1 Problématique

$$a^n = \underbrace{a \times \dots \times a}_{nfois}$$
 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.

```
def puissance_star(x:int,n:int)->int:
    return x**n

def puissance_builtin(x:int,n:int)->int:
    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.

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



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

$$a^n = \underbrace{a \times \dots \times a}_{nfois}$$
 et $a^0 = 1$

Activité 2:

- 1. Implémenter la fonction $puissance_perso(x:int,exp:int)$ ->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).

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

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

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

```
def puissance_recursif(x:int,n:int)->int:
    if n==0:
        return 1
    else:
        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)

FIGURE 1 – Exponentiation rapide

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

```
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

```
def puissance_iteratif_rapide(x,n):
1
2
      res=1
      while n>0:
3
          if n % 2 == 0:
4
5
             x = x*x
              n = n // 2
6
7
          else:
              res = res * x
8
9
              n = n - 1
10
      return res
```

