LMI 1 - Semestre 2

Université d'Artois
Faculté des Sciences Jean Perrin

ALGO2 – Algorithmique et Programmation en Python 2

Fiche de TP numéro 2

Tests unitaires

Chaque définition de fonction commencera par un commentaire, qui contiendra toujours les mêmes informations.

1. La spécification de la fonction, c'est à dire le type de ses paramètres et son type de retour.
2. Une description textuelle de la fonction
3. Des tests unitaires, qui montrent le fonctionnement attendu de la fonction. Ces tests ont l'avantage d'être exécutables, et permettent de tester si la fonction écrite retourne les résultats attendus sur un certain nombre d'exemples caractéristiques.

```
def f(x):
    '''int -> int
       une fonction qui retourne 2*x+1

       tests unitaires : on simule une conversation avec l'interprete
       python. Ne pas oublier l'espace entre les >>> et l'appel de la
       fonction. Ne pas oublier la ligne vide avant les tests.

       >>> f(0)
       1
       >>> f(1)
       3
       >>> f(-0.5)
       0.0
       >>> f(100)
       201
    '''
    return 2*x+1
```

Pour vérifier si la fonction créée correspond aux tests, vous avez deux solutions :

1. il suffit de lancer dans une console la commande suivante, si la fonction se trouve dans un fichier `intro.py` :
   `python3 -m doctest intro.py -v`
2. en mode interactif, il suffit de taper les trois commandes suivantes pour obtenir le même résultat :

```
>>> import doctest
>>> import intro
>>> doctest.testmod(intro)
```

Vous obtenez alors l'affichage suivant (on a rajouté des commentaires) :

```
Trying:
    f(0)          ## python va évaluer f(0)
Expecting:        ## dans les tests unitaires, il est indiqué que
    1             ## le résultat doit être 1
ok                ## c'est bien le cas, la fonction a retourné 1
Trying:           ## test suivant ...
    f(1)
Expecting:
    3
ok                ## et ainsi de suite pour les autres tests
...
...               ## suivis d'un résumé
1 items had no tests:  ## pas de test unitaire général pour le module intro
    intro
1 items passed all tests:  ## 4 tests unitaires pour la fonction f
   4 tests in intro.f
4 tests in 2 items.
4 passed and 0 failed.     ## Tous les tests sont ok
Test passed.
```

Le contenu de votre réfrigérateur est mémorisé dans une structure du type `TIngredients = dico(string:int)` où la clé est le nom d'un produit alimentaire, et la valeur la quantité de ce produit.

Une recette de cuisine est une instance de la classe `Recette` qui contient deux attributs : le nom de la recette et les ingrédients de la recette (on ne s'intéresse ici qu'aux ingrédients nécessaires à la réalisation d'une recette).

Une collection de recettes est une liste d'éléments de type Recette.

Exemples :

```
# TIngredients : dico(str,int)
# On supposera les variables suivantes définies pour les exemples
>>> ingrGateauChoc = {'oeufs':4,'sucre':150,'farine':80,\
                      'beurre':200,'chocolat':200}
>>> ingrQuatreQuarts = {'oeufs':4,'sucre':250,'farine':250,'beurre':250}
>>> lesRecettes = ['omelette':{ 'oeufs': 4, 'lait (en cl)': 5},\
    'soupe':{'poireau': 4, 'pommes de terre': 2},\
    'fondant au chocolat':ingrGateauChoc,\
    'quatre-quarts':ingrQuatreQuarts]
```

Exercice 1 : Spécifiez puis écrivez une fonction `liste_ingredients(nom_recette, lesRecettes)` qui retourne la liste des ingrédients de la recette (on veut uniquement le nom des ingrédients).

Exercice 2 : Spécifiez puis écrivez une fonction`contient(ingr,recette,lesRecettes)` qui dit si un ingrédient fait partie de la recette

```
>>> contient('oeufs','omelette',lesRecettes)
True
>>> contient('basilic','omelette',lesRecettes)
False
```

Exercice 3 : Spécifiez puis écrivez une fonction `quantite(ingr,recette,lesRecettes)` qui retourne la quantité nécessaire pour un ingrédient (0 si l'ingrédient n'est pas nécessité par la recette) ;

Les exercices suivants, sauf mention contraire, concernent des fonctions qui manipulent des `Recette`.

Exercice 4 : Spécifier puis écrire la fonction `recette_possible(frigo,uneRecette)` qui prend en paramètre un élément de type `TIngredients` pour le `frigo` et une instance de `Recette` et qui retourne vrai si la recette peut être faite à partir du contenu du réfrigérateur.

```
>recette_possible({'oeufs':3, 'tomates': 4},lesRecettes,'omelette')
False
```

Exercice 5 : Spécifier puis écrire la fonction `ajoute_courses(frigo,courses)` qui ajoute dans le réfrigérateur les courses qui viennent d'être faites. Les courses sont aussi de type `TIngredients`.

```
>ajoute_courses({'oeufs':3, 'tomates':4},{'oeufs':2, 'melon':1})
{'oeufs':5, 'tomates':4, 'melon':1}
```

Exercice 6 : Spécifier puis écrire la fonction `recettes_possibles(frigo,lesRecettes)` qui retourne la liste des noms des recettes qui peuvent être faites à partir du contenu du réfrigérateur. Chaque recette de la liste retournée doit pouvoir être faite, mais on ne peut pas forcément faire toutes les recettes de la liste.

```
>recettes_possibles({'oeufs':4, 'sucre':350, 'farine':300,
'chocolat':200, 'beurre':250}, lesRecettes)
['fondant au chocolat','quatre-quarts']
```

Exercice 7 : Spécifier puis écrire la fonction `cuisine_recette(frigo,uneRecette)` qui retourne le contenu du réfrigérateur après avoir fait `uneRecette`. Attention, si vous prenez complètement un ingrédient, il ne doit plus apparaître dans le résultat (pas de quantité nulle).

```
>cuisineRecette({'oeufs':4, 'sucre':350, 'farine':300,
'chocolat':200, 'beurre':250}, 'fondant au chocolat', lesRecettes)
{'sucre':200,'farine':220,'beurre':50}
```

Exercice 8 : Spécifier puis écrire la fonction `cuisine_les_recettes(frigo,lesRecettes)` qui retourne le contenu du réfrigérateur après avoir fait toutes `lesRecettes`.

Exercice 9 : Spécifier puis écrire la fonction `courses_pour_recette(frigo,uneRecette)` qui, cette fois-ci, retourne la liste des courses à faire pour pouvoir faire une recette.

Exercice 10 : Spécifier puis écrire la fonction `toutes_les_courses(frigo,lesRecettes)` qui retourne la liste des courses nécessaires pour faire toutes les recettes de `lesRecettes` (de type `TRecettes`).

Un repas est maintenant représenté par une liste de noms de recettes.

```
# TRepas : liste de noms de recettes
# TRepas : list(str)
```

Exercice 11 : Spécifier puis écrire la fonction `les_ingredients` qui, pour un repas et un livre de recettes, retourne tous les ingrédients (et leur quantité) nécessaires pour faire ce repas.

```
>>> les_ingredients(['soupe',omelette'],lesRecettes)
{ 'oeufs': 4, 'lait (en cl)': 5, 'poireau': 4, 'pommes de terre': 2}
```

Exercice 12 : On organise un peu mieux les informations, et maintenant les recettes sont associées à des catégories (*recettes végétariennes*, *entrées*, *potages*, *recettes à mijoter*,...). Une recette peut être associée à plusieurs catégories.

Modifiez votre classe `Recette` afin de répondre au cahier des charges suivant ;

— on peut créer une recette en lui associant une liste de catégories. Par défaut, cette liste est vide ;

— on peut obtenir la liste des catégories d'une recette (attention, vous penserez à retourner une copie de la liste)

— on peut savoir si une recette est associée à une certaine catégorie

Exercice 13 : Spécifier puis écrire la fonction `recettes_d_une_categorie` qui, pour une catégorie `categ` et un repas, retourne les recettes du repas qui sont associées à `categ`. Rappel : un repas est représenté par une liste de noms de recettes.

```
>>> recettes_d_une_categorie('recettes végétariennes',['escargots de Bourgogne','so
['soupe']
```

Exercice 14 : On veut maintenant mémoriser les recettes associées à chaque catégorie. On représente cette classification par un dictionnaire dont la clé est le nom d'une catégorie et la valeur la liste des noms de recettes qui entrent dans cette catégorie, ce qui correspond au type suivant :

```
# TClassification : dico(str,liste(str))
# L'exemple suivant servira à illustrer les questions suivantes :
>>> laClassification = {'recettes végétariennes' : ['omelette', 'soupe'],\
        'entrées' : ['soupe'],\
        'desserts' : ['quatre-quarts', 'fondant au chocolat'] }
```

Spécifier puis écrire la fonction `classification_des_recettes` qui, pour une liste de recettes, construit la classification de ces recettes.

Exercice 15 : Spécifier puis écrire la fonction `convient` qui, pour un repas `rep`, une liste de catégories `categs` souhaitées, et des informations sur les catégories des recettes, retourne vrai si `rep` contient au moins une recette pour chaque catégorie de `categs`, faux sinon. Il est possible qu'une même recette corresponde à plusieurs catégories souhaitées, ou que le repas contienne des recettes hors classification.

```
>>> convient(['soupe','carbonnade','tarte au sucre'],
['recettes végétariennes','poissons'],laClassification)
False
>>> convient(['escargots de Bourgogne','soupe','fondant au chocolat'],
['recettes végétariennes','entrées'],laClassification)
True
```