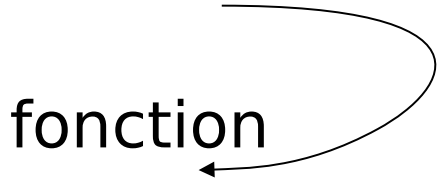
RECURSIVITE

Projet informatique – Eric Thomas

Récusivité concept

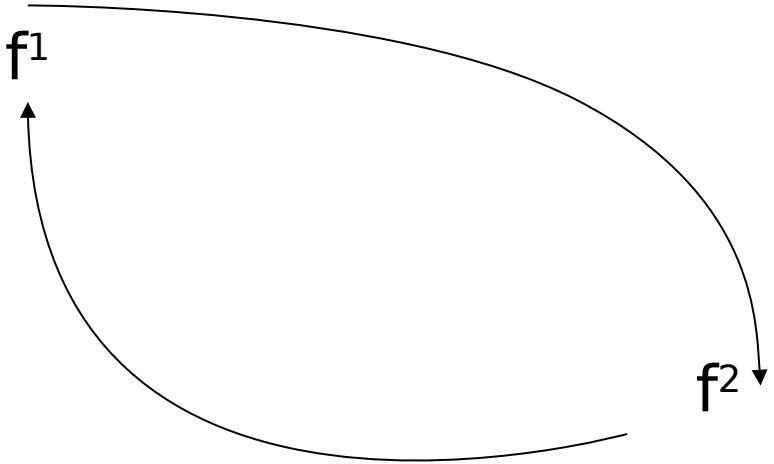
Une technique de programmation où une fonction s’appelle elle-même indirectement ou directement

Appel Direct

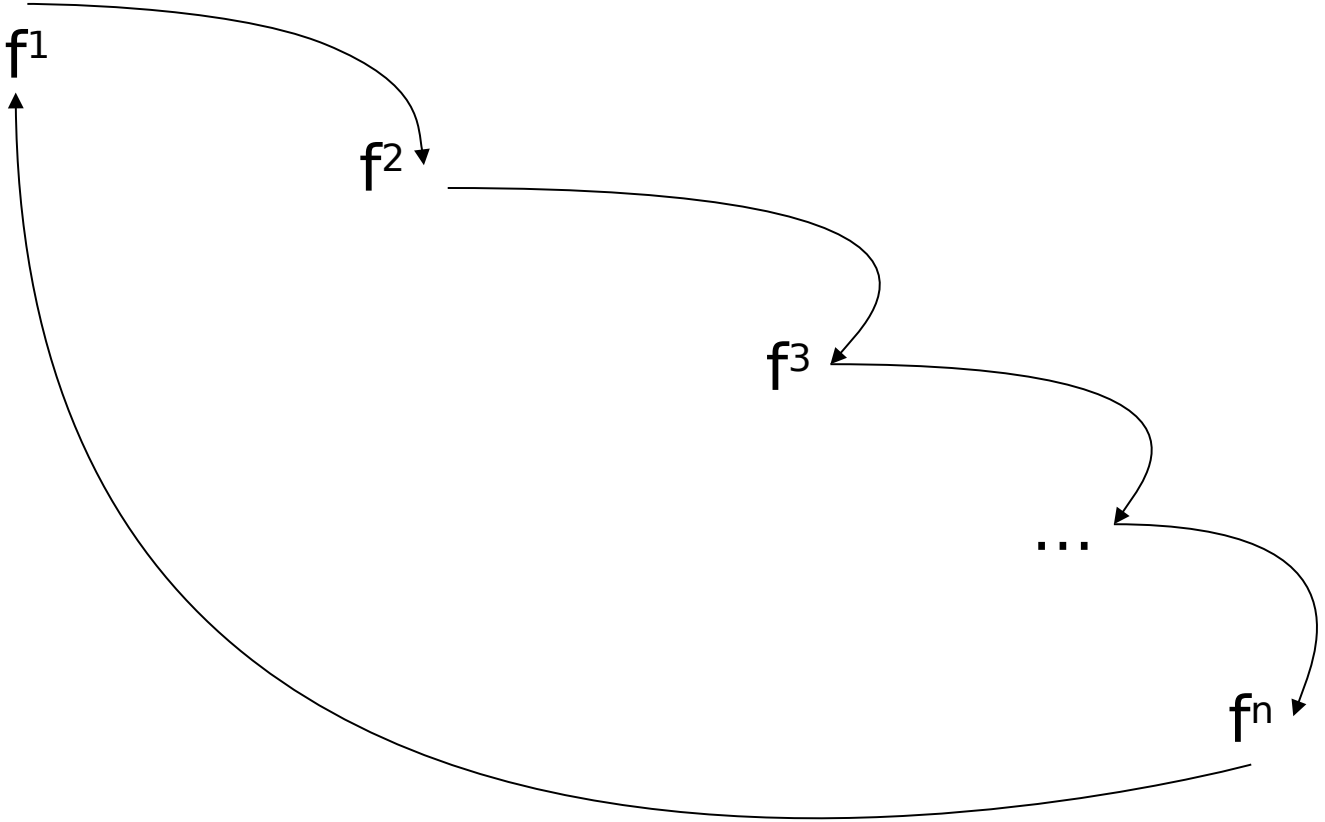



```
def jouer ():
    …
jouer ()
…
```

Appel indirect



Appel indirect complexe



Exemple appel indirect

```
def jouer1():
    jouer2()

def jouer2():
    jouer1()

jouer1()
```

Récursivité « Tail »

Une récursivité est dites « Tail » quand la dernière ligne est un appel récursif à la fonction

```
def tail(no):
    if (no <= 3):
        print (no)
        tail(no+1)
    return()

tail(1)
```

Récursivité non-tail

Une récursivité est « non-Tail » quand la dernière ligne n'est pas un appel récursif à la fonction

```
def nonTail(no):s
    if (no < 3):
        nonTail(no+1)
    print(no)
    return()

nonTail(1)
```

Quand utiliser la récursivité ?

- Quand le problème peut être divisé en étapes
- Le résultat d'un étape peut être utilisé dans l'étape précédente
- Toutes les étapes permettent de résoudre le problème

Quand ne pas utiliser la récursivité ?

Quand une boucle permet de résoudre le problème

Bénéfice de la récursivité

- Solution simple et plus élégante pour beaucoup de problèmes
- Code plus lisible

Limites de la récursivité

- Utilisation de plus de mémoire
- Moins rapide qu’une boucle

Les erreurs courantes

- Boucles infinies
- Consommation de ressources trop importantes

Boucle infinie

```
def sum (no):
    if (no == 1):
        return 1
    else:
        return (no + sum (no))
```

Cette fonction récursive ne convergera jamais faire la condition de fin

Boucle infinie (suite)

```
def sum (no):
  return (no + sum (no - 1))
```

Pas d'arrêt!

Utilisation de trop de ressources

```
def fun(no):
    print(no)
    aList = []
    for i in range (0, 10000000, 1):
       aList.append("*")
    no = no + 1
    fun(no)

fun(1)
```