Archivos y múltiples arrays

Objetivo

- Leer y procesar archivos de múltiples líneas y varios registros por línea.

Introducción

El escenario mas frecuente a la hora de trabajar con archivos es que una archivo tenga varios registros por línea y múltiples línaes, por ejemplo un archivo podría tener un set de datos de empresas y ventas por periodo.

```
Empresa1, 100, 20, 30, 50
Empresa2, 200, 10, 20, 30
```

Otros ejemplos

O ser un registro de alumnos y notas

```
Camilo, 90, 50 80, 90
Francisca 100, 40, 100, 100
...
Verónica 60, 70, 80, 100
```

En este capítulo aprenderemos a leer estos archivos, guardar la información en arreglos e iterar los resultados.

Archivos CSV

Los archivos como los que vimos en la introducción tienen un formato llamado CSV, es un formato similar al de un excel pero en formato plano, esto quiere decir que lo podemos leer sin necesidad de protocolos complejos desde cualquier programa. coma.

Leyendo un CSV

Para leer un CSV lo primero que haremos será incorporar la clase CSV. Esto lo podemos hacer con la instrucción `require`

```
require 'csv'
```

Luego podemos leer el archivo utilizando:

```
CSV.open('data.csv').readlines #  => [["Camilo", " 90", " 50 80", " 90"], ["Francisca 100", " 40", " 100", " 100"], ["Verónica 60", " 70", " 80", " 100"], []]
```

Obviamente para seguir trabajando con los datos lo guardaremos en una variable.

```
data = CSV.open('data.csv').readlines
```

Removiendo las líneas vacías

En este caso nos quedó un arreglo vacío al final porque hay un salto de línea al final del archivo, pueden haber casos donde haya una o incluso mas de una.

Para removerla ocuparemos `.reject`

```
data.reject! {|x| x.empty? }
```

Seleccionando una fila

- Si queremos obtener la primera fila lo podemos hacer con `a[0]`.
- Podemos seleccionar la primera fila con `a[1]`.
- Podemos seleccionar la última fila con `a[-1]`.
- Otra forma de seleccionar la última fila es con `a[a.length - 1]`

Seleccionando un elemento de una fila

Para seleccionar solo un elemento, ya sea un nombre o una nota necesitamos dos índices, uno para la fila y el otro para el elemento dentro de la fila. Ejemplo:

Si queremos obtener el nombre de la primera fila utilizaremos `a[0][0]`.

Si queremos obtener el último nombre lo podemos hacer con `a[-1][0]` o podríamos hacer `a[a.length - 1][0]`

Limpiando y transformar datos

Por defect `CSV.open('data.csv').read` devuelve todos los datos como string, si bien podemos pedirle de forma automática que intente transformar automaticamente los números a tipo Integer con `CSV.open('data.csv', converters: :numeric).read` Esto nos quita una oportunidad de practicar lo aprendido con .map

Nota: Algunos métodos como por ejemplo `.open` pueden reciben como argumento un hash, esto lo estudiaremos en la próxima unidad.

Transformando datos

La primera pregunta es si al iterar ocuparemos `.times` o `.each`. Podemos ocupar cualquiera de las dos pero si queremos cambiar los datos dentro del mismo arreglo directamente utilizaremos `.times`, porque sin el índice tendríamos que crear un arreglo nuevo con todos los datos.

Modificando una columna

Por ejemplo supongamos que queremos aumentar 15 puntos extra a todos los alumnos en la segunda nota.

Algoritmo

- Cargamos la biblioteca de CSV.
- Leemos el archivo transformando los datos automáticamente a número.
- Contamos la cantidad de líneas del archivo.
- Por cada línea:
    - Sumamos 10 puntos a la segunda nota (tercer elemento)

```
require 'CSV'

data = CSV
       .open('examples/data.csv',
           converters: :numeric)
       .readlines
       .reject! {|x| x.empty? }

lines = data.length

lines.times do |i|
   data[i][2] += 15
end

print data
```

```
[["Camilo", 90, 65, 80, 90], ["Francisca", 100, 55, 100, 50], ["Verónica", 60, 85, 80, 100]]
```

Agregando una columna

Supongamos que los alumnos son tan simpáticos que queremos agregar una columna con nota `100`

Algoritmo

El proceso es muy similar

- Cargamos la biblioteca de CSV.
- Leemos el archivo transformando los datos automáticamente a número.
- Contamos la cantidad de líneas del archivo.
- Por cada línea:
    - Agregamos una nota 100

```
require 'CSV'

data = CSV
       .open('examples/data.csv',
           converters: :numeric)
       .readlines
       .reject! {|x| x.empty? }
lines = data.length

lines.times do |i|
   data[i] << 100
end

print data
```

```
[["Camilo", 90, 50, 80, 90, 100], ["Francisca", 100, 40, 100, 50, 100], ["Verónica", 60, 70, 80, 100, 100]]
```

Guardando los resultados en un archivo

Es bastante probable que después de haber modificado el archivo queramos guardar los datos, para eso aprenderemos a abrir un archivo nuevo y guardar una línea

```
#require 'csv'
require 'CSV'
CSV.open("archivo.csv", "w") do |csv|
  csv << ["Juan", 80, 21, 55]
end
```

```
<#CSV io_type:File io_path:"myfile.csv" encoding:UTF-8 lineno:1 col_sep:"," row_sep:"\n"
quote_char:"\"">
```

Para guardar todos los resutados simplemente tenemos que iterar los elementos y guardarlos por línea.

```
require 'CSV'

data = CSV
       .open('examples/data.csv',
           converters: :numeric)
       .readlines
       .reject! {|x| x.empty? }

lines = data.length

CSV.open("archivo.csv", "w") do |csv|
   lines.times do |i|
      csv << data[i]
   end
end
```

```
3
```

Comentarios importantes respecto a arreglos y archivos

Es posible que rara vez nos enfrentemos a una situación como la de este capítulo, hoy en día existen múltiples motores de bases de datos están fuertemente difundidas y son mucho mas flexibles y rápidas a la hora de guardar, actualizar y recuperar datos, sin embargo este tipo de ejercicios y la forma de enfrentarlos nos ayudan a resolver diversos tipos de situaciones y la lógica a la hora de trabajar con Bases de datos o APIs es muy similar, tendremos grandes cantidades de datos que tendremos que iterar para poder obtener los resultados que deseamos.