Exponentiation
Notion de récursivité
Christophe Viroulaud

Terminale - NSI

Exponentiation

Notion de récursivité

Christophe Viroulaud

Terminale - NSI

Exponentiation Notion de récursivité

ude de la nction native

Tester un programme Préconditions Mettre en place des tests

> plémenter la action puissance ppuyer sur la définition

chématique rection de l'algorithme

mulations ursives ation mathématique

tion mathématique émentation velle formulation lématique 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 $a^0 = 1$

≥ 2⁴ → 3 opérations. ≥ 2701¹⁰³⁰⁵⁶ → 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?



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

mplémentation Jouvelle 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 Equations Dython "built in"
- 1.2 Tester un programme
 Préconditions
 - Mettre en place des tests
 Durée d'exécution
- 2. Implémenter la fonction puissance
- 3 Formulations récursives
- on *puissance*

Exponentiation

Notion de

récursivité

Tester un programme

Correction de l'algorithme

Préconditions

Nous décidons de nous limiter au cas positif.

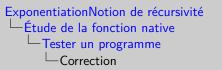
A retenir

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

Préconditions



2021-08-21



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

2021-08-21

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

onction native

Tester un programme
Préconditions

Mettre en place des tests

Durée d'exécution

nplémenter la

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

Correction de l'algorithme

nulations rsives

n mathématique entation

formulation atique

ExponentiationNotion 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 Préconditions

Mettre en place des tests Durée d'exécution

fonction puissanc
S'appuyer sur la définitio
mathématique

Correction de l'algorithme
Formulations

lotation mathématique

ouvelle formulation athématique 2021-08-21

Durée d'exécution

Durée d'exécution

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

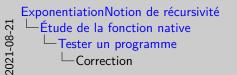
la durée d'exécution de la fonction puissance_star pour calculer 270119406.

> Activité 3 : À l'aide de la bibliothèque time mesurer la durée d'exécution de la fonction puissance_star pour calculer 2701¹9406.

Exponentiation Notion de récursivité

Durée d'exécution

Correction de l'algorithme





Correction

```
from time import time

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

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

nplémenter la onction *puissance*

S'appuyer sur la définition mathématique

Correction de l'algorithme

Correction de l'aigorithme

ursives

Votation mathématique mplémentation

velle formulation hématique



S appuyer sur la définition mathématique

a' = 2 = 2 de de d' = 1

Activité 4:

1. Implémente la fonction pulsanee, persof; z int., n. 140) - let caus utilize la fonction ballois de 2. Mettre en pless une said et vérification de la fonction.

3. Mesure la tourpe d'adoction de la fonction.

1. Experience de la fonction personnées (2701.13406).

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

Étude de la fonction native

Tester un programme Préconditions

nplémenter la

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

Formulations

ÉCURSIVES Notation mathématique

Implémentation

Nouvelle formulation
mathématique



Correction

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

```
opérande ** 0.006058692932128906

fonction pow() 0.005688667297363281

fonction personnelle 0.13074541091918945
```

Code 4 – Les résultats sont significatifs.

de la on native

nction native onctions Python "built-

Exponentiation

Notion de

récursivité

Préconditions Mettre en place des tes Durée d'exécution

Implémenter la fonction puissance S'appuyer sur la définition mathématique

rrection de l'algorithr

cursives otation mathématique

tation mathématique

plémentation ouvelle formulation athématique

15 / 31



Sommaire

- 2. Implémenter la fonction puissance
- 2.2 Correction de l'algorithme

Exponentiation

Notion de

récursivité

- Correction de l'algorithme

A retenir

Un invariant de bourde est une propriété qui si elle est vaie avant l'enécution d'une itération le démeure après l'exécution de l'ideation.

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

nplémenter la

appuyer sur la définition athématique

Correction de l'algorithme

écursives

Notation mathématique

velle formulation hématique

ExponentiationNotion de récursivité Implémenter la fonction puissance Correction de l'algorithme

1 rez = 1
2 for i in range(n):
3 code S – La propriété rez = x' 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

Etude de la fonction native

Tester un programme

Mettre en place des tests Durée d'exécution

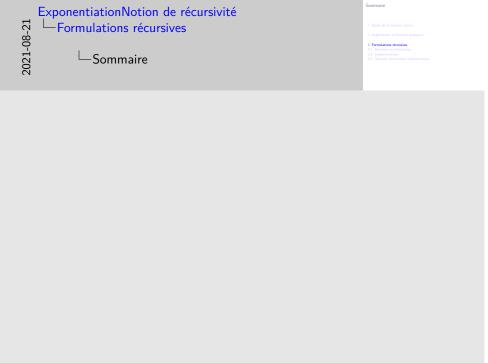
plémenter la nction *puissance*

thématique

Correction de l'algorithme

ormulations écursives

tation mathématique plémentation



Sommaire

- 1 Étude de la fonction native
- 2. Implementer la fonction *puissanc*
- 3. Formulations récursives
- 3.1 Notation mathématique
- 3.1 Notation mathematique
- 3.2 Implémentation

Correction de l'algorithme
Formulations
récursives

Exponentiation

Notion de

récursivité

19/31

Notation mathématique

Notation mathématique

À retenir

 $\mathsf{pwissance}(\mathsf{x}, \mathsf{n}) = \left\{ \begin{array}{ll} 1 & \text{si } \mathsf{n} = 0 \\ \mathsf{x}.\mathsf{pwissance}(\mathsf{x}, \mathsf{n} - 1) & \text{si } \mathsf{n} > 0 \end{array} \right.$

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. Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme Préconditions

nplémenter la inction *puissance*

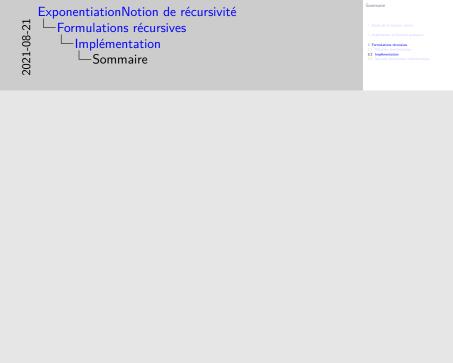
S'appuyer sur la définition mathématique Correction de l'algorithme

Correction de l'algorithme

ursives

Notation mathématique

émentation velle formulation hématique



Sommaire

- 3. Formulations récursives
- 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.

Exponentiation

Notion de

récursivité

Correction de l'algorithme

Implémentation

22 / 31

```
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

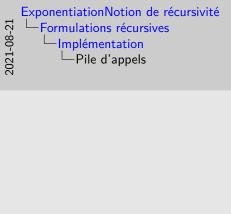
onction puissance

Correction de l'algorithme

ursives tation mathématique

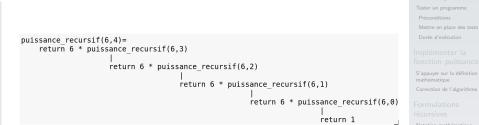
Notation mathématique Implémentation

Nouvelle formulation mathématique





Pile d'appels



Visualisation

Implémentation

Exponentiation

Notion de

récursivité

2021-08-21

À retenir

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

Exponentiation Notion de récursivité

Étude de la fonction native

Tester un programme Préconditions

Mettre en place des tests Durée d'exécution

nction puissance

mathématique Correction de l'algorithme

> rmulations rursives

> tation mathématique

Implémentation

Nouvelle formulation

ExponentiationNotion 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

fonction puissance

mathématique Correction de l'algorithme

Correction de l'aigorithme

écursives

Notation mathématique

Implémentation

ExponentiationNotion 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

Préconditions

Mettre en place des tests Durée d'exécution

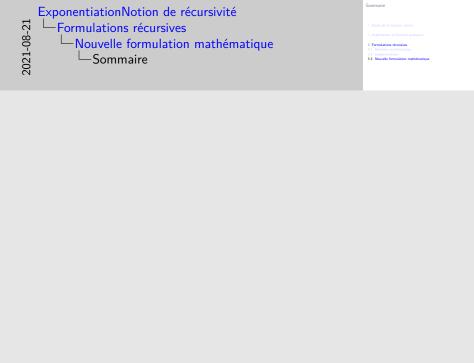
onction puissance S'appuyer sur la définition

Correction de l'algorithme

écursives

Implémentation

npiementation louvelle formulation



Sommaire

- 3. Formulations récursives

- 3.3 Nouvelle formulation mathématique

Exponentiation

Notion de

récursivité

Correction de l'algorithme

Nouvelle formulation

mathématique

ExponentiationNotion de récursivité Formulations récursives Nouvelle formulation mathématique

-Nouvelle formulation mathématique

Nouvelle formulation mathématique

FIGURE 1 – Exponentiation rapide

Exponentiation Notion de récursivité

Etude de la fonction native

Fonctions Python "built-ir Tester un programme Préconditions

Ourée d'exécution

nction puissance
appuyer sur la définition

Correction de l'algorithme
Formulations

lotation mathématique

mplémentation

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 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)→ int qui traduit la formulation récursive précédente.

Exponentiation Notion de récursivité

Etude de la fonction native

Tester un programme Préconditions

Implémenter la fonction *puissance*

mathématique

Correction de l'algorithme

Eormulations

écursives

Implémentation

Nouvelle formulation

mathématique

```
Correction

| ord princesson_vectorid_repide(x: isi, a: isi) ->
| ord princesson_vectorid_repide(x: isi, a: isi) ->
| ord princesson_vectorid_repide(xx, a//2) |
| ord princesson_vectorid_repide(x: isi, a: isi) ->
| ord princesson_vectorid_repide(x: isi) ->
| ord princesson_vectorid_repi
```

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

Implémenter la fonction puissance

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

ormulations cursives

Notation mathématique Implémentation

Nouvelle formulation mathématique

ExponentiationNotion de récursivité Formulations récursives Nouvelle formulation mathématique

1 fonction récursive rapide
0.02107537841796875

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

- 1. Implémentation des fonctions builtin en C
- Implementation des fonctions builtin er

 2 '

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 enction puissance

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

> nulations rsives ion mathématique

Notation mathématique Implémentation Nouvelle formulation

mathématique