Exercice 1 :

On cherche à résoudre le problème suivant :

On part d'une liste de nombres et on souhaite déterminer la longueur de la plus longue succession ininterrompue de nombres rangés dans l'ordre strictement croissant qu'on peut trouver dans cette liste.

1) On considère la liste suivante : [4, 9, 1, 3, 5, 7, 6, 2, 8, 0].

a) La suite 4, 9 est-elle une succession ininterrompue de nombres rangés dans l'ordre strictement croissant ? Justifier.

b) La suite 5, 7, 6, 2 est-elle une succession ininterrompue de nombres rangés dans l'ordre strictement croissant ? Justifier.

c) Donner la plus longue suite de nombres rangés dans l'ordre strictement croissant qu'on peut trouver dans la liste. Donner sa longueur.

2) On considère la fonction suivante qui résout ce problème :

```
def longueur_croissante(liste):
    """
    Renvoie la longueur de la plus grande suite strictement croissante qu'on
peut trouver dans la liste de nombres, non vide.
    """
    longueur_max = 1
    longueur = 1
    precedente = liste[0]
    for valeur in liste[1:]:
        print(valeur, precedente, longueur, longueur_max)
        if precedente >= valeur:
            longueur = 0
        longueur += 1
        precedente = valeur
        longueur_max = max(longueur_max, longueur)
    return longueur_max
```

Compléter le tableau des informations affichées par la fonction appelée avec la liste [8, 5, 7, 4, 6] par l'instruction print.

valeur	précedente	longueur	longueur_max
a			

Exercice 2 : récursivité

On considère la fonction suivante :

```
def repeter(n, motif):
    # n est un entier naturel et motif une chaîne
    if n == 0:
        return ""
    else:
        return motif + repeter(n - 1, motif)
```

1) Pourquoi la fonction **repeter()** est-elle récursive ?

2) Quelle est la condition d'arrêt de la récursivité ?

3) Qu'est-ce qui décroît à chaque appel récursif et permet d'affirmer que la récursivité s'arrête ? Argumenter.

4) Que renvoient les appels suivants ?

repeter(0, "*") : **repeter(3, "+")**

Exercice 3 : récursivité

On souhaite écrire une fonction récursive **somme(n)** qui renvoie la somme des chiffres du nombre entier naturel **n** écrit en base 10.

1) Donner la valeur renvoyée par chacun des appels suivants :

2) **somme(3)** : **somme(7)** : **somme(36)** : **somme(582)** :

2) Quelle condition d'arrêt de la récursivité doit-on privilégier naturellement ?

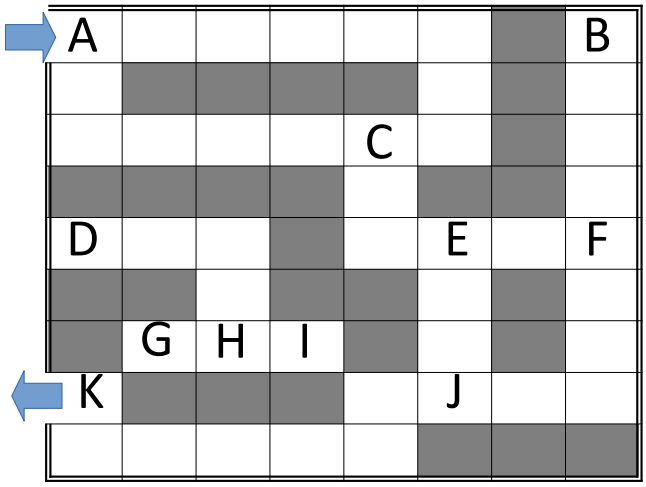
3) Écrire la fonction **somme(n)** de manière récursive.

```
def somme(n):                    # n est un entier naturel
```

Rappel : a // b *donne le quotient de la division entière de* a *par* b*, et* a % b *le reste.*

Exercice 4 - Labyrinthe et implémentation des données sous forme de graphe

On vous propose d'étudier le labyrinthe représenté ci–dessous :



Le point d'entrée est représenté par la lettre A
et le point de sortie par la lettre K.

Chaque intersection ou extrémité d'un parcours est représentée par une lettre et la distance entre deux lettres est déterminée par le nombre de cases à parcourir pour aller d'une lettre à la suivante.
Exemple : la distance de E à F est de 2 : partant de E on avance de deux cases pour arriver à F.

1 - Représenter ce labyrinthe par un graphe où :
- chaque intersection ou extrémité occupera un sommet
- chaque arête sera pondérée par la distance qui sépare les deux sommets.

2 - En déduire la matrice d'adjacence :

	A	B	C	D	E	F	G	H	I	J	K
A											
B											
C											
D											
E											
F											
G											
H											
I											
J											
K											

3 - Comment peut-on justifier cette matrice soit symétrique par rapport à la diagonale ?

Pour simplifier l'interprétation du graphe, **on va implémenter les connexions entre un sommet et ses sommets connexes par un dictionnaire.**

Pour chaque sommet, correspondant à une clé, on associe les sommets connexes sous forme d'une liste de tuples : sommet, distance.

4 - Indiquer ce que sera le dictionnaire d'adjacence du graphe représenté ci-dessus.

On donne une implémentation du graphe en langage Python ci-dessous :

```
class Graphe ( ) :

  def __init__ ( self ):
       self.sommets = { } # Dictionnaire d'adjacence

  #======================================== adjacents ( … )
  def adjacents ( self, sommet ) :
       return self.sommets [sommet]
```

5 - Quel est le but de la fonction *adjacents* fonction qui ne contient qu'une instruction ?
Vous préciserez en détail le type de la ou les donnée(s) retournée(s) par l'instruction return

```
  #======================================== ajouter_sommet ( … )
  def ajouter_sommet (self, nom ) :
      """
      Si le sommet n'est pas déjà enregistré dans le dictionnaire d'adjacence, cette
      fonction ajoute alors un nouveau sommet au dictionnaire avec aucun sommet
      connexe donc une liste vide.
      """
```

6 - Quelles instructions en langage Python permettraient d'implémenter cette fonction ?

```
  #======================================== ajouter_arete ( … )
  def ajouter_arete ( self, s1, s2, pond ) :
      """
      Ajouter un arc du sommet s1 vers le sommet s2 avec une pondération : pond
      Si s1 ou s2 ne sont pas encore répertoriés dans le graphe, cette fonction les
      ajoute au dictionnaire d'adjacence.
      """
```

7 - Quelles instructions en langage Python permettraient d'effectuer ce que doit faire cette fonction ?

Pour la suite, on considère que sont déjà implémentées et donc disponibles :

- la classe File ( ) de type FIFO avec les méthodes :
. enfiler (val) *: insérer une valeur dans la file*
.defiler ( ) *: retirer la valeur située en tête de file*
.estVide ( ) *: retourne True si la file est vide sinon False*

- la classe Pile ( ) de type FILO avec les méthodes :
. empiler (val) *: déposer une valeur sur la pile*
.depiler ( ) *: retirer la valeur située au dessus de la pile*
.estVide ( ) : retourne True si la pile est vide sinon False

- la fonctions liste_voisins de la classe Graphe :
. liste_voisins (self. S) *: retourne la liste des sommets connexe à s*

De plus, toujours dans la classe Graphe, on dispose de la fonction parcours ci-dessous :

```
#======================================== ajouter_arete ( … )
def parcours ( self, sommet, objectif ) :
    """
    Entrées :
      sommet : point de départ d'un parcours
      objectif : point d'arrivée pour lequel on cherche à savoir s'il existe un chemin
    """
    vus=
    []
    pile = Pile ( )
    pile.empiler ( sommet )
    while not pile.estVide (
    ) :
        sommet = pile.depiler()
        if sommet in vus :
            continue # retour au while pour passer à l'itération suivante
        vus.append(sommet)
        for voisin in list(self.liste_voisins(sommet)):
            pile.empiler ( voisin )
    return objectif in vus
```

8 - Tous les sommets et arêtes ont été implémentés dans l'instance de classe **mon_graphe** afin de modéliser le graphe d'adjacence de l'exercice. **Quelle serait alors la valeur retournée par cette fonction** si elle était appelée avec les arguments :

`mon_graphe.parcours ( 'A', 'G' )` *(justifier votre réponse)* :

9 - A partir des instructions de la fonction `.parcours ( ... )`, en déduire s'il s'agit d'un parcours du graphe en largeur ou en profondeur. Quelles instructions vous permettent de justifier votre réponse ?

10 - En déduire l'ordre des sommets qui seront parcourus et ceux pour lesquels on ne prolonge pas l'exploration parce qu'ils ont déjà été visités.

EXERCICE 5 : Base de données relationnelles et langage SQL

LA SANDWICHERIE

1: Structure de la base :

Après vos déboires dans la location de voitures et dans le monde du football, vous décidez d'ouvrir une sandwicherie effectuant des livraisons à domicile. Gaël, le responsable du service client et Elouarn votre comptable se propose de partager une représentation de votre activité à travers trois tables Excel présentées ci-dessous :

nom	adresse	ville
Chenadec	5 rue du Port	Ploemeur
Robic	10 quai des bavards	Lorient
Le Calvez	5 route de Larmor	Lorient

date	nom	sandwich	nombre
08/12/20	Elouarn Chenadec	Cheeseburger	1
08/12/20	Elouarn Chenadec	Triple	1
09/12/20	Elouarn Chenadec	Double	2
09/12/20	Eliaz Robic	Triple	1
09/12/20	Eliaz Robic	Fois gras Deluxe	2
10/12/20	Eliaz Robic	Triple	1
10/12/20	Eliaz Robic	Fois gras Deluxe	3
10/12/20	Gaël Le Calvez	Deluxe	3
11/12/20	Maewenn Guenezan	Cheeseburger	2
11/12/20	Maewenn Guenezan	Triple	1

sandwich	prix
Cheeseburger	2,00
Double Cheeseburger	3,50
Triple Cheeseburger	5,00
Fois gras Deluxe	10,00

1.1 : Signalez les défauts de leur proposition et **indiquez quelques avantages** que procurerait l'utilisation d'une base de données relationnelle.

Vous arrivez à les convaincre de basculer sur une base de données.
Après réflexion, vous décidez d'utiliser la structure présentée en annexe 1.

1.2 : Expliquez à Gaël pourquoi vous n'avez pas souhaité conserver le nom du client comme clé primaire.

1.3 : Pour chacune de ces tables, il manque : le type d'un des attributs.
Complétez les informations manquantes dans chacune des tables.

1.4 : Indiquez par des flèches sur l'annexe 1 les relations associées à cette base.

1.5 : Cette base de données permet-elle de distinguer le contenu de deux commandes passées le même jour par un même client ? Les tables précédentes le pouvaient-elles ? Justifiez votre réponse …

2 : Interrogation de la base :

Vous venez de lancer votre activité et actuellement la base n'est pas très fournie.
Le détail est présenté sur l'annexe 2 en page 15

2.1 : Rédigez un petit paragraphe précisant toutes les informations que vous pouvez extraire de cette base au sujet de votre amie Maewenn. Vous indiquerez notamment le total de sa dernière facture.

Vous décidez de faire analyser votre activité pour comprendre pourquoi elle ne démarre pas plus vite. Malheureusement votre service « relation client » n'est pas très compétent en gestion de base de données.
Aussi vous décidez de réaliser un petit listing de requêtes SQL lui permettant d'obtenir les informations suivantes. On rappelle que les fonctions d'agrégation sont : COUNT, MIN, MAX, AVG et SUM.

2.2: Le nombre de clients :
2.3 : La liste des villes des clients (une seule fois) :
2.4 : La liste des noms des clients habitant Lorient ou Lanester :
2.5 : Le nombre de sandwichs dont le prix est inférieur à 4€.
2.6 : L'identifiant du client dont le prénom est Eliaz.
2.7 : Le nombre de commandes effectuées par le client d'identifiant 2.
2.8 : Le nombre de sandwichs commandés par le client d'identifiant 2.
2.9 : La somme que vous doit le cuisinier, soit le client d'identifiant 2.

3: Modification de la base :

3.1 : Elouarn vous annonce que Maewenn du service « approvisionnement » préfère annuler son Triple Cheese et commander un Double Cheese à la place.
Indiquez la requête SQL poureffectuer cette modification.

3.2 : Vous décidez de faire une promotion sur le Foie gras Deluxe et de le proposer finalement à 9€ seulement. Vous décidez d'ajouter un sandwich « Foie Gras Deluxe Pacher » à 9€ dans la liste des sandwichs.
Indiquer la requête SQL correspondante.

3.3 : Pourquoi n'avez-vous pas souhaité modifier directement le prix du « Foie gras Deluxe » comme vous le proposait Eliaz ?

3.4 : Finalement Maewenn ne se sent pas très bien et rentre chez elle. Elle souhaite annuler sa commande. Indiquez dans l'ordre les requêtes permettant de faire disparaître l'ensemble des enregistrements associés à cette commande. Justifiez cet ordre.

ANNEXE 1 : Structure de la base

Sandwichs		
id	INT	la clef primaire
nom		le nom
prix	REAL	le prix

Commandes		
id	INT	la clef primaire
date	DATE	la date
client		le client

Clients		
id	INT	la clef primaire
nom	TEXT	le nom
adresse		l'adresse
ville	TEXT	la ville
telephone	TEXT	le téléphone

Lignes		
id	INT	la clef primaire
commande	INT	la commande
sandwich	INT	le sandwich
nombre		le nombre de sandwich

ANNEXE 2 : Détail de la base

Clients				
id	**nom**	**adresse**	**ville**	**téléphone**
1	Chenadec Elouarn	5 rue du Port	Ploemeur	07 70 00 77 00
2	Robic Eliaz	10 quai des bavards	Lorient	06 77 66 88 24
3	Le Calvez Gael	5 route de Larmor	Lorient	07 84 85 65 54
4	Guenezan Maewenn	10 pont du bonhomme	Lanester	07 68 69 26 54

Sandwichs		
id	**nom**	**prix**
1	Cheeseburger	2,00
2	Double Cheese	3,50
3	Triple Cheese	5,00
4	Foie gras Deluxe	10,00

Commandes		
id	date	client
1	08/12/20	1
2	09/12/20	1
3	09/12/20	2
4	10/12/20	2
5	10/12/20	3
6	11/12/20	4

Lignes			
id	commande	sandwich	nombre
1	1	1	1
2	1	3	1
3	2	2	2
4	3	3	1
5	3	4	2
6	4	3	1
7	4	4	3
8	5	4	3
9	6	1	2
10	6	3	1

Exercice 2 : récursivité

On considère la fonction suivante :

```
def repeter(n, motif):
# n est un entier naturel et motif une chaîne
if n == 0:
   return ""
else:
   return motif + repeter(n - 1, motif)
```

1) Pourquoi la fonction **repeter()** est-elle récursive ?

2) Quelle est la condition d'arrêt de la récursivité ?

3) Qu'est-ce qui décroît à chaque appel récursif et permet d'affirmer que la récursivité s'arrête ? Argumenter.

4) Que renvoient les appels suivants ?

repeter(0, "*") : **repeter(3, "+")**

1) La fonction repeter() est récursive car il y a un appel à elle-même dans son corps.

2) La condition d'arrêt est n == 0.

3) n est un entier naturel (positif) et le premier paramètre dans l'appel récursif à la fonction repeter() est n – 1.

Donc dans la suite des appels récursifs n décroit strictement.

n sera donc égal à 0 après n appels récursifs et avec la condition d'arrêt n == 0, la récursivité s'arrêtera donc.

4) Repeter(0, **"*") renvoie une chaine vide.**

Repeter(3, **"+") renvoie la chaîne " +++ ".**

Exercice 3 : récursivité

On souhaite écrire une fonction récursive **somme(n)** qui renvoie la somme des chiffres du nombre entier naturel **n** écrit en base 10.

1) Donner la valeur renvoyée par chacun des appels suivants :

2) **somme(3)** : **somme(7)** : **somme(36)** : **somme(582)** :

2) Quelle condition d'arrêt de la récursivité doit-on privilégier naturellement ?

3) Écrire la fonction **somme(n)** de manière récursive.

```
def somme(n):                    # n est un entier naturel
```

Rappel : a // b *donne le quotient de la division entière de* a *par* b*, et* a % b *le reste.*

1) somme(3) renvoie 3

   somme(7) renvoie 7

   somme(36) renvoie 3 + 6 = 9

   somme(582) renvoie 5 + 8 + 2 = 15

2) On peut choisir comme condition d'arrêt n == 0.

3)
```
def somme(n) :
    if n == 0 :
        return 0
    return n % 10 + somme(n//10)
```

cf un repl.it avec plusieurs implémentations de la fonction somme (2 itératives et 2 récursives) : https://repl.it/@hmalherbe/somme#main.py

EXERCICE 5 : Base de données relationnelles et langage SQL

LA SANDWICHERIE

1.1

Redondance d'informations,

Pas de domaine de valeurs associés aux attributs : Incohérence entre les noms des clients dans les différentes tables et des noms de sandwichs dans la même table,

Possibilité d'avoir deux lignes strictement identiques (une même personne commandant le même sandwich deux fois dans la journée)

Risques liés à une modification simultanée des tables,

1.2 : Risque d'avoir deux clients du même nom et donc non-unicité de la clé.

1.3 : Clients.adresse TEXT

Sandwichs.nom TEXT

Commandes.client INT (car clé étrangère sur Clients.id)

Lignes.nombre INT

1.4 : Commandes.client → Clients.id

Lignes.commande → Commandes.id

Lignes.sandwich → Sandwichs.id

1.5 : Oui car des enregistrements différents dans Commandes les distingueront.

Non, car seule la date est fournie

2.1 : Maewenn habite 10 pont du bonhomme à Lanester.
Son numéro de téléphone est le 07 68 69 26 54.
Elle a commandé le 11/12/2020, 2 Cheeseburger à 2 € l'unité et 1 Triple Cheese à 5 €.
Le montant de sa facture était : $2\times2 + 5\times1 = 4 + 5 = 9$ €

2.2 : 4 => SELECT COUNT(*) FROM Clients;

2.3 : Ploemeur, Lorient, Lanester

=> SELECT DISTINCT ville FROM Clients;

2.4 : Robic Eliaz , Le Calvez Gaël, Guenezan Maewenn

SELECT nom FROM Clients WHERE ville in ('Lorient','Lanester');

2.5 : 2 => SELECT COUNT(*) FROM Sandwichs WHERE prix<4.0;

2.6 : 2 => SELECT id FROM Clients WHERE nom LIKE '%Eliaz%';

2.7 : 2 => SELECT COUNT(*) FROM Commandes WHERE client=2;

2.8 : 7 => SELECT SUM(nombre) FROM Lignes

JOIN Commandes ON Lignes.commande=Commandes.id

WHERE client=2;

2.9 : 60 => SELECT SUM(nombre*prix) FROM Lignes

```
JOIN Commandes ON Lignes.commande=Commandes.id
JOIN Sandwichs ON Lignes.sandwich=Sandwichs.id
WHERE client=2;
```

3.1 : UPDATE Lignes SET sandwich=2 WHERE id=10;

3.2 : INSERT INTO Sandwichs(id,nom,prix) VALUES(5,'Foie gras Deluxe Pacher',9.0);

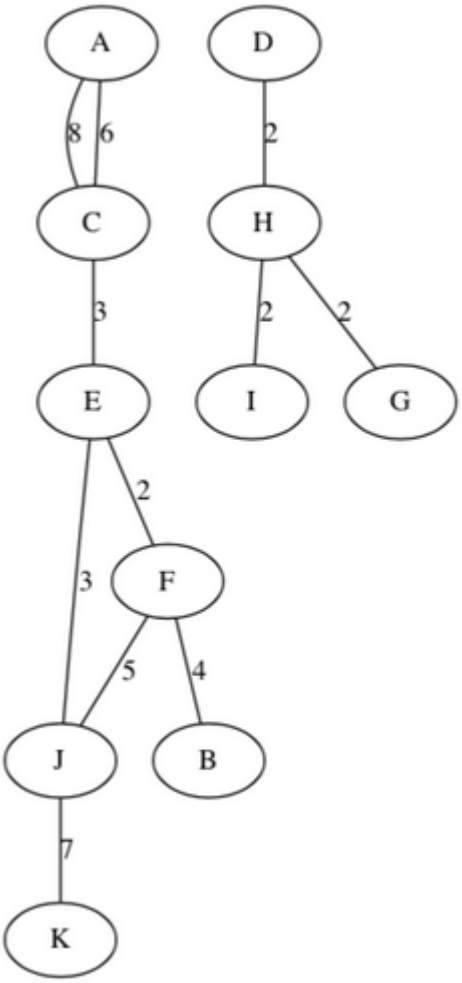
3.3 : Dans le cas contraire ce ne serait plus possible de calculer le juste prix des anciennes factures et Elouarn ne devrait plus que 55€.

3.4 : DELETE FROM Lignes WHERE commande=6;

DELETE FROM Commandes WHERE id=6;

Exercice 4 - Labyrinthe et implémentation des données sous forme de graphe

1)



2 - En déduire la matrice d'adjacence :

	A	B	C	D	E	F	G	H	I	J	K
A	0	0	6	0	0	0	0	0	0	0	0
B	0	0	0	0	0	4	0	0	0	0	0
C	6	0	0	0	3	0	0	0	0	0	0
D	0	0	0	0	0	0	0	2	0	0	0
E	0	0	3	0	0	2	0	0	0	3	0
F	0	4	0	0	2	0	0	0	0	5	0
G	0	0	0	0	0	0	0	2	0	0	0
H	0	0	0	2	0	0	2	0	2	0	0
I	0	0	0	0	0	0	0	2	0	0	0
J	0	0	0	0	3	5	0	0	0	0	7
K	0	0	0	0	0	0	0	0	0	7	0