Correction du DS 2

Informatique pour tous, première année

Julien Reichert

Exercice 1

1. C'est bien entendu la réponse c) : 2^{32}, chaque séquence de 32 bits code bien un nombre.

2. En complément à deux sur 16 bits, les nombres de -2^{15} à $2^{15}-1$ sont représentés. Tout autre nombre sera représenté modulo 2^{16}, en particulier la représentation de 50000 sera celle de -15636, qui est négatif, d'où le 1 du début, et pair, d'où le 0 de la fin. Le passage à l'opposé nécessite de changer tous les bits puis d'additionner 1, donc -50000 n'a pas la même représentation à un bit près. En l'occurrence, la réponse g) est aussi exacte (en plus de b), c) et f), donc), et peut s'obtenir en convertissant 15636 en binaire puis en utilisant l'algorithme d'obtention de l'opposé, ou en convertissant 50000 en binaire puis en déclarant que c'est la représentation naturelle de -15636.

3. Il est tout à fait possible d'accéder à un élément d'une séquence fournie explicitement, même si on trouve plus souvent des accès à un élément d'une séquence en tant que variable. C'est simplement dû au fait que les constantes sont rares en pratique. L'indice `-3` est le troisième en partant de la fin, mais le dernier ne peut évidemment pas être `-0`, donc la réponse est d) : le nombre `7`.

4. Les entiers n'ont pas de longueur, la réponse `a)` est donc fausse. Les chaînes de caractères en ont une, la réponse `b)` est donc vraie. Quant aux listes, leur longueur est définie comme le nombre d'éléments sans considérer leur valeur, donc la liste `c)` est de longueur un, alors que la liste `d)` est de longueur sept. De même pour la liste `e)` qui est en fait `['a','b','c','d','e','f','g']`. Enfin, un objet `range` a certes une longueur, mais les éléments de `range(1,7)` sont les nombres de `1` à `6`.

5. La liste `l1` est composée de 4 éléments : les listes `[1]`, `[2,2]`, `[3,3,3]` et `[4,4,4,4]`. Ceci est dû à l'utilisation de la méthode `append` qui n'ajoute qu'un élément en fin de liste, quel que soit son type (en l'occurrence, ici, une liste). Au contraire, la liste `l2` est effectivement « plate » et de longueur 10. La liste `l3` est composée de 10 listes (type des éléments ajoutés par `append`) toutes égales à `[42]`. En effet, quand `i3` est pair, cette liste est ajoutée deux fois, et quand `i3` est impair, elle est ajoutée trois fois.

La liste `l4` est composée des éléments de la liste des quarante-deux premiers entiers naturels aux positions de `2` (incluse) à `22` (exclue) par pas de `2`. Il y en a donc effectivement dix. La liste `l5` est composée des éléments de `l4` du début au dernier exclu, en ajoutant au début un deuxième `0`. La taille reste de dix. La liste `l6` est composée des éléments de `l5` entre la position `1` (incluse) et la fin. Le premier ayant été écarté, il n'en reste que neuf.

Exercice 2

Le script (a) ne provoque pas d'erreur et imprime les entiers de `0` à `41` sur des lignes successives de la console.

Le script (b) ne provoque pas d'erreur mais il n'imprime rien car l'objet `range` est vide (le premier paramètre de la fonction qui l'engendre est supérieur au deuxième sans qu'un troisième paramètre strictement négatif ne soit fourni).

Le script (c) provoque une erreur lors de l'exécution du script. En effet, la ligne `return n`, n'étant pas indentée, n'est pas dans la définition de la fonction, et un `return` en-dehors d'une fonction cause une erreur.

Le script (d) provoque une erreur lors de l'appel de la fonction. En effet, le paramètre `l` est une liste, et il est interdit d'appeler la fonction `range` sur autre chose qu'un entier.

Le script (e) ne provoque pas d'erreur à proprement parler, mais il déclenche une boucle infinie imprimant un nombre illimité de lignes contenant `42`. En effet, la ligne `n-1` ne provoque pas de modification de la variable `n`, ce qui fait que le test d'entrée dans la boucle est toujours vérifié.

Le script (f) ne provoque pas d'erreur, malgré le test qui devrait faire faire une division par zéro. En fait, le premier tour de la boucle déclenche la fin de la fonction en raison du `return`, et seul `0` sera donc retourné, et éventuellement imprimé dans la console si c'est là que l'appel avait été effectué.

Exercice 3

```
def minmax(l):
    mini = l[0]
    maxi = l[0]
    for i in range(len(l)):
        if l[i] > maxi:
            maxi = l[i]
        if l[i] < mini:
            mini = l[i]
    return (mini, maxi)
```

Une version optimisée organise les éléments de `l` par paires et fait moins de tests, mais il est encore trop tôt pour en parler.

Exercice 4

```
def nombre_de_pairs(l):
    s = 0
    for element in l:
        if element % 2 == 0:
            s = s + 1
    return s
```