Structures de Données Simples 2017 (CSV'17)

TD 1 : Fonctions et types composés énumérables

Pour chaque exercice programmer une ou plusieurs fonctions et plusieurs tests unitaires.

Exercice 1. Implémenter les fonctions suivantes :

$$
\begin{aligned}
&f(n) = n^2 + n + 1,\\
&g(n) = f(0) + f(1) + \ldots + f(n),\\
&h(n,m) = \begin{cases} f\left(\frac{m}{2} + n\right) & \text{si } m \text{ est divisible par } 2,\\ g\left(\frac{m+1}{2} + 2n\right) & \text{sinon.}\end{cases}
\end{aligned}
$$

Exercice 2. Implémenter la fonction récursive `fib` pour calculer les nombres de fibonacci définis avec la formule suivante :

$$
\begin{aligned}
&\texttt{fib}(0) = 0,\\
&\texttt{fib}(1) = 1,\\
&\texttt{fib}(n) = \texttt{fib}(n-2) + \texttt{fib}(n-1) \quad \text{si } n \geq 2.
\end{aligned}
$$

Observer le temps necessaire pour calculer les valeurs $\texttt{fib}(n)$ pour les valeurs n consécutives.

Exercice 3. Implémenter la fonction `fib` d'une manière plus efficace en utilisant la technique de mémorisation (stockage des valeurs ultérieurement calculées dans une liste). Observer le temps d'évaluation.

Ensuite, implémenter la fonction suivante qui retourne des nombres réels :

$$\texttt{phi}(n) = \frac{\texttt{fib}(n)}{\texttt{fib}(n-1)}, \quad n > 0.$$

Observer la convergence des les valeurs de $\texttt{phi}(n)$ pour les valeurs n croissants (une convergence vers le nombre d'or $\phi = \frac{1\sqrt{5}}{2}$).

Exercice 4. Implémenter la fonction `fib` avec la formule close

$$\texttt{fib}(n) = \frac{\phi^n - \psi^n}{\phi - \psi},$$

où $\phi = \frac{1+\sqrt{5}}{2}$ et $\psi = 1 - \phi$.

Exercice 5. Implémenter une fonction `sum` qui calcule la somme des valeurs de la collection donnée sur l'entre.

Exercice 6. Implémenter une fonction `overlapping` qui prend deux séquences et retourne `True` si les deux listes ont un élément en commun, et `False` sinon.

Exercice 7. Implémenter une fonction `growing` qui prend une séquence de numéros vérifie que tout élément et supérieur ou égal à tout élément précédent.

Exercice 8. Implémenter une fonction `telescope` qui pour une séquence donnée $(a_1, \ldots, a_n)$ calcule sa somme télescopique : $a_1 - a_2 + a_3 - a_4 + \ldots \pm a_n$.

Exercice 9. Implémenter une fonction `palindrome` qui vérifie qu'une chaîne de caractères donnée est bien un palindrome. Un palindrome est un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche

à droite ou de droite à gauche. Par exemple, `'radar'`, `'anna'`, `'mon nom'`, mais aussi `'Un roc cornu'` et `'Engage le jeu que je le gagne'` (alors on ignore les espaces).

Exercice 10. Implémenter une fonction `balanced` qui vérifie qu'une chaîne de caractères donnée est bien imbriquée. Une chaîne de caractères est bien imbriquée si toute parenthèse ouvrante a une parenthèse fermante correspondante qui la suit. Par exemple, `'((()())())'` est bien imbriquée mais `'((()()))()('` ne l'est pas.

Exercice 11*. Implémenter une fonction `well_formed` qui vérifie qu'une séquence de balises HTML est bien formée. Une balise HTML ouvrante a la forme suivante <*nom*> et une balise HTML fermante a la forme </*nom*>. Une séquence de balises est bien formée si toute balise ouvrante a une balise fermante correspondante (avec le même nom) et deux paires de balises correspondantes ne se chevauchent jamais. Par exemple, la sequence suivante est bien formée

```
['<a>', '<b>', '</b>', '<c>', '<b>', '</b>', '</c>', '<a>', '</a>', '</a>']
```

mais pas celle-ci :

```
['<a>', '<b>', '<c>', '</b>', '</c>', '<a>', '</b>', '</a>']
```

Exercice 12.** Implémenter une fonction `inverse_polish` qui prends une expression arithmétique postfixée (connue aussi comme la notation polonaise inversée) et l'évalue. Une expression postfixée utilise les opérateurs arithmétiques binaires (+, −, ∗ et /) après les arguments. Par exemple `3 4 +` s'évalue à 7, `2 2 *` à 4, `3 2 2 * +` à 7 et `3 4 + 2 1 + * 6 /` devient 3.5.