

Langages et programmation (1)

Réversivité

Résoudre un problème de façon récursive c'est décomposer un problème en sous-problèmes identiques de plus en plus petits jusqu'à obtenir un problème suffisamment petit pour qu'il puisse être résolu de manière triviale.

Les trois règles d'un algorithme récursif :

- Il s'**appelle lui même** ;
- Il doit avoir un **état trivial**, ce qui permet de définir une condition d'arrêt ;
- Il doit conduire vers cet **état d'arrêt**, pour éviter les boucles infinies.



Initialisation de la variable

Boucle FOR

```
def somme_for(vec):  
    somme = 0  
    for i in range(len(vec)):  
        somme += vec[i]  
    return somme
```

Condition d'arrêt

S'arrête quand toutes les valeurs de i ont été passées en revue.

Boucle WHILE

```
def somme_while(vec):  
    somme = 0  
    while len(vec)>0:  
        somme += vec[0]  
        vec = vec[1:]  
    return somme
```

S'arrête quand la condition du while est obtenue. Attention aux boucles infinies!

Récursif

```
def somme_recu(vec):  
    if len(vec) == 1:  
        return vec[0]  
    else :  
        return vec[0] + somme_recu(vec[1:])
```

S'arrête quand la condition du if est obtenue. Attention aux boucles infinies!

Pour éviter de surcharger un langage informatique, on utilise des briques logiciels appelées **modules** (= bibliothèques). Un module est un fichier qui contient des variables, fonctions, objets, méthodes...



```
from math import *
```

Importer toute les fonctions de la bibliothèque math

```
from random import uniform as uf
```

Importer la fonction uniform de la bibliothèque random en donnant le nom uf à la fonction

Une **API** (*Application Programming Interface*) est une interface de programmation d'application qui facilite les échanges entre différents programmes informatiques.

Documentation de programme (commentaires et docstrings)

La bonne structure d'un code (nom de fonctions bien choisies, conventions conservées, code facile à lire) doit être complétée par une **documentation** (**docstring**) et par l'anticipation des erreurs des utilisateurs (**assertion**).

DOCSTRINGS

Ce texte d'aide est entouré par ''' et suit la convention PEP 257

```
def est_palindrome(mot):
    '''Test si un mot est un palindrome

    Cette fonction test si un mot est un palindrome

    :param str mot: Le mot à tester
    :return: Retourne un booléen qui dit si le mot est un palindrome
    :rtype : Bool
    '''
    mot=mot.lower() # tout en minuscules
    for i in range(len(mot)//2):
        if mot[i]!=mot[-i-1]:
            return False
    return True
```

COMMENTAIRES

Un commentaire est précédé d'un # et donne des indications sur le code plus que sur ce que fait la fonction.

Gestion de bug (assertion)

assert condition, 'Message'

Si la condition n'est pas remplie, le programme s'arrête avec une erreur d'assertion en indiquant le Message écrit entre guillemets.

```
def sup_10(x):
    assert type(x) == int, "Il faut que x soit un entier (type int)"
    return x > 10
```

Condition
Message

```
print(sup_10(20))    # True
print(sup_10(2))     # False
print(sup_10("2"))   # AssertionError
```