Fonctions Récursives

Cours de CPGE PSI, 4 novembre 2026

Adeline Pierrot

Une fonction peut en appeler une autre.

Exemple

```
def f(x):
    if x%2 == 0:
        return x//2
    else:
        return x - 1

def g(x):
    x = x + 1
    return f(3*x)

print(g(4))
```

Une fonction peut en appeler une autre.

Exemple

```
def f(x):
    if x%2 == 0:
        return x//2
    else:
        return x - 1

def g(x):
    x = x + 1
    return f(3*x)

print(g(4))
```

Notion de pile d'appels (ou pile d'exécution)

Définition

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

Définition

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

Exemple

Fonction récursive avec **return**:

```
def puissance2(n):
    if n == 0:
        return 1
    else :
        return 2 * puissance2(n-1)
```

Exemple

Fonction récursive sans **`return`**:

```
def continuer():
    reponse = input("Continuer ? ")
    if reponse == "oui":
        continuer()
```

Exemple

Fonction récursive sans **`return`**:

```
def continuer():
    reponse = input("Continuer ? ")
    if reponse == "oui":
        continuer()
```

Remarque : si on remplace le **`if`** par un **`while`** dans la fonction ci-dessus, cela fait une boucle infinie !

L'endroit où on fait l'appel récursif est important:

```
def nombres(n):
    x = input("Nombre ? ")
    print(x)
    if (n>1):
        nombres(n-1)

def nombres2(n):
    x = input("Nombre ? ")
    if (n>1):
        nombres2(n-1)
    print(x)
```

L'endroit où on fait l'appel récursif est important:

```
def nombres(n):
    x = input("Nombre ? ")
    print(x)
    if (n>1):
        nombres(n-1)

def nombres2(n):
    x = input("Nombre ? ")
    if (n>1):
        nombres2(n-1)
    print(x)
```

La première fonction affiche les nombres dans l'ordre dans lequel ils ont été entrés par l'utilisateur, tandis que la seconde les affiche dans l'ordre inverse !

Certains problèmes se résolvent naturellement avec des fonctions récursives :

```
def dichotomie(t,v):
    n = len(t)
    if n == 0:
        return False
    if t[n//2] == v:
        return True
    elif t[n//2] > v:
        return dichotomie( t[0:n//2], v )
    else :
        return dichotomie( t[n//2+1:n], v )
```

Certains problèmes se résolvent naturellement avec des fonctions récursives :

```
def dichotomie(t,v):
    n = len(t)
    if n == 0:
        return False
    if t[n//2] == v:
        return True
    elif t[n//2] > v:
        return dichotomie( t[0:n//2], v )
    else :
        return dichotomie( t[n//2+1:n], v )
```

Remarque : Ne pas oublier les deux cas de base (pour pouvoir renvoyer True ou False)

Certains problèmes se résolvent naturellement avec des fonctions récursives :

```
def dichotomie(t,v):
    n = len(t)
    if n == 0:
        return False
    if t[n//2] == v:
        return True
    elif t[n//2] > v:
        return dichotomie( t[0:n//2], v )
    else :
        return dichotomie( t[n//2+1:n], v )
```

Remarque : Ne pas oublier les deux cas de base (pour pouvoir renvoyer True ou False)

Remarque : Cette version n'a pas une bonne complexité (à cause de la recopie due au slicing)

Remarques

- Le principe des fonctions récursives est similaire au principe de la preuve par récurrence.

Remarques

- Le principe des fonctions récursives est similaire au principe de la preuve par récurrence.
- Attention à la terminaison des fonctions récursives (erreur RecursionError: maximum recursion depth exceeded):
  - Ne pas oublier le(s) cas de base
  - Ne pas se tromper en faisant l'appel récursif sur n au lieu de $n-1$ (ou d'une autre valeur différente de n)

Remarques

- Le principe des fonctions récursives est similaire au principe de la preuve par récurrence.
- Attention à la terminaison des fonctions récursives (erreur RecursionError: maximum recursion depth exceeded):
  - Ne pas oublier le(s) cas de base
  - Ne pas se tromper en faisant l'appel récursif sur n au lieu de $n-1$ (ou d'une autre valeur différente de n)
- Pour l'appel récursif, ne pas écrire ~~f(n) = ...~~ mais utiliser **`return`** (ou un appel seul sur sa ligne pour une procédure)

Remarques

- Le principe des fonctions récursives est similaire au principe de la preuve par récurrence.
- Attention à la terminaison des fonctions récursives (erreur RecursionError: maximum recursion depth exceeded):
  - Ne pas oublier le(s) cas de base
  - Ne pas se tromper en faisant l'appel récursif sur n au lieu de $n-1$ (ou d'une autre valeur différente de n)
- Pour l'appel récursif, ne pas écrire ~~f(n) = ...~~ mais utiliser **`return`** (ou un appel seul sur sa ligne pour une procédure)
- Pour prouver qu'une fonction récursive est correcte, on utilise une preuve par récurrence

Remarques

- Le principe des fonctions récursives est similaire au principe de la preuve par récurrence.
- Attention à la terminaison des fonctions récursives (erreur RecursionError: maximum recursion depth exceeded):
  - Ne pas oublier le(s) cas de base
  - Ne pas se tromper en faisant l'appel récursif sur n au lieu de $n-1$ (ou d'une autre valeur différente de n)
- Pour l'appel récursif, ne pas écrire ~~f(n) = ...~~ mais utiliser **`return`** (ou un appel seul sur sa ligne pour une procédure)
- Pour prouver qu'une fonction récursive est correcte, on utilise une preuve par récurrence
- Tout ce qui peut se faire avec une fonction récursive peut aussi se faire avec une fonction non récursive en utilisant des boucles (fonction itérative).