Exercices d'informatique 1
Polynômes

I Avec des listes

On représente le polynôme $P : x \mapsto \sum_{k=0}^{n} a_k x^k$ par la liste `P` de longeur $n+1$ telle que `P[k]` est exactement le coefficient a_k de x^k. Par exemple le polynôme $P = x^3 + 4x^2 - 5x + 8$ sera représenté par

```
P = [8, -5, 4, 1]
```

On prendra garde aux remarques suivantes :

1. Le fait que la liste soit de longueur $n+1$ n'implique pas que P est de degré $n+1$: les coefficients peuvent très bien être nuls. Le polynôme nul est donc aussi bien la liste vide `[]` qu'une liste comme `[0, 0, 0]`. Le polynôme x est `[0, 1]` mais on peut toujours étendre la liste par des zéros.
2. On convient de la notation suivante : les variables `n`, `m` sont utilisées pour représenter non pas les longueurs des listes, mais leur longueur moins un. C'est à dire le degré du polynôme (s'il n'y a pas des zéros en trop). Le coefficient dominant de `P` est alors `P[n]` et pour parcourir toute la liste il faut une boucle avec `range(n+1)` ; rappelons que sinon, pour une liste `L` de longueur `n`, le dernier élément est `L[n-1]` et qu'elle se parcourt en entier avec `range(n)`.
3. Les fonctions ne doivent pas modifier leur argument, mais renvoyer une nouvelle liste. Ainsi elles commencent en général par calculer le degré `r` du résultat puis initaliser une nouvelle liste `R` (résultat) de taille `r+1` avec la syntaxe

   ```
   R = [0 for _ in range(r+1)]
   ```

Exercice 1. Écrire les fonctions suivantes :

1. `degre(P)` : donne le degré de P.
   On pourra retourner -1 pour le polynôme nul, et si on s'y prend bien c'est en fait naturel et pas un cas à traiter à part.
2. `produit_constante(P, a)` : polynôme aP.
3. `produit(P, Q)` : produit $P \times Q$.
4. `somme(P, Q)` : somme $P + Q$.
   Attention car il faut gérér le cas où les deux listes ne sont pas de la même longeur !
5. `puissance(P, m)` : puissance P^m, itérativement ou bien récursivement.
6. `evalue(P, a)` : prend un réel a et calcule $P(a)$.
7. `compose(P, Q)` : composition $P \circ Q$.
8. `derive(P)` : dérivée P'
9. `nderive(P, r)` : dérivée r-ième $P^{(r)}$, itérativement ou bien récursivement.

Exercice 2. Écrire les fonctions suivantes :

1. `est_racine(P, a)` : renvoie `True` si a est une racine de P et `False` sinon.
2. `divise(P, a)` : en supposant que a est racine de P, renvoie le polynôme Q tel que $P(x) = (x-a)Q(x)$.
   On pourra s'inspirer de la preuve du cours et avec la fonction `compose`.
3. `multiplicite_racine(P, a)` : renvoie la multiplicité de a comme racine de P.
   On pourra s'inspirer de la preuve du cours sur la factorisation par $(x-a)^m$. On rappelle que a est racine de multiplicité 0 si et seulement si a n'est pas racine de P, si on s'y prend bien ce cas n'est donc pas à traiter à part.
4. `racine_evidente(P)` : cherche une racine évidente de P parmi les nombres entiers dans l'intervalle $[-5, 5]$ et renvoie une solution si elle en trouve une. Sinon, la fonction renvoie `None`.
   On pourra aussi agrandir l'intervalle de recherche.
5. `racines(P)` : en cherchant des solutions évidentes, donne la liste de toutes les racines de P (où une racine multiple doit apparaitre plusieurs fois).
   On pourra suivre la méthode habituelle en factorisant à chaque fois. La fonction s'écrit aussi bien récursivement. À la fin on obtient aussi un quotient qu'on ne peut plus factoriser.

II Avec des dictionnaires

On s'intéresse maintenant aux **polynômes creux**. Ce sont des polynômes de degré très grand, mais ne contenant qu'un petit nombre de coefficients non-nuls. C'est le cas par exemple de $P : x \mapsto x^{100} + 5x^2 - 3$. Pour le représenter avec la méthode précédente, il faudrait une très grande liste

```
P = [3, 0, 5, 0, ..., 0, 1]
```

de longueur 101, mais avec essentiellement des zéros.
À la place, on représente le polynôme P par un dictionnaire `P`, où les clés sont les degrés des monômes présents et où `P[i]` désigne simplement le coefficient de x^i dans P. L'exemple précédent est donc représenté par

```
P = {0: -3, 2: 5, 100: 1}
```

Le monôme ax^n est tout simplement `{n: a}` et le polynôme nul est le dictionnaire vide `{}`.
Le but est de ré-écrire toutes les fonctions précédentes dans ce contexte. Quelques remarques :

1. L'ordre des clés dans le dictionnaire n'a aucune importance.
2. Les fonctions n'ont pas besoin de connaitre à l'avance le degré des polynômes ni le nombre de clés. Elles peuvent très bien manipuler des monômes « en vrac ».
3. Il faut toujours faire attention à ce qu'on ne peut pas accéder à `P[i]` si la clé `i` n'est pas déjà présente dans le dictionnaire. Dans de nombreux cas il faut distinguer les cas selon que la clé `i` est bien ans le dictionnaire (alors, l'utiliser), sinon considérer que le coefficient correspondant est 0.
4. Il faut faire attention à ce qu'on ne suppose pas nécessairement que tous les coefficients présents sont non-nuls. Par exemple le polynôme nul peut être donné par `{1: 0, 5: 0, 10: 0}`. Cela est en fait pratique car lors d'une somme peuvent naturellement apparaitre des coefficients qui s'annulent et donc il est plus simple de les laisser. Alternativement, on peut écrire une fonction qui nettoie le polynôme (supprime toutes les clés dont les coefficients sont nuls) et alors toutes les fonctions de calcul doivent nettoyer le polynôme avant de le renvoyer.
5. Comme précédemment, les fonctions ne doivent jamais modifier leur argument mais elles renvoient un nouveau dictionnaire. Elles démarrent donc souvent en initialisant un dictionnaire vide `R = {}`.
6. Les fonctions telles que `puissance` ou `nderive` font uniquement appel aux autres fonctions déjà écrites (respectivement `produit`, `derive`), on n'a donc pas spécialement besoin de les ré-écrire dans le contexte des polynômes creux.

Exercice 3. Écrire une fonction `nettoyer(P)` qui supprime de `P` les clés correspodant à des coefficients nuls.

Exercice 4. Ré-écrire les fonctions précédentes pour les polynômes creux, notamment : `degre`, `produit_constante`, `produit`, `somme`, `evalue`, `derive`.