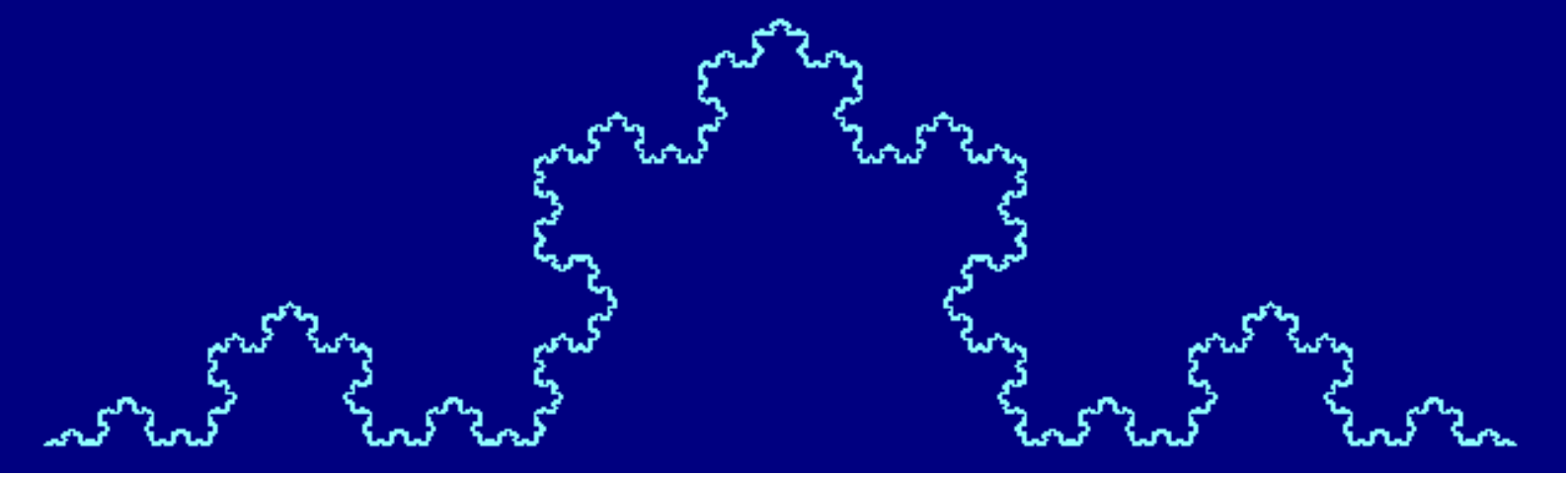

Activité

On considère la situation suivante.

Source : https ://glassus.github.io/terminale_nsi/T2_Programmation/

? Décrire la situation, en précisant quels sont les éléments qui ont permis à notre héros déterminer le prix du livre.

Objectifs

Dans ce chapitre, il faudra :

- Connaître les éléments caractéristiques d'une fonction récursive,
- Prédire le résultat d'un appel à une fonction récursif,
- Déterminer le nombre d'appels à la fonction que déclenche un appel à une fonction récursif,
- Écrire un programme récursif.

Cours

Définition

Définition

Récursivité

On dira d'une fonction qu'elle est `récursive` si cette fonction s'appelle elle-même. C'est à dire que dans définition de la fonction, on fait un appel à cette fonction.

! Cette définition n'est pas suffisante pour produire un code fonctionnel.

Exemple

On considère la fonction `probleme`, qui est **récursive** car elle s'appelle elle-même.

```
1 def probleme():
2     print("Code non fonctionnel")
3     probleme()
```

Et son exécution, donnerait :

```
1 probleme()
2 >>> Code non fonctionnel
3 >>> Code non fonctionnel
4 >>> Code non fonctionnel
5 >>> ...
```

Les appels à la fontion `probleme` ne s'arreteront jamais et l'interpréteur renverra une erreur.

Voir l'éxécution de ce code dans pythontutor

Eléments caractéristiques

i Pour être sur qu'une fonction récursive est bien définie, elle doit comporter certains éléments :

- une condition d'arrêt. C'est la terminaison des appels
- la fonction contient un ou plusieurs appel.s à elle-même à l'intérieur de celle-ci,
- une situation de convergence vers la terminaison.

Exemple

On considère la fonction `somme` qui prend en paramètre un entier naturel `n` et qui renvoie la somme des entiers inférieurs ou égaux à `n`.

Par exemple : `somme(5)` = 0 + 1 + 2 + 3 + 4 + 5

Exemple

De manière récursive, on pourrait décrire la fonction ainsi.

> Fonction **somme**
> **Paramètres :**
> — `n`
>
> **Instructions :**
> **si** $n = 0$ **alors**
> | renvoyer 0
> **sinon**
> | renvoyer n + **somme**(n - 1)

Une fois codée, en Python on obtiendrait le code suivant :

```
1  def somme(n):
2      if n == 0:
3          return n
4      else:
5          return n + somme(n - 1)
```

Regardons ce que fait un appel à cette fonction `somme` avec l'argument 5.

`somme(5)` → 5 + `somme(4)`
→ 5 + (4 + `somme(3)`)
→ 5 + (4 + (3 + `somme(2)`))
→ 5 + (4 + (3 + (2 + `somme(1)`)))
→ 5 + (4 + (3 + (2 + (1 + `somme(0)`))))
→ 5 + (4 + (3 + (2 + (1 + (0)))))
→ 5 + (4 + (3 + (2 + (1))))
→ 5 + (4 + (3 + (3)))
→ 5 + (4 + (6))
→ 5 + (10)
→ 15

La pile d'appels

? L'éxécution de la fonction `somme(5)` fonctionne parfaitement bien, mais l'éxécution de la fonction `somme(1000)` génére une erreur. On peut se demander pourquoi ?

Lors d'un appel à un fonction récursive, le processeur utilise une structure de `pile` pour stocker les résultats temporaire de chaque appel. Il va donc empiler les réusltats temporaire et une fois arrivée à la terminaison, il pourra `dépiler` chaque appel.

! En pyhton, la taille maximale de la pile d'appels est limitée à 986.
Quand l'on dépasse cette valeur on obtient une erreur de type :

```
RecursionError: maximum recursion depth exceeded in comparison
```

Exercices

Exercice 1 ★

On considère la fonction `mystere` qui prend en paramètre un nombre entiers naturel `n` définit par le code ci-dessous.

```
def mystere(n):
    if n  == 0:
        return 1
    else :
        return 5*(n-1)
```

1. La fonction `mystere` est-elle récursive ? Justifier.
2. Déterminer la relation de entre `mystere(n)` et `mystere(n-1)`
3. Complète la pile d'éxécution de l'appel `mystere(4)`. Combien d'appels à la fonction l'appel initial déclenche-t-il ?

	mys(3) : 5 × mys(2)			
mys(4) : 5 × mys(3)	mys(4) : 5 × mys(3)			

Exercice 2 ★ Algorithme d'Euclide

On considère la fonction `pgcd` qui prend en paramètre deux entiers, `a` et `b`, tel que `a>b` et qui renvoie le pgcd (plus grand diviseur commun) de `a` et de `b`.

```
def pgcd(a,b):
    if a % b == 0
        return b
    else:
        return pgcd(b, a % b)
```

1. La fonction `pgcd` est-elle récursive ? Justifier
2. Quelle est la condition d'arret de la fonction `pgcd` ?
3. Quelle valeur va retourner l'appel à la fonction `pgcd(24,18)` ? Combien d'appels à la fonction l'appel initial déclenche-t-il ?