# TP1 - Filtrage par motifs

> **Remarque générale** (*valable pour tous vos prochains TP dans l'UE et au-delà*) : **ce n'est pas parce que tous les tests automatisés fournis passent que votre implémentation est forcément correcte, algorithmiquement optimale, ou suffisamment lisible :)** D'où les trois recommandations suivantes :

1. Implémenter vos fonctions de manière "incrémentale", en testant des petites unités de code (par exemple avec le *Toplevel*) au fur et à mesure que vous écrivez vos fonctions, plutôt que de coder 24 lignes d'un coup à déboguer ensuite !
2. Ne pas hésiter à "refactoriser" ensuite votre code pour le simplifier, découper… ou plus généralement le rendre plus lisible (cf. le document PFITA et qualité de code disponible sur Moodle, à lire dans les prochains jours si ce n'est pas déjà fait)
3. Solliciter votre encadrant de TP en cas de question, ou juste pour valider certains aspects de votre code (lisibilité et extensibilité/maintenabilité)

## Exercice 1 - Fonctions par cas

**1a)** Écrire la fonction `estZero_v1` qui retourne `"zero"` si son argument est `0` (en utilisant un filtrage, mais en n'effectuant pas de tests supplémentaires).

Que se passe-t-il si l'on applique la fonction `estZero_v1` à `2022` ?

Pour expérimenter cela, vous pouvez évaluer votre code et utiliser l'onglet *Toplevel* de l'environnement learn-ocaml.

**Remarque :** après chaque clic sur `Compiler`, le warning généré par le code de cette question subsistera pendant le reste du TP (ce qui est fait exprès ici !). D'autre part, notez que le texte des warnings (ou des erreurs de syntaxe/typage) apparaît dans l'éditeur si l'on survole le numéro des lignes en question avec la souris.

**1b)** Écrire la fonction `estZero_v2` qui reprend et complète le code précédent et renvoie `"nonZero"` pour tout autre argument que `0`.

**2)** Écrire la fonction `voyelle` qui indique par un booléen si un caractère est une voyelle (minuscule) ou pas.
Concernant le code à écrire, vous vous efforcerez d'obtenir une implantation la plus concise et lisible possible.

## Exercice 2 - Fonctions manipulant des dates

On représente les jours de la semaine par les chaînes de caractères `"lundi"`, `"mardi"`, `"dimanche"`, etc.

**1)** Écrire la fonction `rang` qui prend en argument un jour de semaine et retourne son rang dans la semaine (lundi est le jour de rang 1 et dimanche est de rang 7 dans la semaine). La fonction est partielle et on peut la compléter en associant l'entier 0 à un jour inconnu :

```
rang "jeudi" => 4
rang "monday" => 0
```

**2)** Écrire la fonction `inf` prenant en arguments 2 jours et retournant `true` si et seulement si le premier jour est la veille du deuxième. Les jours étant ici représentés par de simples chaînes de caractères, votre fonction `inf` devra retourner `false` en cas de jour non valide.

**Conseil :** pour cette fonction, nous vous proposons d'utiliser un filtrage simultané (cf. diapositive 14/17 du CTD2), en utilisant aussi la fonction `rang` (mais il y a plusieurs autres implémentations possibles !-)

**3)** Écrire la fonction `jsem` qui prend en argument un entier de l'intervalle `[1..7]` et retourne la chaîne de caractères représentant le jour de la semaine correspondant. Si l'entier est incorrect la fonction retourne la chaîne `"jour inconnu"` :

```
jsem 4 => "jeudi"
jsem 0 => "jour inconnu"
```

**4)** On veut écrire plusieurs versions de la fonction `jourSucc` qui prend en argument un jour de semaine et retourne son successeur dans la semaine : le successeur de lundi est mardi, celui de dimanche est lundi (en renvoyant `"jour inconnu"` en cas d'erreur) :

- 1ère version `jourSucc1` en utilisant uniquement le filtrage ;
- 2ème version `jourSucc2` en utilisant les fonctions `jsem`, et `rang`, sans utiliser `mod` et sans filtrage (mais `if-then-else` autorisés). On choisira une version qui évite de calculer 2 fois la même expression ;
- 3ème version `jourSucc3` en utilisant les fonctions `jsem`, et `rang` et l'opérateur prédéfini `mod` (et un seul `if-then-else`).

> **Remarque :** même si les tests fonctionnels passent pour ces questions, il se peut que votre version soit sous-optimale ou ne respecte pas pleinement l'énoncé. Faites valider vos réponses par votre encadrant de TP.

**5)** Écrire les 3 mêmes versions `jourPred1`, `jourPred2`, `jourPred3` pour la fonction jourPred retournant le jour précédent d'un jour donné. Vous respecterez les même contraintes que pour `jourSucc1`, `jourSucc2` et `jourSucc3`. On notera que `(-1) mod 7 = -1`.

**6)** Écrire la fonction `bissextile` prenant en argument une année et retournant un booléen indiquant si l'année est bissextile ou non. Une année est bissextile si elle est divisible par 4 et si elle n'est pas divisible par 100 à moins qu'elle soit divisible par 400.

**7)** Écrire la fonction `nbjour` prenant en argument un numéro de mois (entier entre `1` et `12`) et une année, et retournant le nombre de jour de ce mois dans cette année.