université
PARIS-SACLAY

Programmation en Python
Master 2 Réseaux Télécoms
TP 3

Les tests unitaires pour ce TP se trouvent dans les fichiers `testArguments.py`, `testDecorators.py` et `testLambdas.py`.

1 Arguments des fonctions

Dans le fichier `arguments.py` modifiez les signatures des fonctions (et complétez leur code si nécessaire), en respectant les consignes suivantes :

1. `f1` : Cette fonction doit prendre deux arguments strictement positionnels et un argument positionnel-ou-nommé, dénommé `arg`. La fonction ne doit rien retourner.
2. `f2` : Cette fonction doit prendre quatre arguments strictement positionnels, dont deux derniers ayant les valeurs par défaut 42 et 5 respectivement. La fonction ne doit rien retourner.
3. `f3` : Cette fonction doit prendre quatre arguments dénommés `a1`, `a2`, `a3` et `a4` dont les deux premiers sont positionnels-ou-nommés, et deux autres sont strictement nommés. La fonction ne doit rien retourner.
4. `f4` : Cette fonction doit prendre six arguments. Les deux premiers sont strictement positionnels, les quatre suivant sont dénommés `a1`, `a2`, `a3` et `a4`. Les arguments `a1` et `a2` sont positionnels-ou-nommés, `a3` et `a4` sont strictement nommés. La fonction ne doit rien retourner.
5. `f5` : Cette fonction doit prendre deux arguments strictement positionnels suivis par des arguments positionnels arbitraires. La fonction doit retourner la liste de ces arguments.
6. `f6` : Cette fonction doit prendre deux arguments strictement nommés, de noms `a1` et `a2` ayant les valeurs par défaut 5 et 6 respectivement, suivis par des arguments nommés arbitraires. La fonction doit retourner un dictionnaire mettant en correspondance les noms des arguments et leurs valeurs.

7. `f7` : Cette fonction doit prendre un nombre quelconque d'arguments positionnels suivis par un nombre quelconque d'arguments nommés. Elle retournera un tuple composé des nombres d'éléments de chaque type. Par exemple, `f7(1,2,coucou=3)` retournera `(2,1)`

2 Décorateurs

Dans le fichier `decorateurs.py`, complétez le code des décorateurs `keep_calm` et `insist` . Ces décorateurs sont censés être appliqués à une fonction susceptible de lever une `Exception`. Le décorateur `keep_calm` appelle la fonction décorée et retourne soit la valeur retournée par cette dernière, soit `None` si une `Exception` a été levée. Le décorateur `insist` continue à appeler la fonction décorée tant qu'elle lève une `Exception`, et retourne uniquement la valeur donnée par la fonction décorée dans le cas d'exécution normale. Par exemple, pour le code suivant

```
def bad_function():
    if random()<0.1:
        return 42
    raise Exception('This is kind of normal for this function to fail')

@keep_calm
def calm_function():
    return bad_function()

@insist
def good_function():
    return bad_function()
```

les appels de `calm_function` retourneront `None` dans 90% des cas et la valeur 42 dans 10% des cas, tandis que les appels de `good_function` retourneront 42 dans tous les cas.

3 Expressions λ"

Complétez le code dans le fichier `lambdas.py` :

1. La fonction `is_multiple_of(n)` doit retourner une expression lambda associant à un argument une valeur booléenne. Cette valeur est vraie si et seulement si l'argument de l'expression lambda est un multiple de `n`.
2. La fonction `binary(op)` doit retourner une expression lambda qui accepte deux arguments et retourne la valeur de l'opération binaire sur eux, corres-

pondante au contenu de la chaine de caractères `op` (par exemple, `op` peut être égale à `'+'`, `'-'`, `'=='` etc.). *Indice* : pensez à utiliser la fonction `eval`.