# Révisions PTSI

### Objectifs

Reprendre python après une longue interruption de cours.

### Bon usage

On rappelle qu'un programme de qualité utilise des noms de variable explicités et est pertinemment commenté afin de pouvoir être repris plus tard ou par quelqu'un d'autre. On créé un fichier avec un nom explicite, par exemple `TP1_NOM.py`.

Le `Memento Python 3 pour l'Oral de la Banque PT` est disponible uniquement à l'oral et non aux écrits. Il pourra néanmoins vous être utile pour réviser rapidement durant l'année.

## Introduction

1. Ecrire une fonction `pwd_generator(Alpha, alpha, digit, spe)` qui prend comme arguments les cinq entiers `Alpha`, `alpha`, `digit` et `spe` et qui retourne un mot de passe sous forme d'une chaîne de caractères comportant :
   - `Alpha` lettres majuscules
   - `alpha` lettres minuscules
   - `digit` chiffres
   - `spe` caractères parmi #@$%?_

   On utilisera les méthodes `choices` et `shuffle` après les avoir importé depuis la bibliothèque `random`.
2. Écrire une fonction `is_prime(n)` qui prends en argument un entier naturel `n` et retourne un booléen traduisant la primalité de `n`.
3. Écrire une fonction `next_prime(n)` qui retourne le plus petit des nombres premiers strictement supérieur à l'entier `n`.
4. Écrire un script construisant un tableau `prime` contenant les `100 000` plus petits nombres premiers.
5. Implémenter la fonction `factorielle(n)` par un codage récursif.

## Les algorithmes gloutons

Les algorithmes gloutons sont utilisés dans des problèmes d'optimisation. Un problème d'optimisation consiste à déterminer les valeurs des paramètres permettant de :

- minimiser ou maximiser une fonction objectif ;
- satisfaire une ou des fonctions contraintes (il existe des problèmes avec ou sans contrainte).

Ils correspondent à une solution optimale obtenue en effectuant une suite de meilleurs choix pour chaque étape de l'algorithme (à chaque étape, on fait le meilleur choix possible).

Il n'y a pas de retour en arrière : quand un choix est fait à une étape, il n'est pas modifié ultérieurement et il ne modifie pas les choix précédents.

L'autre caractéristiques des algorithmes gloutons est que lorsqu'un choix est fait, on tente de résoudre un problème plus petit. On appelle cela la progression descendante.

## Rendu de monnaie

Le problème du rendu de monnaie est un problème d'algorithmique qui s'énonce de la façon suivante : étant donné un système de monnaie, comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?

Dans la zone euro, le système en vigueur, en mettant de côté les centimes d'euros, met à disposition des pièces ou billets de : 1€, 2€, 5€, 10€, 20€, 50€, 100€, 200€, 500€.

On suppose que nous avons à notre disposition un nombre illimité de ces pièces ou billets.

## Premier contact

Supposons que la somme à rendre est de 7€. Faire la liste de toutes les façons de rendre une telle somme. Quel est le nombre minimal de pièces ou billets à rendre ? Supposons que la somme à rendre soit maintenant de 463 €. Trouver une méthode permettant d'effectuer un rendu de monnaie en utilisant le moins possible de pièces sans avoir à énumérer toutes les possibilités. Combien de pièces ou de billets semblent nécessaires ?

Il est très probable qu'en résolvant cette dernière question, vous ayez mis en oeuvre la méthode dite "gloutonne" : tant qu'il reste quelque chose à rendre, on choisit la plus grosse pièce (ou plus gros billet) qu'on peut rendre (sans rendre trop).

A chaque étape d'une méthode gloutonne, le choix fait est celui qui maximise un critère. Le critère est ici : diminuer au maximum la somme qu'il reste à rendre.

## Est-ce toujours une bonne stratégie ?

Supposons que le système de monnaie mette à disposition uniquement des pièces de 1€, 3€et 4€(on ne tient toujours pas compte des centimes d'euros) et supposons que nous devions rendre la somme de 6 €. Combien de pièces l'algorithme glouton nous amenerait-il à rendre ? Est-ce optimal ?

Pour le système monétaire européen (qui ne comporte aucune pièce ni aucun billet de 3€ni de 4€), une telle situation ne se produirait pas ! Pour cette raison, un tel système de monnaie est qualifié de canonique. Sans s'en rendre compte, tout individu met en oeuvre un algorithme glouton pour rendre la monnaie.

## Mise en oeuvre d'une solution

Pour la zone euro on considère une liste qui contient les différentes pièces et billets. Notez que les éléments sont triés par ordre décroissant vis à vis du critère.

```
systeme_euro = [500, 200, 100, 50, 20, 10, 5, 2, 1]
```

6. Écrire une fonction `rendu_monnaie(somme, systeme)` qui prend en entrée la somme à rendre et le système de monnaie contenant les valeurs des pièces et billets et qui renvoie la liste des billets et pièces. Par exemple :

```
>>> rendu_monnaie(97, systeme_euro)
[50, 20, 20, 5, 2]
```

# Les graphes

## Définition

On appelle graphe $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un couple où :

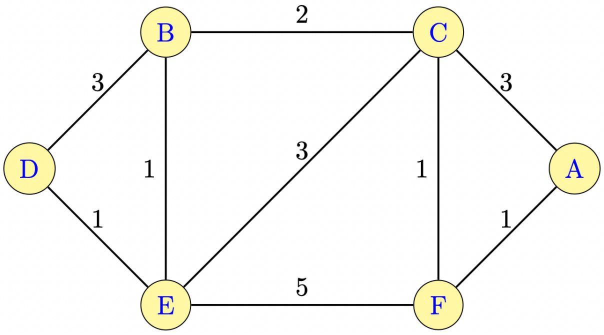
— $\mathcal{S}$ est un ensemble, appelé ensemble des **sommets**,

— $\mathcal{A}$ est une partie de $\mathcal{S} \times \mathcal{S}$, appelée ensemble des **arêtes**.

Dans la suite, on supposera $\mathcal{S}$ fini.

## Exemple

On considère le graphe suivant, donné par sa représentation graphique.




Le graphe $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ est ici défini par :

— l'ensemble des sommets $\mathcal{S} = \{A, B, C, D, E, F\}$

— l'ensemble des arêtes $\mathcal{A} = \{(A, B), \dots, (E, F), \dots (F, A)\}$

## Remarque

Le graphe présenté ici possède les caractéristiques suivantes.

— Il n'est pas **orienté** : si $(u, v) \in \mathcal{A}$ alors $(v, u) \in \mathcal{A}$. Ainsi, si on peut aller de $u$ à $v$, alors on peut aussi aller de $v$ à $u$.

— Il est **simple** : il y a au plus une arête entre deux sommets.

— Il est **pondéré** : à chaque arrête on associe une valeur positive appelée poids. Ce poids représente la distance entre les deux sommets.

**7.** On implémente les sommets $\mathcal{S} = \{A, B, C, D, E, F\}$ de la manière suivante :

```
S = [0, 1, 2, 3, 4, 5]
G = [
[0,-1,3,-1,-1,1],
[-1,0,2,3,1,-1],
[3,2,0,-1,3,1],
[-1,3,-1,0,1,-1],
[-1,1,3,1,0,5],
[1,-1,1,-1,5,0]
]
```

Que représente `G` ? Que représente `-1` ?

**8.** Quelle remarque peut-on faire sur cette matrice ? De quelle propriété provient cette caractéristique ?

**9.** Que représente `G2` dans le code suivant ? Quel est son type ?

```
G2 = {
    0: {1: 1},
    1: {3: 2},
    2: {1: 3},
    3: {0: 1, 4: 1},
    4: {0: 3, 1: 2, 2: 2, 5: 2},
    5: {2: 4, 7: 0},
    6: {0: 3, 3: 0, 5: 2},
    7: {2: 3}
    }
```

**10.** Représenter ce graphe à la main sur votre feuille.

### Algorithme de Dijkstra

**11.** Lire le document `Dijkstra.pdf` en ligne. L'appliquer à la main sur l'exemple 2.

**12.** Lire le document `Dijkstra_bis.png` en ligne. L'appliquer à la main sur l'exemple 4.

## Codage de message

On utilise la méthode de codage suivante :

— on définit une clé de codage : un nombre entier `n` compris entre 1 et 25

— pour chaque lettre du message, on va prendre la lettre décalée de `n` dans l'alphabet. Il faut considérer l'alphabet cyclique, la lettre « a » succède au « z ».

Le message initial pourra être composé de phrases qui possèdent des minuscules, des majuscules, des lettres accentuées, des nombres, des espaces, des ponctuations... pour simplifier, on suppose qu'on ne code que les lettres minuscules et majuscules.

Pour votre programme, il peut être utile de définir deux chaines de caractères :

```
minuscules = 'abcdefghijklmnopqrstuvwxyz'
majuscules = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

**13.** Écrire une fonction `permutation(chaine,x)` qui retourne `chaine_permutee`, c'est-à-dire `chaine` après une permutation de `x` caractères vers la gauche. On suppose que `x` est positif et inférieur à la longueur de la chaine.

```
>>> permutation('charlyTANGO71', 2)
'arlyTANGO71ch'
```

**14.** Écrire une fonction `indice(caractere,chaine)` qui retourne l'indice de `caractere` dans la `chaine` ou `-1` si `caractere` est absent de `chaine`. On suppose que `caractere` n'est prèsent au maximum qu'une seule fois dans `chaine`.

**15.** Écrire une fonction `codage_minuscules(chaine,n)` qui retourne `chaine_modifiee`, c'est-à-dire `chaine` chifrée avec un décalage de `n` lettres (on pourra utiliser les fonctions définies dans les questions précédentes). On suppose ici que `chaine` ne comporte que des caratères minuscules.

```
>>> codage_minuscules('bonjour', 3)
'erqmrxu'
```

**16.** Écrire une fonction `codage(chaine, n)` qui retourne `chaine_modifiee`, c'est-à-dire `chaine` chiffrée avec un décalage de `n` lettres (on pourra utiliser les fonctions définies dans les questions précédentes). Cette fonction traitera le cas des phrases qui possèdent des minuscules, des majuscules, des lettres accentuées, des nombres, des espaces, des ponctuations... Pour simplifier, on suppose qu'on ne code que les lettres minuscules et majuscules.

```
>>> codage('Bonjour, coucou, ça va ?', 3)
'Erqmrxu, frxfrx, çd yd ?'
```

**17.** Écrire une fonction `decodage(chaine, n)` qui permet de décoder un message. On utilisera pour cela la fonction `codage`. Tester votre fonction sur l'exemple précédent.

**18.** Décoder le message suivant qui a été chiffré avec `n=3`.

```
Eudyr, yrxv dyhc uéxvvl à géfrghu ! Bhv, zh glg lw !
```

Bon courage et bon travail ! ☺