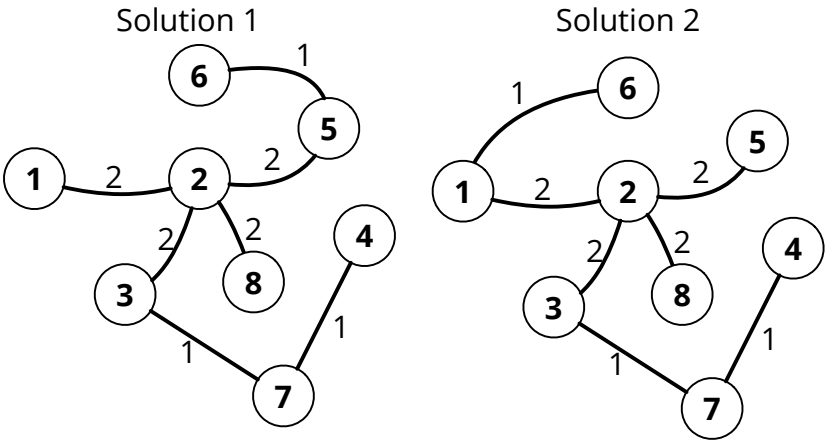
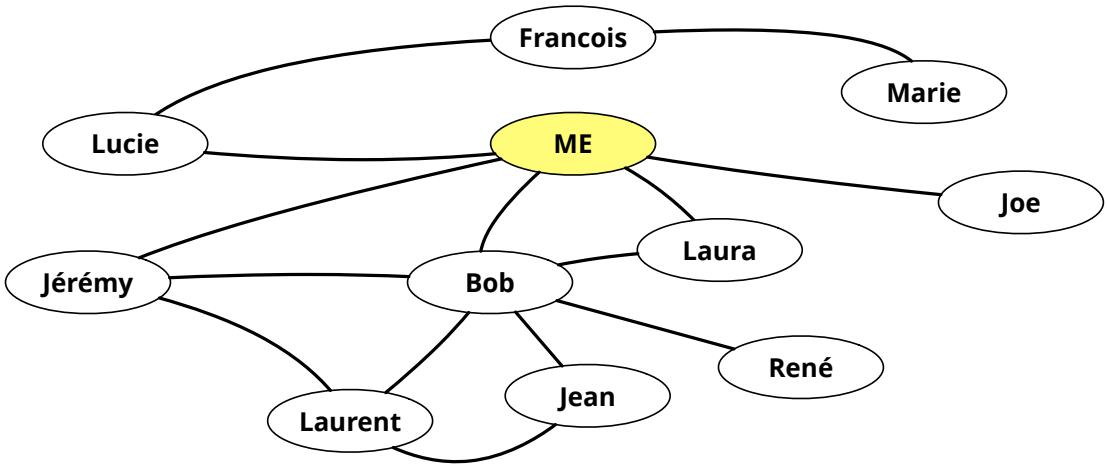
EXO0:



Exo1:

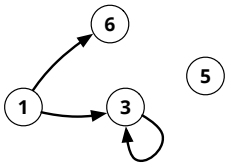
Noeud = Personne
Arête = un lien d'amitié
Le graph est non-dirigé (l'amitié est réciproque)

1. Celui-qui possède le plus d'amis est celui qui compte le plus d'arêtes: Bob.
2. Mes amis qui se connaissent sont les personnes qui sont connectées avec moi (mes amis) ET qui sont connectées entre-elles: Bob-Laura et Bob-Jérémy
3. Les amis qui pourraient me présenter quelqu'un sont: ceux qui sont connectés avec moi ET qui possèdent une relation que je n'ai pas: Lucie, Jérémy, Bob. Notez que Laura et Joe ne peuvent pas me présenter quelqu'un.
4. Une personne qui n'a pas d'amis n'a pas de lien. Aucune ci-dessous.



Exo2:

SOLUTION :



|G| = 4 = number of vertices
||G|| = 3 = number of edges
Il est clairsemé.

Exo3:

{ v = {1,6,3}, e = {{1,6}, {1,3}, {3,3}} }

Exo4:

a : Breadth-First Search (BFS)
b : Il permet de s'avoir si un nœud est atteignable à partir d'un nœud donné.
c : H n'apparaît pas car il n'est pas atteignable depuis A.

Exo5:

1. Somme des degrés d'un graphe = 2* nombre d'arêtes.
2. Même relation pour un arbre
3. Nombre d'arêtes = Nombre de noeuds -1 (pour un arbre). Il n'existe pas de telle relation pour un graphe qui n'est pas un arbre.
4. Solution pour le programme

```
def check_tree(N, listN):
    Sum_degrees = sum(listN)
    Edges = Sum_degrees/2
    if Edges == N-1:
        return True
    else:
        return False
print(check_tree(3, [1, 2, 1]))
```

Exo6:

https://networkx.github.io/documentation/networkx-1.10/reference/algorithms.shortest_paths.html

```
import networkx as nx
import matplotlib.pyplot as plt

# Dessine un graph circulaire
# (évite de devoir générer soit-même un graph)
G=nx.cycle_graph(30)
nx.draw(G)   #draws the graph

plt.show()   #Displays the graph on the screen

# connaitre le chemin le plus court entre le noeud 0 et 20
print (nx.shortest_path(G, source=0, target=20, weight=None))
# output [0, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20]
```