Tâche 1 - Extraction d'informations à partir de recettes

Construisez des expressions régulières pour repérer les aliments et les quantités de chaque item du fichier ***data/t1_ingredients.txt***. Compléter la fonction ***get_ingredients*** de la Section 3 de ce *notebook* afin de retourner la quantité et l'ingrédient d'un item. Par exemple,

```
get_ingredients("2 cuillères à café de poudre à pâte")
```

devrait retourner la paire :

```
"2 cuillères à café", "poudre à pâte"
```

Par la suite, présentez à la Section 4 les performances que vous obtenez avec ces expressions régulières sur le fichier ***data/t1_test.json***. Discutez des principales erreurs commises par vos expressions régulières.

Section 1 - Lecture du fichier pour la construction d'expressions régulières

In [31]:

```
ingredients_fn = "./data/t1_ingredients.txt"
```

In [32]:

```
def load_ingredients(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        raw_items = f.readlines()
    ingredients = [x.strip() for x in raw_items]
    return ingredients
```

In [ ]:

```
ingredients = load_ingredients(ingredients_fn)
len(ingredients)
```

In [34]:

```
#ingredients
```

Les résultats attendus des 50 exemples utilisés pour la construction de vos expressions régulières sont:

In [ ]:

```
import json

solution_fn = 'data/t1_ingredients_solution.json'

with open(solution_fn, 'r', encoding='utf-8') as fp:
    solutions = json.load(fp)

len(solutions)
```

In [36]:

```
#solutions
```

Section 2 - Vos expressions régulières

Mettez dans cette section toutes les expressions régulières que vous avez construites à partir du fichier ***data/t1_ingredients.txt***. Vous pouvez ajouter des cellules dans cette section du *notebook* si cela permet de rendre la description plus claire.

In [37]:

```
# Vos expressions régulières
auboutpattern=r"(.*)\s(au goût)"
#Cette regex extrait le texte qui est suivi de l'expression "au goût"

quantitypattern= r"(Q\w+\ssom\w+|[\d½¼¾.,/\s\-]+(?:[.,]\d+)?(?:\s*(?:à)?\s*\d+(?:[.,]\d+)?)?\s*(?:
#Cette regex extrait les quantités d’ingrédients.
#Elle capture :
#les quantités (chiffres, fractions, avec ou sans unité de mesure, mais aussi 2 à 3)
#les unités de mesure (comme tasses, cuillères, gousses, grammes, millilitres etc..)
#des expressions facultatives entre parenthèses exemple: (environ 1 tasse)
#capture le reste sur lequel on va appliquer la prochaine expression régulière et enlève d ou d' de

ingredientpattern=r"([^,]+?)(?= en\s|pour|,|ou\s|\s\w{4,}(ées|és|ée|é|u|us)\b|pelé[es]?|râpé[es]?|
#Cette regex se débarasse des descriptions préparatoires des ingrédients:
#avec des termes se finissant par dans beaucoup de cas par éées|és|ée|é|u|us
#(comme vous pouvez le voir nous avons rajouté pelé et rapé car l'on capture que les mots de plus
#ou avec une préposition comme en, pour, ou, dans, etc.
```

Section 3 - Fonction pour l'extraction des ingrédients

La fonction principale ***get_ingredients*** prend en entrée une ligne de texte et retourne une paires de valeurs: la quantité et l'aliment.

Vous pouvez ajouter autant de sous-fonctions que vous le souhaitez. Il est cependant important de ne pas modifier la signature de la fonction principale afin de faciliter notre travail de correction.

In [38]:

```
import re

def get_ingredients(text):

    # ---- gérer le cas des au goût ---- #
    auboutpattern = re.match(r"(.*)\s(au goût)", text.strip(),re.IGNORECASE)
    pre_quant = ""

    if auboutpattern:
        pre_ingred = auboutpattern.group(1).strip()
        pre_quant = auboutpattern.group(2).strip()
    #----------------------------------- #

    quantitypattern = r"(Q\w+\ssom\w+|[\d½¼¾.,/\s\-]+(?:[.,]\d+)?(?:\s*(?:à)?\s*\d+(?:[.,]\d+)?)?\s
    match = re.match(quantitypattern, text.strip(),re.IGNORECASE)

    # ---- case 1 : if we have an ingredient with a quantity ---- #
    if match:
        quantity = match.group(1).strip()
        ingredient = match.group(2).strip()
        new_ingredient=re.match(r"([^,]+?)(?= en\s|pour|,|ou\s|\s\w{4,}(ées|és|ée|é|u)\b|pelé[es]?
        if pre_quant != "":
            return quantity + " ou au goût", new_ingredient.group(1).strip()
        return quantity, new_ingredient.group(1).strip()

    # ---- case 2 : if we have an ingredient without a quantity ---- #
    else:
        if pre_quant != "":
            return "au goût", pre_ingred

        new_ingredient=re.match(r"([^,]+?)(?= en\s|pour|,|ou\s|\s\w{4,}(ées|és|ée|é|u|us)\b|pelé[es
        return "", new_ingredient.group(1).strip()
```

Vous pouvez mettre ici tout commentaire qui nous aiderait à comprendre votre fonction et pour expliquer votre démarche. De plus, indiquez si vous réussissez à extraire toutes les informations du fichier d'ingrédients. Sinon, donnez des précisions.

Nous avons 3 regex qui sont expliquées en commentaire dans la section 2. Concernant la démarche, nous avons commencé par essayer d'extraire toutes les informations de quantité, une fois celle-ci ayant atteint une performance acceptable nous sommes passés à l'extraction des ingrédients. En dernier lieu nous avons rajouté la regex auboutpattern afin d'améliorer notre performance globale.

La fonction vérifie d’abord si la quantité mentionnée est “au goût”. Si c’est le cas, elle sépare l’ingrédient de cette indication et conserve cette information pour plus tard. Ensuite, la fonction applique une expression régulière pour repérer les quantités (si elle existe, sinon elle renvoie un string vide pour la quantité) ainsi que les ingrédients correspondants. Nous matchons avec re.IGNORECASE afin de gérer tous les problèmes de upper et lower case en une fois et nous enlevons les espaces potentiels de la réponse avec strip().

Concernant les quantités, nous avons réussi à extraire toutes les informations comme vous pouvez l'observer dans la premiere cellule de la section 4.

Concernant les ingrédients, une instance ne fonctionne pas "langoustines surgelées et décongelées" comme vous pouvez le voir dans la deuxième cellule de la section 4. Le mot "surgelées" est considérée dans la partie descriptive se finissant par "ées" de laquelle on se débarasse. Nous avons donc seulement langoustines dans les ingrédients.

Section 4 - Évaluation et analyse de vos résultats

Décrivez ici les résultats obtenus et présentez l'évaluation obtenue sur le fichier de test ***data/t1_test.json***. Présentez des exemples d'erreurs. Vous pouvez ajouter le nombre de cellules que vous souhaitez pour faire votre analyse.

Dans un premier temps, on monte en mémoire les exemples de tests. Vous deviez en avoir 26. Chacun contient le texte de l'ingrédient, la quantité et l'aliment.

In [39]:

```
import json

def load_test_set(filename):
    with open(filename, 'r', encoding='utf-8') as fp:
        tests = json.load(fp)
    return tests
```

In [ ]:

```
test_fn = "./data/t1_test.json"

test_examples = load_test_set(test_fn)
len(test_examples)
```

In [46]:

```
#test_examples
```

Les prochaines cellules présentent l'évaluation (dont le code pour mener l'évaluation) et votre analyse des résutlats.

In [ ]:

```
#test quantity on training set
true=0
false=0
for i in range(50):
    result = get_ingredients(ingredients[i])

    solution1 = solutions[i]['quantity']

    if result[0] == solution1:
        true+=1
    else:
        print(result[0])
        print(solution1)
        false+=1

print("True: ", true)
print("False: ", false)
print("Accuracy quantity: ", true/(true+false))
```

In [ ]:

```
#test ingredient on training set
true=0
false=0
for i in range(50):
    result = get_ingredients(ingredients[i])

    solution2 = solutions[i]['ingredient']

    if result[1] == solution2:
        true+=1
    else:
        print(solution2)
        print(result[1])
        false+=1

print("True: ", true)
print("False: ", false)
print("Accuracy ingredient: ", true/(true+false))
```

In [ ]:

```
#compute overall accuracy on training set
import re
true=0
false=0
for i in range(50):
    result = get_ingredients(ingredients[i])
    solution1 = solutions[i]['quantity']
    solution2 = solutions[i]['ingredient']

    if result[1] == solution2 and result[0] == solution1:
        true+=1
    else:
        print(solution2)
        print(result[1])
        false+=1

print("True: ", true)
print("False: ", false)
print("Accuracy : ", true/(true+false))
```

In [ ]:

```
true = 0
false = 0
for i in range(len(test_examples)):
    result = get_ingredients(test_examples[i]['text'])

    if result[0] == test_examples[i]['quantity'] and result[1] == test_examples[i]['ingredient']:
        true += 1
    else:
        false += 1

        if result[0] != test_examples[i]['quantity']:
            print(f"{false} Quantity Solution".ljust(22)+" vs Result: ",
                  test_examples[i]['quantity'].ljust(40), "\t vs\t", result[0].ljust(40))

        if result[1] != test_examples[i]['ingredient']:
            print(f"{false} Ingredient Solution".ljust(22) + " vs Result: ",
                  test_examples[i]['ingredient'].ljust(40), "\t vs\t", result[1].ljust(40))

print("True: ", true)
print("False: ", false)
print("Accuracy: ", true / len(test_examples))
```

Analyse des performances:

Nous avons atteint une accuracy de 0.98 sur le training set et de 0.93 sur le test set. Vous pouvez observer ci-dessus les exemples qui ne fonctionne pas dans le test set.

La 1 et 2 ont l'air d'être des erreurs encodés dans les tests car d'autres exemples montrent un comportement different pour lesquels notre regex capture bel et bien le bon résultat.

Nous pouvons observer que sur les 9 exemples non trouvés, il y 4 exemples (3,4,7,9) pour qui les ingrédients sont coupés trop tôt car une partie est considérée comme descriptif de préparation comme pour l'exemple "langoustines surgelées" du test set.

Le 5, 6 et 8 sont des cas que nos regex ne sont également pas capables de capturer, ce sont des cas que nous n'avons malheureusement pas pris en compte. Notamment le cas du *facultatif* que l'on a pas vu à l'entrainement.

Section 5 - Section réservée pour notre correction

Ne pas retirer les cellules de cette section.

In [ ]: