Nom Prénom :

Eval_1 TNSI partie 1

Programmation Orientée Objet :

Exercice 1

On considère l'extrait de code ci-dessous :

```
bart = Simpson("fils", 12)
homer = Simpson("père", 41)
homer.ronfle()
```

Compléter les phrases ci-dessous :

1. `bart` est une ..... de la ..... `Simpson`.
2. C'est donc un ..... de type .....
3. À la construction, l'..... `homer` possède (au moins) deux ..... qui valent `"père"` et `41`.
   Il possède aussi (au moins) une ..... appelée ..... .

Exercice 2

On s'intéresse aux joueurs du Top14 de rugby.

1. Construire une classe `Joueur` où chaque instance contiendra les attributs `nom`, `club` et `age`.

```
class Joueur :
```

2. Instancier le joueur Nans_DUCUING, joueur de 27 ans évoluant à l'UBB.

```
```

3. Écrire une fonction prenant en paramètres deux joueurs de la classe `Joueur` et renvoyant le nom du joueur le plus âgé, ou celui des deux joueurs dans le cas d'une égalité d'âge.

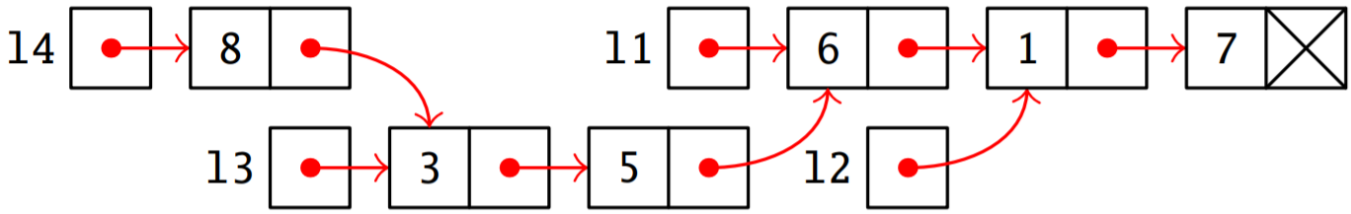
```
def compare(joueur1, joueur2) :
```

Listes chainées :

On considère des listes chaînées, avec la liste vide notée `nil` et les fonctions suivantes :

Fonction	Description
`tete(liste)`	Renvoie la valeur du premier maillon de `liste`, qui ne doit pas être vide.
`queue(liste)`	Renvoie la liste sur laquelle pointe le premier maillon de `liste`, qui ne doit pas être vide.
`cons(valeur, liste)`	Renvoie une nouvelle liste correspondant à l'ajout de `valeur` en début de `liste`.
`est_vide(liste)`	Renvoie un booléen indiquant si `liste` est vide ou non.

Le schéma suivant correspond à la représentation en mémoire des listes chaînées `l1`, `l2`, `l3` et `l4`.



Exercice 1 :

Pour chacune des commandes suivantes, indiquer la réponse obtenue.

1. `tete(l2)` : .....
2. `queue(l1)`: .....
3. `est_vide(queue(l1))`: .....
4. `tete(queue(queue(queue(l4))))`: .....
5. `est_vide(queue(queue(l2)))`: .....

Exercice 2 :

L'instruction `l1 = cons(6, cons(1, cons(7, nil)))` définit la liste `l1`. Donner les définitions des listes `l2`, `l3` et `l4` à l'aide des fonctions `cons` et `queue` et en réutilisant les listes déjà définies pour définir les suivantes :

1. `l2` = ...
2. `l3` = ...
3. `l4` = ...

Exercice 3 :

En utilisant les fonctions données ci-dessus, proposer une version itérative ~~et une version récursive~~ pour la fonction `longueur(liste)` qui renvoie la longueur de la liste chaînée `liste` telle que :

```
>>> longueur(nil)
0
>>> longueur(cons(7, cons(0, cons(12, nil))))
3
```

```
def longueur(liste) : # version itérative
```