Exponentiation Notion de récursivité

Christophe Viroulaud
Terminale - 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
Mettre en place des tests

mplémenter la fonction puissance

nthématique rrection de l'algorithme

> nulations rsives ion mathématique

émentation velle formulation hématique L'exponentiation est une opération mathématique définie par :  $a^a = \underbrace{a \times \dots \times a}_{\text{in fois}} \qquad \text{et } a^0 = 1$ 

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

L'exponentiation est une opération mathématique définie par :

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

Tester un programme Préconditions Mettre en place des tests

Durée d'exécution

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

nulations rsives

tion mathématique mentation elle formulation ≥ 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

nplémenter la onction *puissance* 

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

orrection de l'algorithme

rsives

ation mathematique lémentation ivelle formulation thématique



#### Fonctions Python "built-in"

```
1  def puissance_star(x:int,n:int)->int:
2    return x**n
3  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.

Exponentiation Notion de récursivité

Étude de la fonction native

Fonctions Python "built-in"

Préconditions Mettre en place des tests

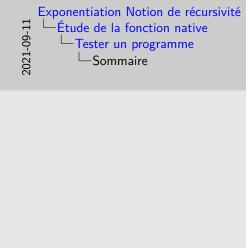
onction puissance
S'appuyer sur la définition

mathématique Correction de l'algorithme

Formulations

Notation mathématique

ouvelle formulation athématique



1. Étude de la fonction native 1.2 Tester un programme Mettre en place des tests

Sommaire

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

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 paintif.

A retemir

La programation définaire conside à anticiper les prolimines destruits.

Activité 2: Mettre en place on test qui livera une
Assertiantierer ul l'oposant est négot.

#### 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* 

mathématique

Correction de l'algorithme

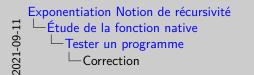
Correction de l'algorithme

ormulations

Notation mathématique

lotation mathématique

elle formulation ématique





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

8/31

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

ctude de la conction native

Tester un programme
Préconditions
Mettre en place des tests

mplémenter la

S'appuyer sur la définition mathématique

Correction de l'algorithme

Formulations

ursives ation mathématique

nentation lle formulation matique

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

```
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

Tester un programme

Mettre en place des tests Durée d'exécution

fonction puissance
S'appuyer sur la définitio
mathématique

Correction de l'algorithme Formulations

écursives Notation mathématique

Implémentation Nouvelle formulation mathématique 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 9406.

Activité 3 : À l'aide de la bibliothèque time mesurer la durée d'exécution de la fonction puissance\_star pour calculer 2701<sup>1</sup>9406.

Exponentiation Notion de récursivité

Étude de la fonction native

Fester un programme
Préconditions

Mettre en place des tests

Durée d'exécution

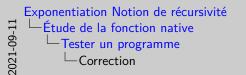
onction puissance

mathématique Correction de l'algorithme

Correction de l'algorithme

nulations irsives

ementation
elle formulation





#### 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

ster un programme

Mettre en place des tests

Durée d'exécution

plémenter la

S'appuyer sur la définition

Correction de l'algorithme

Correction de l'algorithme

ursives

lotation mathématique nplémentation

hématique



- 2. Implémenter la fonction puissance

Exponentiation

Notion de

récursivité

Implémenter la

fonction puissance

Correction de l'algorithme

13/31

#### S'appuyer sur la définition mathématique

$$a^n = \underbrace{a \times .... \times a}_{\text{pfois}}$$
 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

Étude de la fonction native

Préconditions

Mettre en place des tests

Implementer la 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)
        256
        >>> puissance perso(2,9)
 5
        512
        11 11 11
        res = 1
        for i in range(n):
10
            res*=x
11
        return res
```

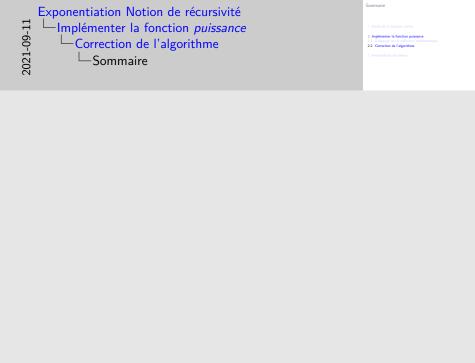
```
opérande ** 0.006058692932128906
fonction pow() 0.005688667297363281
fonction personnelle 0.13074541091918945
```

Code 4 – Les résultats sont significatifs.

Exponentiation Notion de récursivité

S'appuver sur la définition mathématique

15 / 31



### Sommaire

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

place des test xécution

Correction de l'algorithme

Exponentiation

Notion de

récursivité

S athématique tion

16 / 31

Exponentiation Notion de récursivité

Implémenter la fonction puissance

Correction de l'algorithme

Correction de l'algorithme

À retenir

Un invariant de boucle est une propriété qui si elle est vaixe avant l'exécution d'une itération le demeure après l'exécution de ultréation.

Correction de l'algorithme

#### Correction de l'algorithme

#### À retenir

Un **invariant de boucle** est une propriété qui si elle est vraie avant l'exécution d'une itération le demeure après l'exécution de l'itération.

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme Préconditions

Durée d'exécution
nplémenter la

appuyer sur la définition

Correction de l'algorithme

ormulations cursives

Notation mathématique

rementation rvelle formulation hématique

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

1 rea = 1
2 dor 4 in range(n):
3 rea+x

Code 5 - La propriété res = x<sup>2</sup> est un invariant de boucle.

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

Étude de la fonction native

Tester un programme

Mettre en place des tests

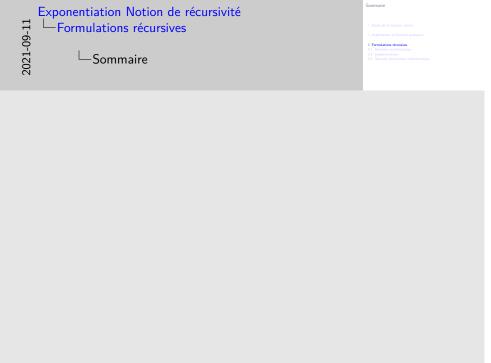
plémenter la nction *puissance* 

thématique

Correction de l'algorithme

ormulations cursives

tation mathématique



## Sommaire

- 3. Formulations récursives

Correction de l'algorithme

Formulations récursives

Exponentiation

Notion de

récursivité

récursivité = technique de programmation // impératif

### Notation 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 ellemême. Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme Préconditions

nplémenter la

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

Correction de l'algorithme

ursives

Notation mathématique

nentation lle formulation matique



## Sommaire

- 3. Formulations récursives
- 3.2 Implémentation

Exponentiation

Notion de

récursivité

Correction de l'algorithme

Implémentation

Exponentiation Notion de récursivité

Formulations récursives

Implémentation

Implémentation

Implémentation

#### Implémentation

### À retenir

Une fonction récursive :

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

Exponentiation Notion de récursivité

Étude de la fonction native

ster un programme réconditions

Durée d'exécution

onction puissance
S'appuyer sur la définition

mathématique Correction de l'algorithme

Correction de l'algorithme

cursives

otation mathématique

Implémentation

```
1 def prisence_recursif(x: int, n: int) >> int;
2 if n == 0: # cas limits
3 return 1
4 slate # spp1 recursif
5 return reprisence_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
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 Préconditions

Durée d'exécution

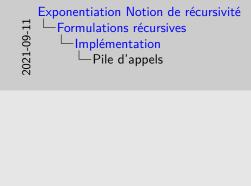
onction puissance l'appuyer sur la définition nathématique

Correction de l'algorithme

rmulations cursives

Notation mathématique

Implémentation
Nouvelle formulation





### 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)
                                                    return 6 * puissance recursif(6,0)
                                                                    return 1
```

#### Visualisation

Exponentiation

Notion de

récursivité

Correction de l'algorithme

Implémentation

# 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

Etude de la onction native

Tester un programme Préconditions

> Mettre en place des tests Durée d'exécution

nction puissance
appuyer sur la définition

mathématique Correction de l'algorithme

Formulations récursives

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 7 – Augmenter le nombre de récursions

#### Exponentiation Notion de récursivité

Etude de la fonction native

Fonctions Python "built-in' Tester un programme

Préconditions

Mettre en place des tests

Durée d'exécution

conction puissance

mathématique Correction de l'algorithme

ormulations

Notation mathématique

Notation mathematique

nplémentation ouvelle formulation

# 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 Gonction native

Tester un programme

Mettre en place des tests Durée d'exécution

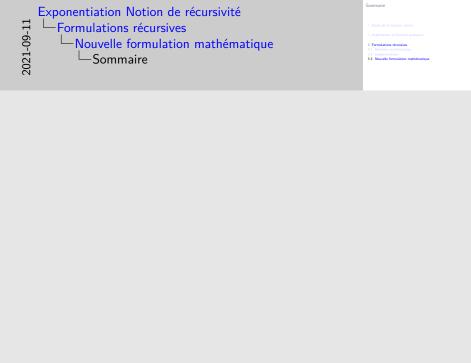
onction puissance S'appuyer sur la définition

Correction de l'algorithme

écursives

Implémentation

louvelle formulation nathématique



### Sommaire

- 2. Implémenter la fonction *puissance*
- 3. Formulations récursives
  - athématique
- 3.2 Implémentation
- 3.3 Nouvelle formulation mathématique

Implémentation

Nouvelle formulation
mathématique

Correction de l'algorithme

Exponentiation

Notion de

récursivité

27 / 31

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

Nouvelle formulation mathématique

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

CUrSIVES lotation mathématique

nplémentation

Nouvelle formulation mathématique

```
Activité 5 : Implémenter la fonction
```

```
puissance(x + x, n/2) si n > 0 et n pair
 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

| off publication, recursif, repide(s: int, n: int) =>
| off n = 0 + int interest
| off n = 0 + interest | option |
| off n = 0 + interest | option |
| off n = 0 + interest | option |
| off n = 0 + interest | option |
| off n = 0 + interest |
| off n =
```

#### 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

Motation mathematique

Nouvelle formulation mathématique

# Exponentiation Notion de récursivité Formulations récursives Nouvelle formulation mathématique

1 | fonction récursive rapide 0.021007537841796875 | Code 10 - Les résultats s'améliorent sans égaler la fonction r

- 1. Implémentation des fonctions builtin en C
- 2

1 fonction récursive rapide 0.021007537841796875

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

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

Exponentiation Notion de récursivité

tude de la onction native

Tester un programme Préconditions Mettre en place des tests

> nplémenter la nction *puissance*

Correction de l'algorithme
Formulations

ursives

Implémentation

Nouvelle formulation
mathématique

21 / 21