Exponentiation Notion de récursivité

Christophe Viroulaud
Terminale - NSI
Lang 05

Exponentiation Notion de récursivité

Christophe Viroulaud

Terminale - NSI

Lang 05

Exponentiation Notion de récursivité

tude de la onction native

Fester un programme
Préconditions

mplémenter la conction *puissance* 

> appuyer sur la définition athématique rrection de l'algorithme

> > nulations rsives

ation mathématique lémentation Ivelle formulation chématique 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 :

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

2<sup>4</sup> → 3 opérations,
 2701<sup>100056</sup> → 103055 opérations.

Comment calculer la puissance d'un nombre de manière

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

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

Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme

Mettre en place des tests

onction puissance

Correction de l'algorithme

rmulations

rsives tion mathématique

lémentation rvelle formulation hématique



### 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

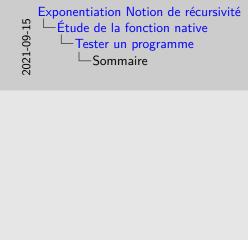
onction puissance

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
- 1.1 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

. . .

Exponentiation

Notion de

récursivité

Tester un programme

Correction de l'algorithme

Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme

Préconditions

Préconditions

Nous décidens de nous limiter au cas positif.

A retenir

La programmation définaive consiste à anticiper les prolimines destratul.

Activité 2 : Mettre en place un test qui livera une
Assertiantieren si reposant est négat.

### 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

Etude de la fonction native

Tester un programme Préconditions

Mettre en place des tests

nplémenter la

S'appuyer sur la définition mathématique

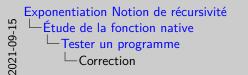
Correction de l'algorithme

Formulations

écursives

Notation mathématique

olémentation uvelle formulation thématique





```
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

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

tude de la onction native

Tester un programme
Préconditions

Mettre en place des tests

Durée d'exécution

nplémenter la

Onction puissance
S'appuyer sur la définition
mathématique

Correction de l'algorithme

rmulations cursives

tion mathématique mentation

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

```
| import doctors
| definition |
```

```
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

Fonctions Python "built-Tester un programme

Mettre en place des tests

Implémenter la fonction *puissance* 

mathématique

Correction de l'algorithme

ormulations cursives

Notation mathématique Implémentation

Durée d'exécution

Durée d'exécution

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

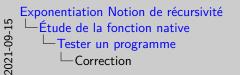
la durée d'exécution de la fonction puissance\_star pour calculer 2701 19006.

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

Durée d'exécution

Correction de l'algorithme



```
1 from time import time
2 minor-time()
3 minorise plan (7700,19608)
6 firstime()
6 print("operado **",fin-debut)
```

#### 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

lester un programme

Mettre en place des tests

Durée d'exécution

S'appuyer sur la définitio

mathématique Correction de l'algorithme

Correction de l'algorithme

ursives

lotation mathématique

olémentation uvelle formulation



# Sommaire

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

Exponentiation

Notion de

récursivité

Implémenter la

fonction puissance

Correction de l'algorithme

# Exponentiation Notion de récursivité Implémenter la fonction puissance S'appuyer sur la définition mathématique S'appuyer sur la définition mathématique

### 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

Tester un programme
Préconditions
Mettre en place des tests

fonction puissance
S'appuyer sur la définition
mathématique

Correction de l'algorithme

écursives

Notation mathématique Implémentation



#### 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):
10        res*=x
11    return res
```

```
opérande ** 0.006058692932128906

fonction pow() 0.005688667297363281

fonction personnelle 0.13074541091918945
```

Code 4 – Les résultats sont significatifs.

Exponentiation Notion de récursivité

nction native

lester un programme Préconditions Mettre en place des test

Implémenter la fonction puissance

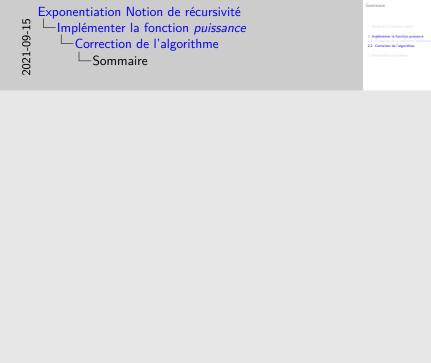
mathématique Correction de l'algorit

rmulations ursives

rtation mathématique plémentation

nplémentation ouvelle formulation athématique

15 / 32



# Sommaire

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

on

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



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 Préconditions

Durée d'exécution

nplémenter la

anction puissance

Correction de l'algorithme

ormulations

écursives Notation mathématique

tation mathématique olémentation

# 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

Fester un programme

Mettre en place des tests Durée d'exécution

plémenter la nction *puissance* 

thématique

Correction de l'algorithme

ormulations cursives

tation mathématique

- 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 :  $x^0 = 1 = res$ .
- ► Supposons la propriété vraie au rang p.

Exponentiation Notion de récursivité

fonction native

Tester un programme Préconditions

pplémenter la

puyer sur la définition nématique

Correction de l'algorithme Formulations

cursives
tation mathématique

olémentation Invelle formulation Intématique

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

► Supposons la propriété vraie au rang p.

 $\blacktriangleright$  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

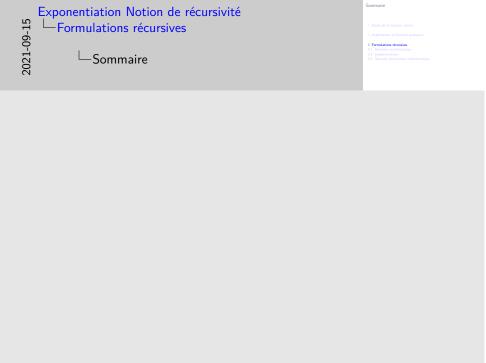
Correction de l'algorithme

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

Supposons la propriété vraie au rang p.

<sup>▶</sup> Vérifions au rang p + 1 :

à la fin de l'itération p. res = x\* x x = x\*\*\* donc au début de l'itération p+1, res = x<sup>p+1</sup>



# Sommaire

- 1. Étude de la fonction native
- 2 Implémenter la fonction puissance
- 3. Formulations récursives
- 3.1 Notation mathématique
- 3.2 Implémentation
- 3.2 Implementation

Exponentiation

Notion de

récursivité

Correction de l'algorithme
Formulations
récursives

# Exponentiation Notion de récursivité Formulations récursives Notation mathématique Notation mathématique

récursivité = technique de programmation // impératif

# Notation mathématique

Notation mathématique

 $pwissance(x, n) = \begin{cases} 1 \end{cases}$ 

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

À retenir

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

### À retenir

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

Notion de récursivité

Exponentiation

fonction native

Tester un programme
Préconditions

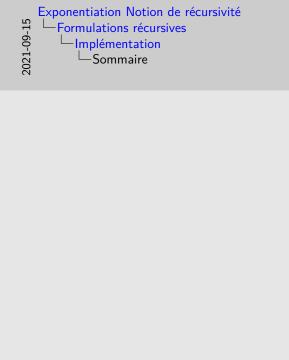
nplémenter la nction *puissance* 

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

> mulations ursives

Notation mathématique

nentation lle formulation matique



# Sommaire

3.2 Implémentation

- 3. Formulations récursives
- 3.2 Implémentation

Correction de l'algorithme

Exponentiation

Notion de

récursivité

Implémentation

Implémentation

### **Implémentation**

## À retenir

Une fonction récursive :

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

Exponentiation Notion de récursivité

Correction de l'algorithme

Implémentation

```
Exponentiation Notion de récursivité

Formulations récursives

Implémentation
```

```
i def paismance_recursif(x: int, n: int) -> int:

if n == 0: 0 cms limits
return
slams = appal recursif
return repaismance_recursif(x, n=1)

Code 6 - Traduction de la formule mathématique
```

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

urée d'exécution

nction puissance

Correction de l'algorithme Formulations

cursives tation mathématique

Implémentation

Nouvelle formulation

# Pile d'appels

Visualisation

ction native

Exponentiation

Notion de

récursivité

lettre en place des te urée d'exécution

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

formulations écursives Notation mathématique

Notation mathématique
Implémentation

velle formulation thématique

# Exponentiation Notion de récursivité Formulations récursives Implémentation



# À retenir

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

#### Exponentiation Notion de récursivité

tude de la onction native

ster un programme Préconditions

urée d'exécution

nplémenter la nction puissance 'appuyer sur la définition

mathématique Correction de l'algorithme

> ormulations cursives

> otation mathématique

Implémentation
Nouvelle formulation
mathématique

# Exponentiation Notion de récursivité Formulations récursives 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

Étude de la fonction native

Fonctions Python "built-in Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

Fonction puissance S'appuyer sur la définition

Correction de l'algorithme

rmulations

Notation mathématique

Implémentation

# Exponentiation Notion de récursivité Formulations récursives 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

Mettre en place des tests Durée d'exécution

onction puissance S'appuyer sur la définition

Correction de l'algorithme

écursives

Notation mathématique

Nouvelle formulation



# Sommaire

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

Correction de l'algorithme
Formulations
récursives
Notation mathématique

Exponentiation

Notion de

récursivité

Implémentation

Nouvelle formulation mathématique

28 / 32

# 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

Étude de la fonction native

Tester un programme
Préconditions

olémenter la ction *puissance* 

Correction de l'algorithme
Formulations

UrSIVES tation mathématique

lation mathematique Démentation

Nouvelle formulation mathématique

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

```
puissance(x + x, n/2) si n > 0 et n pair
Activité 5 : Implémenter la fonction
```

```
x.puissance(x + x, (n - 1)/2) si n > 0 et n impair
puissance_recursif_rapide(x: int, n: int)-
int qui traduit la formulation récursive précédente.
```

```
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}
```

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

#### Exponentiation Notion de récursivité

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)
```

Code 9 – Exponentiation rapide

Visualisation

Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme
Préconditions

Durée d'exécution Implémenter la

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

ormulations

cursives lotation mathématique

Notation mathématique Implémentation Nouvelle formulation

mathématique

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

0.021007537841796875 Code 10 - Les résultats s'améliorent sans égaler la fonction nativ

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

fonction récursive rapide 0.021007537841796875

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

Exponentiation Notion de récursivité

Correction de l'algorithme

Nouvelle formulation mathématique