Introduction

- Les exercices sont à effectuer dans l'ordre.
- Le choix du langage est libre.
- Un fichier README contiendra tout commentaire jugé utile pour comprendre votre programme, ou votre raisonnement vis-à-vis de certains problèmes.
- La notation dépend de la qualité du code et du respect des énoncés.
- Une *fonction* n'est pas un programme. Si vous ne parvenez pas à effectuer le dernier exercice dans les temps, un programme qui compile n'est donc pas demandé.
- Même si un exercice demande une fonction, vous pouvez utiliser n'importe quel nombre de sous-fonctions si vous le désirez.
- **Attention** : Il est interdit d'utiliser des librairies tierces faites pour parser le CSV.
- Les fichiers CSV fournis dans le .zip peuvent être utilisés comme entrée à votre programme. Pour les 3 premiers exercices, vous pourrez utiliser le premier fichier CSV, qui ne comporte aucune erreur potentielle d'intégration (voir exercice 4).

Fichiers CSV

Les fichiers CSV fournis avec ce sujet ont les champs suivants, dans l'ordre :
`firstname, lastname, email, birthdate`

Ces champs définissent un "contact". Ce contact a une adresse e-mail unique.

Exercices

Exercice 1

Ecrire une fonction `parseCSV` qui prend en paramètre un nom de fichier csv, et retourne le contenu du fichier sous la forme d'un tableau de tableaux de chaînes de caractères.
La fonction doit afficher le nombre de lignes correctement parsées. Le fichier CSV contiendra toujours, en première ligne, les noms des colonnes.
Sa signature en pseudo-code : `function parseCSV(string filename) string[][]`

Exemple :

```
firstname,lastname,email,inscription_date
John,Dupont,john.d@mail.com,01/02/1990
Jane,Dupond,j.dupond@mail.com,01/02/1990
```

renvoie :

```
Array [
        0 => Array ["John","Dupont","john.d@mail.com","01/02/1990"]
        1 => Array ["Jane","Dupond","j.dupond@mail.com","01/02/1990"]
]
```

Exercice 2

Ecrire une fonction `appendCSV` qui prend en paramètre un nom de fichier csv, ainsi qu'un tableau résultant du premier exercice (potentiellement vide ou non-initialisé). La fonction doit retourner un nouveau tableau de tableaux, qui contient la jointure du tableau donné en paramètre, ainsi que le nouveau contenu du CSV parsé.
Comme à l'exercice 1, la fonction affichera le nombre de lignes correctement parsées, ainsi que la nouvelle taille du tableau résultat.
Sa signature : `function appendCSV(string filename, string[][] data) string[][]`

Exemple :

```
# Fichier CSV
Alice,Foo,a.foo@mail.com,01/02/1990T15:53:26Z

# Tableau data
Array [
        0 => Array ["John","Dupont","john.d@mail.com","01/02/1990"]
        1 => Array ["Jane","Dupond","j.dupond@mail.com","01/02/1990"]
]
```

renvoie :

```
Array [
        0 => Array ["John","Dupont","john.d@mail.com","01/02/1990"]
        1 => Array ["Jane","Dupond","j.dupond@mail.com","01/02/1990"]
        2 => Array ["Alice","Foo","a.foo@mail.com","01/02/1990T15:53:26Z"]
]
```

Exercice 3

Ecrire un programme 'convertCsvToJson' qui traite le contenu d'un fichier CSV **ou** d'un dossier contenant plusieurs fichiers CSV et enregistre le tout sous le format JSON.
Les dates enregistrées dans le JSON respecteront le format : `YYYY-MM-DD HH:MM:SS`

Ce fichier JSON respectera le format suivant (à partir de l'exemple de l'exercice 2) :

```
{
        "contacts": [
                {
                        "firstname": "John",
                        "lastname": "Dupont",
                        "email": "john.d@mail.com",
                        "inscription_date": "1990-02-01 00:00:00"
                },
                {
                        "firstname": "Jane",
                        "lastname": "Dupond",
                        "email": "j.dupond@mail.com",
                        "inscription_date": "1990-02-01 00:00:00"
                },
                {
                        "firstname": "Alice",
                        "lastname": "Foo",
                        "email": "a.foo@mail.com",
```

```
                        "inscription_date": "1990-02-01 15:53:26"
                }
        ]
}
```

Le nom du fichier JSON doit être sous la forme : `<nom_du_fichier_csv_original>.json`
Par exemple : `20180101_132200_contactstream2.json`

Exercice 4

Les fichiers `[...]_contactstream2.csv`, `[...]_contactstream3.csv` et `[...]_contactstream4.csv` contiennent des erreurs (champs manquants, e-mail dupliqués, etc.).

Des restrictions s'appliquent sur ces champs pour qu'ils soient valides :

Champ	Règles
firstname	Longueur max : 50 caractères
lastname	Longueur max : 50 caractères
email	Longueur max : 255 caractères ; format : e-mail valide
inscription_date	Longueur max : 255 caractères ; format : `DD/MM/YYYY` **ou** au format RFC3339 (`YYYY-MM-DDTHH:MM:SSZ`)

En modifiant votre programme de l'exercice 3, la donnée du JSON final devra :

- respecter ces règles
- respecter l'unicité de l'adresse e-mail. Si un contact se trouve dupliqué dans un fichier, la date d'inscription la plus récente doit écraser les informations du contact précédent.
- afficher des logs verbeux d'exécution, et d'erreur. Ces statistiques portent sur :
    - Combien de lignes étaient dans le fichier à l'origine ?
    - Combien de contacts sont dupliqués via leur e-mail ?
    - Combien de lignes cassent les règles du `firstname` ? du `lastname` ? de l'`email` ? de l'`inscription_date` ?
    - Au final, combien de contacts uniques a-t-on pu intégrer dans le fichier JSON correctement ?

Exercice 5

Modifiez votre programme de l'exercice 4 pour trier par date d'inscription dans l'ordre décroissant les contacts dans le fichier JSON final.