

Exponentiation Notion de récursivité

Christophe Viroulaud

Terminale - NSI

Lang 05

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

L'*exponentiation* est une opération mathématique définie
par :

$$a^n = \underbrace{a \times \dots \times a}_{n \text{ fois}} \quad \text{et } a^0 = 1$$

- ▶ $2^4 \rightarrow 3$ opérations,
- ▶ $2701^{103056} \rightarrow 103055$ opérations.

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

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

1. Étude de la fonction native

1.1 Fonctions Python "built-in"

1.2 Tester un programme

2. Implémenter la fonction *puissance*

3. Formulations récursives

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
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)
```

Code 1 – Fonctions natives

Activité 1 : Tester les deux fonctions du code 1.

1. Étude de la fonction native

1.1 Fonctions Python "built-in"

1.2 Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

2. Implémenter la fonction *puissance*

3. Formulations récursives

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Nous décidons de nous limiter au cas positif.

À retenir

La programmation *défensive* consiste à anticiper les problèmes éventuels.

Activité 2 : Mettre en place un test qui lèvera une `AssertionError` si l'exposant est négatif.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
1 def puissance_star(x: int, n: int) -> int:
2     assert n >= 0, "L'exposant doit être positif."
3     return x**n
```

Code 2

Mettre en place des tests

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

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)
```

Code 3 – Tester une fonction

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Activité 3 : À l'aide de la bibliothèque `time` mesurer la durée d'exécution de la fonction `puissance_star` pour calculer 2701^{19406} .

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
1 from time import time
2
3 debut=time()
4 puissance_star(2701,19406)
5 fin=time()
6 print("opérande **",fin-debut)
```

1. Étude de la fonction native
2. Implémenter la fonction *puissance*
 - 2.1 S'appuyer sur la définition mathématique
 - 2.2 Correction de l'algorithme
3. Formulations récursives

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

S'appuyer sur la définition mathématique

$$a^n = \underbrace{a \times \dots \times a}_{n \text{ fois}} \quad \text{et} \quad a^0 = 1$$

Activité 4 :

1. Implémenter la fonction **puissance_perso(x : int, n : int) → int** sans utiliser les fonctions builtin 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).

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
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
```

```
1 opérande ** 0.006058692932128906
2 fonction pow() 0.005688667297363281
3 fonction personnelle 0.13074541091918945
```

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

1. Étude de la fonction native

2. Implémenter la fonction *puissance*

2.1 S'appuyer sur la définition mathématique

2.2 Correction de l'algorithme

3. Formulations récursives

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

À retenir

Un **invariant de boucle** est une propriété qui est vraie avant l'exécution de chaque itération.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
1 res = 1
2 for i in range(n):
3     res*=x
```

Code 5 – La propriété $res = x^i$ est un invariant de boucle.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

- Si $i = 0$ (début de la première itération) la propriété est vérifiée : $x^0 = 1 = res$.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

- ▶ Si $i = 0$ (début de la première itération) la propriété est vérifiée : $x^0 = 1 = res$.
- ▶ Supposons la propriété vraie au rang p .

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

- ▶ Si $i = 0$ (début de la première itération) la propriété est vérifiée : $x^0 = 1 = res$.
- ▶ Supposons la propriété vraie au rang p .
- ▶ Vérifions au rang $p + 1$:
 - ▶ au début de l'itération p , $res = x^p$
 - ▶ à la fin de l'itération p , $res = x^p * x = x^{p+1}$
 - ▶ donc au début de l'itération $p+1$, $res = x^{p+1}$

1. Étude de la fonction native
2. Implémenter la fonction *puissance*
3. Formulations récursives
 - 3.1 Notation mathématique
 - 3.2 Implémentation
 - 3.3 Nouvelle formulation mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
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** est une fonction qui s'appelle elle-même.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

1. Étude de la fonction native
2. Implémenter la fonction *puissance*
3. Formulations récursives
 - 3.1 Notation mathématique
 - 3.2 **Implémentation**
 - 3.3 Nouvelle formulation mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

À retenir

Une fonction récursive :

- ▶ s'appelle elle-même,
- ▶ possède un **cas limite** pour stopper les appels.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

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

Code 6 – Traduction de la formule mathématique

Pile d'appels

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

```
puissance_recuratif(6,4)=  
    return 6 * puissance_recuratif(6,3)  
        |  
        return 6 * puissance_recuratif(6,2)  
            |  
            return 6 * puissance_recuratif(6,1)  
                |  
                return 6 * puissance_recuratif(6,0)  
                    |  
                    return 1
```

Visualisation

À retenir

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

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Remarques

- Python limite la pile d'appels à 1000 récursions.

```
1 import sys
2 sys.setrecursionlimit(20000)
```

Code 7 – Augmenter le nombre de récursions

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Remarques

- Python limite la pile d'appels à 1000 récursions.

```
1 import sys
2 sys.setrecursionlimit(20000)
```

Code 8 – Augmenter le nombre de récursions

- La durée d'exécution ne s'est pas améliorée.

```
1 fonction récursive
   0.16802310943603516
```

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

1. Étude de la fonction native
2. Implémenter la fonction *puissance*
3. Formulations récursives
 - 3.1 Notation mathématique
 - 3.2 Implémentation
 - 3.3 Nouvelle formulation mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

FIGURE 1 – Exponentiation rapide

Nouvelle formulation
mathématique

$puissance(x, n) =$

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

Activité 5 : Implémenter la fonction

`puissance_recuratif_rapide(x: int, n: int) → int` qui traduit la formulation récursive précédente.

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

Étude de la fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations récursives

Notation mathématique

Implémentation

Nouvelle formulation
mathématique

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

Code 9 – Exponentiation rapide

Visualisation

Étude de la
fonction native

Fonctions Python "built-in"

Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Implémenter la
fonction *puissance*

S'appuyer sur la définition
mathématique

Correction de l'algorithme

Formulations
récursives

Notation mathématique

Implémentation

**Nouvelle formulation
mathématique**

```
1 fonction récursive rapide  
    0.021007537841796875
```

Code 10 – Les résultats s'améliorent sans égaler la fonction native.