Ruby & Web

Juan P. Sandoval

Cosas inusuales de Ruby

Bloques

```
1.  # Form 1: recommended for single line blocks
2.  [1, 2, 3].each { |num| puts num }
3.                   ^^^^^ ^^^^^^^^
4.                     block   block
5.                   arguments body
```

```
1.  # Form 2: recommended for multi-line blocks
2.  [1, 2, 3].each do |num|
3.    puts num
4.  end
```

Bloques

```ruby
def explicit_block(&block)
  block.call # same as yield
end

explicit_block { puts "Explicit block called" }
```

Notice the `&block` parameter...

Symbol vs String

Estructuras de Datos Utiles

Arreglos

```ruby
# ############
# Arreglo

an_array = Array.new

an_array.push 2
an_array.push 7
an_array.push 1
an_array.push 4

puts an_array.inspect
# imprime [2,7,1,4]

# otra forma:
an_array = [2,7,1,4]
puts an_array.inspect
```

Rango

```ruby
# ###########
# Rango

a_range = Range.new(1,20) # del 1 al 20, inclusive
a_range = Range.new(1,20,true) # del 1 al 19, inclusive

a_range = 1..20 # del 1 al 20, inclusive
a_range = 1...20 # del 1 al 19, inclusive

puts a_range.to_a.inspect
# imprime [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
#          14, 15, 16, 17, 18, 19]

a_range = ('a'..'f')

puts a_range.to_a.inspect
# imprime ["a", "b", "c", "d", "e", "f"]
```

Hashes

```
dictionary = { "one" => "eins", "two" => "zwei", "three" => "drei" }

puts dictionary["one"]
```

Hashes

```
dictionary = { "one" => "eins", "two" => "zwei", "three" => "drei" }
dictionary["zero"] = "null"
puts dictionary["zero"]
```

This prints out `null`.

Hashes

```
{ :one => "eins", :two => "zwei", :three => "drei" }
{ one: "eins", two: "zwei", three: "drei" }
```

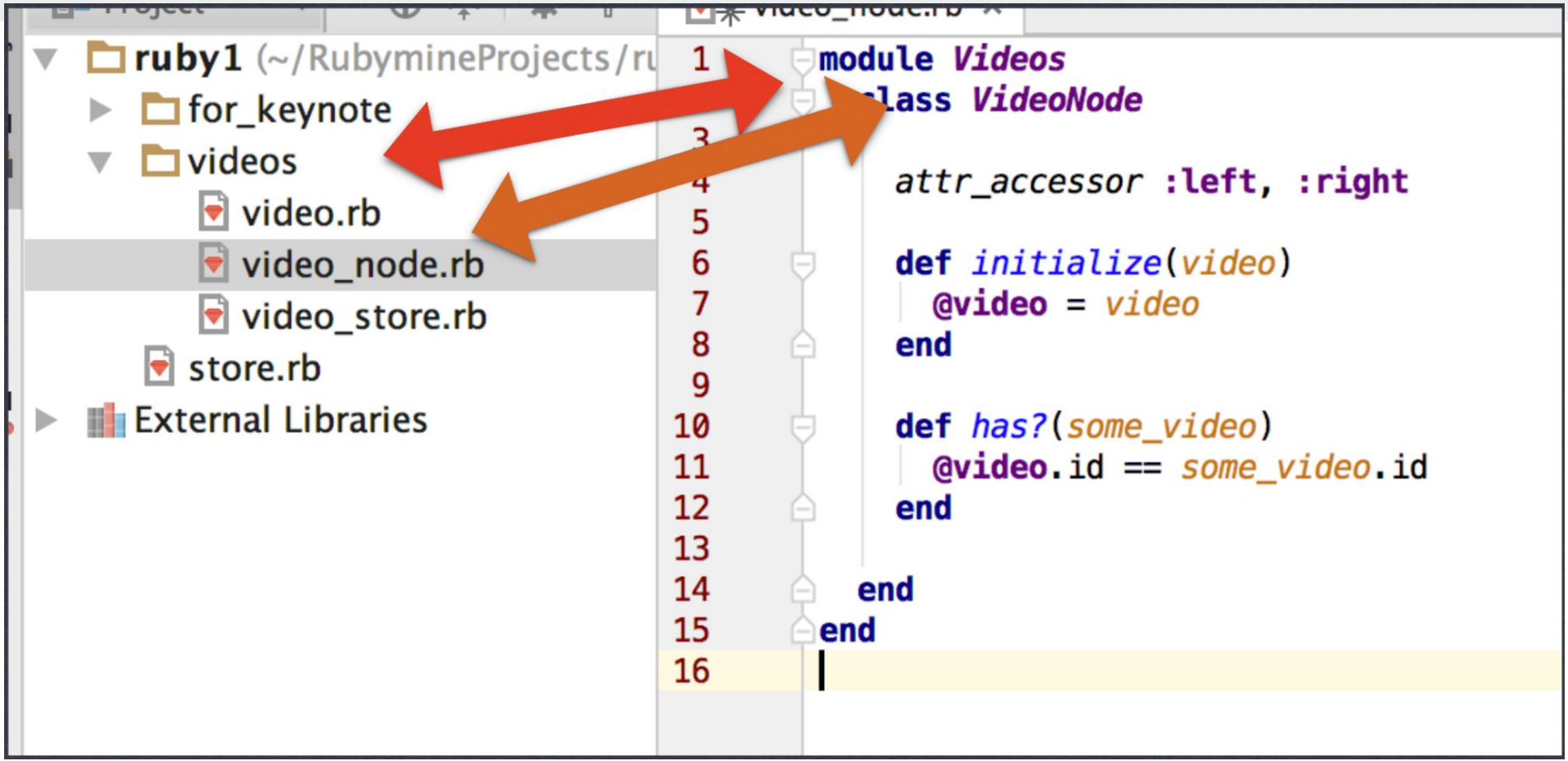
Hash keys

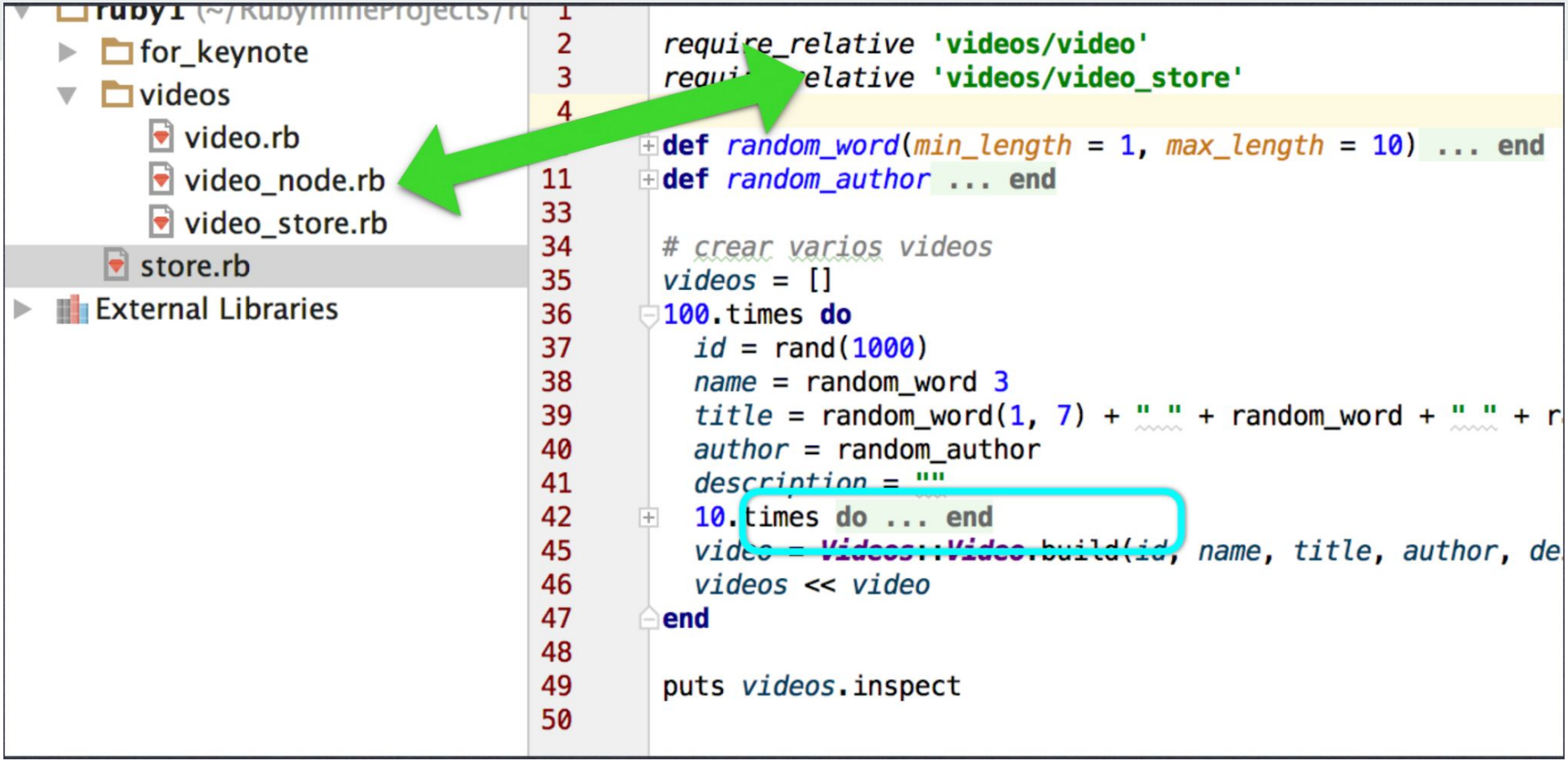
```
$ irb
> dictionary = { "one" => "eins", "two" => "zwei" }
> dictionary.keys
=> ["one", "two"]
```

Hash methods

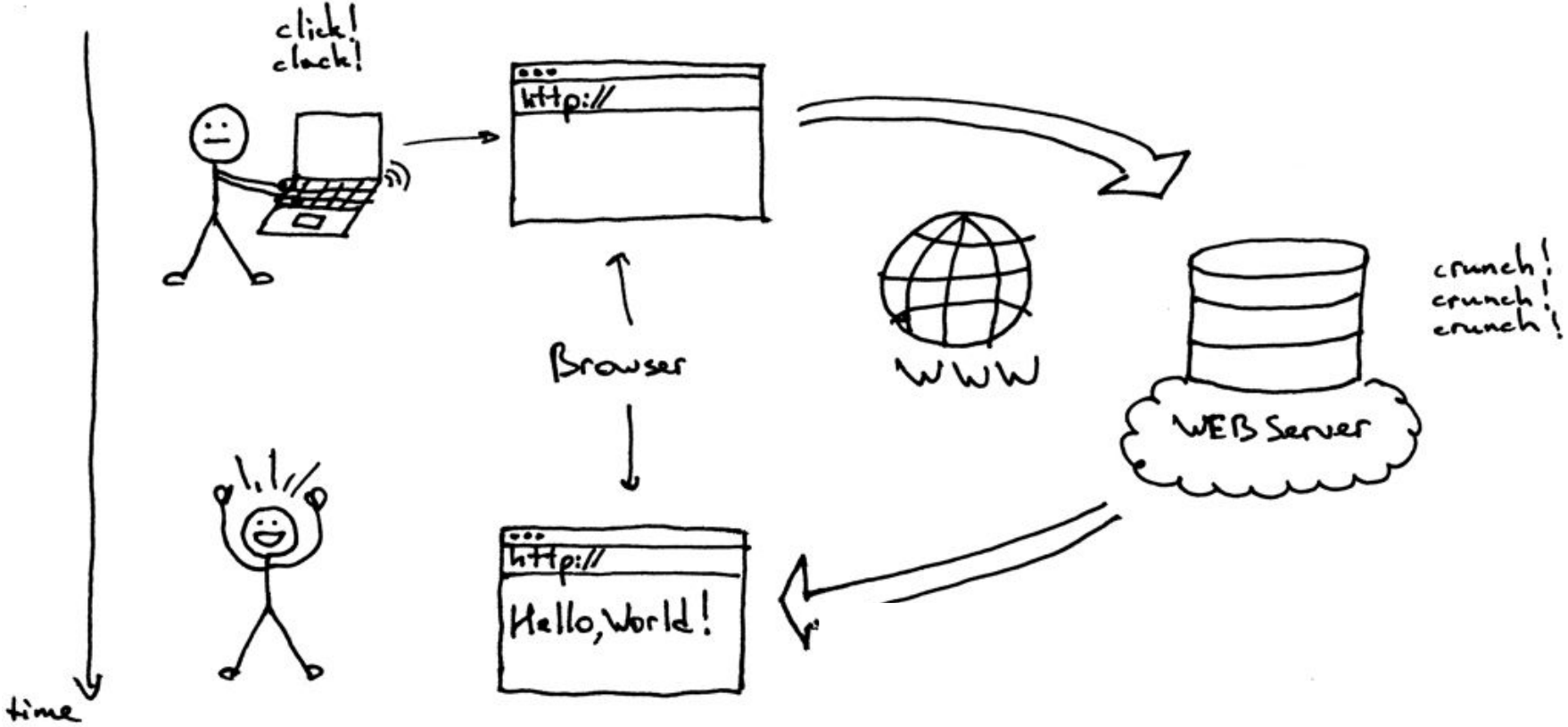
```
$ irb
> dictionary = { "one" => "eins", "two" => "zwei" }
> dictionary.length
=> 2
> dictionary.size
=> 2
```

Archivos


ruby1 (~/RubymineProjects/ru
for_keynote
videos
video.rb
video_node.rb
video_store.rb
store.rb
External Libraries
module Videos
class VideoNode
attr_accessor :left, :right
def initialize(video)
@video = video
end
def has?(some_video)
@video.id == some_video.id
end
end
end


ruby1 (~/RubymineProjects/ru
for_keynote
videos
video.rb
video_node.rb
video_store.rb
store.rb
External Libraries
require_relative 'videos/video'
require_relative 'videos/video_store'
def random_word(min_length = 1, max_length = 10) ... end
def random_author ... end
# crear varios videos
videos = []
100.times do
id = rand(1000)
name = random_word 3
title = random_word(1, 7) + " " + random_word + " " + r
author = random_author
description = ""
10.times do ... end
video = Videos::Video.build(id, name, title, author, de
videos << video
end
puts videos.inspect

Web


click!
clack!
http://
Browser
WWW
WEB Server
crunch!
crunch!
crunch!
http://
Hello, World!
time

00110010 00110000 00110010 0011

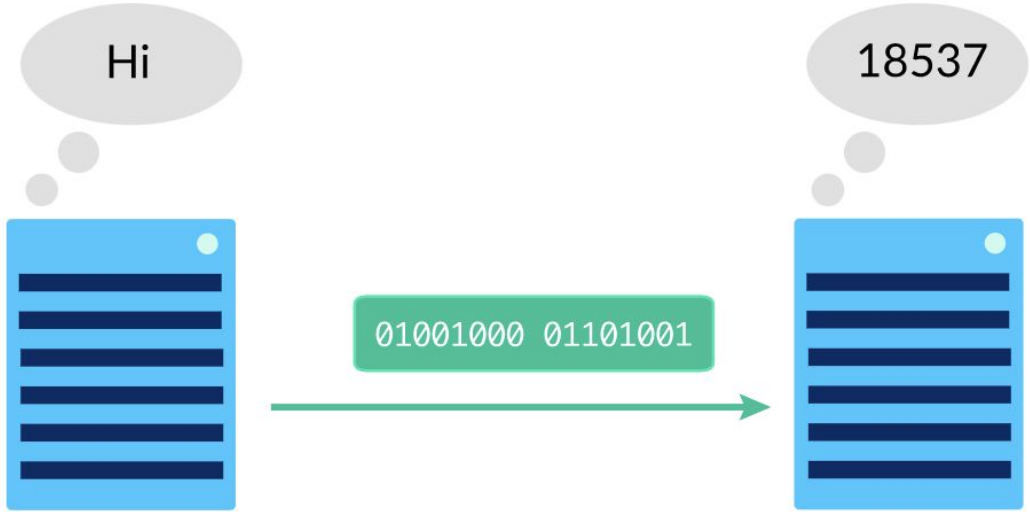
00110010 00110000 00110010 0011



50 48 50 en decimal

“2022” en String

Desafortunadamente, la computadora B piensa que está recibiendo un número e interpreta el mensaje como el número decimal 18537.



necesitamos un protocolo

Creemos un pequeño server en Ruby

```ruby
require 'socket'
server = TCPServer.new 5678

while session = server.accept
  session.puts "Hello world!"
  session.close
end
```

server

```ruby
# tcp_client.rb
require 'socket'
server = TCPSocket.new 'localhost', 5678

while line = server.gets
  puts line
end

server.close
```

client

que pasa si el cliente es un browser

```
3   require 'socket'
4   server = TCPServer.new 5678
5
6   while session = server.accept
7     session.puts "Hello world!"
8     session.close
9   end
```

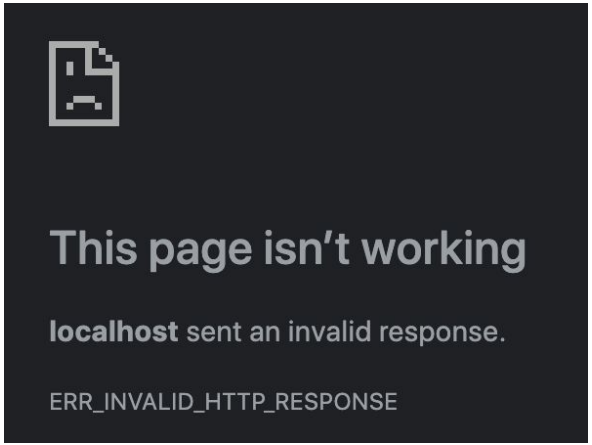
server

client

El browser no entiende el formato en que le mandamos la información

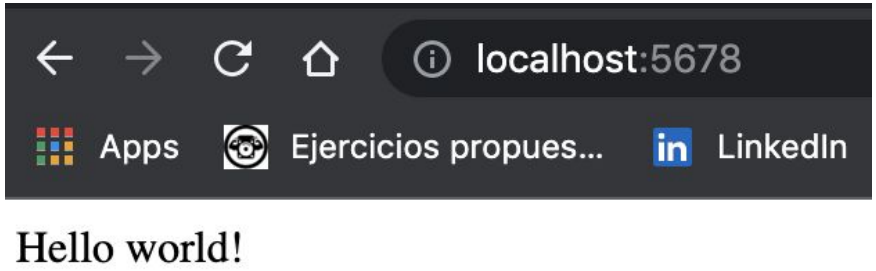
```
3   require 'socket'
4   server = TCPServer.new 5678
5
6   while session = server.accept
7     session.puts "Hello world!"
8     session.close
9   end
```

server

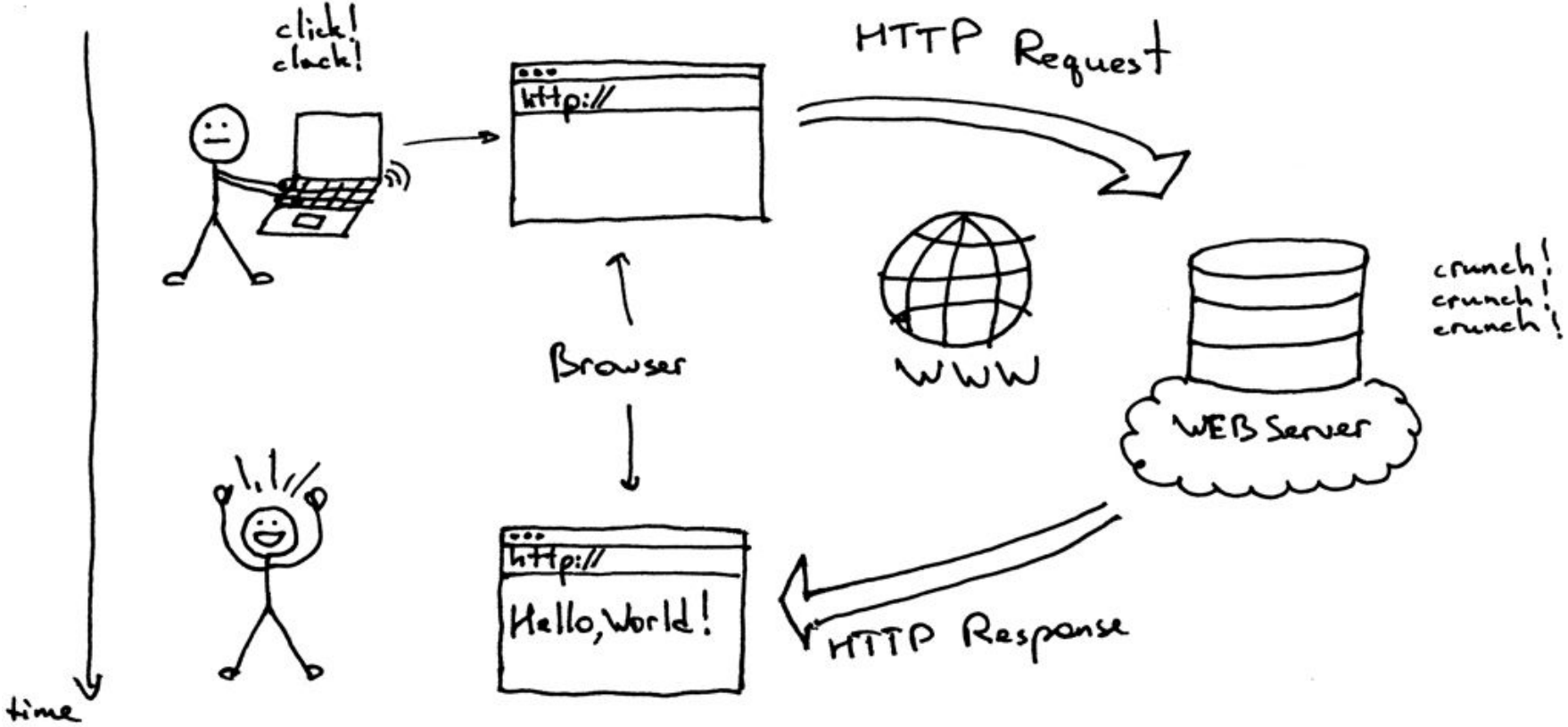


client

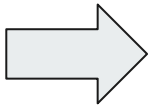
```ruby
# http_server.rb
require 'socket'
server = TCPServer.new 5678

while session = server.accept
  request = session.gets
  puts request

  session.print "HTTP/1.1 200\r\n" # 1
  session.print "Content-Type: text/html\r\n" # 2
  session.print "\r\n" # 3
  session.print "Hello world!" #4

  session.close
end
```



Server 2.0


click!
clack!
http://
HTTP Request
WWW
crunch!
crunch!
crunch!
WEB Server
Browser
http://
Hello, World!
HTTP Response
time

pero qué información me mando el browser?



```ruby
# http_server.rb
require 'socket'
server = TCPServer.new 5678

while session = server.accept
    http_request = ""
    while (line = session.gets) && (line != "\r\n")
        http_request += line
    end
    STDERR.puts http_request
    STDOUT.flush
    session.close
end
```

```
GET / HTTP/1.1
Host: localhost:5678
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHT
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
```


```
GET / HTTP/1.1
Host: localhost:5678
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHT
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
```

Okay

- Los browsers normalmente tienen un protocolo en el cual se basan para mandar y recibir e interpretar los mensajes.
  - **Hypertext Transfer Protocol (HTTP)**

Como puedo generar contenido dinámicamente?

- Quiero una pagina:
  - que muestre mi lista de tareas
  - que muestre mi lista de compras
- **Pero como?**

```ruby
server = TCPServer.new 5678

while session = server.accept
    http_request = ""
    while (line = session.gets) && (line != "\r\n")
        http_request += line
    end

    if http_request.match(/tareas/) != nil
        tareas(session)
    elsif http_request.match(/compras/) != nil
        compras(session)
    else
            session.print "HTTP/1.1 204\r\n" # 1
            session.print "Content-Type: text/html\r\n" # 2
            session.print "\r\n" # 3
            session.print "" #4
    end
    STDOUT.flush
    session.close
end
```

```ruby
def tareas(session)
    session.print "HTTP/1.1 200\r\n" # 1
    session.print "Content-Type: text/html\r\n" # 2
    session.print "\r\n" # 3

    session.print "<html><body><ul>" #4
    10.times {|index|
        session.print "<li>Tarea No #{index}</li>" #4
    }
    session.print "<html></body></ul>" #4

end
```

Menos mal tenemos librerías/frameworks que nos hacen la mayoría de la pega anterior

la siguiente clase

HTML + CSS + javascript demo

https://onecompiler.com/html

https://www.w3schools.com/html/

https://www.w3schools.com/css/

JUST DO IT