

Initiation à l'algorithmique et programmation

2025-2026

Travaux Pratiques 2

Exercice 1. *Imprimer des nombres*

1. Ecrivez une fonction utilisant la récursivité pour imprimer des nombres compris entre n et 0.
2. Ecrivez une fonction utilisant la récursivité pour imprimer les nombres de 0 à n . (Il vous suffit de changer une ligne dans le programme) du problème 1).

Exercice 2. *Longueur d'une Chaîne*

Créez une fonction récursive `longueur_chaine` qui retourne la longueur d'une chaîne de caractères sans utiliser la fonction intégrée `len()`.

Exercice 3. *Inversion d'une Chaîne*

Implémentez une fonction récursive `inverse_chaine` qui prend en entrée une chaîne de caractères et renvoie la chaîne inversée.

Exercice 4. *Parenthèses Équilibrées*

Une chaîne de caractères contenant uniquement des parenthèses "(" et ")" est dite *équilibrée* si chaque parenthèse ouvrante "(" a une parenthèse fermante correspondante ")" qui apparaît après elle, et si chaque paire de parenthèses est correctement imbriquée et appariée.

Écrire une fonction récursive `parentheses_equilibrees` qui vérifie si les parenthèses dans une chaîne de caractères sont équilibrées.

Exemples :

```
> > > parentheses_equilibrees("")
True
> > > parentheses_equilibrees("()")
True
> > > parentheses_equilibrees("(())")
False
> > > parentheses_equilibrees("((()))")
True
```

Exercice 5. *Sac à dos*

Le problème du sac à dos consiste à essayer de remplir complètement un sac à dos ayant une capacité maximale de S kg avec des objets de différents poids non-nuls donnés dans un ensemble E .

On représente l'ensemble E par une liste d'entiers distincts. Le but est d'écrire une fonction qui détermine s'il existe une solution F ou non. Un algorithme possible est le suivant :

- Si $S = 0$ alors il existe une solution ;
- Sinon, pour chaque entier k dans E :
 - s'il existe une solution pour $S - k$ n'utilisant pas k , alors il existe une solution pour s ;
 - sinon, on cherche une solution pour S n'utilisant pas l'entier k .

Questions :

1. Détailler le déroulement de l'algorithme ci-dessus pour $S = 8$ et $E = \{2, 4, 5, 8, 6, 1, 9\}$.
2. Écrire une fonction `sac_a_dos_booleen(somme, entiers, index)` qui renvoie `True` s'il existe une solution au problème pour le poids total `somme` en utilisant les éléments de `entiers` à partir de l'indice `index`, et `False` sinon.

3. Écrire une fonction `sac_a_dos_sol(somme, entiers, index)` qui renvoie une solution au problème sous la forme d'une liste d'entiers, ou `None` s'il n'en existe aucune.
4. Écrire une fonction `sac_a_dos_liste_sol(somme, entiers, index)` qui renvoie la liste de toutes les solutions au problème.

Exercice 6. *Monnayeur*

Ce problème est une variante du précédent, dans laquelle il est possible d'utiliser plusieurs fois la même valeur. On dispose de pièces de différentes valeurs en nombre illimité, et on veut rendre la monnaie pour une somme entière $S \geq 0$. La liste des valeurs de pièces disponibles sera fournie sous la forme d'une liste d'entiers *pieces* (ce n'est pas forcément l'habituelle suite 1, 2, 5, 10, 20, 50, 100, . . .).

Exemples :

```
> > > monnayeur_nb(127, [1, 10, 97, 100])
```

```
4
```

1. Écrire une fonction `monnayeur_nb(somme, pieces, index)` qui renvoie le nombre minimum de pièces nécessaires pour atteindre *somme* en n'utilisant que des pièces de la liste *pieces* (donnée par ordre décroissant, et contenant forcément 1) à partir de l'indice *index*, ou `None` s'il n'y a pas de solution.
2. Écrire une fonction `monnayeur_sol(somme, pieces, index)` qui renvoie une solution de longueur minimale sous la forme d'une liste de valeurs de pièces (qui contiendra en général des entiers répétés), ou `None` s'il n'y a pas de solution.