



# Clase 06:

# HTTP



profesor: **Patricio López Juri** { [patricio@lopezjuri.com](mailto:patricio@lopezjuri.com) }  
créditos: **10**  
horario: **J:7-8**  
sala: **H3**



**Supuesto que hago  
yo en este curso:**

*Ustedes son personas inteligentes y pueden aprender y deducir reglas lógicas en base a ejemplos.*

---

OK, en la Web nos mandamos y recibimos información.

---

OK, en la Web nos mandamos y recibimos información.

**¿Cómo funciona esto?**

---

# Las máquinas entienden binario

010010100100101001001101010  
010101010101010101011010101  
011010101010101010101010101

---

Podemos definir un vocabulario de  
**números a letras.**

**01000001 = 65 = "A"**

---

Entonces entre computadores nos mandaremos información en forma de texto.

---

Entonces entre computadores nos mandaremos información en forma de texto.

**YA LO HACEMOS CON HTML**



---

## Pero HTML como texto no es suficiente.

- ¿Dónde se manda? IP destino
- ¿Qué formato? HTML? Si es un video? o un formulario?
- ¿Desde donde? Mi IP
- ¿Tengo permisos? Cómo hago página que requiere *login*?
- ¿Error 404? Les suena? Hay más errores?

HTTP

**HyperText Transfer Protocol**

---



# HTTP es el corazón de la web

- Hacemos peticiones
- Los servidores nos contestan



# Nosotros ya usamos HTTP

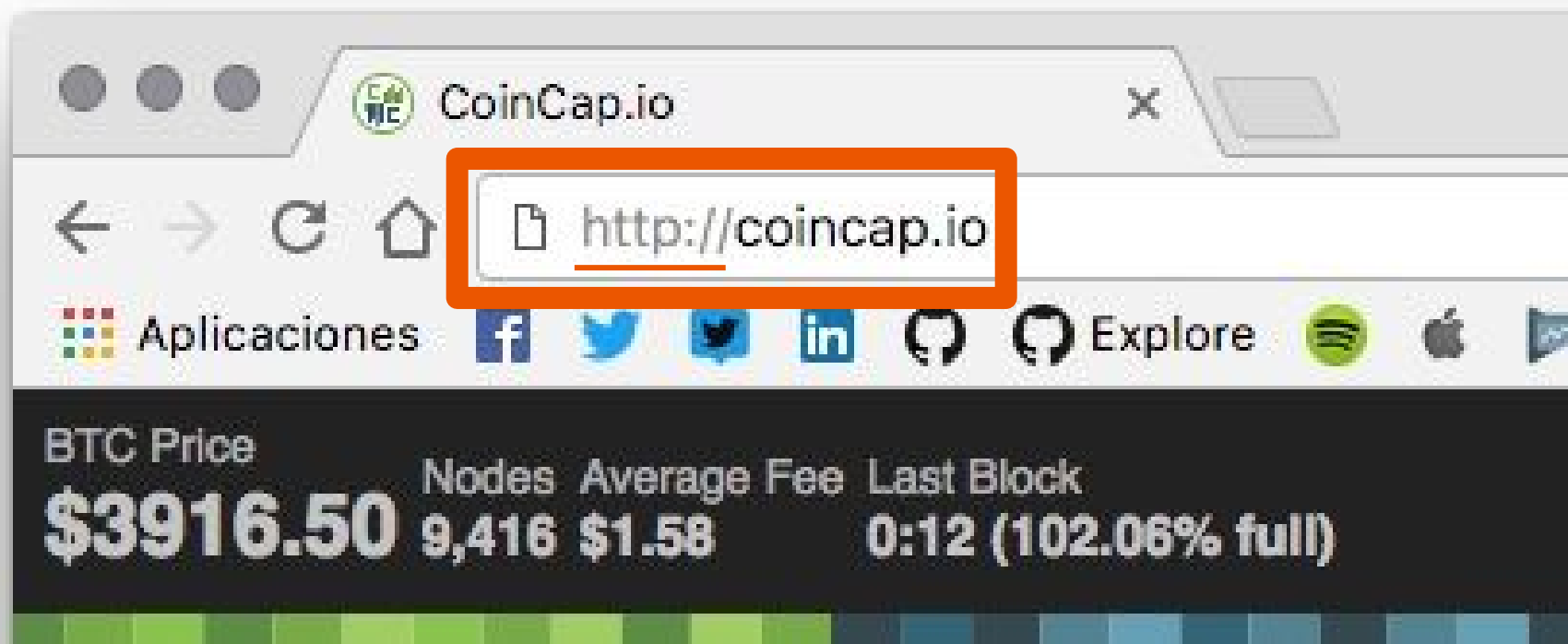
Hasta el momento hemos estado haciendo:

- Pedimos el index.html
- El servidor entrega el index.html
- El navegador web (Chrome u otro) nota que necesitamos fonts, archivos CSS, Bootstrap, etc
- El navegador pide los .CSS y fonts a los servidores correspondientes
- Los servidores correspondientes nos mandan los archivos
- El navegador muestra la página final



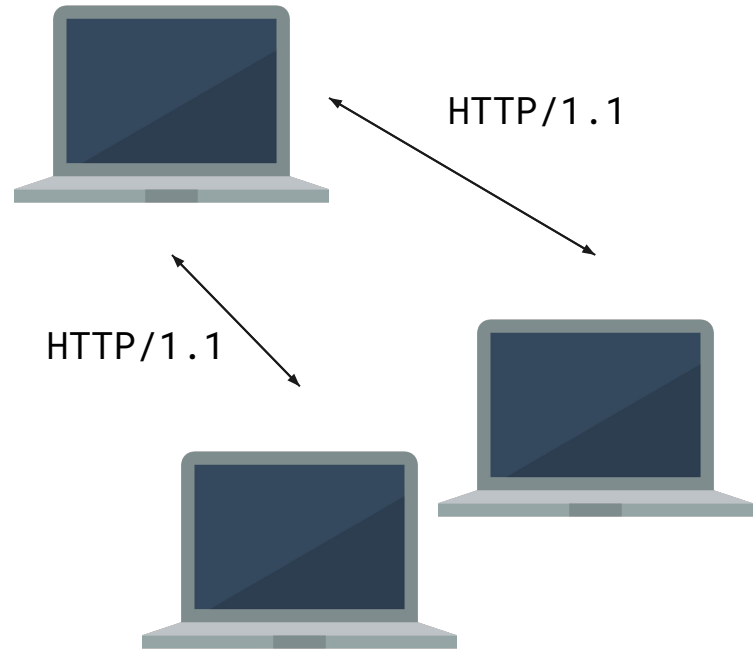
Esto ve el usuario final

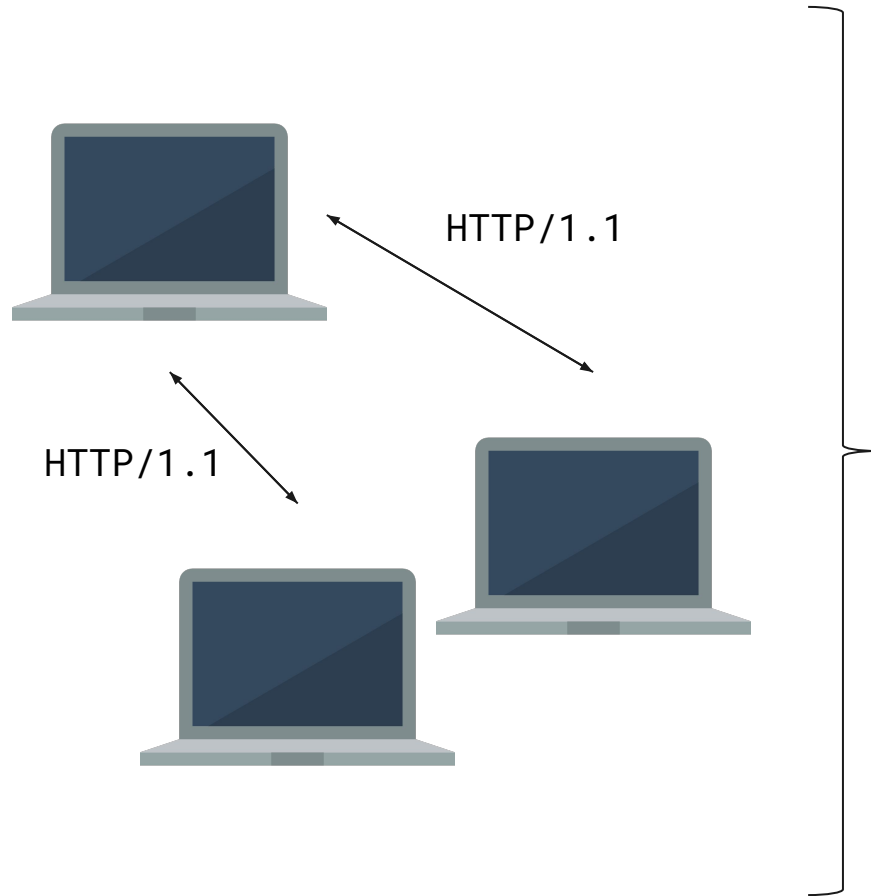
Ya lo hemos visto cuando pedimos un sitio web:



# ¿Entonces para qué sirve?

A través del protocolo HTTP podemos comunicar computadores conectados por internet.





**¡La World Wide Web  
nace del protocolo HTTP!**



---

¿Y SI QUIERO **COMPARTIR MIS**  
**SITIOS WEB QUE TENGO** EN EL  
COMPUTADOR?

---

**Necesitamos un programa** que entregue mis HTML y CSS cuando me los pidan

---

Ese programa se llama **Web  
Server**

---

Ese programa se llama **Web Server**

“Escucha” o “espera” peticiones HTTP y las responde.

**Nosotros usaremos**  
**Ruby on Rails**, un  
***framework*** para  
crear ***web servers***.

Pero eso para la próxima clase...





## Vamos a partir por algo más simple: Sinatra

Solo para entender HTTP en esta clase.

No tienen que aprenderlo.

# Sinatra



¿Qué es un framework?

¿Cuál es la diferencia entre Sinatra, Ruby on Rails... y Ruby?

¿Por qué se llama Sinatra?  
... cómo Frank Sinatra?



¿No íbamos a usar Ruby?  
... y ahora se llama Ruby EN RIELES?

Me da miedo programar

Un framework son módulos de código pre-hecho que me proveen una base para programar y mucha ayuda

Ruby es un lenguaje de programación

Sinatra y Ruby on Rails son frameworks escritos en Ruby.

Sí, por Frank Sinatra



Reitero, si usamos Ruby y además un framework para Ruby.

Les contaré por qué en "Rieles"

HAY QUE SALIR DE LA ZONA DE CONFORT!



---

ALERTA:

**VAN A ENCONTRARSE CON INFINITOS  
ERRORES DE PROGRAMACIÓN Y VAN  
A SER SIEMPRE SU CULPA POR  
PROGRAMAR MAL**

---

**ALERTA:**

**PARA EVITAR ERRORES NECESITAN  
EXPERIENCIA**

---

**ALERTA:**

**PARA GANAR EXPERIENCIA TIENEN  
QUE COMETER MUCHOS ERRORES Y  
APRENDER DE ESTOS**

---

**ALERTA:**

**Y ESTO APLICA PARA TODO EN LA  
VIDA**

Comienza lo difícil  
**LET'S GO**

---



# Un proyecto en Ruby está compuesto por:

Archivos:

- **Gemfile**
- **main.rb** o algún archivo de “entrada”.



## Gemfile contiene módulos de código externo

Esto es como cuando usábamos Bootstrap en HTML/CSS. Este era módulo externo de código que incluimos en el nuestro.

En Ruby los módulos externos se llaman “Gemas”.

En otros lenguajes se les suele llamar “librerías”.



## **Gemfile contiene módulos de código externo**

En el Gemfile guardamos las referencias a las librerías que usamos





## Gemfile de ejemplo (para Sinatra)

```
source 'http://rubygems.org/'
```

```
gem 'sinatra', '~> 2.0.0'
```

```
group :test, :development do
```

```
  gem 'ruby-debug-ide'
```

```
end
```



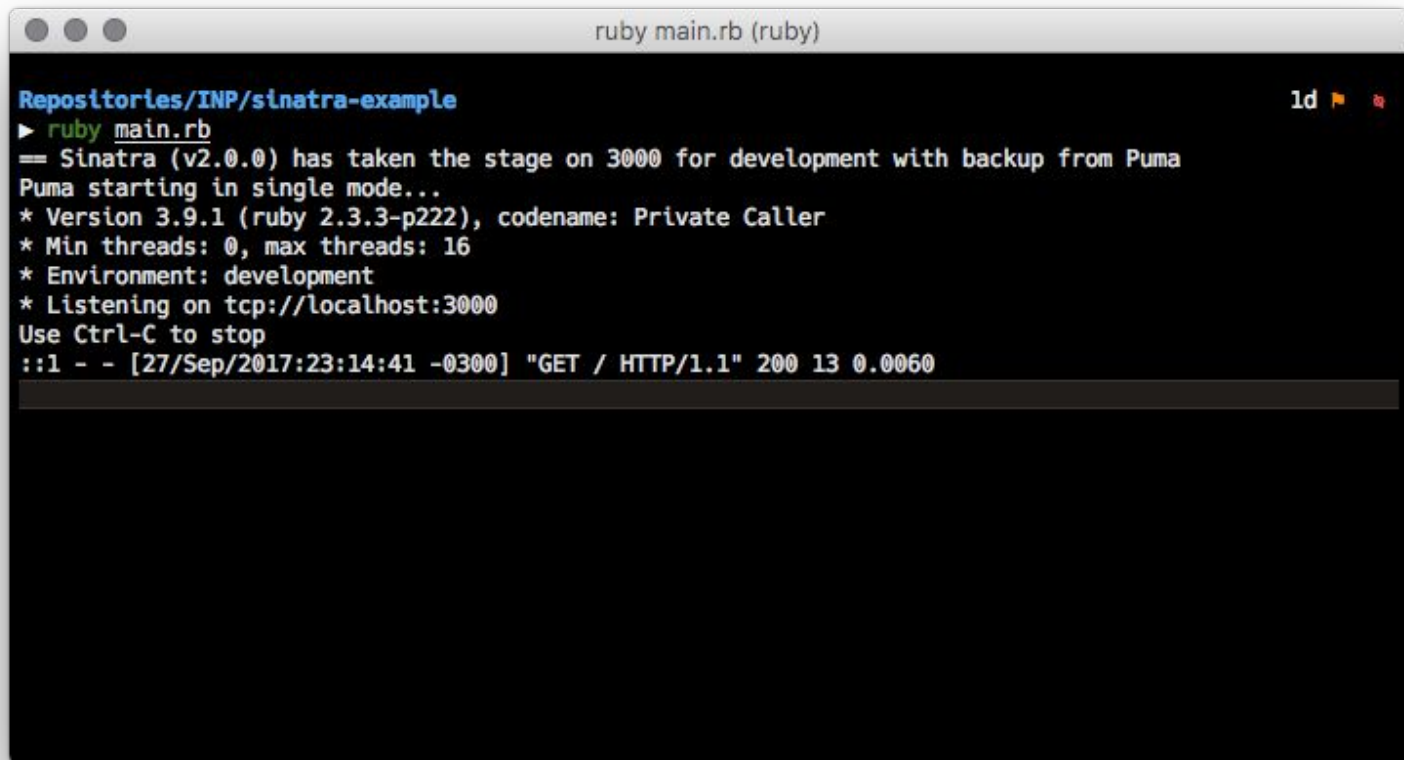
## Ahora el archivo con código Ruby

- Debe tener extensión .rb
- Es un archivo de texto como cuando escribíamos HTML y CSS
- HACER TUTORIAL DE <http://TRYRUBY.ORG>



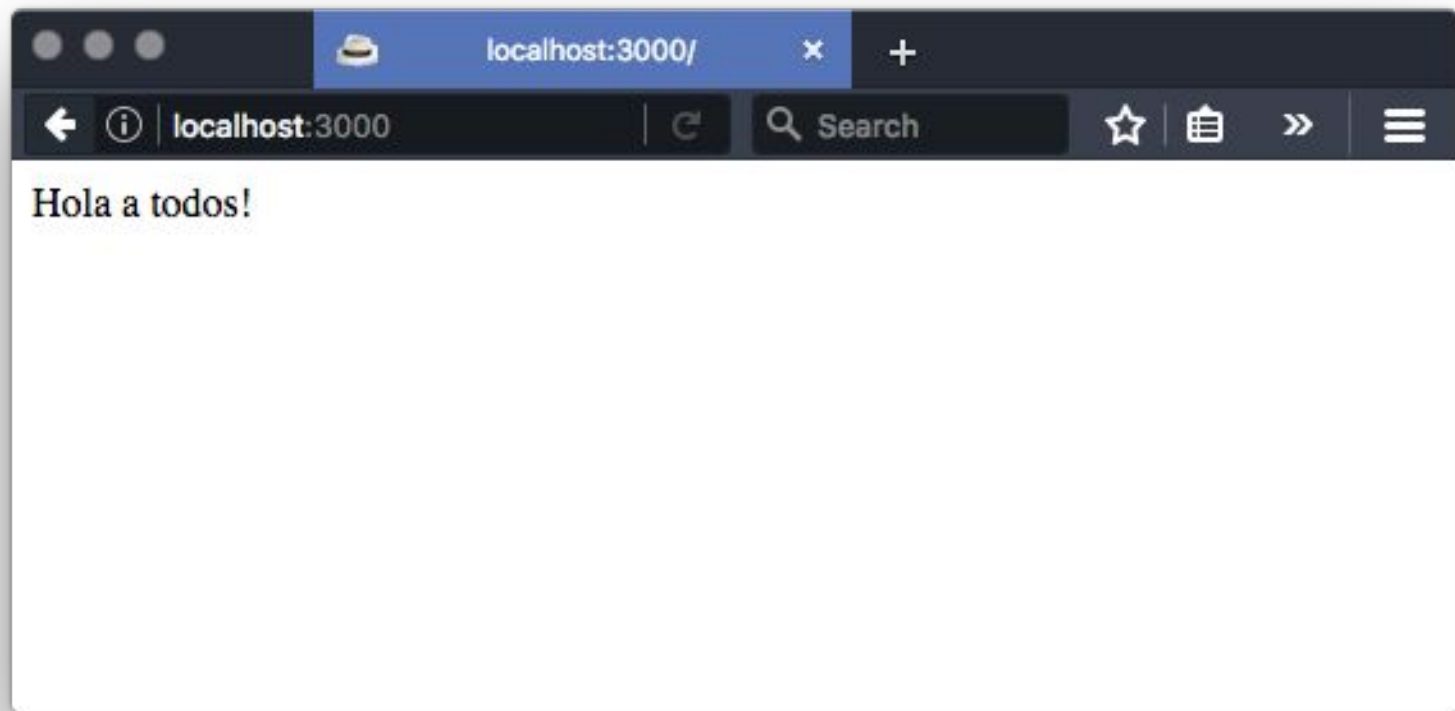
## App simple en Sinatra

```
require 'sinatra'  
  
set :port, 3000  
  
get '/' do  
  "Hola a todos!"  
end
```

A terminal window titled 'ruby main.rb (ruby)' with a dark background. It shows the execution of 'ruby main.rb' in the directory 'Repositories/INP/sinatra-example'. The output indicates that Sinatra (v2.0.0) is running on port 3000 with Puma as the server. It lists configuration details like Ruby version (2.3.3-p222), environment (development), and thread settings. A successful HTTP GET request is also shown.

```
Repositories/INP/sinatra-example 1d
► ruby main.rb
= Sinatra (v2.0.0) has taken the stage on 3000 for development with backup from Puma
Puma starting in single mode...
* Version 3.9.1 (ruby 2.3.3-p222), codename: Private Caller
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop
::1 - - [27/Sep/2017:23:14:41 -0300] "GET / HTTP/1.1" 200 13 0.0060
```

Ejecutamos el interprete de Ruby sobre el código



Si visitamos <http://localhost:3000>



## **Hicimos una app web**

Cuando la gente ingresa muestra un texto.



## ¿Qué pasa si mejor respondemos con HTML?

```
require 'sinatra'

set :port, 3000

get '/' do
  "<html>
    <body>
      <h1>Hola a todos!</h1>
    </body>
  </html>"
end
```



# Podemos incluir lógica!

```
require 'sinatra'

set :port, 3000

get '/' do
  if rand < 0.3 # 30% probabilidad
    "NADA POR AQUÍ"
  else
    "<html>
      <body>
        <h1>Hola a todos!</h1>
        <p>Son las: #{Time.now}</p>
      </body>
    </html>"
  end
end
```





# Rutas

```
require 'sinatra'

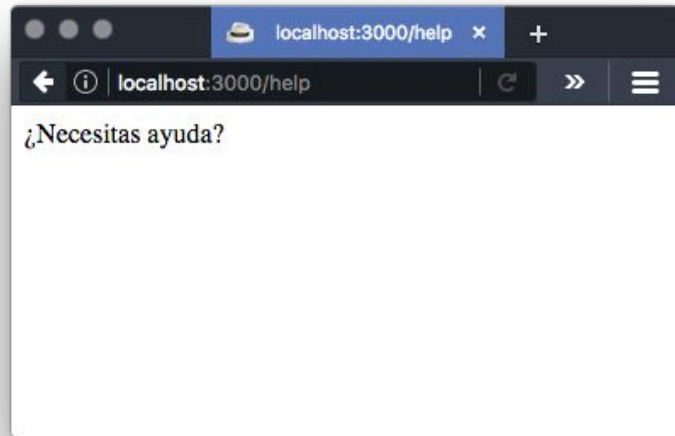
set :port, 3000

get '/' do
  # . . .
end

get "/help" do
  "¿Necesitas ayuda?"
end
```



# Rutas



# HTTP en detalle

---

# Una vista más *bruta* de una request HTTP:

Esto ve el computador

---

## Request Header (Encabezado de la petición)

Lo que mandamos

Lo que recibimos

## Response Header (Encabezado de la respuesta)

## Request Body (Cuerpo)

En este caso  
es nuestro  
HTML



```
patriciolopez@lopezjuripatricio: ~ (zsh)
▶ http http://mi-genial-sitio-web.bitballoon.com/ -p HBhb
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: mi-genial-sitio-web.bitballoon.com
User-Agent: HTTPie/0.9.9

HTTP/1.1 200 OK
Age: 347890
Cache-Control: public, max-age=0, must-revalidate
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 1054
Content-Type: text/html; charset=UTF-8
Date: Sun, 27 Aug 2017 14:39:56 GMT
Etag: "125763b21bd6de64f8dd4f41fb9937d1-df"
Server: Netlify
Vary: Accept-Encoding

<!DOCTYPE html>
<html prefix="og: http://ogp.me/ns#" lang="en">

<head>
  <title>Mi Blog</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="Sitio de arte y blogs">
  <meta name="keywords" content="Arte,Blog,Genial">
  <meta name="author" content="Patricio López">

  <meta property="og:title" content="Mi Blog de arte genial" />
  <meta property="og:type" content="image/jpeg" />
  <meta property="og:url" content="http://mi-genial-sitio-web.bitballoon.com/" />
  <meta property="og:description" content="Sitio de arte y blogs." />
```

- **Ver una página web** es una request http
- **Bajar un .css** es una request http
- **Mandar un formulario** es una request http
- **Subir un archivo** es una request http

---

# Métodos HTTP



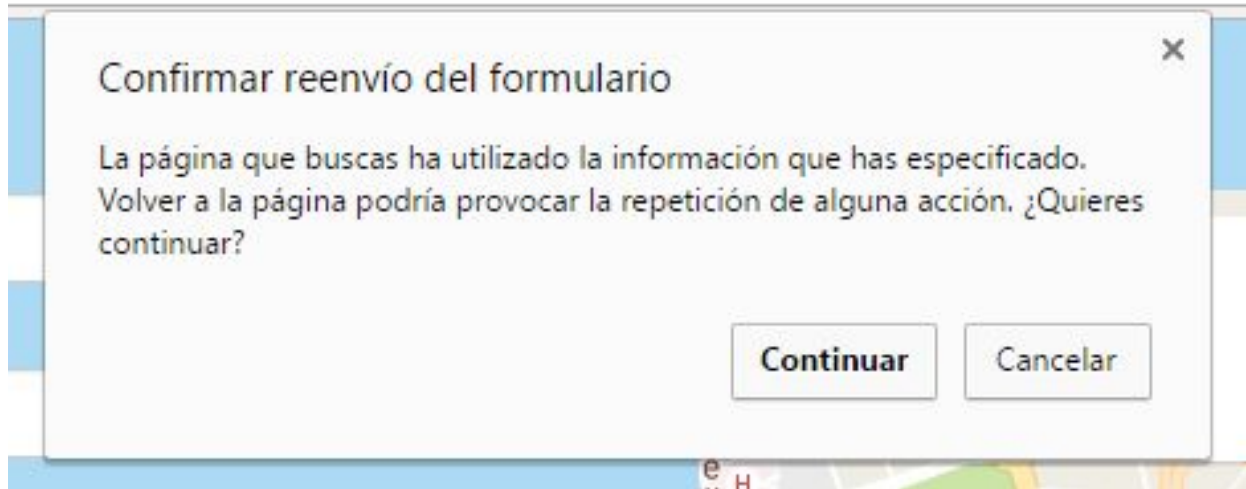
## Métodos HTTP:

# Es distinto “pedir” a “mandar” algo

- Así también es distinto “actualizar” y “borrar” algo.
- “Mandar” algo para crear un comentario debería crearlo bien, ¿pero qué pasa si mandamos muchas veces la misma información?
  - Creará un nuevo comentario duplicado.
- “Pedir” no debería tener problemas y podemos, por ejemplo, “pedir” las veces que queramos una imagen.



Les ha aparecido esto en Chrome cuando refrescan o navegan hacia atrás?



---

En el protocolo HTTP/1.1 (actual)  
encontramos los siguientes métodos:

Fuente: <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

---

# GET: pedimos un recurso

- Por lo general sin efectos secundarios
  - NO DEBERÍA CREAR UN RECURSO
  - Pero **si permitir o no acceder a un recurso público o privado**
    - No quieren ustedes que cualquier persona haga GET a sus fotos de Facebook.
- **Hasta el momento siempre le hemos hecho GET a nuestro index.html, .css e imágenes.**

---

# POST: mandamos información y/o creamos un recurso.

- Cuando:
  - creamos una cuenta
  - publicamos un comentario
  - subimos una foto

---

# DELETE: borramos un recurso

Si se hace DELETE dos veces o más a un recurso, la segunda vez en adelante debería fallar porque el recurso ya no existe.

---

# **PUT:** reemplaza completamente el recurso

Dado un identificador, reemplaza el recurso completo.

---

# **PATCH:** reemplaza parcialmente el recurso

Dado un identificador, reemplaza solo los cambios nuevos en el recurso.

---

**HEAD:** es como GET, pero sin el *body* de la *request*. Solo los *headers* de la *request*

Ojo que no estamos hablando del <head /> del HTML.



---

HAY UNOS MÁS  
PERO LO DEJAREMOS HASTA AHÍ.

# RUBY ON RAILS

---





## Ruby on Rails es un framework en Ruby

- Es mucho más grande y completo que Sinatra
- Es mucho más complejo
- Automatiza muchas de las tareas
- Usado ampliamente en la industria y Startups



# Demo de cómo se usa Ruby on Rails

La próxima clase entramos de lleno.





## PRÓXIMA CLASE

TRAER INSTALADO RUBY MINE

BAJAR COMO ESTUDIANTE USANDO EL CORREO UC EN:

<https://www.jetbrains.com/shop/eform/students>

