# 4TPM205U: Algorithmique des tableaux: feuille 2

## Itérer avec la boucle `while`

## Travaux dirigés sur machine

Télécharger le fichier `http://dept-info.labri.fr/ENSEIGNEMENT/algotab/src/tp02.py`
Il contient des fonctions utilisées pour les tests. Vous pouvez y ajouter les fonctions demandées.

**Exercice 1**

1. Tester la fonction `sommeChiffres` vue en TD.
2. Écrire une fonction `sommeChiffresRangPair` qui retourne la somme des chiffres de rang pair d'un nombre entier `n` passé en paramètre (le chiffre le plus à droite est considéré comme étant de rang 0). Par exemple, si `n` vaut 5741, la fonction retournera 8, c'est-à-dire $7+1$.
3. Écrire une fonction qui retourne la somme des chiffres de rang impair d'un nombre `n` passé en paramètre. Cette fonction appelera la fonction `sommeChiffresRangPair`. Tester ces deux dernières fonctions en exécutant la fonction fournie dans le fichier `tp02.py`.

**Exercice 2** – Écrire et tester les fonctions du TD simulant des expériences de lancers de dés : `nbLancers6()`, `nbLancersDouble()`, `moyenneTentativesDouble(n)` et `nbLancersPourSomme(n, valeur)`.

La fonction `lancerDe()`, qui retourne aléatoirement un entier entre 1 et 6, est fournie dans le fichier `tp02.py`. Elle utilise une fonction du module `random` de `python`.

**Exercice 3** – Un nombre entier $n$ est divisible par 11 si les sommes de ses chiffres de rang pair et de rang impair sont égales modulo 11. Par exemple, si $n$ vaut 387958948949, votre fonction retournera `True` car les sommes de ses chiffres de rang pair et impair font respectivement 47 et 36, et $(47 \bmod 11) = 3 = (36 \bmod 11)$.

Utiliser cette propriété pour écrire une fonction booléenne `multipleDeOnze(n)` qui, sans utiliser l'opérateur `%` ni la fonction `reste`, retourne `True` si `n` est un multiple de 11 et `False` sinon. Pour ce faire vous tirerez partie de la propriété mathématique $(a \bmod k) = (b \bmod k)$ si et seulement si $a - b$ est divisible par $k$.

**Exercice 4**

1. Écrire une fonction `racineEntiere` implémentant l'algorithme demandé dans la feuille de TD#2.
2. Utiliser cette fonction pour écrire une fonction `racineApprochee(n,d)` affichant la valeur approchée par défaut à $10^{-d}$ prés de $\sqrt{n}$. Par exemple, `racineApprochee(2,1)` affichera 1.4 et `racineApprochee(2,3)` affichera 1.414.

   Exécuter la fonction `testeRacineCarree` fournie dans le fichier `tp02.py`.

**Exercice complémentaire 1 – *(Multiplication alexandrine)***

Pour multiplier deux entiers naturels $x$ et $y$, on répète, tant que $y$ n'est pas nul, la double opération suivante : *multiplier* $x$ *par* $2$ *et diviser* $y$ *par* $2$ *(quotient entier)*. Le résultat est égal à la somme des multiples de $x$ correspondant à des quotients de $y$ impairs (le couple initial compris).

Cela revient à déterminer la suite :

$$\begin{cases} (x_0, y_0) = (x, y) \\ (x_{n+1}, y_{n+1}) = (2x_n, \lfloor \frac{y_n}{2} \rfloor) \end{cases}$$

où $\lfloor \frac{a}{b} \rfloor$ dénote mathématiquement la division entière de $a$ par $b$.

Le produit $xy$ est égal à la somme des $x_n$ pour lesquels $y_n$ est impair.

1. Faire tourner l'algorithme pour calculer $7 \times 9$ (donner le tableau des valeurs successives de $n$, $x_n$ et $y_n$).
2. Écrire une fonction `multiplicationAlexandrine` implémentant cet algorithme.
   Déterminer, en fonction de $x$ et de $y$, le nombre d'opérations élémentaires effectuées (multiplications par $2$, divisions par $2$, comparaisons, additions).
3. Sur le modèle de la multiplication alexandrine, imaginer un algorithme permettant de calculer $x^n$.

**Exercice complémentaire 2 – *(Convergence de l'approximation de la racine carrée)***

L'exercice 6 de la feuille de TD machine de la première semaine demande d'écrire la fonction `sqrt(x)` (qui retourne une approximation de $\sqrt{x}$) et de la tester avec les valeurs $x = \{2, 3, 4, 10\}$ et $n = 10$.

La tester maintenant avec $x = 1000$.

On constate que la convergence est lente, on n'est pas sûr que $10$ itérations suffisent pour obtenir une bonne approximation de la racine. Remplacer la boucle `for` par une boucle `while` pour continuer le calcul tant que la différence entre deux termes consécutifs est plus grande que $0.0000001$. Tester avec $x = 100000$.