1 Problématique

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

Un calcul comme 3^4 ne pose pas de problème mais 2701^{103056} peut prendre un certain à effectuer par le langage de programmation.

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

2 Étude de la fonction native

2.1 Fonctions Python "built-in"

fournies par Python et/ou langages de haut-niveau.

```
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)
```

2.2 Tester un programme

2.2.1 Préconditions

Nous nous limitons au cas positif.

évoquer la *programmation défensive*

Activité 1 : Mettre en place un test qui lèvera une *AssertionError* si l'exposant est négatif.

2.2.2 Mettre en place des tests

Il existe plusieurs modules (*doctest*) qui facilitent les phases de test.

```
1  import doctest
2
3  def puissance_star(x:int,n:int)->int:
4      """
5      >>> puissance_star(2,8)
6      256
7      >>> puissance_star(2,9)
8      512
9      """
10     return x**n
11
12 doctest.testmod(verbose=True)
```


Christophe Viroulaud

2.3 Temps d'exécution

Observons la durée d'exécution de nos fonctions, pour de grandes valeurs de paramètres.

```
from time import time

debut=time()
puissance_star(2701,19406)
fin=time()
print("opérande **",fin-debut)
```

3 Implémenter la fonction *puissance*

3.1 S'appuyer sur la définition mathématique

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

Activité 2 :

1. Implémenter la fonction **puissance_perso(x : int, n : int) → int** sans utiliser les fonctions buitin de Python.
2. Mettre en place un test de vérification de la fonction.
3. Mesurer le temps d'exécution de la fonction en l'appelant avec les paramètres (2701,19406).

```
def puissance_perso(x:int,n:int)->int:
    """
    >>> puissance_perso(2,8)
    256
    >>> puissance_perso(2,9)
    512
    """
    res = 1
    for i in range(n):
        res*=x
    return res
```

3.2 Invariant de boucle

Il permet de prouver la *correction* d'un algorithme.
On appelle *invariant d'une boucle* une propriété qui si elle est vraie avant l'exécution d'une itération le demeure après l'exécution de l'itération.
La propriété $res = x^i$ est un invariant de boucle. C'est en fait un raisonnement par récurrence comme en mathématiques.

3.3 Temps d'exécution

Christophe Viroulaud

Activité 3 : Que peut-on conclure à propos de l'implémentation de la fonction *exponentielle* fournie par Python ?

durée ×20 par rapport aux fonctions builtin. Qu'en conclure ?

4 Formulations récursives

4.1 Notation mathématique

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

4.2 Traduction en code

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

visualisation via pythontutor
tester sur petites valeurs
pour grandes valeurs : python limite la pile d'exécution à 1000 récursion
ne change rien pour la durée d'exécution pour l'instant.

4.3 Nouvelle formulation mathématique

s'appuie sur les maths pour optimiser nos programmes//il y a une relation étroite maths/info (EDF recrute des matheux en Python)

$$a^{2048} = \left(\left(\left(\left(\left(\left(\left(\left(\left(\left(a^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2\right)^2.$$

FIGURE 1 – Exponentiation rapide

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

Christophe Viroulaud

```python
def puissance_recursif_rapide(x,n):
    if n==0:
        return 1
    elif n%2==0:
        return puissance_recursif_rapide(x*x,n//2)
    else:
        return x*puissance_recursif_rapide(x*x,n//2)
```

pourquoi n'obtient-on pas encore une durée similaire aux fonctions builtin ?

— Implémentation des fonctions builtin en C

— itératif plus rapide car appels fonction coûtent ; mais récursif donne souvent code plus clair/lisible

— python pas optimisé pour récursif

code Python est opensource.
on peut passer impératif <-> récursif

```python
def puissance_iteratif_rapide(x,n):
    res=1
    while n>0:
        if n % 2 == 0:
            x = x*x
            n = n // 2
        else:
            res = res * x
            n = n - 1
    return res
```


Christophe Viroulaud