

Genaver!

por Dave Aronson

T.Rex-2023@Codosaur.us

twitter.com/DaveAronson

linkedin.com/in/DaveAronson

Speaker notes

NOTE TO READERS: If viewing this in Preview (not sure about Acrobat etc.), try adding another page in the front, and viewing it two slides at a time (Command-3). That way you'll have the current slide on the left, and its corresponding speaker notes on the right. Eventually maybe I'll figure out a way to make Reveal.JS put them both on one page in portrait orientation, like Keynote does.

Genaver!

por Dave Aronson

T.Rex-2023@Codosaur.us

twitter.com/DaveAronson

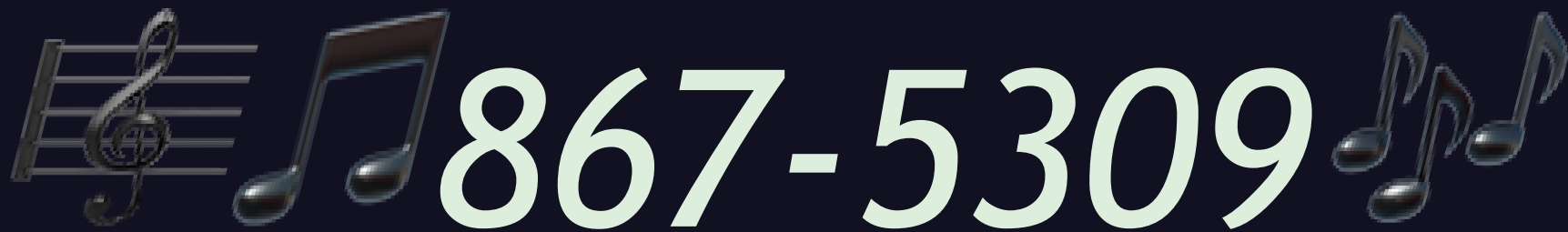
linkedin.com/in/DaveAronson

Speaker notes

Olá, São Paulo! Meu nome é Dave Aronson, sou o T. Rex de Codosaurus, e voei aqui no meu pterodátilo de estimação para falar sobre a minha gambiarra, que eu chamo de GENAVER!

Geralmente, quando eu faço uma palestra num país que não fala inglês, eu faço uma introdução como isso, na linguagem local, terminando com "Mas . . . vou fazê-lo em inglês." Mas neste caso, não é a verdade! Vocês vão ouvir minha primeira palestra fora do inglês!

Mas, voltando ao assunto: O nome Genaver não é uma referência ao nome de menina . . .



867-5309

Speaker notes

. . . Jennifer, nem . . .



Speaker notes

. . . à Rainha Guinevere, nem . . .



Speaker notes

. . . a bebida holandês, em verdade pronunciada je-NEE-ver, como eu descobri só este ano em maio. Genaver é uma abreviação para uma frase em inglês,

GENerate

A

VERsion

Speaker notes

. . . GENerate A VERsion, ou em português, gerar uma versão.

Mas, uma versão do que, e por que? Vocês que viram . . .



Speaker notes

. . . a GambiConf do ano passado, em Lisboa, podem lembrar de . . .



 **Dave
Aronson**

**Kill All
Mutants!**
(Intro to
Mutation Testing)

**GAMBI
CONF EU**

03/09/2022
Lisbon,
Portugal

Partners

R


Fronteiras da
Engenharia de
Software


Speaker notes

. . . mim, falando (em inglês) sobre Teste de Mutação, ou em inglês, Mutation Testing.

Não foi a única vez que eu fiz essa apresentação. Eu a fiz cerca vinte-cinco vezes, com a introdução em muitas linguagens humanas, o código em várias linguagens de programação, e com durações diferentes, de vinte minutos a setenta. Eu digo tudo isso não para gabar, hmmm, talvez um pouco, mas principalmente para contar por que eu criei Genaver.

Manter uma apresentação em PowerPoint ou Keynote, faz um arquivo muito grande e binário. Por isso, é muito difícil de compará-los, ou de encontrar uma boa frase de uma versão anterior, para colocá-la numa versão nova.

Por estas razões, eu queria um jeito de manter uma . . .

Apresentação

Introdução em Português

Introdução em Japonês

Introdução em Klingon

...

Explicação Longa

Explicação Média (da mesma coisa)

Explicação Curta

...

Código em Elixir

Código em Ruby

Código em Befunge

...

Speaker notes

. . . "versão primária", com a introdução e o código em todas as linguagens que eu usei, e com todas as descrições, todos os pedaços que não mudam, etc. Então, eu poderia gerar uma versão com, por exemplo, a introdução em japonês, o código em Ruby, e cabendo em quarenta e cinco minutos, ou com a introdução sueca, o código em Elixir (uma linguagem criada por um brasileiro!), e cabendo em trinta minutos, sem precisar editar o arquivo . . . muito. Idealmente, eu poderia fazer uma nova versão por adicionar as novas seções nas novas linguagens, e passar novos argumentos para um programa para escolher essas linguagens, e as explicações longas, curtas, ou médias.

Eu sabia que o primeiro passo seria pôr a apresentação num formato textual (além das imagens). Felizmente, há muitos sistemas de fazer apresentações em HTML ou Markdown. O mais famoso é RevealJS, qual eu escolhi, e qual estou usando para esta apresentação agora.

Depois disso, pensei que precisaria . . .

```
<dave-special-tag linguagem-humana="português">  
  Isto é em português  
</dave-special-tag>
```

```
<dave-special-tag linguagem-humana="igpay-atinlay">  
  Isthay isay inay igpay atinlay  
</dave-special-tag>
```

```
<dave-special-tag linguagem-de-programação="elixir">  
  IO.puts("Olá, mundo!")  
</dave-special-tag>
```

```
<dave-special-tag linguagem-de-programação="c">  
  printf("Olá, mundo!\n")  
</dave-special-tag>
```

Speaker notes

. . . marcar o HTML com tags especiais, e escrever um programa para aceitar argumentos que expressam meus desejos, analisar essa HTML, e . . .

Com argumentos de escolher
português e Elixir:

Isto é em português

```
IO.puts("Olá, mundo!")
```

Speaker notes

. . . fazer um novo arquivo de acordo com meus desejos. Eu me esforcei um pouco em pensar nisso, e . . . procrastinei. Meses mais tarde, tenho a ideia de fazer um abuso grosseiro do . . .


```
#define ENGLISH 1
#define FRANCAIS 2
#define PORTUGUÊS 3
/* edit below to prep new version */
#define WANTED_LANGUAGE PORTUGUÊS
/* ... */
#if WANTED_LANGUAGE == ENGLISH
    Hello, world!
#elif WANTED_LANGUAGE == FRANCAIS
    Bonjour, monde!
#elif WANTED_LANGUAGE == PORTUGUÊS
    Olá, mundo!
#endif
```

Speaker notes

. . . pré-processador da linguagem C, uma gambiarra verdadeira, usando principalmente as diretivas "define" e "if", e talvez . . .

```
#include "introduction.html"  
#include "problem_description.html"  
#include "proposed_solution.html"  
#include "tech_description.html"  
#include "pros_and_cons.html"  
#include "conclusion.html"
```

Speaker notes

. . . "include", para organizar os pedaços de uma apresentação. Isso daria muito menos trabalho! Mas, de novo, eu procrastinei, porque não gostei muito dessa solução. Ela me pareceu um pouco ... suja demais.

Semanas mais tarde, eu discuti a ideia do programa com um outro palestrante. Ele recomendou que eu usasse uma biblioteca em Python para processamento de texto, com que ele teve muito sucesso. Ele começou a usar isso porque ele queria se afastar do processamento de texto em JavaScript.



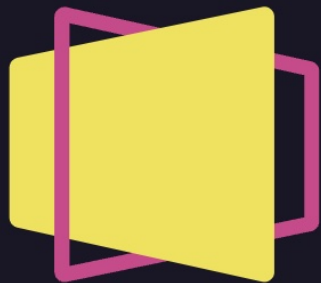
Speaker notes

DING! Ou talvez, D'OH!

Finalmente, eu tive meu momento de lâmpada! O texto já foi em HTML, e mudar a aparência de HTML é exatamente o papel de JavaScript! Melhor ainda, é uma das tarefas (na minha opinião, das poucas tarefas) para as quais o JavaScript é verdadeiramente bom! Eu só tinha que mudar minha ideia de copiar os pedaços bons, a esconder ou remover os ruins!

Então, eu fiz um pequeno Proof of Concept, ou em português, Prova de Conceito.

Já comecei escrever minhas novas apresentações em . . .



reveal.js

THE HTML PRESENTATION FRAMEWORK

Created by [Hakim El Hattab](#) and [contributors](#)

Sponsored by



codeless



slides

Speaker notes

. . . RevealJS, e ia fazer uma, sobre Algoritmos Genéticos, nas semanas que vem. Eu ia fazê-la uma vez em vinte e cinco minutos com a introdução em espanhol, uma vez em trinta e cinco minutos com a introdução em holandês, e uma vez em cinquenta minutos com a introdução em norueguês. Eu já tive as três versões separadas, e as combinei, numa "versão primária" para testar minha gambiarra. E . . .



Speaker notes

. . . ela deu certo! Eu pudesse passar argumentos diferentes ao JavaScript, para escolher a introdução, e remover pedaços que precisam tempo demais, às vezes substituindo pedaços menores.

Não ainda confiei nela, suficiente para usá-la nas conferências, mas continuei a adicionar recursos, como passar os argumentos na URL, substituindo pelos valores passado direto ao JavaScript. Então, pudesse usar exatamente o mesmo HTML para muitas versões de uma apresentação, e configurá-lo só com argumentos na URL.

Mas, como ela funciona?

```

function generate_version(args=[]) {
  const url_params = Array.from(new URLSearchParams(window.location.search));
  var specs;

  specs = parse_specs(args, {});
  specs = parse_specs(fix_url_params(url_params), specs);

  for (const key of Object.keys(specs)) {
    var op, val;
    [op, val] = specs[key];
    var data_key = `data-${key}`;
    switch(op) {
      case '=': filter_str_same(data_key, val); break;
      case '!=': filter_str_diff(data_key, val); break;
      case '<': filter_num(data_key, val, is_lt); break;
      case '<=': filter_num(data_key, val, is_le); break;
      case '>': filter_num(data_key, val, is_gt); break;
      case '>=': filter_num(data_key, val, is_ge); break;
      case '@': filter_num_at(data_key, val); break;
    }
  }
}

const SPEC_REGEX = /^[a-zA-Z0-9_]\s*([<=>|@])\s*(.+)$/;

function parse_specs(args, so_far) {
  for (var arg of args) {
    var key, op, val;
    [key, op, val] = SPEC_REGEX.exec(arg).slice(1, 4);
    so_far[key] = [op, val];
  }
  return so_far;
}

function fix_url_params(url_params) {
  const fixed_params = [];
  for (var [key, val] of url_params) {
    if (key.startsWith('gv-')) {
      const key2 = key.replace(/gv-/ , '');
      fixed_params.push(val ? `${key2}=${val}` : key2);
    }
  }
  return fixed_params;
}

function filter_num_at(key, value) {
  filter_num(`${key}-min`, value, is_le);
  filter_num(`${key}-max`, value, is_ge);
}

function is_ge(x, limit) { return x >= limit }
function is_gt(x, limit) { return x > limit }
function is_le(x, limit) { return x <= limit }
function is_lt(x, limit) { return x < limit }

function filter_num(key, value, comp_fn) {
  const limit = parseInt(value);
  const all = document.querySelectorAll(`[${key}]`);
  for (elt of all) {
    if (!comp_fn(parseInt(elt.getAttribute(key)), limit)) {
      elt.style.remove();
    }
  }
}

function filter_str_diff(key, value) {
  const all =
    document.querySelectorAll(`[${key}=${value}]`);
  for (elt of all) elt.style.remove();
}

function filter_str_same(key, value) {
  const all = document.querySelectorAll(`[${key}]`);
  for (elt of all) {
    if (elt.getAttribute(key) != value) elt.style.remove();
  }
}

```

Speaker notes

Aqui está o código inteiro de Genaver. Não, não espero que vocês leiam tudo isso, numa fonte tão pequena! Vamos ver isto peça por peça. Mas antes, mais um pequeno aviso:



Speaker notes

Eu não sou especialista em JavaScript, então, a maneira como eu fiz isto, pode ser não a melhor. Não tome isso como um exemplo de código ótimo de JavaScript. Mas, vamos dar uma olhada.

Inicialmente, . . .

```
function generate_version(args=[]) {  
  const url_params = Array.from(new URLSearchParams(window.location.search));  
  var specs;  
  
  specs = parse_specs(args, {});  
  specs = parse_specs(fix_url_params(url_params), specs);  
}
```

Speaker notes

. . . ela aceita só um argumento, opcional, passado ao JavaScript, que é uma lista de filtros, padrão para uma lista vazia. Cada filtro é um string. A lista é analisada por . . .


```
const SPEC_REGEX = /^[a-zA-Z0-9_]+\s*([<=>!@]+\s*(.+))$/;

function parse_specs(args, so_far) {
  for (var arg of args) {
    var key, op, val;
    [key, op, val] = SPEC_REGEX.exec(arg).slice(1, 4);
    so_far[key] = [op, val];
  }
  return so_far;
}
```

Speaker notes

. . . uma função que usa um regex, ou "expressão regular", para dividir a string em uma chave, uma comparação, e um valor, e os armazena num dicionário.

As chaves representam escolhas como linguagem humana, linguagem de programação, tempo mínimo ou máximo de qual uma coisa precisa, ou qualquer outra coisa que querem escolher.

As comparações são . . .

```
switch(op) {  
  case '=': filter_str_same(data_key, val); break;  
  case '!=': filter_str_diff(data_key, val); break;  
  case '<': filter_num(data_key, val, is_lt); break;  
  case '<=': filter_num(data_key, val, is_le); break;  
  case '>': filter_num(data_key, val, is_gt); break;  
  case '>=': filter_num(data_key, val, is_ge); break;  
  case '@': filter_num_at(data_key, val); break;  
}
```

Speaker notes

. . . igual, não igual, menos, menos ou igual, maior, maior ou igual, e uma coisa especial que explicarei mais tarde. As comparações de ordem são usadas só para os valores numéricos, e igual ou não-igual são para os números e os strings. Por isso, vocês podem usar um filtro como "linguagem é português" (ou especificamente português-brasileiro) mas não "linguagem é maior que português". Isso simplesmente não faz sentido!

Cada comparação está implementado por . . .

```
switch(op) {  
  case '=': filter_str_same(data_key, val); break;  
  case '!=': filter_str_diff(data_key, val); break;  
  case '<': filter_num(data_key, val, is_lt); break;  
  case '<=': filter_num(data_key, val, is_le); break;  
  case '>': filter_num(data_key, val, is_gt); break;  
  case '>=': filter_num(data_key, val, is_ge); break;  
  case '@': filter_num_at(data_key, val); break;  
}
```

Speaker notes

. . . uma função diferente, ou a mesma função com argumentos diferentes, e vamos ver estas funções um pouco mais tarde.

Cada chave se torna uma chave num dicionário, cujo o valor é uma lista de duas coisas: a comparação, e o valor, do filtro. Por exemplo, . . .

```
["ling = pt", "tempo-min <= 30"]
```

se torna

```
{ "ling"      : ["=", "pt"],  
  "tempo-min": ["<=", 30] }
```

Speaker notes

. . . a lista dos filtros linguagem é português e tempo-min é menos ou igual a trinta, se torna o dicionário com as chaves linguagem e tempo-min, e os valores como isto. (Observam que o trinta é um numero, sem aspas!)

Então, . . .


```
specs = parse_specs(args, {});  
specs = parse_specs(fix_url_params(url_params),  
                    specs);
```

Speaker notes

. . . ela faz a mesma coisa com os arguments passados na URL. Mas, um sinal de igual pode quebrar um filtro na URL! Por isso, Genaver tem . . .

```
function fix_url_params(url_params) {
  const fixed_params = [];
  for (var [key, val] of url_params) {
    if (key.startsWith('gv-')) {
      const key2 = key.replace(/^gv-/, '');
      fixed_params.push(val ? `${key2}=${val}` : key2);
    }
  }
  return fixed_params;
}
```

Speaker notes

. . . uma função que remonta os filtros quebrados, assim. Ela leva os URL-parameters, como pars de chave e valor, e trata só os que começam com "gv-". Isso é como sabemos que é para Genaver, não algum outro programa. Se tem um valor, junta-lo com a chave e um sinal de igual. Se não, usa só a chave. Depois disto, temos uma lista de filtros, num formato exatamente igual à lista passado no JavaScript, e . . .

```
specs = parse_specs(args, {});  
specs = parse_specs(fix_url_params(url_params),  
                    specs);
```

Speaker notes

. . . a enviamos para a mesma função. Mas, em vez de enviar também um dicionário vazia, para encher, enviamos o dicionário dos filtros passado no JavaScript. Então, se passarmos na URL uma comparação ou um valor diferente para a mesma chave que no JavaScript, o que passamos na URL substitui o que passamos no JavaScript. Então, os filtros no JavaScript podem ser usados como padrões, para talvez serem substituídos pelos filtros na URL, ou não. Por exemplo . . .

```
generate_version([ "ling=es" ] );
```

Speaker notes

se temos a chamada para Genaver assim, especificando a linguagem espanhol, mas a URL é . . .

talk.html?gv-ling=pt

Speaker notes

. . . assim, especificando a linguagem português, vamos ver a apresentação em português, não espanhol.

Agora que Genaver tem todos os argumentos, ela . . .

```
for (const key of Object.keys(specs)) {  
  var op, val;  
  [op, val] = specs[key];  
  var data_key = `data-${key}`;  
  switch(op) {  
    . . .  
  }  
}
```

Speaker notes

. . . os aplica ao documento, filtro por filtro. Para cada um, . . .

```
var op, val;  
[op, val] = specs[key];  
var data_key = `data-${key}`;  
switch(op) {  
    . . .  
}
```

Speaker notes

. . . ela precede a chave com a palavra "data", como se faz para atributos personalizados, e faz . . .

```
case '=': filter_str_same(data_key, val); break;
case '!=': filter_str_diff(data_key, val); break;
case '<': filter_num(data_key, val, is_lt); break;
case '<=': filter_num(data_key, val, is_le); break;
case '>': filter_num(data_key, val, is_gt); break;
case '>=': filter_num(data_key, val, is_ge); break;
case '@': filter_num_at(data_key, val); break;
```

Speaker notes

. . . algo que depende da comparação.

Para a comparação de "não igual", é muito simples —


```
function filter_str_diff(key, value) {  
  const all =  
    document.querySelectorAll(`[${key}="${value}"]`);  
  for (elt of all) elt.style.remove();  
}
```

Speaker notes

. . . ela usa `querySelectorAll` para encontrar todos os elementos com esse valor para esse atributo, e os remove.

Para "igual", . . .

```
function filter_str_same(key, value) {
  const all = document.querySelectorAll(`[${key}]`);
  for (elt of all) {
    if (elt.getAttribute(key) !== value) {
      elt.style.remove();
    }
  }
}
```

Speaker notes

. . . é quase mesmo. Ela encontra todos os elementos com esse atributo, de qualquer valor, itera através deles, e remove os que não tem esse valor.

Para . . .

```
case '<': filter_num(data_key, val, is_lt); break;  
case '<=': filter_num(data_key, val, is_le); break;  
case '>': filter_num(data_key, val, is_gt); break;  
case '>=': filter_num(data_key, val, is_ge); break;
```

Speaker notes

. . . as comparações de ordem, é um pouco mais complexo que isso, porque precisa . . .

```
function filter_num(key, value, comp_fn) {
  const limit = parseInt(value);
  const all = document.querySelectorAll(`[${key}]`);
  for (elt of all) {
    if (!comp_fn(parseInt(elt.getAttribute(key)), limit)) {
      elt.style.remove();
    }
  }
}
```

Speaker notes

. . . converter os atributos de strings em números, e verificar seus relações ao argumento. Ela faz isso por encontrar todos os elementos com o atributo, e para cada um, usar uma . . .


```
function filter_num(key, value, comp_fn) {
  const limit = parseInt(value);
  const all = document.querySelectorAll(`[${key}]`);
  for (elt of all) {
    if (!comp_fn(parseInt(elt.getAttribute(key)), limit)) {
      elt.style.remove();
    }
  }
}
```

Speaker notes

. . . função de comparação para verificar se a relação desejado é verdadeira, e se não, remove o elemento. As . . .

```
function is_ge(x, limit) { return x >= limit }  
function is_gt(x, limit) { return x > limit }  
function is_le(x, limit) { return x <= limit }  
function is_lt(x, limit) { return x < limit }
```

Speaker notes

. . . funções de comparação são muito simples. Eu poderia usar literais de função, ou em inglês, function literals, em vez de funções reais, mas então não poderia fazer a próxima comparação tão simples. Isso é a coisa especial, de que eu falei anteriormente: a comparação de arropa!

```
function filter_num_at(key, value) {  
    filter_num(`${key}-min`, value, is_le);  
    filter_num(`${key}-max`, value, is_ge);  
}
```

Speaker notes

Isto é útil quando queremos mostrar os elementos, cujo algum atributo mínimo, com um nome que termina em "min", é menor que algo limite, que nós declaramos no filtro, e algum outro atributo máximo, com quase o mesmo nome mas terminando em "max" em vez de "min", é maior que o mesmo limite. Principalmente, eu uso isto para marcar alguns elementos com o mínimo e máximo quantidade de tempo para os incluir. Por exemplo, . . .

```
<p data-tempo-max="19">
```

```
  Explicação curta.
```

```
</p>
```

```
<p data-tempo-min="20" data-tempo-max="29">
```

```
  Explicação média.
```

```
</p>
```

```
<p data-tempo-min="30">
```

```
  Explicação longa.
```

```
</p>
```

Speaker notes

. . . se tenho algo que eu queira dizer, e poderia dizê-lo em um minuto, cinco, ou dez, talvez eu quereria dizê-lo de maneira curta quando tenho menos que vinte minutos para falar, de maneira média quando tenho vinte a vinte-nove minutos, e de maneira longa quando tenho trinta minutos ou mais. Se tenho por exemplo quinze minutos, e uso o filtro "tempo arroba quinze", . . .


```
function filter_num_at(key, value) {  
  filter_num(`${key}-min`, value, is_le);  
  filter_num(`${key}-max`, value, is_ge);  
}
```

Speaker notes

. . . isto faz a mesma coisa como se eu tivesse passado os filtros "tempo-min é menor ou igual a quinze" e "tempo-max é maior ou igual a quinze". Isto é como eu faço uma versão com aproximadamente a duração que quero.

E agora, uma pequena demonstração.

```
generate_version([]);
```

Speaker notes

Aqui está a chamada para Genaver, no fundo desta apresentação. Veja que não tem nenhum filtros. E . . .

```
<p data-ling="en">
```

```
  This is in English.
```

```
</p>
```

```
<p data-ling="fr">
```

```
  Ceci est en français.
```

```
</p>
```

```
<p data-ling="pt">
```

```
  Isto é em português.
```

```
</p>
```

Speaker notes

. . . aqui está o código do próximo slide. Se irmos alí, o que vocês pensam que aparecerá? Depende também na URL, então vamos lá, e vemos que não temos nenhum filtros lá também. Quem pensa que não veremos nada? Levante uma mão ou faça um barulho. E quem pensa que veremos tudo? De novo, mão ou barulho. Algumas outras respostas? . . . Vamos-lá e ver!

This is in English.

Ceci est en français.

Isto é em português.

Speaker notes

Vemos tudo, porque Genaver não funciona por copiar os pedaços bons, mas remover os ruins. Não dissemos o que queremos, então, Genaver não sabe o que remover, e não remove nada.

Mas agora, vamos mudar a URL, por adicionar o argumento `gv-ling=en`. Se agora nós recarregarmos a página, agora vemos só a versão em inglês. E podemos fazer a mesma coisa com francês e português. Podemos fazer a mesma coisa com linguagens de programação, ou qualquer outra coisa, por simplesmente usar uma outra chave. Por exemplo .

..


```
<p data-ling-prog="c">  
    printf( "Genaver!\n" );  
</p>  
<p data-ling-prog="elixir">  
    IO.puts( "Genaver!" )  
</p>  
<p data-ling-prog="ruby">  
    puts "Genaver!"  
</p>
```

Speaker notes

. . . aqui está o código do próximo slide. Vamos-lá e ver o que aparecerá!

```
printf( "Genaver! \n" );
```

```
IO.puts( "Genaver!" )
```

```
puts "Genaver!"
```

Speaker notes

De novo, vemos tudo, porque de novo, Genaver não sabe o que remover. Mas vamos mudar a URL, por adicionar o argumento `gv-ling-prog=c`, e recarregamos a página. Agora vemos só a versão em C. E podemos fazer a mesma coisa com Elixir e Ruby.

Mas, que tal a duração?

```
<p data-tempo-max="19">
  Explicação curta.
</p>
<p data-tempo-min="20"
  data-tempo-max="29">
  Explicação média.
</p>
<p data-tempo-min="30">
  Explicação longa.
</p>
```

Speaker notes

De novo, aqui está o código do próximo slide. Observa que as explicações são marcadas com os atributos "tempo-min", "tempo-max", ou talvez ambos. Queremos a explicação curta se temos menos que vinte minutos para falar, a média se temos de vinte a vinte-nove, e a longa se temos trinta ou mais.

Vamos-lá e ver o que aparece!

Explicação curta.
Explicação média.
Explicação longa.

Speaker notes

Como antes, vemos tudo, porque não dissemos a Genaver o que remover. Mas vamos mudar a URL de novo.

Podemos dizer, se temos por exemplo quinze minutos para falar, tempo-min menos ou igual a quinze, e tempo-max maior ou igual a quinze, e isso dá certo. Mas, é um pouco chato, precisar dar ambos os tempos mínimo e máximo, quando são geralmente o mesmo número, né? Podemos dizer, em vez de desses dois argumentos, tempo (sem min ou max) arroba quinze, e Genaver interpreta isso como o min e o max, assim. E mesmo a dezenove. Mas a vinte, vemos agora a explicação média, assim de novo com vinte-nove, e a trinta (ou mais), vemos a explicação longa.

Podemos usar as comparações de ordem para outros propósitos, mas eu pessoalmente não tenho nenhum.

Agora, eu espero que vocês entedam como, . . .


```
generate_version([ "ling-humana = pt",  
                  "ling-prog   = elixir",  
                  "tempo      @ 30" ])
```

OU

```
talk.html?gv-ling-humana=pt&gv-ling-prog=elixir&gv-tempo@30
```

Speaker notes

. . . com a minha gambiarra chamada de Genaver, eu posso alcançar meu objetivo, de manter uma "versão primária" de uma apresentação, e produzir, rapidamente e facilmente, uma versão com uma linguagem humana, uma linguagem de programação, e uma duração (aproximada), de minha escolha.

E agora, . . .

?????

T.Rex-2023@Codosaur.us
twitter.com/DaveAronson
linkedin.com/in/DaveAronson

github.com/CodosaurusLLC/genaver
www.codosaur.us/reds/genaver-gambi-23-slides

Speaker notes

. . . se há perguntas, aceitarei-las agora, e se vocês pensarem em algo mais tarde, aqui estão minhas informações de contato, e as URLs de Genaver e dos slides desta apresentação, completo com um roteiro. Mas, eu peço um favor pequeno. Eu posso falar o português rápido, mas não posso entendê-lo rápido, então se há perguntas, eu peço que vocês perguntem um pouco devagar, e muito claramente . . . ou em inglês. Há perguntas?

