4TPM205U: Array Algorithms : Sheet 2

Iterate using a `while` loop

Travaux dirigés sur machine

Download the file `http://dept-info.labri.fr/ENSEIGNEMENT/algotab/src/tp02.py`
It has the functions used for the tests. You can add the functions asked here to the same file.

Exercice 1

1. Test the function `sommeChiffres` discussed in TD.
2. Write a function `sommeChiffresRangPair` which returns the sum of the digits in the even positions of an integer `n` passed as an argument (the right most digit is considered to be at position 0). For example, if `n` is 5741, the function returns 8, which is $7 + 1$.
3. Write a function which returns the sum of digits in the odd positions of an integer `n` passed as an argument. This function should call the function `sommeChiffresRangPair`. Test these last two functions by executing the function provided in the file `tp02.py`.

Exercice 2 – Write and test the functions discussed in the TD, simulating the rolling of dices : `nbLancers6()`, `nbLancersDouble()`, `moyenneTentativesDouble(n)` and `nbLancersPourSomme(n, value)`.

The function `lancerDe()`, which returns randomly an integer between 1 and 6, is provided in the file `tp02.py`. It uses a function belonging to the module `random` of `python`.

Exercice 3 – An integer n is divisible by 11 if the sum of its digits in the even positions and the sum of the digits in the odd positions leave the same remainder when divided by 11. For example, if n is 387958948949, your function should return `True` because the sum of its digits in the even positions and in the odd positions make respectively 47 and 36, and $(47 \bmod 11) = 3 = (36 \bmod 11)$.

Use this property to write a boolean function `multipleDeOnze(n)` which, without using the operator `%` nor the function `reste`, returns `True` if `n` is a multiple of 11 and `False` otherwise. To do this, note the mathematical property $(a \bmod k) = (b \bmod k)$ if and only if $a - b$ is divisible by k.

Exercice 4

1. Write a function `racineEntiere` implementing the algorithm discussed in the sheet TD#2.
2. Use this function to write a function `racineApprochee(n,d)` returns the value approximated to 10^{-d} closer to $\sqrt{n}$. For example, `racineApprochee(2,1)` returns 1.4 and `racineApprochee(2,3)` returns 1.414.

   Execute the function `testeRacineCarree` provided in the file `tp02.py`.

Exercice complémentaire 1 – *(Multiplication alexandrine)*

To multiply two natural integers x and y, we repeat, as long as y is not zero, the following double operations : *multiplu* x *by* 2 *and divide* y *by* 2 *(integer quotient)*. the result is equal to the sum of the multiples of x corresponding to the odd quotients of y (the initial tuple included).

This determines the following :

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

where $\lfloor \frac{a}{b} \rfloor$ denotes mathematically the integer division of a by b.

the product xy is equal to the sum of x_n where y_n is odd.

1. Run the algorithm to calculate 7×9 (given the table of succesive values of n, x_n and y_n).
2. Write a function `multiplicationAlexandrine` implementing this algorithm.
   Determine, as a function of x and of y, the number of basic operations peformed (multiplications by 2, divisions by 2, comparisons, additions).
3. In terms of the alexandrine multiplication model, imagine an algorithm that calculates x^n.

Exercice complémentaire 2 – *(Convergence of the approximation of square root)*

Exercise 6 of the TD machine machine of the first week asks to write the function `sqrt(x)` (which returns an approximation of $\sqrt{x}$) and test it with the values $x = \{2, 3, 4, 10\}$ and $n = 10$.

Test it now with $x = 1000$.

We see that the convergence is slow, we are not sure that 10 iterations are enough to get a good approximation of the square root. Replace the loop `for` with a loop `while`, continuing the calculations as long as the difference between two consecutive terms is greater than 0.0000001. Test it with $x = 100{,}000$.