Comment calculer la puissance d'un nombre de manière optimisée ?

1 Étude de la fonction native

1.1 Fonctions Python "buit-in"

```
1    def puissance_star(x:int,n:int)->int:
2        return x**n
3
4    def puissance_builtin(x:int,n:int)->int:
5        return pow(x,n)
6
```

Code 1 – Fonctions natives

1.2 Tester un programme

À retenir

La programmation *défensive* consiste à anticiper les problèmes éventuels. On peut utiliser des **assertions**.

Il existe plusieurs modules permettant d'automatiser les tests : `doctest` est un exemple.

2 Implémenter la fonction puissance

2.1 S'appuyer sur la définition mathématique

$$a^n = \underbrace{a \times .... \times a}_{n fois} \qquad et \qquad a^0 = 1$$

```
1  def puissance_perso(x:int,n:int)->int:
2      """
3      >>> puissance_perso(2,8)
4      256
5      >>> puissance_perso(2,9)
6      512
7      """
8      res = 1
9      for i in range(n):
10         res*=x
11     return res
```

2.2 Correction de l'algorithme

À retenir

Christophe Viroulaud

Un **invariant de boucle** est une propriété qui si elle est vraie avant l'exécution d'une itération le demeure après l'exécution de l'itération.

3 Formulations récursives

3.1 Notation mathématique

$$puissance(x,n) = \begin{cases} 1 & \text{si } n = 0 \\ x.puissance(x, n-1) & \text{si } n > 0 \end{cases}$$

À retenir

Une fonction récursive :

— s'appelle elle-même,

— possède un **cas limite** pour stopper les appels.

```
1 def puissance_recursif(x: int, n: int) -> int:
2     if n == 0: # cas limite
3         return 1
4     else: # appel récursif
5         return x*puissance_recursif(x, n-1)
```

Code 2 – Traduction de la formule mathématique

À retenir

La **pile d'appels** stocke les appels successifs de la fonction récursive.

3.2 Nouvelle formulation mathématique

$$puissance(x,n) = \begin{cases} 1 & \text{si } n = 0 \\ puissance(x*x, n/2) & \text{si } n > 0 \text{ et n pair} \\ x.puissance(x*x, (n-1)/2) & \text{si } n > 0 \text{ et n impair} \end{cases}$$

```
1 def puissance_recursif_rapide(x: int, n: int) -> int:
2     if n == 0: # cas limite
3         return 1
4     elif n % 2 == 0: # pair
5         return puissance_recursif_rapide(x*x, n//2)
6     else: # impair
7         return x*puissance_recursif_rapide(x*x, n//2)
```

Code 3 – Exponentiation rapide

Christophe Viroulaud