EP type n° 1

Exercice 1 – *(Guidé, programme à trous)* **– La classe** `Chien`

On souhaite dans cet exercice créer une classe Chien ayant deux attributs :

- un nom `nom` de type `str`,
- un poids `poids` de type `float`.

Cette classe possède aussi différentes méthodes décrites ci-dessous (`chien` est un objet de type `Chien`) :

- `chien.donne_nom()` qui renvoie la valeur de l'attribut `nom` ;
- `chien.donne_poids()` qui renvoie la valeur de l'attribut `poids` ;
- `chien.machouille(jouet)` qui renvoie son argument (paramètre), à savoir la chaîne de caractères `jouet`, privée de son dernier caractère ;
- `chien.aboie(nb_fois)` qui renvoie la chaine `'Ouaf' * nb_fois`, où `nb_fois` est un entier passé en argument (paramètre) ;
- `chien.mange(ration)` qui modifie l'attribut `poids` en lui ajoutant la valeur de l'argument `ration` (de type `float`).

Chien
nom : str poids : float
donne_nom() : str donne_poids() : float machouille(jouet : str) : str aboie(nombre : int) : str mange(ration : float) : bool

On ajoute les contraintes suivantes concernant la méthode `mange` :

- on vérifiera que la valeur de `ration` est comprise entre 0 (exclu) et un dixième du poids du chien (inclus) ;
- la méthode renverra `True` si `ration` satisfait ces conditions et que l'attribut `poids` est bien modifié, `False` dans le cas contraire.

Exemples d'utilisation dans la console :

```
>>> medor = Chien('Médor', 12.0)
>>> medor.donne_nom()
'Médor'
>>> medor.donne_poids()
12.0
>>> medor.machouille('bâton')
'bâto'
>>> medor.aboie(3)
'OuafOuafOuaf'
>>> medor.mange(2.0)
False
>>> medor.mange(1.0)
True
```

```
>>> medor.donne_poids()
13.0
>>> medor.mange(1.3)
True
```

Programme à compléter (identique au fichier Python fourni) :

```python
class Chien:
    def __init__(self, nom, poids):
        self.... = nom
        self.... = poids

    def donne_nom(self):
        return self....

    def ...(self):
        return self....

    def machouille(self, jouet):
        resultat = ""
        for i in range(...):
            resultat += jouet[...]
        return ...

    def ...(self, ...):
        ...

    def ...(self, ration):
        if ...:
            ...
            return True
        else:
            return ...


# Tests
medor = Chien('Médor', 12.0)
assert medor.donne_nom() == 'Médor'
assert medor.donne_poids() == 12.0
assert medor.machouille('bâton') == 'bâto'
assert medor.aboie(3) == 'OuafOuafOuaf'
assert not medor.mange(2.0)
assert medor.mange(1.0)
assert medor.donne_poids() == 13.0
assert medor.mange(1.3)
```

Exercice 2 – *(À maîtriser, programme à créer)* **– Recherche des positions d'un élément dans un tableau**

Écrire une fonction `indices` qui prend en paramètres un entier `element` et un tableau `entiers` de nombres entiers et qui renvoie la liste croissante des indices de `element` dans le tableau `entiers`.

Cette liste sera donc vide `[]` si `element` n'apparait pas dans entiers.

/!\ On n'utilisera ni la méthode `index`, ni la méthode `max`.

Exemples d'utilisation dans la console :

```
>>> indices(3, [3, 2, 1, 3, 2, 1])
[0, 3]
>>> indices(4, [1, 2, 3])
[]
>>> indices(10, [2, 10, 3, 10, 4, 10, 5])
[1, 3, 5]
```

Programme à compléter (identique au fichier Python fourni) :

```
def indices(element, entiers):
    ...


# tests

assert indices(3, [3, 2, 1, 3, 2, 1]) == [0, 3]
assert indices(4, [1, 2, 3]) == []
assert indices(1, [1, 1, 1, 1]) == [0, 1, 2, 3]
assert indices(5, [0, 0, 5]) == [2]
```