

Exercice 1

Définissez la fonction `any` qui prend en argument une liste et un prédicat et retourne vrai ssi il existe au moins un élément dans la liste pour lequel ce prédicat est vrai. Votre fonction aura le profil suivant :

```
def any[a](p: a => boolean)(l: List[a]): boolean
```

Définissez ensuite la fonction `every` qui teste si *tous* les éléments d'une liste satisfont un prédicat. (Note : nous vous demandons d'utiliser le filtrage de motifs dans les deux fonctions).

Exercice 2

Les *listes associatives* sont des listes de paires dont le premier élément représente une clef, et le second représente la valeur associée à cette clef. Une clef n'apparaît jamais plus d'une fois dans une telle liste. Par exemple, la liste

```
List(Pair(1965, "Jean"), Pair(1976, "Marcel"), Pair(1982, "Josette"))
```

associe un nom à une année de naissance.

Écrivez une fonction `lookup` qui prend en argument une liste associative et une clef, et qui retourne l'élément associé à cette clef dans la liste. Utilisez la fonction `error` pour signaler une erreur au cas où l'élément n'existe pas. Votre fonction `lookup` aura le profil suivant :

```
def lookup[a,b](lst: List[Pair[a,b]], key: a): b
```

Conseil : utilisez le filtrage de motif vu au cours pour obtenir une solution élégante et concise.

Exercice 3

Les listes associatives de l'exercice précédent sont un exemple de structures de données associatives, qui lient une valeur à une clef. Le désavantage principal de ces listes est que la fonction de recherche (`lookup`) a une complexité moyenne $O(n)$, où n est le nombre d'éléments dans la liste. Comme vous le savez certainement, les arbres binaires

