Boucle Tant que

Nous avons commencé à écrire un **algorithme** pour convertir un nombre entier en binaire

Dans cet algorithme certains passages compréhensibles par un humain doivent être précisées

Par exemple *Extraire la plus grande puissance de 2 contenue dans l'entier N*

Une des briques des algorithmes nous permet de résoudre ce problème, c'est la boucle Tant que

1 Boucle Tant que

La **donnée** de notre problème est un entier naturel `N` par exemple 124

Le **résultat** de notre algorithme est la plus grande puissance de 2 contenue dans `N` ici 64

Comment faire ça ?

On va initialiser une variable `P` à 1 et **Tant que** `P` reste inférieur ou égal à `N` on va exécuter `P` $\leftarrow$ `P` *2

On va mettre en forme cet algorithme

Algorithme 1 : La plus grande puissance de 2 contenue dans un entier

Données : un entier naturel `N`
Résultat : la plus grande puissance `P` de 2 contenue dans `N`

```
1 début
2 |  P ← 1
3 |  tant que  P <= N faire
4 |  |  P ← P *2
5 |  fin
6 |  P ← P //2
7 fin
```

Commentaires

1. On dit que `P` est **variant de boucle** car à force de multiplier la valeur de `P` par 2 puisque `P` est initialisé à 1, `P` va croitre et dépasser tout nombre entier fixé au préalable et dépasser la valeur de `N`

   Il est très important de s'assurer qu'un programme contenant une boucle Tant que ,s'arrête au bout d'un moment

   Autrement dit un algorithme contenant une boucle **Tant que** nécessite plus de rigueur qu'un algorithme ne contenant pas de boucle **Tant que**

2. Mais comment être sûr que `P` contient à la fin pour tous les `N` possibles la plus grande puissance de 2 contenue dans `N` ? On peut se convaincre en faisant quelques tests mais on aimerait avoir une sorte de **preuve** que c'est toujours vraie

3. En effet à la sortie de boucle P > N donc en divisant par 2 on se retrouve avec la valeur de P juste inférieure ou égale à N

On traduit cet **algorithme** dans le langage Python par une fonction

```
def plus_grande_puissance(n):
    p = 1
    while p <= n:
        p = p *2
    p = p // 2
    return p
```

En fait ce qui nous intéresse c'est *l'exposant de la plus grande puissance de 2 contenue dans `N`*

Il suffit pour cela d'ajouter une variable `E`, pour exposant, initialisée à 0 et augmentant de 1 à chaque tour de boucle

Algorithme 2 : Exposant de la plus grande puissance de 2 contenue dans un entier

Données : un entier naturel `N`
Résultat : la plus grande puissance `P` de 2 contenue dans `N`

```
1 début
2 |  P ← 1
3 |  E ← 0
4 |  tant que P <= N faire
5 |  |  P ← P *2
6 |  |  E ← E + 1
7 |  fin
8 |  E ← E - 1
9 fin
```

Traduit en Python cela donne

```
def plus_grande_puissance(n):
    p = 1
    e = 0
    while p <= n:
        p *= 2
        e += 1
    e -= 1
    return e
```

Exercices

Ex 1

1. Ecrire un algorithme d'abord en **français** puis sous une forme un peu plus structurée, qui permet d'obtenir le nombre de chiffres d'un entier naturel écrit en base 10
2. Définir une fonction Python `nb_chiffres(n)` qui retourne le nombre de chiffres

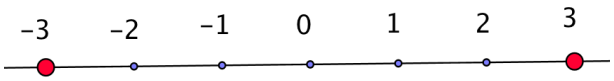
Ex 2

1. Ecrire un algorithme d'abord en **français** puis sous une forme un peu plus structurée, qui permet d'obtenir le nombre de bits contenue dans la représentation binaire d'un entier naturel écrit en base 10 par exemple si n = 7 alors $7 = (111)_2$ est représenté par 3 bits
2. Définir une fonction Python `nb_bits(n)` qui retourne le nombre de bits de la représentation binaire de n

Ex 3

Un jeton part de l'origine et se déplace aléatoirement ainsi : Si le lancer d'une pièce a donné **Pile** alors le jeton se déplace d'une unité vers la droite sinon il se déplace d'une unité vers la gauche et le déplacement s'arrête lorsque l'abscisse du jeton est 3 ou -3. Il s'agit de compter **le nombre moyen** de lancers de la pièce par déplacement

1. Compléter l'algorithme suivant (on dira que Pile est traduit par 0 et Face par 1)



Algorithme 3 : Déplacement du jeton

Données : Un jeton à l'origine, un intervalle gradué [-3 ;3] et un générateur de nombres aléatoires
Résultat : Le nombre de lancers de la pièce

```
1  début
2      position ← 0
3      nbLancers ← 0
4      tant que ..... faire
5          piece ← entierAleatoire(0,1)
6          .....
7          .....
8          .....
9          .....
10     fin
11     afficher(nbLancers)
12 fin
```

2. Traduire l'algorithme par une fonction Python `nb_lancers()`
3. Exécuter un "grand nombre de fois" , par exemple 1000 fois la fonction précédente et calculer la moyenne des nombres de lancers.

   N'est ce pas remarquable ?Faire une **conjecture** et tester la

Ex 4

Dans le Jeu **Devine un nombre** le joueur doit deviner un nombre choisi au hasard par l'ordinateur entre 0 et 1000

A chaque tour le joueur propose une solution et l'ordinateur répond si la solution proposée est plus petite ou plus grande que le nombre tiré au hasard au départ

Ecrire un programme permettant de jouer à ce jeu