Exercice 15 Certaines réalisations des ensembles d'entiers vues dans ce chapitre représentent plus précisément des ensembles d'entiers compris entre 0 et un certain entier maximum n_{max}. Lesquelles ? Comment était décidé n_{max} ? Que changer à l'interface de ces programmes pour permettre à l'utilisateur de choisir n_{max} ?

Solution page 431 □

Exercice 16 Les modules décrits par chacun des programmes 6, 9 et 10 réalisent tous l'interface 1 et peuvent donc tous être utilisés par le programme 8. Pour autant, l'un est-il préférable aux autres pour cette application particulière ?

Solution page 431 □

Exercice 17 Considérons une structure de table de hachage telle que réalisée par le programme 10 mais avec seulement 8 paquets. On place dedans les entiers 0, 1, 4, 9, 13, 24 et 30. Donner le contenu de chacun des paquets.

Solution page 431 □

Exercice 18 L'interface des tableaux de Python fournit de nombreuses opérations à l'allure anodine mais cachant une complexité non négligeable. Rien de tel que d'essayer de réaliser soi-même ces fonctions pour s'en rendre compte. Écrire un module réalisant l'interface suivante

fonction	description
`tranche(`t`, `i`, `j`)`	renvoie un nouveau tableau contenant les éléments de t de l'indice i inclus à l'indice j exclu (et le tableau vide si $j \leq i$)
`concatenation(`t_1`, `t_2`)`	renvoie un nouveau tableau contenant, dans l'ordre, les éléments de t_1 puis les éléments de t_2

sans utiliser les opérations + et t`[`i`:`j`]` du langage. Notez qu'aucune de ces fonctions ne doit modifier les tableaux passés en paramètres.

Solution page 431 □

Exercice 19 Les tableaux de Python sont redimensionnables : leur nombre d'éléments peut augmenter au fil du temps, par exemple avec des opérations comme **append**. L'objectif de cet exercice est de définir un module réalisant une interface de tableau redimensionnable, mais sans utiliser les capacités de redimensionnement natives des tableaux Python.

Voici une interface minimale pour une structure de tableau redimensionnable.

fonction	description
`cree()`	crée et renvoie un tableau vide
`lit(`*tr*`, `*i*`)`	renvoie l'élément de *tr* à l'indice *i*
`ecrit(`*tr*`, `*i*`, `*x*`)`	place la valeur *x* dans la case d'indice *i* du tableau *tr*
`ajoute(`*tr*`, `*x*`)`	ajoute le nouvel élément *x* au tableau *tr*, après ses éléments actuels

Notez que ces opérations sont équivalentes aux opérations `[]`, *tr*`[`*i*`]`, *tr*`[`*i*`]` = *x* et *tr*`.append(`*x*`)` des tableaux de Python.

Pour réaliser cette interface, on va représenter un tableau redimensionnable *tr* de *n* éléments par une paire nommée contenant d'une part ce nombre *n* et d'autre part un tableau Python *t* dont la longueur est supérieure ou égale à *n*. Le nombre *n* sera appelé la *taille* de *tr* et la longueur de *t* sa *capacité*. Les *n* éléments sont stockés dans les cases d'indices 0 à $n-1$ du tableau *t*, tandis que les cases suivantes contiennent `None`.

1. Écrire une fonction `cree()` créant et renvoyant un tableau redimensionnable de taille 0 et de capacité 8.
2. Écrire deux fonctions `lit(`*tr*`, `*i*`)` et `ecrit(`*tr*`, `*i*`, `*x*`)` telles que décrites dans l'interface, en supposant que l'indice *i* fourni est bien compris entre 0 inclus et la taille de *tr* exclue.
3. Fonction `ajoute`.
   (a) Écrire une fonction `_ajoute_aux(`*tr*`, `*x*`)` qui ajoute l'élément *x* à la fin du tableau redimensionnable *tr*, en supposant que la capacité de ce dernier est suffisante.
   (b) Écrire une fonction `_double(`*tr*`)` qui double la capacité du tableau redimensionnable *tr*, en conservant ses éléments.
   (c) En déduire une fonction `ajoute(`*tr*`, `*x*`)` telle que décrite dans l'interface. Cette fonction doit doubler la capacité du tableau redimensionnable lorsqu'il ne peut accueillir de nouvel élément.

Autres opérations qui pourraient être ajoutées : ***pop, del, insert, extend.***

Solution page 432 □

Exercice 20 Voici une interface minimale pour une structure de dictionnaire.

fonction	description
`cree()`	crée et renvoie un dictionnaire vide
`cle(`*d*`, `*k*`)`	renvoie `True` si et seulement si le dictionnaire *d* contient la clé *k*
`lit(`*d*`, `*k*`)`	renvoie la valeur associée à la clé *k* dans le dictionnaire *d*, et `None` si la clé *k* n'apparaît pas
`ecrit(`*d*`, `*k*`, `*v*`)`	ajoute au dictionnaire *d* l'association entre la clé *k* et la valeur *v*, en remplaçant une éventuelle association déjà présente pour *k*