





DESAFIO **LATAM**

EN LA EXPERIENCIA DE HOY
HABLAREMOS DE ARCHIVOS

ARCHIVOS

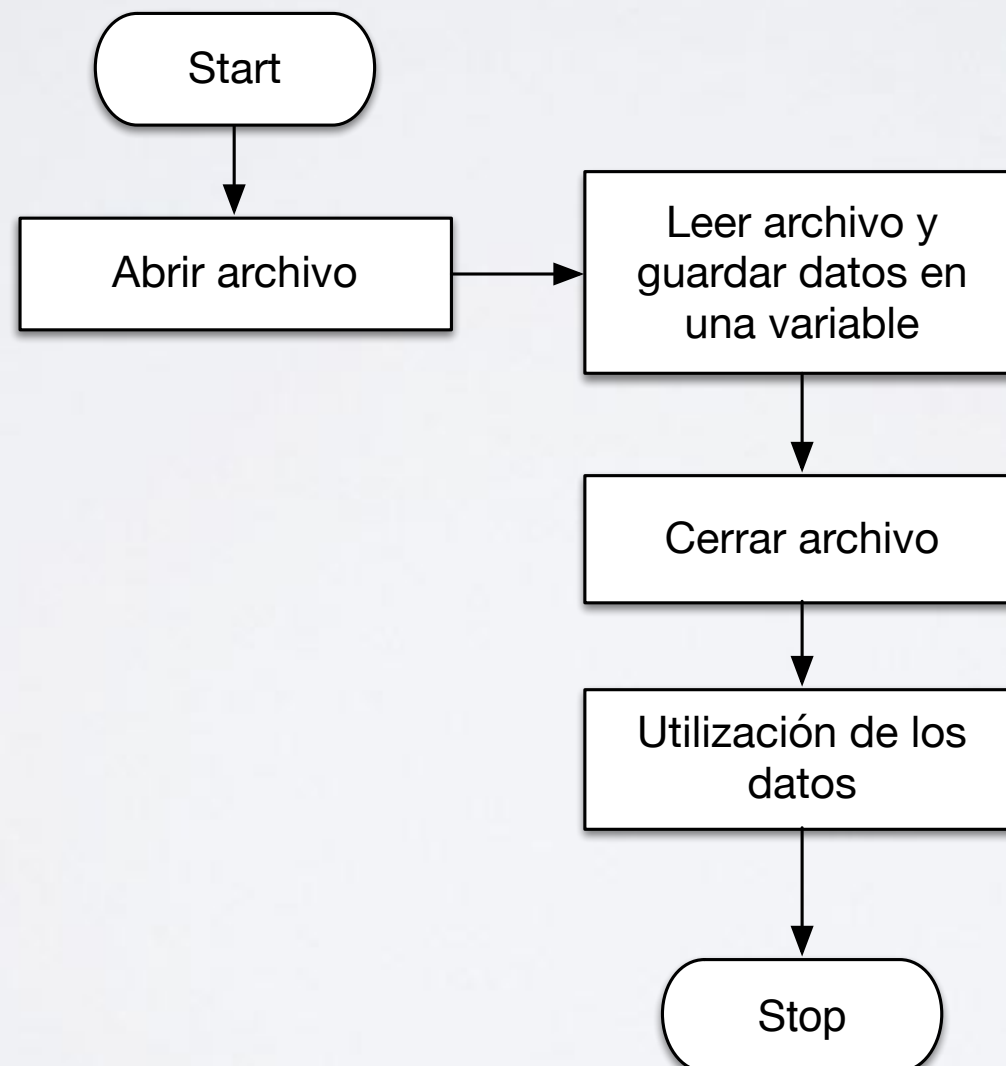
 logo_DL.svg		1 KB	XML
 angelhack_tickes		74 KB	PDF Document
 AngelHack Global Hackathon Series 2017 Participation Agreement.pdf		170 KB	PDF Document
 sitemap.xml		387 bytes	XML

Los archivos son conjuntos de bits que se almacenan en un dispositivo, como por ejemplo un disco duro

ARCHIVOS

Son muy útiles porque nos permiten persistir datos más allá de nuestra aplicación

FLUJO BÁSICO DE UN PROGRAMA QUE LEE UN ARCHIVO

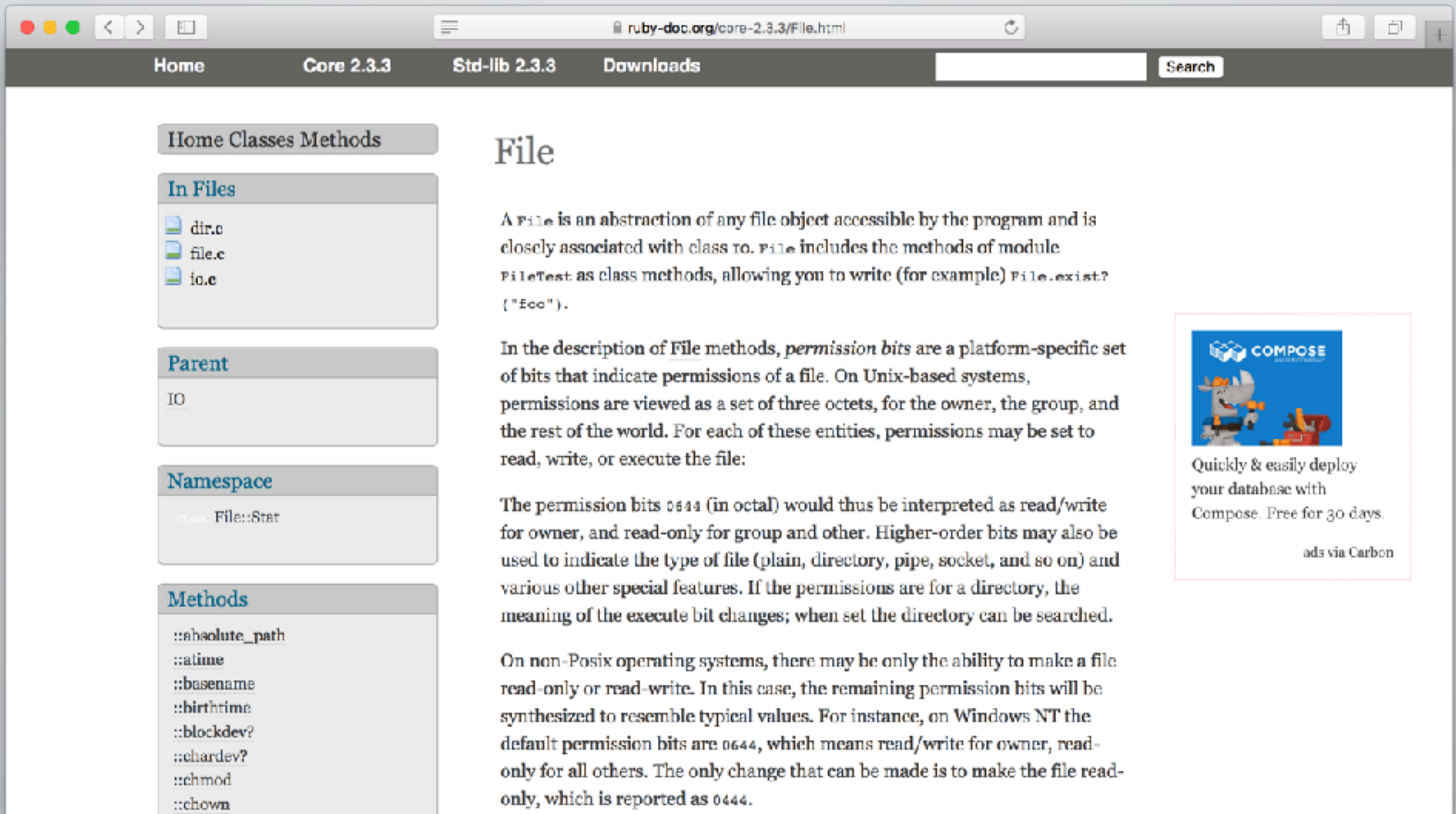


PARA ABRIR Y LEER UN ARCHIVO EN RUBY PODEMOS UTILIZAR LA CLASE FILE

<https://ruby-doc.org/core-2.3.3/File.html>

LA CLASE FILE

<https://ruby-doc.org/core-2.3.3/File.html>



The screenshot shows a web browser window displaying the Ruby documentation for the `File` class. The browser's address bar shows the URL `https://ruby-doc.org/core-2.3.3/File.html`. The page has a navigation bar with links to `Home`, `Core 2.3.3`, `Std-lib 2.3.3`, and `Downloads`, along with a search bar. The main content area is titled `File` and contains a description of the class, its methods, and its parent class `IO`. On the left side, there are several sidebar sections: `Home Classes Methods`, `In Files` (listing `dir.c`, `file.c`, and `io.c`), `Parent` (listing `IO`), `Namespace` (listing `File::Stat`), and `Methods` (listing various methods like `absolute_path`, `atime`, `basename`, etc.). On the right side, there is an advertisement for `COMPOSE` (a Docker ecosystem tool) with the text "Quickly & easily deploy your database with Compose. Free for 30 days." and "ads via Carbon".

File

A **File** is an abstraction of any file object accessible by the program and is closely associated with class `IO`. **File** includes the methods of module `FileTest` as class methods, allowing you to write (for example) `File.exist? {"foo"}`.

In the description of **File** methods, *permission bits* are a platform-specific set of bits that indicate permissions of a file. On Unix-based systems, permissions are viewed as a set of three octets, for the owner, the group, and the rest of the world. For each of these entities, permissions may be set to read, write, or execute the file:

The permission bits `0644` (in octal) would thus be interpreted as read/write for owner, and read-only for group and other. Higher-order bits may also be used to indicate the type of file (plain, directory, pipe, socket, and so on) and various other special features. If the permissions are for a directory, the meaning of the execute bit changes; when set the directory can be searched.

On non-Posix operating systems, there may be only the ability to make a file read-only or read-write. In this case, the remaining permission bits will be synthesized to resemble typical values. For instance, on Windows NT the default permission bits are `0644`, which means read/write for owner, read-only for all others. The only change that can be made is to make the file read-only, which is reported as `0444`.

COMPOSE
Quickly & easily deploy your database with Compose. Free for 30 days.
ads via Carbon

ABRIENDO UN ARCHIVO CON LA CLASE FILE

`File.open`

El método open de la clase File nos permite abrir un archivo (siempre y cuando tengamos los permisos necesarios)

CUANDO ABRIMOS UN ARCHIVO TENEMOS QUE ESPECIFICAR NUESTRAS INTENCIONES

```
File.open(nombre_archivo, intención)
```

```
File.open('sample.txt', 'r')
```

An orange arrow pointing from the word "Read" to the character 'r' in the file mode parameter of the code snippet above.

Read

LEYENDO UN ARCHIVO



El método **read** devuelve un string con todo el contenido dentro del archivo.

O EN FORMA DE BLOQUE

```
File.open('sample.txt', 'r'){ |file| puts file.read }
```

En el bloque el archivo se cierra de forma automática

Y QUÉ SUCEDE SI TENEMOS UNA TABLA COMO ARCHIVO

Producto1	100
Producto2	210
Producto3	3000

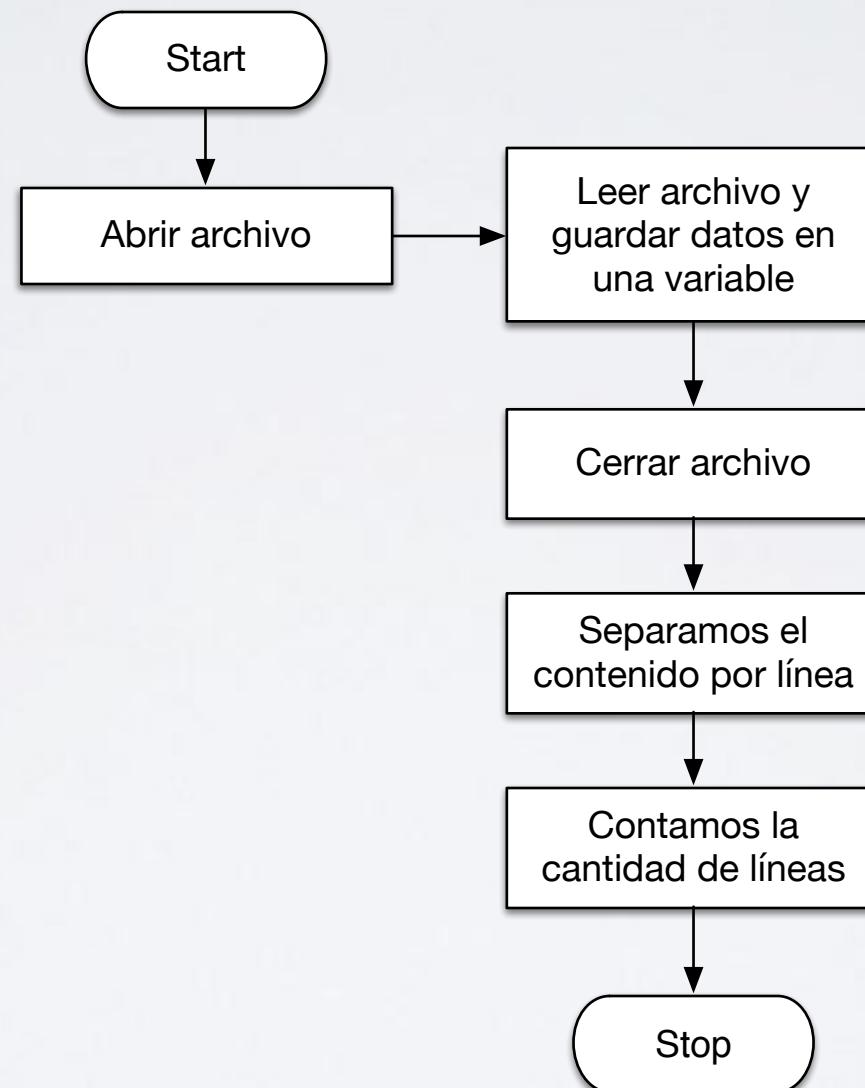
¿Cómo podríamos obtener la cantidad de productos
o el precio promedio?

ESTRATEGIA

La cantidad de productos correspondería a la cantidad de líneas

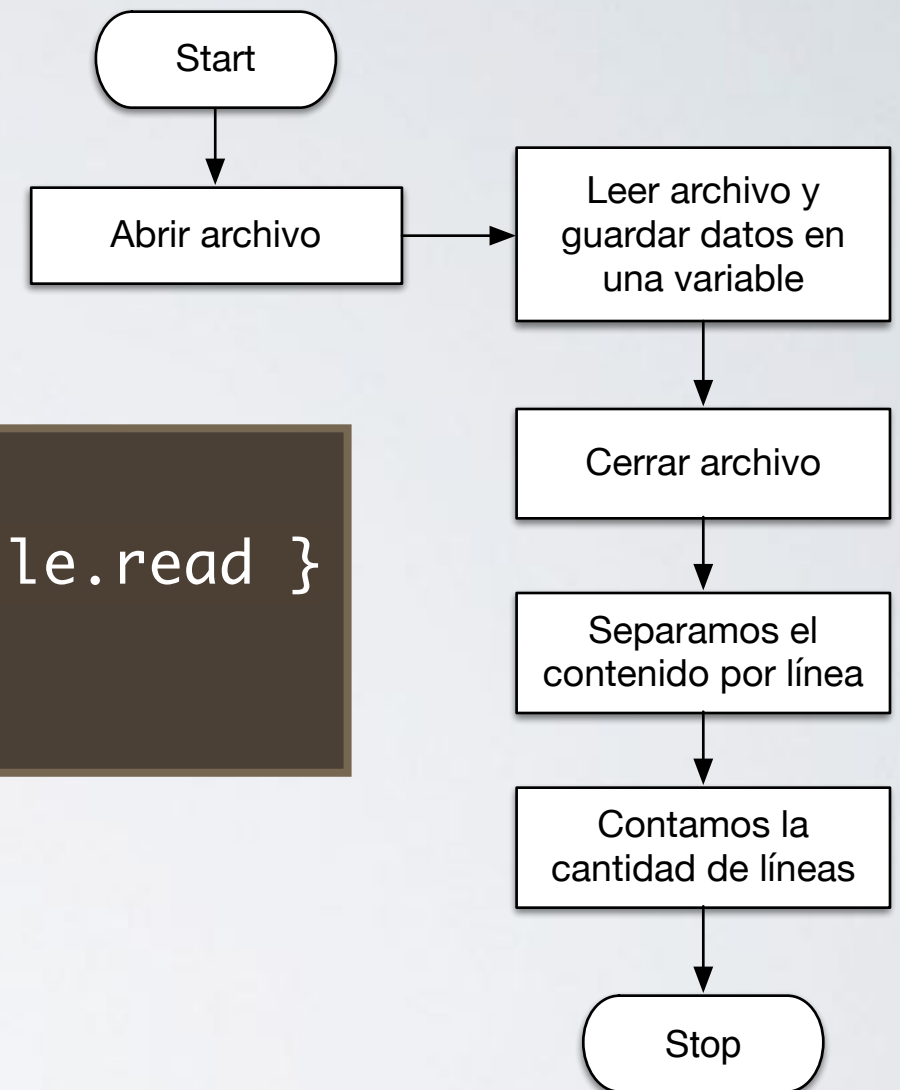
1	Product1	100	↵
2	Product2	210	↵
3	Product3	400	↵
4			

ALGORITMO DE LA SOLUCIÓN



SOLUCIÓN

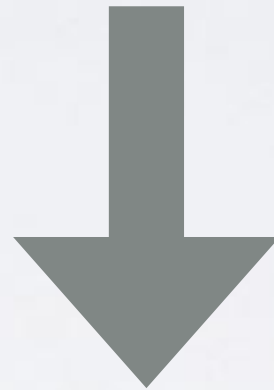
```
data = ''  
File.open('sample.txt', 'r') { |file| data = file.read }  
lines = data.split("\n")  
puts lines.count
```



MEJORANDO LA SOLUCIÓN

La clase File tiene un método llamado readlines que devuelve de forma automática el contenido separado por línea


```
data = ''  
File.open('sample.txt', 'r') { |file| data = file.read }  
lines = data.split("\n")  
puts lines.count
```



```
lines = []  
File.open('sample.txt', 'r') { |file| lines = file.readlines }  
puts lines.count
```

AHORA CALCULAMOS EL PROMEDIO

SOLUCIÓN

```
lines = []
avg = 0
File.open('sample.txt', 'r') { |file| lines = file.readlines }
lines.each do |line|
  price = line.split(' ').last.to_i
  avg += price
end

puts avg / lines.count.to_f
```

SOLUCIÓN 2.0

```
lines = []  
File.open('sample.txt', 'r') { |file| lines = file.readlines }  
print lines.inject(0){ |sum, line| sum + line.split(' ').last.to_i } / lines.count.to_f
```

¿POR QUÉ EL SIGUIENTE CÓDIGO ESTARÍA MALO?

```
a = nil  
File.open('sample.txt', 'r') { |file| a = file.read }  
puts a
```

DESAFÍO

Con el siguiente archivo, abrirlo y sumar todos los valores de la 3° columna

199403	73.35	71.45	72.50	36,065	5,844
199403	72.70	71.30	71.85	36,258	7,048
199403	73.10	71.40	71.55	35,938	6,608
199403	72.20	70.55	72.10	35,921	6,710
199403	72.70	70.65	72.50	35,468	9,432
199403	73.35	72.60	73.05	34,952	4,911
199403	73.35	71.60	71.70	35,563	5,455
199403	73.05	71.40	72.95	35,144	7,552
199403	73.80	72.60	73.45	34,671	6,055
199403	74.40	72.75	73.95	34,849	6,538
199403	74.60	73.25	73.70	34,517	5,785
199403	74.90	73.65	74.65	34,216	6,496
199403	74.10	72.00	72.50	33,792	9,940

EJERCICIO

Se tiene un archivo con un listado de palabras

preliminarily
twain
unenlightened
dupr
abomasum
huntaway
father
servantless
anthologist
equidistant
pastelist
dispellable
polypoid
preforgiving

Crear un método para determinar si una palabra se encuentra dentro de un archivo con un listado de palabras.

LEYENDO UN ARCHIVO MÁS COMPLICADO

```
Johnson Trail
Wed, 21 Dec 2005
Spyglass Entertainment
Action
bad, good, bad, good, bad, bad, good, good, bad, bad, bad, bad
Gregorio Road
Wed, 28 Jun 2006
Embassy Pictures
Documental
good, good, good, good, bad, good, bad, good, bad, bad, bad, bad
Tremayne Way
Thu, 31 Aug 2006
Spyglass Entertainment
Musical
good, bad, bad, good, good, good, good, good, good, bad, good, bad
Nicolas Parkway
Mon, 22 Sep 1997
Five & Two Pictures
Drama
good, good, bad, bad, good, good, bad, good, bad, bad, bad, good
Alfonzo Circles
Sun, 26 Feb 2012
Walt Disney Pictures
Drama
bad, bad, good, bad, bad, good, good, good, bad, bad, good, good
Windler Turnpike
Wed, 04 Dec 2002
Metro Goldwyn Mayer
Thriller
good, good, good, good, good, bad, good, good, good, good, bad, bad
```

Este archivo contiene
datos de películas

línea 1: nombre
línea 2: fecha
línea 3: estudio
línea 4: categoría
línea 5: opiniones

SABEMOS QUE CADA PELÍCULA SE COMPONE DE 5 LÍNEAS, ASÍ QUE AGRUPAMOS DE A 5

```
def show(movie)
  puts "#{movie[0]}: #{movie[1]}"
end

file = File.open "movies.txt"
data = file.readlines
file.close

data.each_slice(5) do |movie|
  show(movie)
end
```

MEJORANDO EL CÓDIGO CON EL OPERADOR SPLAT

```
def show(name, date, studio, category, votes)
  puts "#{name}: #{date}"
end

file = File.open "movies.txt"
data = file.readlines
file.close

data.each_slice(5) do |movie|
  show(*movie)
end
```

El operador splat nos permite pasar un arreglo como si fueran parámetros separados, de esta forma obtenemos un código mucho más ordenado

MEJORANDO EL CÓDIGO CON EL OPERADOR SPLAT

```
def show(name, date, studio, category, votes)
  puts "#{name}: #{date}"
end

file = File.open "movies.txt"
data = file.readlines
file.close

data.each_slice(5) do |movie|
  show(*movie)
end
```

El operador splat nos permite pasar un arreglo como si fueran parámetros separados, de esta forma obtenemos un código mucho más ordenado

DESAFÍO

```
#001, Bulbasaur, Grass
#002, Ivysaur, Grass
#003, Venusaur, Grass
#004, Charmander, Fire
#005, Charmeleon, Fire
#006, Charizard, Fire
#007, Squirtle, Water
#008, Wartortle, Water
#009, Blastoise, Water
#010, Caterpie, Bug
#011, Metapod, Bug
#012, Butterfree, Bug
#013, Weedle, Bug
#014, Kakunaa Bug
#015, Beedrill, Bug
#016, Pidgey, Normal
#017, Pidgeotto, Normal
#018, Pidgeot, Normal
#019, Rattata, Normal
#020, Raticate, Normal
#021, Spearow, Normal
#022, Fearow, Normal
#023, Ekans, Poison
#024, Arbok, Poison
#025, Pikachu, Electric
#026, Raichu, Electric
```

Leer el archivo y mostrar
en pantalla los pokemones
tipo fuego

ESCRIBIENDO UN ARCHIVO

```
file = File.open('hola.txt', 'w')  
file.puts 'Holaaaaaaa!'  
file.close
```

An orange arrow pointing from the word "Write" to the 'w' mode in the code snippet.

Write

Al abrir un archivo que no existe, este se crea, si el archivo existe y tiene contenido se borrará.

O EN FORMA DE BLOQUE

```
File.open("sample.txt", "w"){ |file| file.puts "Hello file!"}
```

No confundir **File** con **file**.

Con mayúscula hace referencia a la clase File,
con minúscula es la variable que contiene el archivo abierto

EJERCICIO

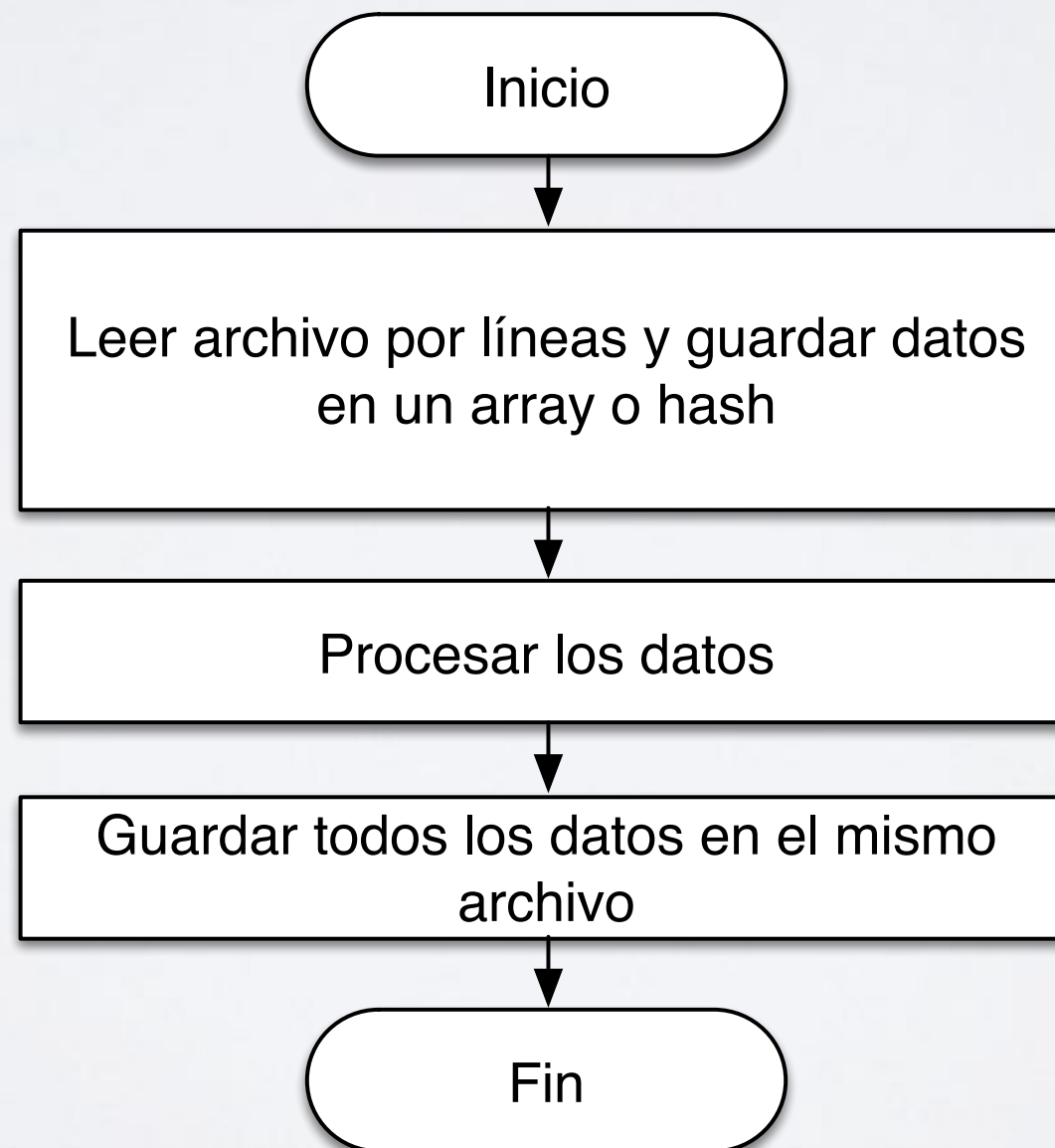
Permitir que el usuario ingrese un texto cualquiera en un archivo

EJERCICIO

El usuario ingresa un producto y un precio, se debe cambiar el precio del producto ingresado en el archivo

1	Product1	100	↵
2	Product2	210	↵
3	Product3	400	↵
4			

FLUJO BÁSICO DE TRABAJO LEYENDO DATOS Y GUARDANDO EN UN ARCHIVO



EJERCICIO DIFICULTAD MEDIA

El usuario ingresa un producto y un precio, se debe cambiar el precio del producto ingresado en el archivo si el producto no existe se debe agregar.

1	Product1	100
2	Product2	210
3	Product3	400
4		

CUANDO ESCRIBIMOS EN UN ARCHIVO BORRAMOS EL CONTENIDO ANTERIOR

Si queremos agregar contenido sin borrar debemos abrir
los archivos con **a**ppend

PERO PODEMOS ABRIR UN ARCHIVO PARA ESCRIBIR AL FINAL

```
File.open('myfile.out', 'a') { |f|  
  f.puts "Hello, world."  
}
```

An orange arrow pointing from the word "append" to the 'a' mode in the code snippet.

append

VIENDO SI UN ARCHIVO O DIRECTORIO EXISTE

```
if File.exists?(filename)
  puts "#{filename} exists"
end
```

EXCEPCIONES

EXCEPCIONES

En ruby cuando sucede algo inesperado, por ejemplo se intenta abrir un archivo que no existe o se llama a un método que no existe, se levanta automáticamente una excepción.

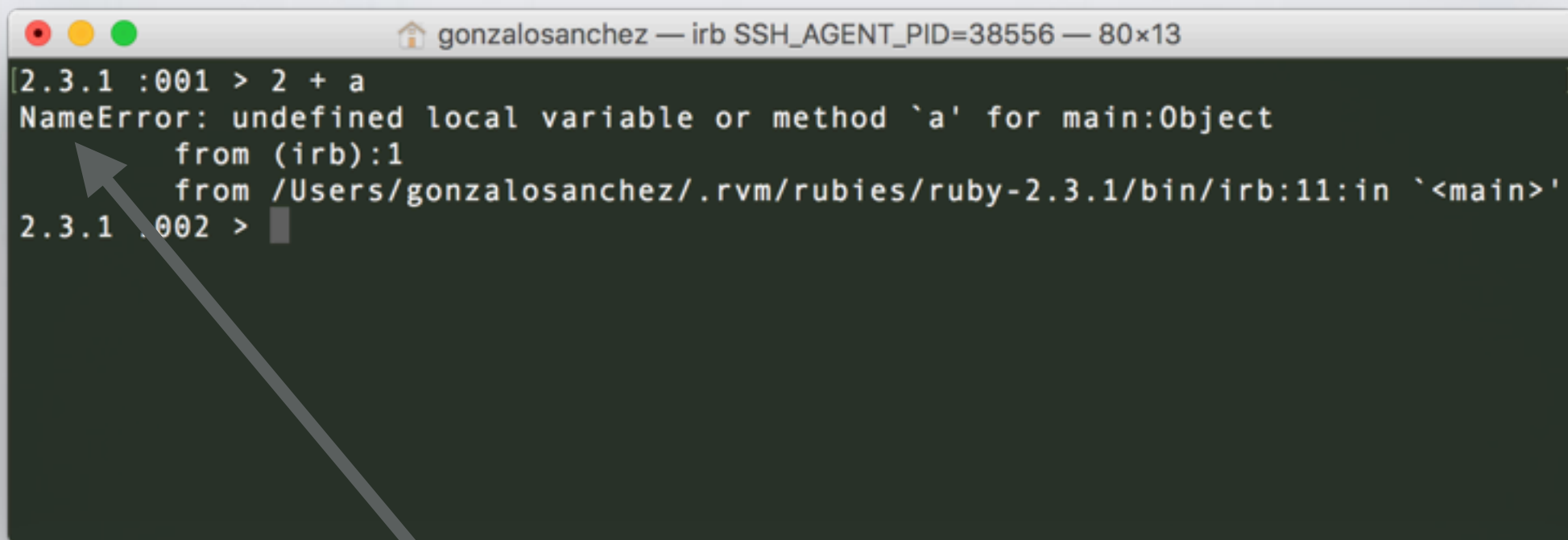
¿POR QUÉ TENEMOS QUE ESTUDIARLAS?

Entender de excepciones nos ayudará a entender mejor los errores que nos muestra ruby y a mejorar la calidad de nuestro código.

¿CÓMO MEJORAR LA CALIDAD DEL CÓDIGO?

Nosotros podemos levantar excepciones o interceptarlas. Son muy útiles para mostrar errores de uso de un código a otros programadores o a tu yo futuro.

LAS EXCEPCIONES SE LEVANTAN DE FORMA AUTOMÁTICA

A screenshot of a macOS terminal window titled 'gonzalosanchez — irb SSH_AGENT_PID=38556 — 80x13'. The terminal shows a Ruby IRB session where the command '2 + a' is entered. This results in a 'NameError: undefined local variable or method `a' for main:Object' exception. The error message is displayed with its backtrace, showing it occurred in the main session. A grey arrow points from the text 'Es el tipo de excepción' to the 'NameError' text in the terminal output.

```
2.3.1 :001 > 2 + a
NameError: undefined local variable or method `a' for main:Object
    from (irb):1
    from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.1/bin/irb:11:in `<main>'
2.3.1 :002 >
```

Es el tipo de excepción

O LAS PODEMOS LEVANTAR MANUALMENTE CON RAISE

A screenshot of a terminal window with a dark background. The window title bar shows "gonzalosanchez — irb SSH_AGENT_PID=38556 — 80x13". The terminal content shows a Ruby session where the command "raise" was entered, resulting in a "RuntimeError" with a stack trace. The prompt "2.3.1 :003 >" is followed by a cursor.

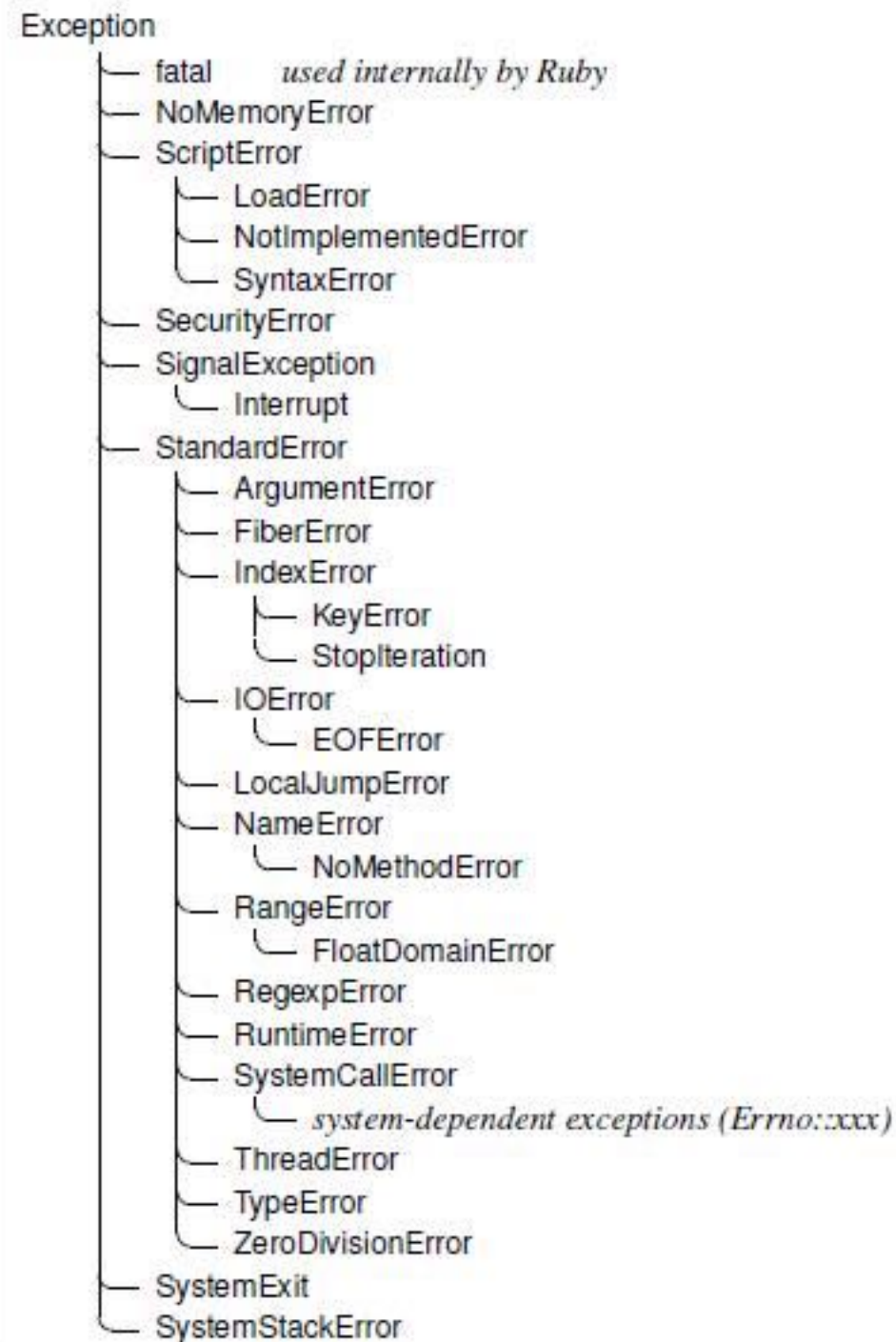
```
gonzalosanchez — irb SSH_AGENT_PID=38556 — 80x13
[2.3.1 :002 > raise
RuntimeError:
  from (irb):2
  from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.1/bin/irb:11:in `<main>'
2.3.1 :003 > █
```

CUANDO LEVANTAMOS UNA EXCEPCIÓN PODEMOS ESPECIFICAR EL TIPO

```
def inverse(x)
  raise ArgumentError, 'Argument is not numeric' unless x.is_a? Numeric
  1.0 / x
end
puts inverse(2)
puts inverse('not a number')
```

TIPOS DE EXCEPCIONES

Figure 9.1. Ruby exception hierarchy



MANEJANDO UNA EXCEPCIÓN

¡¡Rescue to the rescue!!

```
def x
  2 + 2
  raise
rescue
  puts "hola"
end
```

x

rescue permite continuar si
ocurre una excepción

ARCHIVOS Y EXCEPCIONES

```
# Open and read from a text file
# Note that since a block is given, file will automatically be closed when the block terminates
begin
  File.open('p014constructs.rb', 'r') do |f1|
    while line = f1.gets
      puts line
    end
  end

  # Create a new file and write to it
  File.open('test.rb', 'w') do |f2|
    # use "" for two lines of text
    f2.puts "Created by Satish\nThank God!"
  end
rescue Exception => msg
  # display the system generated error message
  puts msg
end
```

ANTI PATRÓN

Un anti patrón es una practica de código abusiva que a primera vista puede ser una buena idea, pero debe ser evitada

EL ANTI PATRÓN BEGIN RESCUE ALL

Las excepciones son para avisarnos de errores inesperados, si ponemos todo el código dentro de un begin y rescue para evitar un error, no podremos verlos ni manejarlos.