "Emily in Melun", "serie"),
 ("06-10-2020", "ITTF 21", "jeu video"),
 ("23-10-2020", "Le Jeu des Petits Chevaux", "serie"),
 ("10-11-2020", "AC: Nogent-le-Rotrou", "jeu video"),
 ("10-12-2020", "Minitelpunk 1977", "jeu video")]
>>> sorties_ajout(automne2020, ("22-12-2021", "La 7eme compagnie 4", "cinema"))
[("27-08-2020", "Yoda", "livre"),
 ("02-10-2020", "Emily in Melun", "serie"),
 ("06-10-2020", "ITTF 21", "jeu video"),
 ("23-10-2020", "Le Jeu des Petits Chevaux", "serie"),
 ("10-11-2020", "AC: Nogent-le-Rotrou", "jeu video"),
 ("10-12-2020", "Minitelpunk 1977", "jeu video"),
 ("22-12-2021", "La 7eme compagnie 4", "cinema")]
```

Précision : Si une (ou plusieurs) sortie(s) avec la même date que `sort` est déjà présente (sont déjà présentes) dans `li`, il n'importe pas que `sort` soit placée avant ou après cette sortie (avant, apres ou entre ces sorties). L'ordre des sorties de même date n'a pas d'importance.



Question 4.5 : [4/48]

Donner la définition d'une fonction `deux_sorties_rassemblees` qui prend en entrée deux listes de sorties `l1` et `l2` (de type `Sorties`) et qui renvoie une liste de sorties correspondant au "rassemblement" des listes de `l1` et `l2`, c'est-à-dire une liste qui contient toutes les sorties présentes dans chacune des deux listes (dans l'ordre chronologique, évidemment).

```
>>> deux_sorties_rassemblees(sorties_categorie(automne2020, "serie"),
                             sorties_categorie(automne2020, "livre"))
[('27-08-2020', 'Yoda', 'livre'),
 ('02-10-2020', 'Emily in Melun', 'serie'),
 ('23-10-2020', 'Le Jeu des Petits Chevaux', 'serie')]
>>> deux_sorties_rassemblees(sorties_categorie(automne2020, "jeu video"),
                             sorties_categorie(automne2020, "serie"))
[('02-10-2020', 'Emily in Melun', 'serie'),
 ('06-10-2020', 'ITTF 21', 'jeu video'),
 ('23-10-2020', 'Le Jeu des Petits Chevaux', 'serie'),
 ('10-11-2020', 'AC: Nogent-le-Rotrou', 'jeu video'),
 ('10-12-2020', 'Minitelpunk 1977', 'jeu video')]
```