Archivos, Arreglos y Strings

Objetivos

- Conocer el concepto de persistencia.
- Leer un archivo con datos y guardarlos en una variable.
- Transformar un array en un string.
- Transformar un string en un arreglo.
- Guardar los resultados en un archivo.

Motivación

En este capítulo seguiremos trabajando con el mismo tipo de problemas de los capítulos anteriores pero, en lugar de agregar el array manualmente dentro del código, aprenderemos a cargar este array desde un archivo.

Esto nos permitirá trabajar con datos que existen mas allá de la ejecución del programa.

Archivos y strings

Para poder leer y guardar información en archivos necesitamos aprender dos métodos que nos harán la vida muy fácil. `.split` y `.join`

- `.split` nos permite generar un array de strings a partir de un string más largo.
- `.join` nos permite convertir un array en un string.

.split

Split nos permite separar un string por algún criterio.

```
palabras = 'palabra1, palabra2, palabra3, palabra4'.split(', ')
#  => ["palabra1", "palabra2", "palabra3", "palabra4"]
```

.join

`.join` nos permite unir un array con un criterio separador.

```
['palabra1', 'palabra2', 'palabra3', 'palabra4'].join(', ')
# => "palabra1, palabra2, palabra3, palabra4"
```

Creando un archivo para leer

Necesitamos un archivo para poder leer desde ruby. Crearemos uno con el editor de texto y lo guardaremos en la misma carpeta de nuestro programa.

Crearemos el archivo data en la misma carpeta en que está nuestro código y lo llamaremos data (sin extensión)

Dentro del archivo guardaremos la siguiente información.

```
1,2,3,4,5,6,7,8,9,10
```

Observación: No dejar espacios entre las comas

Abrir y leer el archivo

Para abrir un archivo ocuparemos el método open, pero open solo nos devuelve un archivo abierto, pero no el texto que contiene. Para leerlo ocuparemos el método .read

```
data = open('data').read
# => "1,2,3,4,5,6,7,8,9,10\n"
```

Si el archivo no está en la carpeta o nos equivocamos de nombre del archivo obtendremos este error:

```
Errno::ENOENT: No such file or directory @ rb_sysopen - data
```

Eliminamos el salo de línea y lo transformamos en array

Para limpiar el salto de línea ocuparemos `.chomp`

```
data = open('data').read.chomp
# => "1,2,3,4,5,6,7,8,9,10"
```

Transformando los datos del archivo a un array

Podemos convertir todos los datos en un array utilizando split.

```
data = open('data').read.chomp.split(',')
```

Transformando los datos a enteros

```
data = open('data').read.chomp.split(',')
array = []
data.each do |d|
    array.push d.to_i
end
print array # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Si transformamos los datos a enteros no es necesario limpiar el salto de línea
```

Leyendo un archivo con información en múltiples líneas

Es posible que los datos vengan en distintas líneas del archivo. Para esto vamos a utilizar un archivo con datos en distintas líneas y lo llamaremos `data2`. Dentro de el guardaremos la siguiente información.

```
21
10
6
9
11
0
2
3
50
```

Leyendo el archivo con .readlines

Para hacer sencilla la lectura del archivo y traer todos los datos como un arreglo ocuparemos .readlines.

```
data = open('archivo2').readlines  ##=> ["21\n", "10\n", "6\n", "9\n", "11\n", "0\n", "2\n", "3\n", "50\n"]
```

Removiendo los saltos de línea

Veremos aquí que tenemos múltiples saltos de línea, uno por cada dato. Para limpiarlo podemos aplicar el método `.chomp` a cada uno de los elementos, pero si los transformamos a enteros se eliminará el `\n` automáticamente.

```
original_data = open('archivo2.txt').readlines
lines = original_data.count
array = []
lines.times do |i|
   array << original_data[i].to_i
end
```

Reutilizando la lógica con un método

El código para leer archivos lo reutilizaremos en algunos ejercicios, así que lo dejaremos dentro de un método para reutilizarlo de forma sencilla.

Dejaremos como parámetro el nombre del archivo y retornaremos los datos como un array.

```
def read_file(filename)
    original_data = open(filename).readlines
    lines = original_data.count
    array = []
    lines.times do |i|
        array << original_data[i].to_i
    end
    return array
end

read_file("archivo2.txt")
# => [21, 10, 6, 9, 11, 0, 2, 3, 50]
```

Ejercicio resuelto

Dado el archivo2 que tiene los siguientes datos

```
21
10
6
9
11
0
2
3
50
```

Se pide un crear un programa que tome los datos de ese archivo y construya un arreglo con los mismos pero transformando todos los valores mayores de 20 a un máximo de 20.

```
## Solución

def read_file(filename)
    original_data = open(filename).readlines
    lines = original_data.count
    array = []
    lines.times do |i|
        array << original_data[i].to_i
    end
    return array
end

data = read_file("archivo2.txt")
n = data.count
n.times do |i|
    data[i] = 20 if data[i] > 20
end
```

Guardando los resultados

Existen muchas formas de guardar datos en un archivo, una de las más sencillas es:

```
File.write('/path/to/file', 'datos')
```

Si no especificamos estaremos utilizando una relativa al archivo que estamos ejecutando, y los datos tienen que ser un string.

```
## Guardando los resultados

def read_file(filename)
    original_data = open(filename).readlines
    lines = original_data.count
    array = []
    lines.times do |i|
        array << original_data[i].to_i
    end
    return array
end


data = read_file("archivo2")
n = data.count
n.times do |i|
    data[i] = 20 if data[i] > 20
end

File.write('output', data.join("\n"))
```

¡¡¡ Cuidado !!!

Al abrir un archivo para guardar datos sobreescribiremos el contenido. Esto es peligroso porque, si no somos cuidadosos, podríamos destruir un archivo que nos sirve.

Persistencia

Se llama **persistencia** cuando los datos viven más allá del programa que los usa, por ejemplo guardando los resultados en archivos que luego otro programa (o el mismo) utiliza.

Los archivos son una forma de hacer persistente los datos pero existen otras, como las bases de datos.

Resumen

En este capítulo aprendimos a abrir archivos y leer datos de ellos, así como transformar los datos leídos y luego volver a guardarlos.

Para leer archivos vimos los métodos:

- open(filename).read
- open(filename).readlines
- .read es útil cuando queremos traer todo el archivo como un solo string a memoria, luego lo separamos ocupando el método split y algún criterio de separación
- .readlines es útil cuando queremos leer todo el archivo como un arreglo donde cada línea es un elemento del arreglo.

Para guardar datos en archivo ocuparemos:

- File.write('output', data)

Donde `data` es un string, pero si tenemos un array, podemos ocupar el método `.join` para transformarlo en un string.