Use a cabeça! Ruby

Jay McGavren

Tradução
Edson Furmankiewicz
(Docware Traduções Técnicas)

ALTA BOOKS
EDITORA
Rio de Janeiro, 2016

Autor de Use a Cabeça! Ruby

Jay McGavren estava trabalhando em automação para uma empresa de serviços de hotelaria quando um colega lhe apresentou o livro *Programming Perl* (também conhecido como o Camel Book). Jay se converteu instantaneamente ao Perl, uma vez que gostava realmente de escrever código em vez de esperar por uma equipe de desenvolvimento de 10 pessoas para configurar um sistema de compilação. Ele também lhe deu a ideia maluca de escrever um livro técnico algum dia.

Em 2007, com a explosão do Perl, Jay estava à procura de uma nova linguagem interpretada. Com sua forte orientação a objetos, um excelente suporte de bibliotecas e uma flexibilidade incrível, o Ruby o conquistou. Desde então, ele usou Ruby para duas bibliotecas de game e um projeto de arte generativa e começou a trabalhar como programador freelancer do Ruby on Rails. Ele trabalha na área de educação on-line para desenvolvedores desde 2011.

Você pode acompanhar Jay no Twitter em *https://twitter.com/jaymcgavren*, ou visitar seu site pessoal em *http://jay.mcgavren.com* — em inglês.

Conteúdo (sumário)

Conteúdo (a coisa real)

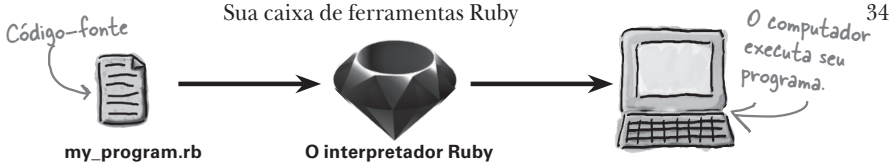
Introdução

Seu cérebro no Ruby. Aqui você está tentando aprender alguma coisa, enquanto aqui seu *cérebro* está fazendo um favor garantindo que a aprendizagem não se *fixe*. Seu cérebro está pensando, "Melhor deixar espaço para coisas mais importantes, como os animais selvagens a evitar e se fazer snowboarding pelado é uma má ideia." Então, como *você* engana seu cérebro fazendo-o pensar que sua vida depende de saber como programar no Ruby?

mais com menos

1 Codifique do seu jeito

Você está querendo saber o que é essa tal de linguagem Ruby, e se ela é certa para você. Deixe-nos perguntar: ***Você gosta de ser produtivo?*** Você acha que todos esses compiladores e bibliotecas extras e todos esses arquivos de classe e atalhos de teclado em sua outra linguagem o aproximam de um **produto acabado, colegas de trabalho impressionados e clientes satisfeitos**? Você gostaria de uma linguagem que **cuida dos detalhes para você**? Se você às vezes pensa em parar de manter código clichê e começar *a trabalhar em seu problema*, então o Ruby é para você. O Ruby permite **fazer mais com menos código.**



métodos e classes

2 Organizando-se

Você está perdendo algo importante. Você tem chamado métodos e criado objetos como um profissional. Mas os únicos métodos que você chamou, e os únicos tipos de objeto que você criou, foram os que o Ruby definiu para você. Agora é sua vez. Você aprenderá a criar seus *próprios* métodos. Você também criará suas próprias **classes** — modelos para novos objetos. Você *decidirá* como serão os objetos baseados em sua classe. Você usará **variáveis de instância** para definir o que esses objetos *sabem* e **métodos de instância** para definir o que eles *fazem*. E, sobretudo, você descobrirá como definir suas próprias classes *pode facilitar a leitura* e a manutenção de seu código.

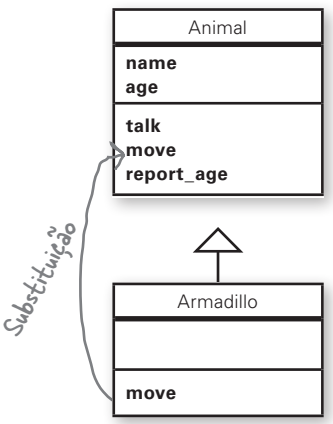
3

herança

Confiando em seus pais

Repetição demais! Suas novas classes que representam os diferentes tipos de veículos e animais são incríveis, é verdade. Mas você está tendo de *copiar os métodos de instância de uma classe para outra*. E as cópias estão começando a sair de sincronia — algumas são boas, enquanto outras têm bugs. As classes não foram feitas para facilitar a manutenção do código?

Neste capítulo, aprenderemos como usar a **herança** para permitir que suas classes *compartilhem* métodos. Menos cópias significa menos dores de cabeça de manutenção!



4 inicializando instâncias
Um bom ponto de partida

Agora, sua classe é uma bomba-relógio. Cada instância que você cria começa como uma lousa em branco. Se você chamar determinados métodos de instância antes de adicionar dados, um erro que derrubará seu programa será gerado.

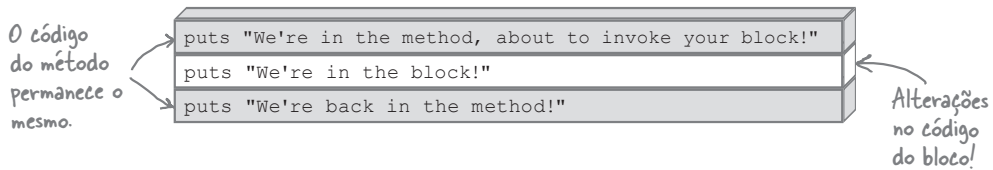
Neste capítulo, mostraremos duas maneiras de criar objetos que são seguras para uso imediato. Começaremos com o método `initialize`, que permite passar vários argumentos para configurar os dados de um objeto *no momento em que você o cria*. Então, mostraremos como escrever **métodos de classe**, os quais você pode usar para criar e configurar um objeto ainda **mais** facilmente.

5

arrays e blocos

Melhor que loops

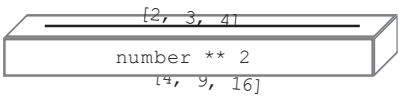
Grande parte do trabalho de programação lida com listas de coisas. Listas de endereços. Listas de números de telefone. Listas de produtos. Matz, o criador do Ruby, sabia disso. Assim, *ele trabalhou arduamente* para se certificar de que o trabalho com listas no Ruby fosse *realmente fácil*. Primeiro, ele assegurou que **arrays**, que mantêm o controle de listas no Ruby, tivessem um grande número de *métodos poderosos* para fazer quase qualquer coisa que você pode precisar com uma lista. Em segundo lugar, ele percebeu que escrever código para *iterar por uma lista* a fim de fazer algo com cada item, embora tedioso, é algo que os desenvolvedores faziam *muito*. Então, ele acrescentou **blocos** à linguagem , e eliminou a necessidade de todo esse código de iteração. O que é um bloco, exatamente? Leia mais para descobrir…

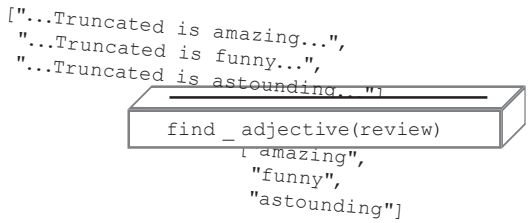


6 Valores de retorno de bloco
Como devo lidar com isso?

Você viu apenas uma fração do poder dos blocos. Até agora, os *métodos* têm se limitado a passar dados de um *bloco* e esperar o bloco lidar com tudo. Mas um *bloco* também pode retornar dados ao *método*. Esse recurso permite que o método obtenha *instruções* do bloco, deixando que ele faça mais de todo o trabalho.

Neste capítulo, mostraremos alguns métodos que lhe permitirão pegar uma coleção *grande*, *complicada* e usar **valores de retorno de bloco** para reduzi-la





7 hashes
Organizando os dados

Colocar coisas em pilhas é bom, até que você precise encontrar algo de novo. Você já viu como criar uma coleção de objetos usando um *array*. Você já viu como processar *cada item* em um array e como *localizar itens* que você quer. Em ambos os casos, você começa no início do array e *examina cada objeto individualmente*. Você também viu métodos que lidam com grandes coleções de parâmetros. Você viu os problemas que isso causa: chamadas de método exigem uma grande e *confusa coleção de argumentos* cuja ordem exata você tem de lembrar.

E se houvesse uma espécie de coleção na qual *todos os dados tivessem rótulos*? Você poderia *encontrar rapidamente os elementos* que precisasse! Neste capítulo, aprenderemos sobre **hashes** do Ruby, que fazem exatamente isso.

Array

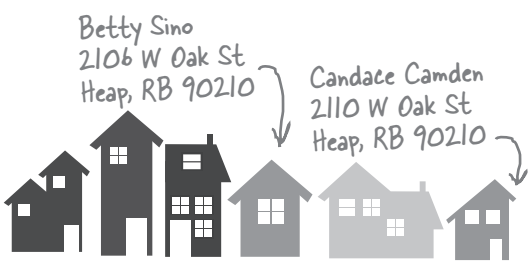
Hash

8

referências

Linhas cruzadas

Já enviou um e-mail para o contato errado? Você provavelmente teve dificuldades para esclarecer a confusão que se seguiu. Bem, *objetos Ruby* são *exatamente como os contatos* do catálogo de endereços, e *chamar métodos* sobre eles é como *enviar mensagens* a eles. Se seu catálogo de endereços *estiver desordenado*, é possível enviar mensagens ao *objeto errado*. Este capítulo o ajudará a *reconhecer os sinais* de que isso está acontecendo e a *fazer seus programas funcionarem novamente*.



Oak Street, na verdade,

Oak Street no livro de endereços do Andy

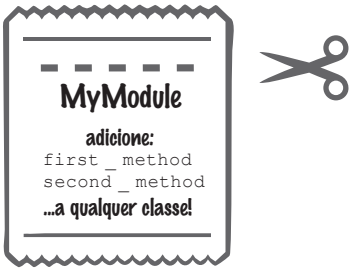
9

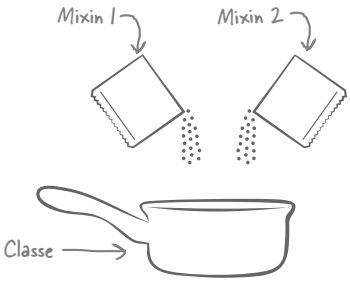
mixins

Misturando tudo

A herança tem suas limitações. Você só pode herdar métodos de uma classe. Mas e se você precisar compartilhar *vários conjuntos de comportamento* entre várias classes? Como métodos para iniciar um ciclo de carga da bateria e relatar seu nível de carga — você pode precisar desses métodos em telefones, furadeiras elétricas e carros elétricos. Você *criará* uma única superclasse para todos *esses métodos*? (Isso não acabará bem se você tentar.) Ou métodos para iniciar e parar um motor. Claro, a furadeira e o carro podem precisar disso, mas o telefone não!

Neste capítulo, aprenderemos sobre **módulos** e **mixins**, uma forma poderosa de *agrupar métodos* e, então, compartilhá-los *apenas com as classes específicas que precisam deles***.**



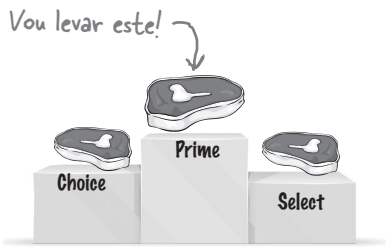


10

comparable e enumerable

Misturas prontas para usar

Você viu que mixins podem ser úteis. Mas você ainda não viu seu poder. A biblioteca de núcleo do Ruby inclui dois mixins que o *surpreenderão*. O primeiro, `Comparable`, é utilizado para comparar objetos. Você usou como operadores <, >, e == em números e strings, mas o `Comparable` permite que você os use em *suas* classes. O segundo mixin, `Enumerable`, é usado para trabalhar com coleções. Lembra-se daqueles métodos superúteis `find_all`, `reject` e `map` que você usou em arrays antes? Eles vieram do `Enumerable`. Mas isso é uma pequena fração do que o `Enumerable` pode fazer. E, mais uma vez, você pode misturá-lo em *suas* classes. Leia mais para ver como!



documentação

11 Leia o manual

Não há espaço suficiente neste livro para ensinar-lhe tudo sobre o Ruby. Há um velho ditado que diz: "Dê um peixe a uma pessoa, e você a alimentará por um dia. Ensine-a a pescar, e você a alimentará por toda a vida." Até aqui nós lhe *demos peixe*. Mostramos como utilizar algumas das classes e módulos do Ruby. Mas existem dezenas de outras coisas, algumas delas aplicáveis a seus problemas, que não temos espaço para abordar. Então é hora de *ensiná-lo a pescar*. Há excelente **documentação** disponível gratuitamente sobre todas as classes, módulos e métodos do Ruby. Você apenas precisa saber onde encontrá-la, e como interpretá-la. Isso é o que este capítulo lhe mostrará.

Documentação para o método de classe "today"

```
.today([start = Date::ITALY]) ⇒ Object

Date.today #=> #<Date: 2011-06-11 ..>

Creates a date object denoting the present day.
```

```
#year ⇒ Integer

Returns the year.
```

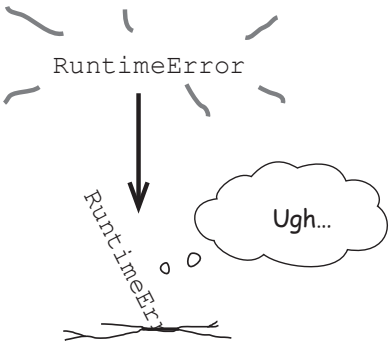
Documentação para o método de instância "year"

exceções

12 Lidando com o inesperado

No mundo real, o inesperado acontece. Alguém pode excluir o arquivo que seu programa está tentando carregar, ou o servidor com o qual seu programa está tentando entrar em contato pode sair do ar. Seu código pode verificar essas situações excepcionais, mas esses controles seriam misturados com o código que manipula a operação normal. (E isso faria uma grande bagunça, tornado o código ilegível.)

Este capítulo ensina tudo sobre o tratamento de exceção do Ruby, que permite escrever código para lidar com o inesperado, e mantê-lo separado de seu código normal.

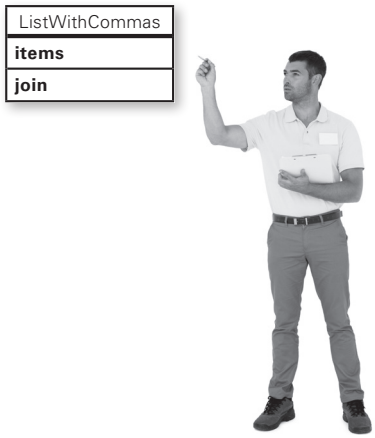


testes de unidade

13 Garantia da qualidade de código

Tem certeza de que seu software está funcionando agora? Certeza absoluta? Antes de enviar aquela nova versão para os usuários, presumivelmente você testou os novos recursos para garantir que todos funcionavam. Mas você testou os *antigos* recursos para garantir que não quebrou nenhum deles? *Todos* os antigos recursos? Se essa pergunta o deixou preocupado, seu programa precisa de testes automatizados. Testes automatizados garantem que componentes de seu programa funcionem corretamente, mesmo depois de alterar seu código.

Os **testes de unidade** são o tipo mais importante e mais comum de teste automatizado. E o Ruby inclui o **MiniTest,** uma biblioteca dedicada a testes de unidade. Este capítulo ensinará tudo que você precisa saber sobre isso!



Se `items` for configurado como `['apple', 'orange', 'pear']`, então `join` deve retornar "`apple, orange, and pear`".

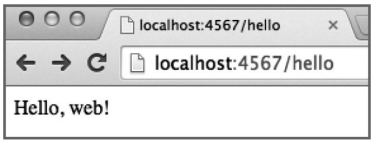
Se `items` for configurado como `['apple', 'orange'], 'laranja']`, então `join` deve retornar "`apple and orange`".

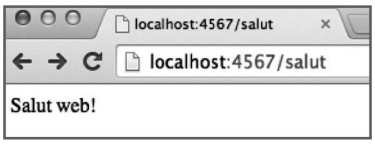
aplicativos web

14 Servindo HTML

Este é o século 21. Os usuários querem aplicativos web. O Ruby também te ajuda nisso! Há bibliotecas disponíveis para ajudá-lo a hospedar seus próprios aplicativos web e torná-los acessíveis a partir de qualquer navegador web. Então, dedicaremos esses dois capítulos finais do livro a mostrar como construir um aplicativo web completo.

Para começar, você precisará do **Sinatra**, uma biblioteca de terceiros para escrever aplicações web. Mas não se preocupe, mostraremos como usar a ferramenta **RubyGems** (incluída no Ruby) para baixar e instalar bibliotecas automaticamente! Então, mostraremos somente o que for necessário sobre código HTML para você criar suas próprias páginas web. E, naturalmente, mostraremos como disponibilizar essas páginas para um navegador!



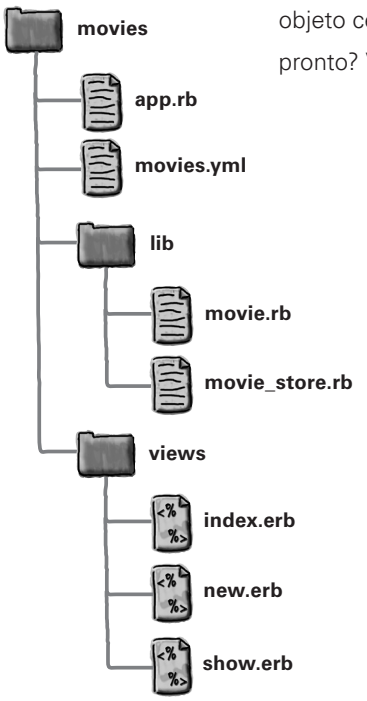


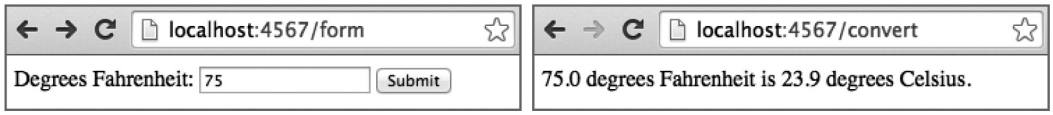
15 salvando e carregando dados

Guarde isso!

Seu aplicativo web ainda não está armazenando os dados dos usuários. Você configurou um formulário para que os usuários *insiram* dados. Eles esperam que você *salve os dados*, de modo que estes possam ser *recuperados* e *exibidos* para outros mais tarde. Mas isso não está acontecendo agora! Qualquer coisa que eles enviam simplesmente *desaparece*.

Neste nosso último capítulo, prepararemos seu aplicativo para salvar as submissões do usuário. Mostraremos como configurá-lo para aceitar dados de formulário. Mostraremos como converter os dados em objetos Ruby, como salvar esses objetos em um arquivo e como recuperar o objeto certo novamente quando um usuário quiser vê-los. Você está pronto? Vamos terminar esse aplicativo!





Apêndice: sobras

Os dez tópicos principais (que não abordamos)

Cobrimos um extenso terreno e você está quase terminando este livro. Sentiremos sua falta, mas antes de deixá-lo ir, não me sentiria bem deixando você só no mundo sem um *pouco* mais de preparação. Não há como caber tudo o que você precisa saber sobre Ruby nestas poucas páginas… (De fato, originalmente nós *realmente* incluímos tudo, reduzindo o tamanho da fonte para 0,00004. Tudo se encaixou, mas ninguém conseguia ler. Então, jogamos fora a maior parte.) Mas mantivemos todas as melhores partes para este apêndice.

Esse *é* realmente o fim do livro. Com exceção do índice, é claro. (Uma leitura obrigatória!)

Um arquivo CSV

```
Associate,Sale Count,Sales Total
"Boone, Agnes",127,1710.26
"Howell, Marvin",196,2245.19
"Rodgers, Tonya",400,3032.48
```

sales.csv

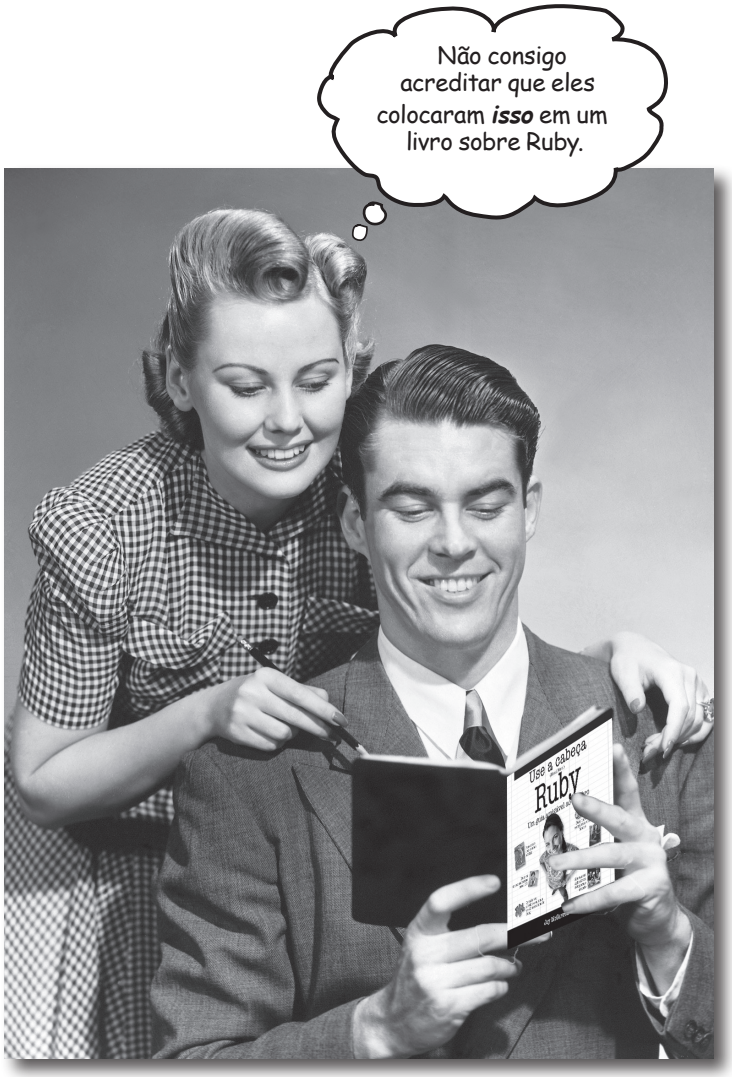
```
p ARGV[0]
p ARGV[1]
```

args_test.rb

```
File Edit Window Help
$ ruby args_test.rb hello terminal
"hello"
"terminal"
```

Como usar este livro

Introdução



Nesta seção, respondemos à pergunta que não quer calar:
"Então, por que eles colocaram isso em um livro sobre Ruby?"

A quem se destina este livro?

Se você responder "sim" a ***todas*** estas perguntas:

1. Você tem acesso a um computador com um editor de texto?

2. Você quer aprender uma linguagem de programação que torna o desenvolvimento **fácil** e **produtivo**?

3. Você prefere **conversas estimulantes em eventos sociais** a **palestras acadêmicas áridas e maçantes**?

este livro é para você.

Quem provavelmente deve fugir deste livro?

Se você responder "sim" a qualquer ***uma*** delas:

1. **Você é <u>completamente</u> iniciante em computadores?**

   (Você não precisa ser experiente, mas deve entender pastas e arquivos, saber como abrir um aplicativo de terminal e saber como usar um editor de texto simples.)

2. Você é um desenvolvedor ninja rockstar à procura de um **livro de *referência***?

3. Você tem **medo de tentar algo novo**? Você prefere fazer um tratamento de canal dentário a misturar listras com xadrez? Você acredita que um livro técnico não pode ser sério se descreve herança de classe usando tatus?

este livro *não* é para você.

[Nota do Marketing: este livro é para qualquer um com um cartão de crédito válido.]

Sabemos o que você está pensando

"Como *isso* pode ser um livro sério sobre o desenvolvimento no Ruby?"

"O que são todas essas imagens esquisitas?"

"Posso realmente *aprender* dessa maneira?"

Sabemos o que seu cérebro está pensando

Seu cérebro anseia por novidade. Ele está sempre em busca, escaneando, *à espera* de algo incomum. Ele foi construído assim, e isso ajuda você a se manter vivo.

Então, o que seu cérebro faz com todas as coisas normais, comuns e rotineiras que você encontra? Tudo o que ele *pode* para impedi-las de interferir no trabalho *real* do cérebro — guardar coisas que *importam*. Ele não se incomoda com salvar as coisas chatas; estas nunca passam pelo filtro "isso obviamente não é importante".

Como é que seu cérebro *sabe* o que é importante? Suponha que você saiu um dia para caminhar e um tigre salta na frente de você — o que acontece dentro de sua cabeça e de seu corpo?

Os neurônios disparam. As emoções ficam à flor da pele. *A adrenalina sobe.*

E é assim que seu cérebro sabe…

Isso deve ser importante! Não se esqueça!

Mas imagine que você está em casa ou em uma biblioteca. É uma zona segura, acolhedora e livre de tigres. Você está estudando. Preparando-se para um exame. Ou tentando aprender algum assunto técnico difícil que seu chefe acha que você levará uma semana ou 10 dias no máximo para aprender.

Só há um problema. Seu cérebro está tentando fazer-lhe um grande favor. Ele está tentando se certificar de que esse conteúdo *obviamente* sem importância não consuma recursos escassos. Recursos que são mais bem gastos armazenando as coisas realmente *importantes*. Como tigres. Como o perigo de incêndio. Como você nunca deveria ter publicado as fotos daquela festa em sua página no Facebook. E não há nenhuma maneira simples de dizer a seu cérebro: "Ei, cérebro, muito obrigado, mas não importa o quão maçante este livro é, e quão baixo estou na escala emocional Richter agora, eu *realmente* quero que você guarde tudo isso."

Pensamos no leitor de "Use a Cabeça!" como um aprendiz.

Então, o que é preciso para aprender alguma coisa? Primeiro, você precisa entendê-la, então, *garantir* que não seja *esquecida*. Não se trata de empurrar fatos em sua cabeça. Com base nas mais recentes pesquisas em ciência cognitiva, neurobiologia e psicologia educacional, a *aprendizagem* exige muito mais do que texto em uma página. Sabemos o que faz seu cérebro ligar.

Alguns dos princípios da aprendizagem da série Use a Cabeça!:

Use recursos visuais. As imagens são muito mais memorizáveis do que palavras apenas, e tornam a aprendizagem muito mais eficaz (uma melhora de até 89% em estudos de recuperação e transferência). As imagens também tornam as coisas mais compreensíveis. **Coloque as palavras dentro ou perto das imagens** com que se relacionam, em vez de na parte inferior ou em outra página, e os alunos terão uma probabilidade até *duas vezes maior* de resolver os problemas relacionados com o conteúdo.

Use um estilo coloquial e pessoal. Em estudos recentes, alunos tiveram um desempenho até 40% melhor em testes posteriores se o conteúdo falasse diretamente ao leitor, em primeira pessoa e na forma coloquial, em vez de em tom formal. Conte histórias em vez de dar palestras. Use uma linguagem casual. Não se leve muito a sério. A que você prestaria mais atenção: uma companhia interessante em um jantar festivo ou uma palestra?

Faça com que o aprendiz pense com mais profundidade. Em outras palavras, a menos que você exercite ativamente seus neurônios, não acontece muita coisa na sua cabeça. Um leitor tem de ser motivado, incentivado, curioso e inspirado para resolver problemas, chegar a conclusões e gerar novo conhecimento. Para isso, você precisa de desafios, exercícios e questões que provoquem o raciocínio, além de atividades que envolvam ambos os lados do cérebro e múltiplos sentidos.

Ganhe — e mantenha — a atenção do leitor. Todos já tivemos a experiência "realmente quero aprender isso, mas não consigo permanecer acordado após a primeira página". Seu cérebro presta atenção a coisas que são fora do comum, interessantes, diferentes, chamativas, inesperadas. Aprender um assunto técnico novo e difícil não precisa ser tedioso. Seu cérebro aprenderá muito mais rapidamente se não for.

Toque nas emoções deles. Agora sabemos que a sua capacidade de lembrar algo é muito dependente do conteúdo emocional. Você lembra daquilo a que dá importância. Você lembra quando *sente* algo. Não, não estamos falando de histórias tristes sobre um garoto e seu cachorro. Estamos falando de emoções como surpresa, curiosidade, diversão, "mas o que...?" e o sentimento de "Sou o máximo!" que vem quando você resolve um problema, aprende o que todas as outras pessoas acham que é difícil ou percebe que sabe algo que o Bob "sou mais técnico do que você" da engenharia *não* sabe.

Metacognição: pensando sobre pensar

Se você realmente quiser aprender, e quiser aprender mais rápido e profundamente, preste atenção em como você presta atenção. Pense em como você pensa. Aprenda como você aprende.

A maioria de nós não fez cursos sobre metacognição ou teoria de aprendizagem quando estávamos crescendo. As pessoas *esperavam* que aprendêssemos, mas raramente nos *ensinavam* a aprender.

Porém, supomos que, se você está segurando este livro, realmente quer aprender como desenvolver aplicativos em Ruby. E provavelmente não quer gastar muito tempo. Se quer usar o que você lê neste livro, você precisa se *lembrar* do que leu. E, para isso, você tem de *entender*. Para tirar o maior proveito deste livro, ou de qualquer livro ou experiência de aprendizagem, assuma a responsabilidade por seu cérebro. Seu cérebro neste conteúdo.

O truque é fazer com que seu cérebro veja o material novo que você está aprendendo como Realmente Importante. Crucial para seu bem-estar. Tão importante quanto um tigre. Caso contrário, você ficará em uma batalha constante, com seu cérebro fazendo seu melhor para impedir que o novo conteúdo permaneça.

Então, como você faz seu cérebro tratar programação como se fosse um tigre faminto?

Há a forma lenta e tediosa, e a mais rápida e eficaz. A forma lenta tem a ver com repetições. Você obviamente sabe que *pode* aprender e lembrar, mesmo os tópicos mais desinteressantes, se ficar repetindo a mesma coisa no seu cérebro. Com repetição suficiente, seu cérebro diz: "Isso não *parece* importante para ele, mas ele continua a olhar para a mesma coisa *uma vez* e *outra* e *outra*, então eu suponho que deve ser."

A forma mais rápida é fazer ***qualquer coisa que aumente a atividade do cérebro***, especialmente *tipos* diferentes de atividade cerebral. As coisas na página anterior são uma parte importante da solução, e são todas coisas que foram provadas como eficazes para ajudar seu cérebro a trabalhar a seu favor. Por exemplo, estudos mostram que colocar palavras *dentro* das imagens que elas descrevem (em vez de em algum outro lugar na página, como um cabeçalho ou no texto do corpo) faz com que seu cérebro tente encontrar sentido no modo como palavras e imagens se relacionam, e isso faz com que mais neurônios disparem. Mais neurônios disparando = mais chances do seu cérebro *entender* que isso é algo que merece atenção, e possivelmente gravar.

Um estilo coloquial ajuda porque as pessoas tendem a prestar mais atenção quando percebem que estão em uma conversa, já que devem acompanhar e esperar seu final. O incrível é que seu cérebro não se *importa* necessariamente que a "conversa" seja entre você e um livro! Por outro lado, se o estilo do texto for formal e árido, seu cérebro a percebe da mesma maneira como quando você assiste a uma palestra, sentado em uma sala cheia de ouvintes passivos. Não é preciso permanecer acordado.

Porém, imagens e estilo coloquial são apenas o início…

Eis o que NÓS fizemos

Usamos ***imagens***, porque seu cérebro é atraído por imagens, não por texto. No que diz respeito a seu cérebro, uma figura realmente *vale por* mil palavras. Quando texto e imagens trabalham juntos, inserimos o texto *na* figura porque seu cérebro funciona mais eficazmente quando o texto está *dentro* do que ele referencia, em contraposição a uma legenda ou enterrado em algum lugar no corpo do texto.

Usamos ***redundância***, dizendo o mesmo de maneiras *diferentes* e com diferentes tipos de mídia e múltiplos sentidos, para aumentar a chance de que o conteúdo se fixe em mais de uma área do cérebro.

Usamos conceitos e imagens de maneiras ***inesperadas*** porque seu cérebro está sintonizado para a novidade, e usamos imagens e ideias com pelo menos *algum conteúdo* ***emocional***, porque seu cérebro está sintonizado para prestar atenção na bioquímica das emoções. Isso faz com que você sinta que algo tem mais chance de ser lembrado, mesmo se esse sentimento não for nada mais do que um pouco de ***humor***, ***surpresa*** ou ***interesse***.

Usamos um ***estilo personalizado***, ***coloquial***, porque seu cérebro é ajustado para prestar mais atenção quando acredita que você está em uma conversa do que se achar que você está ouvindo passivamente uma apresentação. Seu cérebro faz isso mesmo quando você está *lendo*.

Incluímos muitas ***atividades***, porque seu cérebro é ajustado para aprender e lembrar mais quando você ***faz*** coisas do que quando *lê* sobre elas. Tornamos os exercícios desafiadores, mas factíveis, porque é o que a maioria das pessoas prefere.

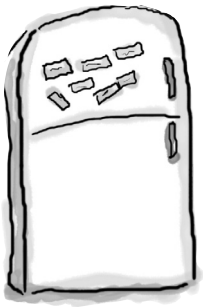
Usamos ***múltiplos estilos de aprendizagem***, porque *você* talvez prefira procedimentos passo a passo ou entender o contexto geral primeiro ou, ainda, só queira ver um exemplo. Mas, independentemente de sua preferência de aprendizagem, *todos* se beneficiam vendo o mesmo conteúdo representado de múltiplas formas.

Incluímos conteúdo para ***ambos os lados do seu cérebro***, pois quanto mais de seu cérebro você envolve, você provavelmente aprenderá e se lembrará de mais, e conseguirá permanecer focado por mais tempo. Como trabalhar um lado do cérebro muitas vezes significa dar ao outro lado uma chance de descansar, você pode ser mais produtivo aprendendo por um período mais longo.

Incluímos ***histórias*** e exercícios que apresentam ***mais de um ponto de vista***, porque seu cérebro é adaptado para aprender mais profundamente quando é forçado a fazer avaliações e julgamentos.

Incluímos ***desafios***, com exercícios, fazendo ***perguntas*** que nem sempre têm uma resposta direta, porque seu cérebro é adaptado para aprender e lembrar quando precisa *trabalhar* com afinco. Pense nisso — você não consegue colocar seu *corpo* em forma apenas *observando* pessoas na academia de ginástica. Fizemos o nosso melhor para assegurar que você trabalhe com afinco nas coisas certas. E que ***você não gastará um dendrito extra*** processando um exemplo difícil de entender, ou analisando um texto conciso demais, cheio de jargões ou difícil.

Usamos ***pessoas***. Em histórias, exemplos, imagens etc., porque, bem, *você é* uma pessoa, e seu cérebro presta mais atenção a *pessoas* do que a *coisas*.



Recorte isso e cole em sua geladeira.

Veja o que fazer para que o seu cérebro se curve em sinal de submissão

Então, fizemos nossa parte. O resto é com você. Estas dicas são um ponto de partida; ouça seu cérebro e descubra o que funciona e o que não funciona para você. Experimente coisas novas.

1 Desacelere. Quanto mais você entende, menos você tem que memorizar.

Não simplesmente leia. Pare e pense. Quando o livro lhe fizer uma pergunta, não apenas pule para a resposta. Imagine que alguém realmente *esteja* lhe fazendo a pergunta. Quanto mais profundamente você forçar seu cérebro a pensar, maior a chance de aprender e lembrar.

2 Faça os exercícios. Faça suas próprias anotações.

Os elaboramos para você, mas, se os fizéssemos por você, seria como se outra pessoa fizesse seus exercícios físicos. Não simplesmente *olhe* os exercícios. Use um lápis. Há bastante evidência de que atividade física *durante* a aprendizagem pode aumentar sua eficácia.

3 Leia as seções "Não existem perguntas idiotas"

Isso significa todas as perguntas. Esses balões de pensamento com uma pergunta que aparecem por todo o livro não são opcionais, são ***parte do conteúdo principal!*** Não os ignore.

4 Faça disso a última coisa que você lê antes de dormir. Ou, pelo menos, a última coisa desafiante.

Parte da aprendizagem (especialmente a transferência para a memória de longo prazo) acontece depois que você fecha o livro. Seu cérebro precisa de tempo sozinho para fazer mais processamento. Se você colocar algo novo na mente durante esse tempo de processamento, um pouco do que você acabou de aprender será perdido.

5 Converse sobre o que está lendo. Em voz alta.

Falar ativa uma parte diferente do cérebro. Se você está tentando entender alguma coisa, ou aumentar sua chance de lembrá-la mais tarde, fale em voz alta. Melhor ainda, tente explicá-la em voz alta para outra pessoa. Você aprenderá mais rapidamente, e pode descobrir ideias que não sabia que estavam lá quando estava lendo sobre o assunto.

6 Beba água. Em grande quantidade.

Seu cérebro trabalha melhor com um belo banho de fluídos. A desidratação (que pode acontecer antes de você sentir sede) diminui a função cognitiva.

7 Ouça seu cérebro.

Preste atenção se seu cérebro está ficando sobrecarregado. Se você está começando a arranhar a mesa com as unhas ou esquecendo o que você acabou de ler, é hora de fazer uma pausa. Depois de passar certo ponto, você não aprenderá mais rápido tentando forçar mais, e você pode até prejudicar o processo.

8 Ouça seu cérebro.

Seu cérebro precisa saber que isso é *importante*. Envolva-se com as histórias. Crie suas próprias legendas para as fotos. Suspirar por causa de uma piada ruim *ainda* é melhor do que não sentir nada.

9 Escreva um monte de código!

Só há uma maneira de aprender a desenvolver aplicações Ruby: **escrevendo um monte de código**. E isso é o que você fará ao longo deste livro. Escrever código é uma habilidade, e a única maneira de ficar bom nisso é praticando. Vamos lhe proporcionar muita prática. cada capítulo tem exercícios que representam um problema para você resolver. Não simplesmente pule esses exercícios — uma grande parte do aprendizado acontece quando você resolve os exercícios. Incluímos uma solução para cada exercício — não tenha medo de **espiar a solução** se você empacar! (É fácil empacar em algo pequeno.) Mas tente resolver o problema antes de ver a solução. E, definitivamente, resolva-o antes de passar para a próxima parte do livro.

Leia-me

Essa é uma experiência de aprendizagem, não um livro de referência. Deliberadamente retiramos tudo o que pode atrapalhar na aprendizagem do que for que estejamos fazendo naquela parte do livro. E, na primeira vez que você ler o livro, você precisa começar do começo, porque o livro faz suposições sobre o que você já viu e aprendeu.

Será útil se você tiver feito um pouco de programação em alguma outra linguagem.

A maioria dos desenvolvedores descobre o Ruby *depois* de aprender alguma outra linguagem de programação. (Eles costumam vir buscando refúgio dessa outra linguagem.) Abordamos os conceitos básicos suficientes para que um iniciante possa começar rápido, mas não entramos em grandes detalhes sobre o que uma variável é, ou como uma instrução if funciona. Sua vida será mais fácil se você tiver feito pelo menos um *pouco* disso antes.

Não abordamos todas as classes, bibliotecas e métodos já criados.

O Ruby vem com um *monte* de classes e métodos incorporados. Claro, todos estes são interessantes, mas não conseguiríamos abordar todos nem que este livro tivesse *o dobro* do tamanho. Nosso foco está nas classes e métodos básicos que são importantes para você, um aprendiz. Temos certeza de que você tem um profundo entendimento deles, e confiança de que você sabe como e quando usá-los. Em qualquer caso, depois de ler *Use a Cabeça! Ruby*, você será capaz de pegar qualquer livro de referência e entender facilmente tudo que ele fala sobre as classes e métodos que ficaram de fora aqui.

As atividades NÃO são opcionais.

Os exercícios e atividades não são suplementos; são parte do conteúdo principal do livro. Alguns são para ajudar a memorizar, outros, a entender, e outros ainda, a aplicar o que você aprendeu. ***Não pule os exercícios.***

A redundância é intencional e importante.

Uma diferença distintiva dos livros desta série é que queremos que você *realmente* tire proveito de sua aquisição. E queremos que você termine o livro lembrando o que aprendeu. A maioria dos livros de referência não tem retenção e recuperação como um objetivo, mas este é um livro de *aprendizagem*; portanto, você verá alguns dos mesmos conceitos mais de uma vez.

Os exemplos de código são o mais enxuto possível.

É frustrante percorrer 200 linhas de código procurando duas linhas que você precisa entender. A maioria dos exemplos neste livro é apresentada no menor contexto possível, de modo que a parte que você está tentando aprender seja clara e simples. Portanto, não espere que o código seja robusto, ou mesmo completo. Esta é *sua* tarefa depois de terminar o livro. Os exemplos do livro são escritos especificamente para *aprender*, e nem sempre são totalmente funcionais.

Colocamos todos os arquivos de exemplo na Web para poder baixá-los. Você pode encontrá-los no site da editora, em www.altabooks.com.br procurando pelo titulo do livro ou em *http://headfirstruby.com/*.