Exponentiation Notion de récursivité

Exponentiation Notion de récursivité

Christophe Viroulaud
Terminals - NSI
Lang 05

Exponentiation
Notion de récursivité

Christophe Viroulaud

Terminale - NSI

Lang 05

Exponentiation Notion de récursivité

Étude de la onction native

Tester un programme
Préconditions

nplémenter la onction *puissance* 

rrection de l'algorithme

mulations ursives tation mathématique

lémentation Ivelle formulation hématique

L'exponentiation est une opération mathématique définie par : 
$$s^a = \underbrace{s \times \dots \times s}_{\text{n fois}} \qquad \text{et } s^0 = 1$$

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

L'exponentiation est une opération mathématique définie par :

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

Exponentiation Notion de récursivité

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

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



### Fonctions Python "built-in"

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

def puissance_builtin(x:int,n:int)->int:
    return pow(x,n)
```

Code 1 – Fonctions natives

**Activité 1 :** Tester les deux fonctions du code 1.

Exponentiation Notion de récursivité

Étude de la fonction native

Fonctions Python "built-in"

Préconditions

Mettre en place des tests

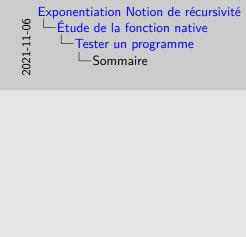
mplementer la onction *puissance* 

mathématique Correction de l'algorithme

Formulations

Notation mathématique

Nouvelle formulation nathématique



## Sommaire

1. Étude de la fonction native

Mettre en place des tests

1.2 Tester un programme

- 1. Étude de la fonction native
- 1.2 Tester un programme Préconditions Mettre en place des tests Durée d'exécution

Tester un programme

Exponentiation

Notion de

récursivité

Correction de l'algorithme

#### Préconditions

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.

Exponentiation Notion de récursivité

fonction native

Tester un programme Préconditions

Mettre en place des tests

nplémenter la onction *puissance* 

S'appuyer sur la définition mathématique

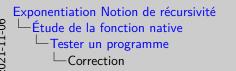
Correction de l'algorithme

Formulations

ecursives

Notation mathématique

uvelle formulation thématique



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

#### Correction

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

Code 2

Exponentiation Notion de récursivité

Préconditions

Correction de l'algorithme

Mettre en place des tests

Il existe plusieurs modules (doctest) qui facilitent les

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

## Exponentiation Notion de récursivité Étude de la fonction native Tester un programme

```
| import doctors
| impo
```

```
import doctest
   def puissance_star(x:int,n:int)->int:
       >>> puissance_star(2,8)
       256
       >>> puissance_star(2,9)
       512
        11 11 11
10
       return x**n
   doctest.testmod(verbose=True)
```

Code 3 – Tester une fonction

Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

fonction puissance S'appuyer sur la définition

Correction de l'algorithme

CUrsives

Notation mathematique Implémentation

Durée d'exécution

Durée d'exécution

Activité 3 : À l'aide de la bibliothèque time mesurer

la durée d'exécution de la fonction puissance\_star pour calculer 2701 19006.

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

Exponentiation Notion de récursivité

Etude de la fonction native

Fester un programme

Mettre en place des tests

Durée d'exécution

onction *puissance* 

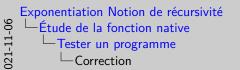
S appuyer sur la definition mathématique

Correction de l'algorithme

mulations ursives

tation mathématique

velle formulation hématique



#### Correction

```
from time import time

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

Exponentiation Notion de récursivité

Étude de la fonction native

ester un programme

réconditions Mettre en place des tests

Durée d'exécution

mplementer la onction *puissance* 

S'appuyer sur la définition mathématique

Correction de l'algorithme

rmulations

Ocursives

Notation mathématique

lotation mathématique mplémentation

velle formulation hématique



## Sommaire

- 2. Implémenter la fonction puissance
- 2.1 S'appuver sur la définition mathématique
- 2.1 Sappuyer sur la definition mathematique
- 2.2 Correction de l'algorithme
- 3 Formulations récursives

Exponentiation

Notion de

récursivité

Implémenter la

fonction puissance

Correction de l'algorithme

S appuyer sur la définition mathématique

s' = \( \frac{x\_1 - y\_0}{x\_1} \) et s' = 1

Activité 4:

1. Implémente la fonction pulsanec\_peros(x int., x int) - int caus utilize la fonction bains de

2. Mattre en place un test de vérification de la fonction.

1. Mesure la treuse d'addiction de la fonction en l'appuise avec les paradites de la fonction en l'appuise avec les paradites (2713,1566)

#### S'appuyer sur la définition mathématique

$$a^n = \underbrace{a \times .... \times a}_{\text{of } i_n}$$
 et  $a^0 =$ 

#### Activité 4 :

- 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).

#### Exponentiation Notion de récursivité

Etude de la fonction native

Préconditions

Mettre en place des tests

fonction puissance
S'appuyer sur la définition
mathématique

Correction de l'algorithme

écursives

notation matnematique
mplémentation
Jouvelle formulation



#### Correction

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

```
opérande ** 0.006058692932128906

fonction pow() 0.005688667297363281

fonction personnelle 0.13074541091918945
```

Code 4 – Les résultats sont significatifs.

Notion de récursivité

Exponentiation

nction native

Préconditions

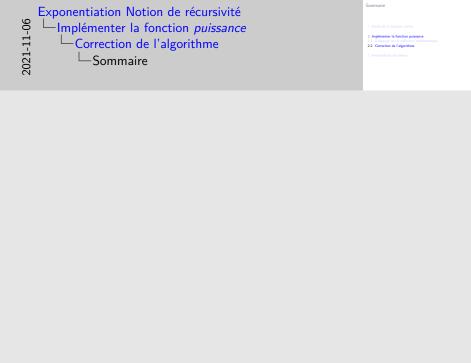
Mettre en place des test

Implémenter la fonction puissance
S'appuyer sur la définition mathématique

orrection de l'algorithm ormulations cursives

tation mathématique plémentation

plémentation ouvelle formulation athématique



## Sommaire

- 2. Implémenter la fonction puissance

- 2.2 Correction de l'algorithme

Exponentiation

Notion de

récursivité

Correction de l'algorithme

16 / 32

Exponentiation Notion de récursivité

Implémenter la fonction puissance

Correction de l'algorithme

Correction de l'algorithme

À retenir
Un invariant de boacle est une propriété qui est vraie avant l'exécution de chaque l'ération.

Correction de l'algorithme

Correction de l'algorithme

### À retenir

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

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

nplémenter la enction *puissance* 

mathématique Correction de l'algorithme

orrection de l'algorithme

écursives

tation mathématique plémentation

## Exponentiation Notion de récursivité Implémenter la fonction puissance Correction de l'algorithme



C'est en fait un raisonnement par récurrence comme en mathématiques.

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

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

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

plémenter la nction *puissance* 

athématique

Correction de l'algorithme

-ormulations écursives

tation mathématique plémentation

▶ Si i=0 (début de la première itération) la propriété est vérifiée :  $\mathbf{x}^0=1=\mathit{res}$ .

Exponentiation Notion de récursivité

onction native

Tester un programme Préconditions

Mettre en place des tests Durée d'exécution

> émenter la tion *puissance* uyer sur la définition

mathématique Correction de l'algorithme

rmulations cursives tation mathématique

mentation elle formulation ématique

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

Exponentiation Notion de récursivité

fonction native

Tester un programme Préconditions

nplémenter la

S'appuyer sur la définition mathématique Correction de l'algorithme

rmulations

ptation mathématique
plémentation
puvelle formulation

Si i = 0 (début de la première itération) la propriété est vérifiée : x<sup>0</sup> = 1 = res.

Supposons la propriété vraie au rang p.

- Si i = 0 (début de la première itération) la propriété est vérifiée : x<sup>0</sup> = 1 = res.
  - Supposons la propriété vraie au rang μ.
  - ▶ Vérifions au rang p + 1 :
  - au début de l'itération p, res = x<sup>p</sup>
     à la fin de l'itération p, res = x<sup>p</sup> = x = x<sup>p+1</sup>
     donc au début de l'itération p+1, res = x<sup>p+1</sup>

- 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:

  - ightharpoonup au début de l'itération p,  $res = x^p$
  - ightharpoonup à la fin de l'itération p,  $res = x^p * x = x^{p+1}$
  - b donc au début de l'itération p+1,  $res = x^{p+1}$

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

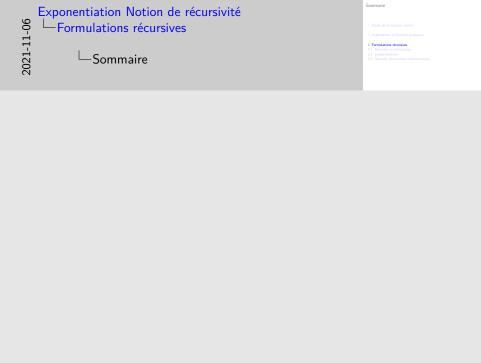
appuyer sur la définition

Correction de l'algorithme

rmulations

écursives Notation mathématique

tation mathematique plémentation



## Sommaire

- 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

Correction de l'algorithme
Formulations
récursives

Exponentiation

Notion de

récursivité

Notation mathématique

Notation mathématique

À retenir

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

Une fonction récursive est une fonction qui s'appelle elle-

récursivité = technique de programmation // impératif

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

### À retenir

Une fonction **récursive** est une fonction qui s'appelle ellemême.



## Sommaire

- 3. Formulations récursives
- 3.2 Implémentation

3.2 Implémentation

Exponentiation

Notion de

récursivité

Correction de l'algorithme

Implémentation

Implémentation

### **Implémentation**

### À retenir

Une fonction récursive :

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

Notion de récursivité

Exponentiation

Correction de l'algorithme

Implémentation

```
def puissance_recursif(x: int, n: int) -> int:
    if n == 0: # cas limite
        return 1
delse: # appel récursif
return x*puissance_recursif(x, n-1)
```

Code 6 – Traduction de la formule mathématique

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme

urée d'exécution

onction puissance
S'appuyer sur la définition

Correction de l'algorithme

ursives

Notation mathématique

plémentation ouvelle formulation athématique





### Pile d'appels

```
puissance recursif(6,4)=
    return 6 * puissance recursif(6,3)
                     return 6 * puissance_recursif(6,2)
                                       return 6 * puissance recursif(6,1)
                                                                                                 Correction de l'algorithme
                                                        return 6 * puissance recursif(6,0)
                                                                         return 1
                                                                                                 Implémentation
```

#### Visualisation

Exponentiation

Notion de

récursivité

25 / 32



## À retenir

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

Exponentiation Notion de récursivité

tude de la onction native

Tester un programme Préconditions

> Mettre en place des tests Durée d'exécution

nplémenter la nction *puissance* 

S'appuyer sur la définition mathématique Correction de l'algorithme

Correction de l'algorithme

rmulations cursives

Notation mathématique

Implémentation



- 1. Il n'y a pas de raison que ça soit mieux : le nombre d'opérations reste le même
- 2. même un peu moins bien : la récursivité est moins bien géré par l'interpréteur Python que par d'autres langages (Ocaml)

#### Remarques

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

```
1 import sys
```

2 sys.setrecursionlimit(20000)

Code 7 – Augmenter le nombre de récursions

#### Exponentiation Notion de récursivité

Etude de la onction native

Fonctions Python "built-i Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

onction puissance

Correction de l'algorithme

ormulations

otation mathématique

Implémentation



- 1. Il n'y a pas de raison que ça soit mieux : le nombre d'opérations reste le même
- 2. même un peu moins bien : la récursivité est moins bien géré par l'interpréteur Python que par d'autres langages (Ocaml)

#### 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 Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme
Préconditions

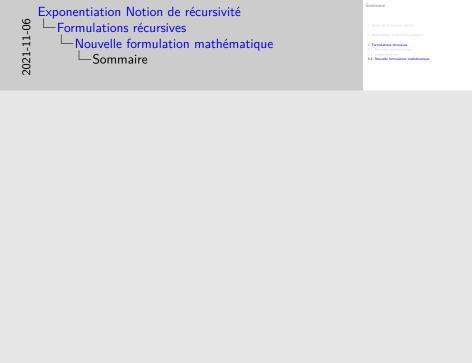
Mettre en place des tests Durée d'exécution

onction puissance

Correction de l'algorithme

écursives

Notation mathématique Implémentation



### Sommaire

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

Correction de l'algorithme Formulations récursives

Exponentiation

Notion de

récursivité

Notation mathématiq Implémentation Nouvelle formulation

Nouvelle formulatio mathématique

# Exponentiation Notion de récursivité Formulations récursives Nouvelle formulation mathématique Nouvelle formulation mathématique

Nouvelle formulation mathématique

 $\label{eq:Figure 1-Exponentiation rapide} Figure \ 1- Exponentiation \ rapide$ 

Exponentiation Notion de récursivité

Etude de la fonction native

Fonctions Python "built-in Tester un programme Préconditions

> olémenter la ction *puissance*

Correction de l'algorithme Formulations

> UrSives tation mathématique

> tation mathématique olémentation

Nouvelle formulation mathématique

## Exponentiation Notion de récursivité Formulations récursives Nouvelle formulation mathématique

```
\begin{aligned} & pubsance(x,n) = \\ \begin{cases} 1 & \text{sin} = 0 \\ pubsance(x+x,n/2) & \text{sin} = 0 \text{ at a pair} \\ x-pubsance(x+x,(n-1)/2) & \text{sin} = 0 \text{ at a pair} \\ x-pubsance(x+x,(n-1)/2) & \text{sin} = 0 \text{ at a impair} \\ \end{aligned} \\ & \textbf{Activité S.: Implimenter la fortition} \\ & \textbf{pubsance.} \end{aligned}
```

```
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a = 0
a =
```

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

puissance(x, n) =

Activité 5 : Implémenter la fonction puissance\_recursif\_rapide(x: int, n: int)→ int qui traduit la formulation récursive précédente.

#### Exponentiation Notion de récursivité

Etude de la fonction native

Préconditions

Mettre en place des tests

Implémenter la fonction *puissance* 

mathématique

Correction de l'algorithme

Correction de l'algorithme

ecursives

Notation mathématique

Nouvelle formulation mathématique

#### Correction

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

•

Visualisation

Code 9 – Exponentiation rapide

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme
Préconditions
Mettre en place des tests

Implémenter la fonction puissance
S'appuyer sur la définition mathématique

Correction de l'algorithme
Formulations
récursives

cursives lotation mathématique

Implémentation

Nouvelle formulation

Nouvelle formulation mathématique 1 fonction récuraire rapide
0.021007537841796875

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

- 1. Implémentation des fonctions builtin en C
- 2. itératif plus rapide car appels fonction coûtent ; mais récursif donne souvent code plus clair/lisible

1 fonction récursive rapide 0.021007537841796875

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

Exponentiation Notion de récursivité

Etude de la fonction native

Fester un programme
Préconditions

Ourée d'exécution

plementer la nction puissance

mathématique

Correction de l'algorithme

orrection de l'algorithme

mulations ursives

ion mathématique mentation

Nouvelle formulation mathématique