Durée du CC : 1h25

L'utilisation de bibliothèques n'est pas autorisée.
Toutes les questions sont indépendantes ; on pourra les traiter dans l'ordre de son choix.

Question 1

1. En utilisant une boucle *for*, stockez dans une variable `d1` le dictionnaire contenant toutes les paires clef-valeur de la forme `n:3*n+1`, pour tous les entiers `n` impairs compris entre 0 et 100.
2. En utilisant une définition par compréhension, définiz un dictionnaire `d2` ayant le même contenu.

Question 2 Écrivez une fonction `premier()` qui attend en argument un entier, et renvoie un booléen indiquant si cet entier est premier.

Pour rappel, un entier est premier s'il est supérieur ou égal à 2, et est divisible uniquement par 1 et par lui-même.

Par exemple, `premier(6)` renverra `False` et `premier(7)` renverra `True`.

Question 3

Construisez une classe `Boucle` dont le constructeur attend un chaîne de caractères `chaine`.
Une `Boucle` doit être itérable. À chaque itération, il faudra renvoyer la prochaine lettre de `chaine`, en repartant du début une fois qu'on est arrivé à la fin.
Le comportement attendu est donc le suivant :

```
boucle = Boucle("ab")
for c in boucle:
    print(c)
# affiche "a" puis "b" puis "a" puis "b" puis "a"...
```

Question 4 Écrivez une fonction `afficheUnique`, qui attend en argument un dictionnaire `dict`, et affiche (dans n'importe quel ordre) les valeurs apparaissant une seule fois (en tant que valeur) dans `dict`.
Le comportement attendu est donc le suivant :

```
dict={1:"foo", "toto":"bar", "bar":"foo", 2:"truc"}
# affiche "bar" et "truc", dans n'importe quel ordre
```

Question 5

1. Construisez une classe `Filtre` dont le constructeur attend
   — une liste `liste`
   — une fonction `condition`, qui renvoie un booléen.
   Un `Filtre` doit être itérable. À chaque itération, il faudra renvoyer le prochain élément de `liste` qui vérifie la condition `condition` (c'est-à-dire pour lequel `condition` renvoie `True`).
   Le comportement attendu est donc le suivant :

```
def pair(n) =
    return n%2==0

filtre = Filtre([1,2,3,4,5,6,7,8],pair)
for n in filtre:
    print(n)
# affiche "2" puis "4" puis "6" puis "8"
```

2. Construisez maintenant une classe `FiltreBis` qui fonctionne comme `Filtre`, si ce n'est que le premier argument du constructeur n'est plus nécessairement une liste, mais n'importe quel itérable.
   Le comportement attendu est donc le suivant :

```
def sup5(n) =
    return n>=5

filtreBis = FiltreBis(filtre,sup5)
for n in filtreBis:
    print(n)
# affiche "6" puis "8"
```