Conseil : Lorsque cela est possible, parcourir une liste par item avec une boucle `for`plutôt que par index.

Partie 1

Recherche du maximum

1.1 Dans une liste python

Proposer une fonction `recherche_max` qui retourne un tuple constitué :

- de la valeur maximale
- du/des indices dans la liste où se situe la valeur max.

Exemple :

```
>>> indices_maxi([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
(9, [3, 8])
```

1.2 Dans un dictionnaire

Proposer une fonction `max_dico` qui retourne un tuple constitué :

- de la valeur maximale
- de la clé où se situe la valeur max.

Exemple :

```
>>> max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50})
('Ada', 201)
```

Partie 2

Recherche d'un élement

2.1 Dans une liste

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la première occurrence de `elt` dans `tab` si `elt` est dans `tab` et -1 sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
2
```

2.2 Dans un arbre

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `noeud`, un objet noeud qui contient un (sous) arbre, et qui renvoie `True` si `elt` est dans `noeud` et `False` sinon.

```
class Noeud:
    def __init__(self, v, g, d):
        self.valeur = v
        self.gauche = g
        self.droite = d

    def recherche(self, elt):
        # a completer
```

2.3 Dans un arbre binaire de recherche (inspiré de 17.2 2022)

```
class Noeud:
    def __init__(self, v, g, d):
        self.valeur = v
        self.gauche = g
        self.droite = d

    def recherche(self, elt):
        # a completer
```