

Langage et algorithmique

- Récursivité -

A. Malek TOUMI

toumiab@ensta-bretagne.fr

ENSTA Bretagne



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Récursivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme récursif
Exemples
Notion de récursivité terminale

3 Horloge



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs

Portée de variable

2 Récursivité

Définition

Pile d'appel

Mécanisme des traitements

Explosion combinatoire

Conception d'algorithme récursif

Exemples

Notion de récursivité terminale

3 Horloge



Fonction fact

toggle

reset

- Paramètres formels

```
def fact(nb):  
    if(nb == 0):  
        ...  
  
def cnp(n, p):  
    num = fact(n)  
    ...
```



Fonction fact

toggle

reset

- Paramètres formels
- Paramètres effectifs

```
def fact(nb):  
    if(nb == 0):  
        ...
```

```
def cnp(n, p):  
    num = fact(n)  
    ...
```



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs

Portée de variable

2 Récursivité

Définition

Pile d'appel

Mécanisme des traitements

Explosion combinatoire

Conception d'algorithme récursif

Exemples

Notion de récursivité terminale

3 Horloge



Rappel

Variables

globales/locales

toggle

reset

Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées



Variables

globales/locales

toggle

reset

Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées
- Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).



Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées
- Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
- Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs



Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées
- Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
- Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
- Les fonctions peuvent modifier des variables globales (les types non modifiables) avec l'instruction **global**



Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées
- Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
- Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
- Les fonctions peuvent modifier des variables globales (les types non modifiables) avec l'instruction **global**
- La fonction **globals()** retourne le dictionnaire des objets globaux.



Remarques

- Les variables globales sont visibles dans les fonctions appelées mais on ne peut les réaffectées
- Le contenu d'une variable globale est modifiable si elle est mutable (modifiable).
- Les variables locales d'une fonction ne sont pas visibles dans les niveaux supérieurs
- Les fonctions peuvent modifier des variables globales (les types non modifiables) avec l'instruction **global**
- La fonction **globals()** retourne le dictionnaire des objets globaux.
- La fonction **locals()** retourne la liste des objets l'espace local en cours



Exemple (Modification d'une variable globale en local)

```
def incremCompt():    # fonction sans param. d'entrée
    global compteur   # définition var. globale
    compteur += 1
    print('Appelé', compteur, 'fois')

compteur = 0 # initialisation du compteur
incremCompt() # => affiche: Appelé 1 fois
incremCompt() # => affiche: Appelé 2 fois
incremCompt() # => affiche: Appelé 3 fois
```



Exemple (Modification d'une variable globale en local)

```
def maville(lm):  
    ville = 'Brest' # variable locale  
    print (ville, cp, lm)  
  
# cp=29200 variable globale vue par la fct  
cp = 29200 # variable globale  
maville('France') # => affiche 'Brest 29200 France'
```



Exemple (Modification d'une variable globale en local)

```
def maville(lm):  
    ville = 'Brest' # variable locale  
    print (ville, cp, lm)  
    test()  
def test():  
    print('test', ville)  
  
# cp=29200 variable globale vue par la fct  
cp = 29200 # variable globale  
maville('France') # => affiche 'Brest 29200 France'
```



Exemple (Modification d'une variable globale en local)

```
def maville(lm):  
    ville = 'Brest' # variable locale  
    print (ville, cp, lm)  
    test()  
def test():  
    print('test', ville)  
  
# cp=29200 variable globale vue par la fct  
cp = 29200 # variable globale  
maville('France') # => affiche 'Brest 29200 France'  
# => affiche 'test, Brest'?
```




Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Récurtivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme récursif
Exemples
Notion de récursivité terminale

3 Horloge



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Récurtivité

Définition

Pile d'appel

Mécanisme des traitements

Explosion combinatoire

Conception d'algorithme récursif

Exemples

Notion de récursivité terminale

3 Horloge



Définition

toggle

reset

Définition (Fonction récursive)

Une fonction ou une méthode est dite récursive si elle se définit à partir d'elle même, c'est-à-dire si elle comporte un appel à elle même dans son corps.



Définition

toggle

reset

Définition (Fonction récursive)

Une fonction ou une méthode est dite récursive si elle se définit à partir d'elle même, c'est-à-dire si elle comporte un appel à elle même dans son corps.

Attention

Éviter les boucles infinies (appel systématique à la fonction).



Définition

[toggle](#)[reset](#)

Définition (Fonction récursive)

Une fonction ou une méthode est dite récursive si elle se définit à partir d'elle même, c'est-à-dire si elle comporte un appel à elle même dans son corps.

Attention

Éviter les boucles infinies (appel systématique à la fonction).

Exemple (Fonction infinie)

```
def sansFin(n):  
    return n+sansFin(n-1)
```



Exemple

toggle

reset

Traduction immédiate des fonctions définies par récurrence.

Exemple (Factorielle)

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$



Exemple

[toggle](#)[reset](#)

Traduction immédiate des fonctions définies par récurrence.

Exemple (Factorielle)

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$

```
def fact(n):
```



Exemple

[toggle](#)[reset](#)

Traduction immédiate des fonctions définies par récurrence.

Exemple (Factorielle)

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$

```
def fact(n):  
    if n==0 :  
        return 1
```




Exemple

toggle

reset

Traduction immédiate des fonctions définies par récurrence.

Exemple (Factorielle)

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$

```
def fact(n):  
    if n==0 :  
        return 1  
    else:  
        return n * fact(n-1)
```



Exemple

toggle

reset

Traduction immédiate des fonctions définies par récurrence.

Exemple (Factorielle)

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$

```
def fact(n):  
    if n==0 : # Condition d'arrêt  
        return 1  
    else:  
        return n * fact(n-1)
```



Condition d'arrêt

toggle

reset

Définition (Condition d'arrêt)

Condition pour laquelle il n'y a pas d'appel récursif.



Condition d'arrêt

toggle

reset

Définition (Condition d'arrêt)

Condition pour laquelle il n'y a pas d'appel récursif.

Important

Toute fonction récursive doit comporter au moins une condition d'arrêt. Sinon : boucle infinie.



Condition d'arrêt

toggle

reset

Définition (Condition d'arrêt)

Condition pour laquelle il n'y a pas d'appel récursif.

Important

Toute fonction récursive doit comporter au moins une condition d'arrêt. Sinon : boucle infinie.

Remarque

Une fonction peut comporter plusieurs conditions d'arrêt.



Condition d'arrêt

toggle

reset

Exemple (Plusieurs conditions d'arrêt)

```
def fact(n):  
    if (n==0):  
        return 1  
    elif (n==1):  
        return 1  
    else :  
        return n * fact(n-1)
```



Condition d'arrêt

toggle

reset

Exemple (Plusieurs conditions d'arrêt)

```
def fact(n):  
    if (n==0):  
        return 1  
    elif (n==1):  
        return 1  
    else :  
        return n * fact(n-1)
```



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Récurtivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme récursif
Exemples
Notion de récursivité terminale

3 Horloge



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| |
|---------|
| |
| |
| fact(2) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| |
|---------------------|
| |
| |
| fact(2) = 2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| |
|---------------------|
| |
| fact(1) |
| fact(2) = 2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| | |
|---------|------------|
| | |
| fact(1) | =1*fact(0) |
| fact(2) | =2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| | |
|---------|------------|
| fact(0) | |
| fact(1) | =1*fact(0) |
| fact(2) | =2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| | |
|---------|------------|
| fact(0) | =1 |
| fact(1) | =1*fact(0) |
| fact(2) | =2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| | |
|---------|------------|
| | |
| fact(1) | =1 |
| fact(2) | =2*fact(1) |



Pile et récursivité

toggle

reset

- Pile d'appel : notion fondamentale pour la récursivité.
- Principe : tous les appels sont empilés, traités, puis dépilés.

```
def fact(n) {  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
}
```

| |
|-------------|
| |
| |
| fact(2) = 2 |



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs

Portée de variable

2 Réversivité

Définition

Pile d'appel

Mécanisme des traitements

Explosion combinatoire

Conception d'algorithme récursif

Exemples

Notion de réversivité terminale

3 Horloge



Exemple (Puissance d'entier : x^n)

```
def puissance(x,n):  
    if n == 0:  
  
        return 1  
    else:  
  
        return x* puissance (x, n-1)  
  
puissance(2,5)  
# sortie
```



Exemple (Puissance d'entier : x^{**n})

```
def puissance(x,n):  
    if n == 0:  
        print("cas de base n :", n)  
        return 1 # cas de base  
    else:  
  
        return x* puissance (x, n-1)
```

```
puissance(2,5)  
# sortie
```



Exemple (Puissance d'entier : x^{**n})

```
def puissance(x,n):  
    if n == 0:  
        print("cas de base n :", n)  
        return 1 # cas de base  
    else:  
        print("prétraitement pour n :", n)  
        return x* puissance (x, n-1)
```

```
puissance(2,5)
```

```
# sortie
```

```
pretraitement de n : 5
```

```
pretraitement de n : 4
```

```
pretraitement de n : 3
```

```
pretraitement de n : 2
```

```
pretraitement de n : 1
```

```
cas de base n : 0
```



Exemple (Puissance d'entier : x^n)

```
def puissance(x,n):
    if n == 0:
        print("cas de base n :", n)
        return 1 # cas de base
    else:
        y = x* puissance (x, n-1)
        print("post-traitement n :", n)
        return y
puissance(2,5)
# sortie

cas de base n : 0
post-traitement de n: 1
post-traitement de n : 2
post-traitement de n : 3
post-traitement de n : 4
post-traitement de n : 5
```



Post et pré-traitement

[toggle](#)[reset](#)

Exemple (Puissance d'entier : $x^{**}n$)

```
def puissance(x,n):  
    if n == 0:  
        print("cas de base n :", n)  
        return 1 # cas de base  
    else:  
        print("prétraitement pour n :", n)  
        y = x* puissance (x, n-1)  
        print("post-traitement n :", n)  
        return y  
puissance(2,5)  
# sortie  
pretraitement de n : 5  
pretraitement de n : 4  
pretraitement de n : 3  
pretraitement de n : 2  
pretraitement de n : 1  
cas de base n : 0  
post-traitement de n: 1  
post-traitement de n : 2  
post-traitement de n : 3  
post-traitement de n : 4  
post-traitement de n : 5
```



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Réversivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme réversif
Exemples
Notion de réversivité terminale

3 Horloge



Suite de Fibonacci

toggle

reset

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \geq 2, F_n = F_{n-1} + F_{n-2} \end{cases}$$



Suite de Fibonacci

toggle

reset

$$\left\{ \begin{array}{l} F_0 = 0 \\ F_1 = 1 \\ \forall n \geq 2, F_n = F_{n-1} + F_{n-2} \end{array} \right.$$

- Deux conditions d'arrêt



Suite de Fibonacci

toggle

reset

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \geq 2, F_n = F_{n-1} + F_{n-2} \end{cases}$$

- Deux conditions d'arrêt
- Une condition de récurrence



Suite de Fibonacci

toggle

reset

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \geq 2, F_n = F_{n-1} + F_{n-2} \end{cases}$$

```
def fibo(n):  
    if n==0:  
        return 0  
    elif n==1:  
        return 1  
    else  
        return fibo(n-1) + fibo(n-2)
```



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|--------|
| |
| |
| |
| $f(3)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|----------------------|
| |
| |
| |
| $f(3) = f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|----------------------|
| |
| |
| $f(2)$ |
| $f(3) = f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| | |
| | |
| $f(2)$ | $= f(1) + f(0)$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| | |
| $f(1)$ | |
| $f(2)$ | $= f(1) + f(0)$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| | |
| $f(1)$ | $= 1$ |
| $f(2)$ | $= f(1) + f(0)$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| $f(0)$ | |
| $f(1)$ | $= 1$ |
| $f(2)$ | $= f(1) + f(0)$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| $f(0)$ | $= 0$ |
| $f(1)$ | $= 1$ |
| $f(2)$ | $= f(1) + f(0)$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| | |
|--------|-----------------|
| | |
| | |
| $f(2)$ | $= 1$ |
| $f(3)$ | $= f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|----------------------|
| |
| $f(1)$ |
| $f(2) = 1$ |
| $f(3) = f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|----------------------|
| |
| $f(1) = 1$ |
| $f(2) = 1$ |
| $f(3) = f(2) + f(1)$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |

- Traduction immédiate de la formule de récurrence



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |

- Traduction immédiate de la formule de récurrence

⇒ peu efficace dans ce cas



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |

- Traduction immédiate de la formule de récurrence
- ⇒ peu efficace dans ce cas
- Nombre d'appel à F en $\Theta(2^n)$



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |

- Traduction immédiate de la formule de récurrence
- ⇒ peu efficace dans ce cas
- Nombre d'appel à F en $\Theta(2^n)$

Important

- Le nombre d'appels est limité par défaut, à 1000 appels.
⇒ **RuntimeError : maximum recursion depth exceeded ...**



Explosion de la pile d'appel

[toggle](#)[reset](#)

Évolution de la pile lors du calcul de $F(3)$:

| |
|------------|
| |
| |
| |
| $f(3) = 2$ |

- Traduction immédiate de la formule de récurrence
- ⇒ peu efficace dans ce cas
- Nombre d'appel à F en $\Theta(2^n)$

Important

- Le nombre d'appels est limité par défaut, à 1000 appels.
⇒ **RuntimeError : maximum recursion depth exceeded ...**
- Exemple : Augmenter la taille à 1500 :
`sys.setrecursionlimit(1500)`



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Réversivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme réversif
Exemples
Notion de réversivité terminale

3 Horloge



Conception d'algorithme récursif

toggle

reset

- Découper l'algorithme en étapes



Conception

toggle

d'algorithme

reset

récurusif

- Découper l'algorithme en étapes
- Déterminer les règles de passage d'une étape à l'autre



Conception d'algorithme récursif

toggle

reset

- Découper l'algorithme en étapes
 - Déterminer les règles de passage d'une étape à l'autre
- ⇒ l'étape n dépend de l'étape $n - 1$ (éventuellement $n - 2$)



Conception

toggle

d'algorithme

reset

récurif

- Découper l'algorithme en étapes
 - Déterminer les règles de passage d'une étape à l'autre
- ⇒ l'étape n dépend de l'étape $n - 1$ (éventuellement $n - 2$)
- Déterminer les conditions d'arrêt



Conception

toggle

d'algorithme

reset

récurif

- Découper l'algorithme en étapes
 - Déterminer les règles de passage d'une étape à l'autre
- ⇒ l'étape n dépend de l'étape $n - 1$ (éventuellement $n - 2$)
- Déterminer les conditions d'arrêt



- Découper l'algorithme en étapes
 - Déterminer les règles de passage d'une étape à l'autre
- ⇒ l'étape n dépend de l'étape $n - 1$ (éventuellement $n - 2$)
- Déterminer les conditions d'arrêt

Remarque

Tout algorithme récursif peut s'écrire de manière itérative, et réciproquement.



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs

Portée de variable

2 Réversivité

Définition

Pile d'appel

Mécanisme des traitements

Explosion combinatoire

Conception d'algorithme récursif

Exemples

Notion de réversivité terminale

3 Horloge



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié
- Principe :



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié
- Principe :
 - comparer l'élément recherché avec le milieu du tableau



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié
- Principe :
 - comparer l'élément recherché avec le milieu du tableau
 - si $>$: recherche dans le tableau \Leftrightarrow recherche dans la partie droite



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié
- Principe :
 - comparer l'élément recherché avec le milieu du tableau
 - si $>$: recherche dans le tableau \Leftrightarrow recherche dans la partie droite
 - si $<$: recherche dans le tableau \Leftrightarrow recherche dans la partie gauche



Recherche par dichotomie

[toggle](#)[reset](#)

- Recherche d'un élément dans un tableau trié
- Principe :
 - comparer l'élément recherché avec le milieu du tableau
 - si $>$: recherche dans le tableau \Leftrightarrow recherche dans la partie droite
 - si $<$: recherche dans le tableau \Leftrightarrow recherche dans la partie gauche
 - si $=$: élément trouvé (condition d'arrêt)



Recherche par dichotomie

toggle

reset

- Recherche d'un élément dans un tableau trié
- Principe :
 - comparer l'élément recherché avec le milieu du tableau
 - si $>$: recherche dans le tableau \Leftrightarrow recherche dans la partie droite
 - si $<$: recherche dans le tableau \Leftrightarrow recherche dans la partie gauche
 - si $=$: élément trouvé (condition d'arrêt)
 - autre condition d'arrêt : taille du tableau ≤ 1



Recherche par dichotomie

toggle

reset

```
def rech(tab, val, deb, fin):  
    n = (deb + fin)//2
```



Recherche par dichotomie

toggle

reset

```
def rech(tab, val, deb, fin):  
    n = (deb + fin)//2  
    if tab[n] == val:  
        return True
```



Recherche

par

dichotomie

toggle

reset

```
def rech(tab, val, deb, fin):  
    n = (deb + fin)//2  
    if tab[n] == val:  
        return True  
    elif deb >= fin :  
        return False
```



```
def rech(tab, val, deb, fin):  
    n = (deb + fin)//2  
    if tab[n] == val:  
        return True  
    elif deb >= fin :  
        return False  
    elif tab[n] > val:  
        return rech(tab, val, deb, n-1)  
    else:  
        return rech(tab, val, n+1, fin)
```



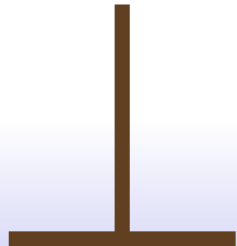
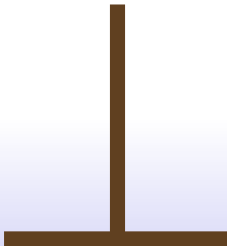
Tours de Hanoï

toggle

reset

Principe :

- Disques empilés : tour
- Ne jamais empiler un disque sur un disque plus petit
- Déplacer un seul disque à la fois





Tours de Hanoï : résolution

[toggle](#)[reset](#)

- Résolution par une fonction récursive
- Étapes de l'algorithme : hauteur de la tour
- Hauteur 0 : évident
- Récurrence :
 - on suppose savoir déplacer une tour de hauteur $h - 1$
 - on veut déplacer une tour de hauteur h de A vers B
 - déplacer les $h - 1$ premiers éléments de A vers C
 - déplacer l'élément restant de A vers B
 - déplacer les $h - 1$ premiers éléments de C vers B



Tours de Hanoï : algorithme

[toggle](#)[reset](#)

Procédure hanoi(entier n, entier source, entier dest, entier tmp)

```
/* n : hauteur de la tour */
/* source, dest, tmp : position d'origine, finale et
   intermédiaire de la tour à déplacer */
si  $n > 0$  alors
    | hanoi (n-1, source, tmp, dest) ;
    | déplacer(source, dest) ;
    | hanoi (n-1, tmp, dest, source) ;
fin
```



Étude de l'algorithme tours de Hanoï

toggle

reset

$C(n)$: nombre d'opération pour déplacer une tour de hauteur n



Étude de l'algorithme tours de Hanoï

toggle

reset

$C(n)$: nombre d'opération pour déplacer une tour de hauteur n

$$\begin{cases} C(n+1) = 2C(n) + 1 \\ C(0) = 0 \end{cases}$$



Étude de l'algorithme tours de Hanoï

[toggle](#)[reset](#)

$C(n)$: nombre d'opération pour déplacer une tour de hauteur n

$$\begin{cases} C(n+1) = 2C(n) + 1 \\ C(0) = 0 \end{cases}$$

Résolution : $C(n) = 2^n - 1$



Étude de l'algorithme tours de Hanoï

[toggle](#)[reset](#)

$C(n)$: nombre d'opération pour déplacer une tour de hauteur n

$$\begin{cases} C(n+1) = 2C(n) + 1 \\ C(0) = 0 \end{cases}$$

Résolution : $C(n) = 2^n - 1$

Légende des tours de Hanoï : dans un temple Bouddhiste, des moines ont reçu pour mission de déplacer une tour de Hanoï de 64 disques. Lorsqu'ils l'auront déplacée, le monde tombera en poussière.



Étude de l'algorithme tours de Hanoï

toggle

reset

$C(n)$: nombre d'opération pour déplacer une tour de hauteur n

$$\begin{cases} C(n+1) = 2C(n) + 1 \\ C(0) = 0 \end{cases}$$

Résolution : $C(n) = 2^n - 1$

Légende des tours de Hanoï : dans un temple Bouddhiste, des moines ont reçu pour mission de déplacer une tour de Hanoï de 64 disques. Lorsqu'ils l'auront déplacée, le monde tombera en poussière.

$$2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$$

Un déplacement par seconde \rightsquigarrow 580 milliards d'années



Procédure triSeg(tableau_entier tab, entier debut, entier fin)

si *debut* < *fin* alors

|



Procédure triSeg(tableau_entier tab, entier debut, entier fin)

si *debut* < *fin* **alors**

 choisir élément pivot p entre debut et fin ;



Procédure triSeg(tableau_entier tab, entier debut, entier fin)

si *debut* < *fin* alors

 choisir élément pivot p entre debut et fin ;
 placer pivot en i , \leq pivot avant, \geq pivot après ;



Procédure triSeg(tableau_entier tab, entier debut, entier fin)

si *debut* < *fin* **alors**

- choisir élément pivot p entre debut et fin ;
- placer pivot en i , \leq pivot avant, \geq pivot après ;
- triSeg (tab, deb, $i-1$) ;
- triSeg (tab, $i+1$, fin) ;

fin



Procédure triSeg(tableau_entier tab, entier debut, entier fin)

si *debut* < *fin* **alors**

si *fin* − *debut* ≤ 1 // cas particulier : 2 éléments à trier

alors

si *tab*[*debut*] > *tab*[*fin*] **alors**

 permuter *tab*[*debut*], *tab*[*fin*] ;

fin

sinon

 choisir élément pivot *p* entre *debut* et *fin* ;

 placer pivot en *i*, ≤ pivot avant, ≥ pivot après ;

 triSeg (*tab*, *deb*, *i*-1) ;

 triSeg (*tab*, *i*+1, *fin*) ;

fin

fin



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Réversivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme réversif
Exemples
Notion de réversivité terminale

3 Horloge



Récurtivité terminale

toggle

reset

- A chaque appel récursif : empilement de données



Récurtivité terminale

toggle

reset

- A chaque appel récursif : empilement de données
- ⇒ limite au nombre d'appels possibles



Récurtivité terminale

toggle

reset

- A chaque appel récursif : empilement de données
- ⇒ limite au nombre d'appels possibles
- Solution : récursivité terminale



Récurtivité terminale

toggle

reset

- A chaque appel récursif : empilement de données
- ⇒ limite au nombre d'appels possibles
- Solution : récursivité terminale

Définition (Récursivité terminale)

Une fonction est dite récursive terminale s'il n'y a aucune opération sur ses appels récursifs.



Récurtivité terminale

toggle

reset

- A chaque appel récursif : empilement de données
- ⇒ limite au nombre d'appels possibles
- Solution : récursivité terminale

Définition (Récursivité terminale)

Une fonction est dite récursive terminale s'il n'y a aucune opération sur ses appels récursifs.

Exemple (Fonction récursive non terminale)

Factorielle :

```
return n*fact(n-1);
```



Écriture de fonction récursive terminale

toggle

reset

- Idée générale : utiliser un accumulateur
- Paramètre qui contient le résultat intermédiaire

Exemple (Factorielle récursive terminale)

```
def fact(int n, int acc):  
    if n==0 :  
        return acc  
    else:  
        return fact(n-1, n*acc)
```



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(4, 1);
```

| | |
|-----|---|
| acc | 1 |
| n | 4 |



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(3, 4);
```

| | |
|-----|---|
| acc | 4 |
| n | 3 |



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(2, 12);
```

| | |
|-----|----|
| acc | 12 |
| n | 2 |



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(1, 24);
```

| | |
|-----|----|
| acc | 24 |
| n | 1 |



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(0, 24);
```

| | |
|-----|----|
| acc | 24 |
| n | 0 |



Appel :

```
fact(4, 1)
```

Si le langage sait optimiser la récursivité terminale, pas d'empilement d'appels \Rightarrow Modification des valeurs dans la pile.

```
fact(0, 24);
```

| | |
|-----|----|
| acc | 24 |
| n | 0 |

Remarque

Python n'optimise pas la récursivité terminale



Sommaire

1 Rappel

Paramètres formels/paramètres effectifs
Portée de variable

2 Récursivité

Définition
Pile d'appel
Mécanisme des traitements
Explosion combinatoire
Conception d'algorithme récursif
Exemples
Notion de récursivité terminale

3 Horloge



Horloge

toggle

reset

