



1.

Sobre este Livro



Kon'nichi wa, Ruby



1. Abrindo Este Livro

Finja que você abriu esse livro (embora você provavelmente já tenha aberto esse livro), e encontrou uma enorme cebola bem no meio da dobra do livro. (Eu pedi ao fabricante do livro que incluisse a cebola.)

Logo você pensa, “Wow, esse livro vem com uma cebola!” (Mesmo que você não tenha um apreço particular por cebolas, eu tenho certeza que você consegue apreciar a logística de se enviar qualquer vegetal discretamente dentro de um suposto manual de programação.)

Então você se pergunta: "Espera um pouco. Eu pensei que isso fosse um livro sobre Ruby, a incrível nova linguagem de programação do Japão. E embora eu consiga apreciar a logística de se enviar qualquer vegetal discretamente dentro de um suposto manual de programação, por que uma cebola? O que eu devo fazer com ela?"

Não. Por favor não pense sobre isso. Você não precisa fazer nada com a cebola. Deixe a cebola de lado e deixe *ela* fazer algo com você.

Eu vou ser direto com você. Eu quero chorar. Lamentar. Me lamuriar docemente. Este livro é um guia **comovente** de Ruby. Isso significa código tão lindo que lágrimas serão derramadas. Isso significa contos nobres e verdades melancólicas que vão fazer você acordar na manhã seguinte nos braços desse livro. Abraçando-o forte por todo o dia. Se necessário, improvise uma bolsa para o (*Comovente*) Guia de Ruby do Why, assim você poderá ter sempre a tenra companhia deste livro.

Você deve soluçar pelo menos uma vez. Ou no mínimo suspirar. Se não der certo, a cebola vai fazer tudo isso acontecer para você.

2. A História do Cachorro

Vamos ver se a história a seguir comove você:

Um dia eu estava caminhando numa dessas ruas movimentadas cheias de revendas de carros (isso foi logo após meu casamento ter sido cancelado) e achei um cachorro órfão na rua. Um cachorro preto, peludo, com olhos verde-avermelhados. Eu meio que me senti órfão também, então eu peguei alguns balões que estavam amarrados a um poste em uma revenda e amarrei eles na coleira do cachorro. Eu decidi, então, que ele seria meu cachorro. Eu dei a ele o nome de Bigelow.

Nós fomos comprar alguns Biscrocks para o Bigelow e, depois, seguimos para minha casa, onde nós poderíamos sentar em cadeiras reclináveis e ouvir Zygotic Mynci de Gorky. Oh, e nós também tivemos que parar em um bazar para comprar uma cadeira reclinável para o Bigelow.

Mas Bigelow não tinha me aceitado como seu mestre. Então, cinco minutos depois, aquele cachorro estúpido mudou de calçada e eu o perdi. Logo, considerando que ele já esteve perdido uma vez, com essa eram duas. Eu diminuí o passo em direção a uma vida de Biscrocks e uma cadeira reclinável extra. Eu tive um cachorro por cinco minutos.

Estúpido cachorro traidor. Eu me sentei num banco e joguei pinhas em uma estátua de três ovelhas atravessando uma ponte. Depois disso, eu chorei por horas. As lágrimas simplesmente vieram. Agora existe algo comovente para começar.

Eu fico imaginando aonde ele foi parar com todos aqueles balões. Aquele cachorro maluco parecia uma festa com patas.

Não foi muito depois que eu fiz meu próprio Bigelow. Eu imprimi um bando de páginas sobre Ruby. Artigos achados na Web. Eu dei uma olhada neles no trem indo para casa um dia. Folheei elas por cinco minutos e desisti. Não me impressionou.

Eu sentei, olhando para o mundo através da janela, um liquidificador gigante misturando grafite e ferro derretido bem na minha frente. *Este mundo é muito grande para uma linguagem tão pequena*, pensei. *A pobrezinha não tem chance. Não tem pernas para se sustentar. Não tem braços para nadar.*

Lá estava eu. Um pequeno homem em um frágil e pequeno trem (e eu ainda tinha um dente-de-leite naquela época) em meio a bilhões de pessoas vivendo numa bola azul flutuante. Como eu posso enfrentar Ruby? Quem me garante que eu não vou morrer engasgado com meu celular mais tarde naquela mesma noite? Why está morto, Ruby ainda vive.

A lápide:

O que é isso na traquéia dele?
Oh, olha, um Nokia!

É só a minha sorte. Finalmente vou ter um gostoso e duradouro cochilo embaixo da terra, só para ser constantemente incomodado pela *Cânone de Pachelbel* tocando incessantemente no meu estômago.

O que eu vou fazer com os lucros massivos deste Livro

Qualquer um que já tenha escrito um livro pode lhe dizer como é fácil para um autor se distrair com delírios de grandeza. Na minha experiência, eu paro duas vezes em cada parágrafo, e quatro vezes em cada tirinha, só para antever a riqueza e prosperidade que este livro vai propiciar ao meu estilo de vida. Eu temo que a confecção deste livro possa parar completamente para dar espaço ao exército de utilitários e carros de luxo que estão passando pela minha cabeça.

Em vez de parar meu trabalho neste (comovente) Guia, eu reservei este espaço como uma zona de segurança para despejar meus vazios e inúteis desejos.

Hoje eu estava nesse restaurante italiano, Granado's, e estava pagando a conta. Aconteceu de eu notar que uma garrafa de vinagre balsâmico estava saindo por \$150. Bem pequena. Eu podia escondê-la na minha mão. Envelhecida vinte e dois anos.

Eu gastei muito tempo pensando naquela garrafa. Às vezes ela é um acessório em algumas dessas fantasias obsessivas. Em uma fantasia, eu entro no restaurante, jogo uma pilha de vegetais sobre o balcão e digo seriamente ao caixa: "Rápido! Eu tenho uma salada importante para fazer!"

Em outra fantasia relacionada, eu estou jogando alface fora. Tamanha brutalidade não vai se beneficiar do meu novo vinagre. Não, eu vou chegar ao ponto onde a fama e a aristocracia vão me corromper completamente. Meu novo alface será dinheiro. Frio, duro dinheiro, Sra. Price.

Logo, eu estarei gastando centenas por um bloco de queijo grego Mizithra.

Minha imaginação agora foi além das posses. Certamente, eu pensei sobre a minha aquisição de urnas gregas, carreatas, linhas aéreas, pirâmides, ossos de dinossauro. Ocionalmente eu verei cidades devastadas no noticiário e vou anotar na minha lista de compras: *Furacão*.

Entretanto, agora eu vejo um objetivo maior. Colocando de forma simples: e se eu acumulasse uma fortuna tão grande que a casa da moeda não conseguisse imprimir o suficiente para atender a minha demanda? Então, todo mundo seria obrigado a usar dinheiro do Banco Imobiliário como moeda. E você teria que ganhar no Banco Imobiliário para colocar comida na mesa. Estes seriam jogos de extrema tensão. Quero dizer, você faz uma hipoteca na

3. Nasce o Sol Vermelho

Então, agora você está se perguntando porque eu mudei de idéia em relação ao Ruby. A resposta rápida é: rolou química.

Como quando você encontra Alguém na faculdade e ela parece com alguém que costumava bater na sua cara com um pincel quando você era criança. E então, impulsivamente, você conclui que esse novo alguém é provavelmente hostil. Você faz careta quando vê o cabelo dessa pessoa. Você desliga o telefone bruscamente nos momentos cruciais das piadas dela. Você resolve usar seu pula-pula bem onde ela está querendo caminhar!

Seis meses depois, de alguma forma, você e esse Alguém estão sentados a beira de uma fonte tendo uma conversa perfeitamente normal. A face dela não lembra mais seu nêmeses infantil. Você conheceu o gêmeo bom. Rolou química.

Logo, apesar de que eu provavelmente devesse estar martelando seus dentes com todo o hype sobre Ruby e todos os acrônimos minuciosamente preparados que acompanham para todo lugar (afiando as gargantas dos seus chefes e dos chefes do seus chefes), em vez disso eu vou te deixar velejar. Vou deixar você mergulhar no código, me intrometendo ocasionalmente com minhas próprias experiências. Vai ser bem fácil, bem natural.

Eu devia, entretanto, te oferecer alguma espécie de motivação. Então, cara pálida, eu vou lhe dar minhas três melhores razões para aprender Ruby e te deixar em paz.

Av. Botafogo e as crianças começam a chorar. Além disso, eu acho que você verá o início do fim daqueles que usam a regra de Estacionamento Grátis como [os cofres subterrâneos](#) para os recursos da cidade.

Apesar disso, você tem que dar algum valor ao dinheiro falso. Dinheiro falso detona. Você pode colocar suas mãos nele muito rápido. Em um momento, parece que você é podre de rico. Quando eu era criança, eu me juntei com algumas crianças da vizinhança e nós construímos uma pequena Tijuana na nossa rua. Nós fizemos nossos próprios pesos, vestimos sombreiros e tudo mais!

Um garoto estava vendendo pamonhas quentes por dois pesos cada. *Dois pesos!* Esse garoto sabia que o dinheiro era falso? Ele era louco? Quem convidou esse garoto? Ele não sabia que isso aqui não era Tijuana de verdade? Talvez ele fosse de Tijuana! Talvez estes fossem pesos *de verdade!* Vamos fazer mais pesos *de verdade!*

Eu acho que nós até tínhamos uma taberna onde você podia ficar completamente bêbado de Ki-Suco. Nada como crianças correndo por aí, resmungando incoerências com os lábios vermelhos como os de um palhaç

1. Saúde mental.

Vitamina R. Vai direto para a cabeça. Ruby vai lhe ensinar a *expressar* suas idéias através de um computador. Você estará escrevendo estórias para uma máquina.

Habilidades criativas, gente. Dedução. Razão. Acenar com a cabeça inteligentemente. A linguagem vai se tornar uma ferramenta para melhorar a comunicação entre sua mente e o mundo. Eu notei que muitos usuários experientes de Ruby parecem pensadores mais claros e objetivos. (Em contraste a: altamente preconceituosos e grosseiros.)

2. Um homem em uma ilha.

Ruby nasceu no Japão. O que é bizarro. O Japão não é conhecido por seu software. E como linguagens de programação são largamente escritas em inglês, quem iria imaginar uma linguagem vinda do Japão?

E ainda assim, aqui temos Ruby. Contra todas as adversidades, Yukihiro Matsumoto criou Ruby em 24 de Fevereiro de 1993. Pelos últimos dez anos, ele tem obstinadamente trazido Ruby para uma audiência global. É triunfante, nobre e tudo mais. Apóie a diversidade. Ajude-nos a dobrar a Terra, só um pouquinho.

3. Grátis.

Usar Ruby não custa nada. O código do próprio Ruby está aberto para todo o mundo inspirar/expirar. Até esse

livro é de graça. É tudo parte de uma enorme doação que deveria ter alguma venda casada junto.

Você poderia pensar que nós faríamos você comprar aspiradores de pó ou tempo de uso, ou falsos Monet. Você poderia pensar que haveria uma palestra de 90 minutos onde o dono da empresa aparece no final e força você a fechar negócio.

Não. É grátis.

Com isso, é hora do livro começar. Você pode pegar seu marca-texto e começar a arrastá-lo sobre cada palavra cativante a partir desta sentença. Eu acho que tenho spray de cabelo e dinheiro de brinquedo o suficiente comigo para me sustentar até a página final.

4. Como Livros Começam

Olha, se você algum dia já leu um livro, você sabe que nenhum livro pode começar direito sem uma quantidade exorbitante de sinergia. Sim, sinergia. Talvez você não soubesse disso. Sinergia quer dizer que eu e você devemos cooperar para fazer desta leitura uma ótima experiência.

Começamos o livro nos dando bem na Introdução. Este companheirismo, esta **sinergia**, nos impulsiona pelo livro, comigo guiando você em seu caminho. Acene com a cabeça ou dê uma risadinha para indicar o seu progresso.

Eu sou Peter Pan segurando a sua mão. Vamos, Wendy! Segunda estrela à direita e em frente até o amanhecer.

Um problema aqui. Eu não me dou bem com gente. Eu não seguro mãos muito bem.

Qualquer um da minha equipe vai lhe dizer. Nas Cerimônias de Abertura Deste Livro (um evento com Buffet e arquibancadas), eu descobri que os sanduíches de pepino não eram servidos em guardanapos de tecido. Como resultado, a manteiga não se ajustou aos pepinos direito... De qualquer forma, Eu fiz uma grande cena e ateei fogo a alguns caminhões de propaganda do lado de fora. Eu deixei um holofote em pedaços, e coisas do tipo. Eu dava risadas maníacas extremamente altas até o final daquela noite. Foi uma bagunça e tanto.

Entretanto, já que eu não me dou bem com gente. Eu não convidei ninguém além de mim para as Cerimônias de Abertura Deste Livro. Então não foi tão embaraçoso assim. Eu mantive o caso sob panos quentes e ninguém jamais ficou sabendo do que aconteceu.

Então você tem que saber que **sinergia** não significa exatamente **sinergia** nesse livro. Eu não posso com a **sinergia** normal. Não, neste livro, **sinergia** significa **raposas em quadrinhos**. O que eu quero dizer é: este livro vai começar com uma quantidade exorbitante de **raposas em quadrinhos**.

E eu vou contar com você para transformá-las em **sinergia**.

3.

Um Rápido (e indolor, eu espero) passeio através do Ruby (com quadrinhos da raposa)





Isso, essas são as duas. Minha asma está atacada, preciso dar uma respirada num ar medicinal agora. Já volto.



Disseram-me que este capítulo combina bem com uma toalha. Algo para você enxugar seu rosto quando o suor começar a escorrer.

De fato, estaremos percorrendo toda a linguagem muito rápido. Será como acender todos os fósforos de uma caixa do jeito mais rápido possível.

1. Linguagem e EU QUERO DIZER Linguagem



Minha consciência não me deixa chamar o Ruby de uma linguagem de *computador*. Isso implica que a linguagem funciona primariamente em termos de computador. Que a linguagem é criada para acomodar o computador, única e exclusivamente. E portanto, nós, os programadores, somos estrangeiros, buscando cidadania no país dos computadores. É a linguagem do computador e nós somos os tradutores para o mundo.

Mas como você a chamaria quando seu cérebro começa a pensar nessa linguagem? Quando você começa a usar as próprias palavras e coloquialismos da linguagem para se expressar. Diga, o computador não consegue fazer aquilo. Como pode então ser a linguagem do computador? Ela é nossa, nós a falamos nativamente!

Nós não podemos mais verdadeiramente chamá-la de linguagem de *computador*. Ela é *papo de programador*. É a linguagem dos nossos pensamentos.

Leia o trecho a seguir em voz alta.

```
5.times { print "Odelay!" }
```

No inglês, as pontuações (pontos, exclamações, parênteses) são mudas. Pontuação agrega significado às palavras, dá pistas da intenção do autor na frase. Vamos então ler o trecho assim: *Cinco vezes imprima "Odelay!"*.

Que é exatamente o que este pequeno programa Ruby faz. A exclamação [espanhol mutante](#) do Beck será impressa

cinco vezes na tela do computador.

Leia o trecho a seguir em voz alta.

```
exit unless "restaurante".include? "aura"
```

Aqui estamos realmente fazendo um teste básico. Nossa programa vai **exit (sair)** (o programa vai terminar) **unless (a menos que)** a palavra **restaurante** contenha (ou **inclua**) a palavra **aura**. Novamente, em Português: *Saia se a palavra restaurante incluir a palavra aura.*

Já viu alguma linguagem de programação usar a pontuação tão eficientemente? Ruby usa a pontuação, como exclamações e interrogações, para melhorar a legibilidade do código. Nós estamos fazendo uma pergunta no código acima, então por quê não deixar isso aparente?

Leia o trecho a seguir em voz alta.

```
['torrada', 'queijo', 'vinho'].each { |comida| print comida.capitalize }
```

Enquanto este pedaço de código é menos legível e parecido com uma frase que o anterior, eu ainda assim o encorajo a ler em voz alta. Enquanto o Ruby algumas vezes soa como Português noutras soa como um breve Português. Totalmente traduzido para Português, você poderia ler desta maneira: *Com as palavras ‘torrada’, ‘queijo’, e ‘vinho’: pegue cada comida e a imprima com a primeira letra maiúscula.*

O computador então responde de forma cortês: **Torrada, Queijo e Vinho.**

Neste ponto, você deve estar se perguntando como essas palavras realmente funcionam juntas. O cara-pálida deve estar imaginando o que os pontos e chaves significam. Vou discutir as várias *partes da conversa* em seguida.

Tudo que você precisa saber até agora é que o Ruby é basicamente construído de sentenças. Elas não são exatamente sentenças em inglês. Elas são pequenas coleções de palavras e pontuação que expressam um simples pensamento. Estas sentenças podem formar livros. Elas podem formar páginas. Elas podem formar romances inteiros, quando combinadas. Romances que podem ser lidos por humanos, mas também por computadores

2. As Partes da Conversa

Assim como a faixa branca nas costas do gambá e a branca e enrolada grinalda da noiva, várias partes da conversa têm sinalizações visuais para te ajudar a identificá-las. Pontuação, maiúsculas e minúsculas, irão ajudar seu cérebro ver pedaços de código e reconhecê-los intensamente. Sua mente vai gritar freqüentemente *Hey, Eu conheço aquele cara!* Você também poderá contar vantagem em conversas com outros Rubistas.

Tente se focar na aparência de cada uma destas partes da conversa. O resto do livro vai entrar em detalhes. Eu darei pequenas descrições de cada parte da conversa, mas você não precisa entender a explicação. Ao término deste capítulo, você deve estar apto a reconhecer cada parte de um programa Ruby.

Variáveis

Qualquer palavra simples, em minúsculas, é uma variável no Ruby. Variáveis podem ser compostas por letras, dígitos e travessões.

x, y, banana2 ou telefone_para_codorna

A respeito do uso comercial do (Comovente) Guia

Este livro foi lançado sob a licença Creative Commons que permite uso comercial ilimitado deste texto. Basicamente, isto significa que você pode vender todas cópias piratas do meu livro e ficar com todo o lucro para você. Eu confio nos meus leitores (e no mundo ao redor deles) para me passar para trás. E colocar uma edição em Xerox com aquela figura das mãos unidas rezando, que nós conhecemos, na capa.

Caras, processar os outros não vale a pena, é muita dor de cabeça. Então eu simplesmente vou apoiar a pirataria aqui. Qualquer um que quiser ler este livro deveria poder fazê-lo. Qualquer um que queira comercializar o livro ou criar edições especiais, eu fico lisonjeado.

Porque eu iria querer o \$\$\$? IGNORE TODAS AS OUTRAS BARRAS LATERAIS: Eu perdi a vontade de ser um ricaço preguiçoso. Soa um

são exemplos.

Variáveis são como apelidos. Lembra quando todo mundo te chamava de Pete Fedido? As pessoas diziam: "Vem aqui, Pete Fedido!" E todo mundo milagrosamente sabia que você era o Pete Fedido.

Com variáveis, você dá um apelido a algo que você use freqüentemente. Por exemplo, vamos dizer que você tem um orfanato. É um orfanato maléfico. E quando o Papai Warbucks vem comprar mais crianças, nós insistimos que ele nos pague **cento e vinte um reais e oito centavos** pelo ursinho de pelúcia da criança, o qual ela ficou apegada nos momentos sombrios que passou vivendo neste pesadelo de custódia.

`taxa_de_urso_de_pelucia = 121.08`

Mais tarde, quando você pegar ele no caixa (uma registradora turbinada que roda Ruby!), você vai precisar juntar as somas em um **total**.

`total = taxa_de_orfao +
taxa_de_urso_de_pelucia + gorjeta`

Estes apelidos para variáveis vêm a calhar. E no submundo sedento das vendas infantis, qualquer ajuda é bem-vinda, tenho certeza.



Números

O tipo mais básico de número é um *integer* (número inteiro), uma **série de dígitos** que podem começar com um **sinal de menos ou de mais**.

`1, 23 e -10000` são exemplos.

Vírgulas não são permitidas em números, mas travessões sim. Então se você acha melhor marcar os milhares para que o número fique mais legível, use um travessão.

`habitantes =`

`12_000_000_000`

Números decimais são chamados de *floats* (números reais ou ponto flutuante) no Ruby. Números reais consistem em números com **cotas decimais** ou **em notação científica**.

pouco desumano, mas eu gosto da minha televisão preto-e-branco. E adoro minha luminária florida. Eu não quero uma carreira de escritor. Dinheiro não vai me inspirar. Não faz sentido.

Então, se dinheiro não significa nada ao cara sortudo (the lucky stiff), para que me enganar quando você pode optar por práticas comerciais exclusivas para literalmente esmagar minha psique e me deixar chorando com meu pulmão negro? Oh, e a ironia de usar meu trabalho contra mim! Morra, garoto comovido!

Para lhe dar uma idéia do que quero dizer, aqui estão alguns conceitos dissimulados que poderiam seriamente matar minha força de vontade de me forçar a reconsiderar algumas coisas, como a existência.

IDÉIA UM: TABACARIA GIGANTE

Compre uma companhia de cigarros. Use meus cartuns dos raposos para abastecer uma agressiva campanha publicitária. Aqui está um anúncio para começar:



Deixe óbvio que seu público alvo são crianças e asmáticos. Então, logo que você tiver tudo pronto, faça com que as pessoas da **verdade** façam um dossiê sobre mim e minha fazenda de raposas de tinta.

Moderninho sensível apoiado na borda da selva urbana: Ele se intitula o cara sortudo.

(Puxa as cortinas para revelar um corpo cinza em uma maca.)

Moderninho: Alguns caras não são tão sortudos assim.

3.14, -808.08 e 12.043e-04 são exemplos.

Strings (Conjunto de Caracteres)

Strings são quaisquer tipos de caracteres (letras, dígitos, pontuação) cercados por aspas. **Aspas**, simples ou duplas, são usadas para se criar strings.

"laboratoriodomar", '2021' ou "Estes quadrinhos são hilários!" são exemplos.

Quando você cerca caracteres em aspas, eles ficam guardados juntos como uma string só.

Pense num repórter que está anotando as baboseiras de uma celebridade incoerente. "Eu estou muito mais sábia," disse Avril Lavigne. "Agora eu sei como as coisas são — o que você tem que fazer e como trabalhar isso."

```
frase_da_avril = "Eu estou muito mais sábia.  
Agora eu sei  
como as coisas são -- o que você tem  
que fazer e como trabalhar isso."
```

Então, assim como nós guardamos um número na variável **taxa_de_urso_de_pelúcia**, agora nós estamos guardando uma coleção de caracteres (uma string) na variável **frase_da_avril**. O repórter envia esta frase para impressão, que por acaso usa Ruby para operar as impressoras.

```
<setup>  
  frase_da_oprah = "O"  
  frase_da_avril = "A"  
  debate_ashlee_simpson = "D"  
</setup>  
print frase_da_oprah  
print frase_da_avril  
print debate_ashlee_simpson
```



Symbols (Símbolos)

Símbolos são palavras que parecem variáveis. Novamente, elas podem conter letras, dígitos e travessões. Mas elas **começam com dois pontos**.

:a, :b, ou :ponce_de_leon são exemplos.

Símbolos são strings leves. Geralmente, símbolos são usados em situações em que você precisa de uma string mas não vai

(Som errático. Raposos em quadrinhos sobrepostos para gerar uma viagem subliminar mental Willy Wonka.)

Yo. Por que você tem esses Big Smokies assim, Holmes?

IDÉIA DOIS: HEY, PELOTÃO DE FUZILAMENTO

Como eu disse, comecem a vender cópias do meu livro, mas corrompam o texto. Estas cópias alteradas conteriam numerosas referências grosseiras (e difamatórias) às agências do governo, como o exército americano e o pentágono. Você me faria com que eu parecesse um completo traidor. Como se eu tivesse todos esses planos para, você sabe, matar certos membros menos desejáveis do exército ou do Pentágono.

Não que existam membros menos desejáveis no exército ou no Pentágono. É, não era isso que eu queria dizer.

Oh, merda.

Oh, merda. Oh, merda. Oh, merda.

Apague as luzes. Abaixe-se.

IDÉIA TRÊS: ANÚNCIOS, PARTE II

Que tal fazer graça dos asmáticos diretamente??



IDÉIA QUATRO: ALEC BALDWIN

Faça uma adaptação do livro para o cinema. E já que eu, você sabe, sou um personagem no livro você poderia conseguir alguém como Alec Baldwin para me interpretar. Alguém que esteja num período muito ruim na carreira.

Você podia fazer como se eu usasse toneladas de drogas. Como se trabalhar comigo fosse algo insano. Como se eu ficasse demitindo pessoas e trancando elas na sala da scooter e obrigando-as a vestir roupas feitas de pão. Isso, como se eu

imprimi-la na tela.

Você pode dizer que um símbolo é pouco mais digerível para o computador. É como um anti-ácido. Os dois pontos indicam as bolhas saindo do estômago do seu computador enquanto ele digere o símbolo. Ah. Doce, doce alívio.



Constantes

Constantes são palavras como variáveis, mas constantes são **maiúsculas**. Se variáveis são os substantivos do Ruby, então pense nas constantes como substantivos próprios.

Time, Array ou Bunny_Lake_Desapareceu são exemplos.

Em inglês, substantivos próprios têm a primeira letra maiúscula. O Edifício Empire State. Você não pode simplesmente mudar o Edifício Empire State. Você não pode simplesmente decidir que o Edifício Empire State é outra coisa. Substantivos próprios são assim. Eles se referem a algo muito específico e que usualmente não muda com o tempo.

Do mesmo modo, constantes não podem ser alteradas depois de criadas.

EdificioEmpireState = "350 5th Avenue, NYC, NY"



pudesse realmente assar as pessoas nas roupas.

Você podia dizer que eu tenho uma forma gigante e que eu prenho as pessoas dentro. Então, eu derramo toda massa e de fato asso eles até o pão crescer e eles estarem quase mortos. E quando o pessoal da televisão vier e eu aparecer no Bom Dia Brasil, eles vão perguntar, “Então, quantas pessoas você empregou na produção do seu livro?” E eu responderia, “Uma dúzia de padeiros!” e eclodiria em gargalhadas maníacas tão altas que forçariam a audiência a tapar os ouvidos.

É claro que, no curso da minha insanidade, eu declararia guerra ao mundo. E daria uma briga boa com as pessoas de pão. Até que o exército americano (ou o Pentágono) construísse um cérebro de macaco robótico gigante (interpretado por Burt Lancaster) para me perseguir.

Aqui você vai me fazer parecer completamente idiota. Eu não sacrificarei apenas todas as pessoas pão (os Starchtroopers, cavaleiros do amido) para me salvar, não apenas me renderei covardemente ao grande cérebro de macaco, mas quando eu escapar por um triz, eu vou gritar para a platéia. Gritando incessantemente que aquele é *MEU* filme e que ninguém deveria mais assisti-lo, eu rasgarei a tela ao meio e o projetor vai rodar seus rolos em vão. E este será o final do filme. As pessoas ficarão *tão* irritadas.

Fiquei pensando aqui. De fato Alec Baldwin fez um trabalho decente de dublagem em *Os Excêntricos Tenenbaums*. A carreira dele deve estar indo bem. Você pode não querer usá-lo. Ele pode não aceitar.

Fazemos o seguinte. Eu interpreto esse papel. Eu tenho uma carreira de papéis ruins.

Métodos

Se variáveis e constantes são substantivos, então métodos são verbos. Métodos geralmente estão ligados ao final das variáveis e constantes por um **ponto**. Você já viu métodos trabalhando.

```
porta_da_frente.abra
```

Acima, **abra** é o método. Nesta ação, o verbo. Em alguns casos, você verá ações ligadas juntas.

```
porta_da_frente.abra.feche
```

Instruímos o computador a abrir a porta da frente e depois imediatamente fechá-la.

```
porta_da_frente.está_aberta?
```

Acima temos uma ação também. Estamos instruindo o computador a testar a porta para ver se está aberta. O método poderia se chamar **Porta.teste_para_ver_se_está_aberta**, mas o nome **está_aberta?** é sucinto e também correto. Exclamações e interrogações podem ser usados em nomes de métodos.

Argumentos de Método

Um método talvez precise de mais informação para realizar sua ação. Se quisermos que o computador pinte a porta, devemos prover uma cor também.

Argumentos de método vão ligados ao final do método. Os argumentos estão geralmente cercados por **parênteses** e separados por **vírgulas**.

```
porta_da_frente.pintar( 3, :vermelho )
```

O código acima pinta a porta da frente com três demões de vermelho.

Pense nisso como um tubo interno que o método carrega, contendo suas instruções extras. Os parênteses formam as redondas e molhadas paredes do tubo. As vírgulas são os pés de cada argumento, saindo para fora da beirada. O último argumento tem o pé preso embaixo então eles não aparecem.

Como um barco de bóias, métodos com argumentos podem ser ligados.

```
porta_da_frente.pintar( 3, :vermelho ).secar( 30 ).fechar()
```

O código acima pinta a porta da frente com três demões de vermelho, seca por trinta minutos e fecha a porta. Mesmo que o último método não tenha argumentos, você pode colocar parênteses se quiser. Um cano vazio não tem muito uso, então os parênteses são geralmente deixados de lado.

Alguns métodos (como **print**) são métodos do kernel. Estes métodos são usados por todo Ruby. Já que eles são tão comuns, você não usa o ponto.

```
print "Veja, sem ponto."
```

Métodos de Classe

Assim como os métodos descritos acima (também chamados métodos *de instância*), métodos de classe são geralmente ligados depois de variáveis e constantes. Ao invés de ponto, usa-se **dois pontos duplo**.

`Porta::new(:carvalho)`

Como foi visto acima, o método de classe `new` é mais usado para se criar coisas. No exemplo cima, estamos pedindo ao Ruby para fazer uma porta de carvalho nova para nós. Claro, o Ruby não tem conhecimento de como fazer uma porta—assim como uma pilha de madeira, madeireiros, e aquelas longas, nervosas, serras acionadas por dois homens.



Variáveis Globais

Variáveis que começam com um **cifrão** são globais.

`$x, $1, $pedacudo` e `$BAcOn_PeDAcUDo` são exemplos.

A maioria das variáveis são temporárias por natureza. Algumas partes do seu programa são como casas. Você entra e elas têm suas próprias variáveis. Em um casa, você pode ter um **pai** que representa Archie, um caixeiro-viajante colecionador de esqueletos. Em outra casa, **pai** pode representar Peter, um domador de leões com grande apreço por flanela. Cada casa tem um sentido próprio para **pai**.

Com variáveis globais, você garante que a variável será a mesma em cada casinha. O sinal de cifrão é bem apropriado. Todo lar Americano respeita o valor do cifrão. Somos doidos pela coisa. Tente bater em qualquer porta na América e dê a eles dinheiro. Eu posso garantir que você não verá a mesma reação se bater em uma porta e oferecer Peter, um domador de leões com grande apreço por flanela.

Variáveis globais podem ser usadas em qualquer lugar no seu programa. Elas nunca saem da vista.

Variáveis de Instância

Variáveis que começam com uma **arroba** são variáveis de instância.

`@x, @y`, e `@somente_o_maior_pedaco_de_bacon_que_eu_ja_viu` são exemplos.

Estas variáveis são muito usadas para se definir atributos de alguma coisa. Por exemplo, você pode prover o Ruby com a largura da `porta_da_frente` criando a variável `@largura` dentro daquela `porta_da_frente`. Variáveis de instância são usadas para se definir características de um objeto em Ruby.

Pense no símbolo **arroba (at)** como significando **atributo**.

Variáveis de Classe

Variáveis que começam com **duas arrobas** são variáveis de classe.

`@@@x@, @@@y@, e
@@@vou_pagar_seus_bacons_pedacudos_e_ensinar_uma_licao_a_voces_dois@` são exemplos.

Variáveis de classe são usadas, também, para se definir atributos. Mas ao invés de definir um atributo a apenas um objeto no Ruby, variáveis de classe dão um atributo a vários objetos relacionados no Ruby. Se as variáveis de instância estipulam atributos para só uma **porta_da_frente**, então variáveis de classe estipulam atributos para tudo que for **Porta**.

Pense no prefixo da **dupla arroba** significando **atribua a todos**. Adicionalmente, você pode pensar em um esquadrão de **AT-ATs** do *Guerra nas Estrelas*, que são comandados pelo Ruby. Você muda uma variável de classe e não apenas uma muda, todas elas mudam.



Blocos

Qualquer código cercado por **chaves** é um bloco.

`2.times { print "Sim, Tenho usado bacon pedaçudo nos meus exemplos, mas
não farei de novo!" }` é um exemplo.

Com blocos, você pode agrupar pedaços de instruções juntas, assim elas podem ser passadas pelo seu programa. As chaves dão a aparência de garras de caranguejo que pegaram o código e o estão segurando. Quando você vir essas duas garras, lembre-se de que o código dentro foi prensado em uma só unidade.

É como uma aquela caixinha da Hello Kitty que eles vendem no shopping que vêm lotadas de pequenos lápis e papel microscópico, espremidos numa bolsa transparente e brilhante que pode ser escondida na palma da mão para operações secretas. Exceto que, blocos não ofuscaram sua visão.

As chaves podem ser substituídas pelas palavras **do** e **end**, o que é bacana se seu bloco for maior que uma linha.

```
loop do
  print "Bem melhor."
  print "Ah. Mais espaço!"
  print "Minhas costas estavam me matando dentro daquelas garras de caranguejo."
end
<stdout>Bem melhor.Ah. Mais espaço!Minhas costas estavam me matando dentro daquelas
garras de caranguejo.</stdout>
```

Argumentos de Bloco

Argumentos de bloco são uma série de variáveis cercadas por símbolos **tal que** e separadas por **vírgulas**.

`|en| |x|, |x,y|, e |cima, baixo, todo_lado|` são exemplos.

Argumentos de bloco são usados no início do bloco.

```
{ |x,y| x + y }
```

No código acima, `|x,y|` são os argumentos. Depois dos argumentos, temos um pedaço de código. A expressão `x + y` soma os argumentos.

Eu gosto de pensar nos símbolos tal que representando um túnel. Ele dão a aparência de uma calha na qual as variáveis estão descendo. (O `x` desce com as pernas abertas, enquanto o `y` harmoniosamente cruza as pernas.) Essa

calha age como uma passagem entre os blocos e o mundo lá fora.

Variáveis são passadas por essa calha (ou túnel) para dentro do bloco.



Ranges (Intervalos)

Um range é formado por dois valores cercados por **parênteses** e separados por **reticências** (na forma de dois ou três pontos).

(1..3) é um range, representando os números de um até três.

('a'..'z') é um range, representando o alfabeto em minúsculas.

Pense neles como um acordeão que fora fechado para se carregar. (Você pode construir muito amor próprio levando consigo um acordeão aberto... mas, às vezes uma pessoa tem que mergulhar em dúvidas e esconder cuidadosamente a sanfona) Os parênteses são os lados deste pequeno acordeão de mão. Os pontos são a fole, mantendo as partes bem unidas.

Normalmente, apenas dois pontos são usados. Se um terceiro ponto for usado, o último valor no range será excluído.

(0...5) representa os números de zero até quatro.

Quando você vê o terceiro ponto, imagine abrir o acordeão um pouco menos. O necessário para uma nota apenas de seu fole. A nota é este último valor. Nós vamos deixar que o céu a coma.

Arrays (Conjuntos, Vetores)

Um array é uma lista cercada por **colchetes** e separada por **vírgulas**.

[1, 2, 3] é um array de números.

['casaco', 'luvas', 'snowboard'] é um array de strings.

Pense nele como uma centopéia que foi grampeada no nosso código. Os dois colchetes são grampos que não deixam a centopéia se movimentar, então você consegue distinguir onde é cabeça e onde é a cauda. As vírgulas são as pernas da centopéia, balançando entre cada seção do seu corpo.

Era uma vez uma centopéia que tinha vírgulas ao invés de pernas. O que significa que ela tinha que dar uma pausa literária a cada passo. As outras centopéias a respeitavam muito por isso e ela veio a ter uma bela presença no comando. Oh, e que filantropo! Ela ficou famosa por dar folhas frescas aos menos afortunados.

Sim, um array é uma coleção de coisas, mas ele também mantém estas coisas em uma ordem específica.

Hashes

Um hash é um dicionário cercado por **chaves**. Dicionários servem para encontrar definições das palavras. O Ruby faz isso com **setas** feitas de sinais de igual, seguida por um sinal de maior que.

`{'a' => 'porco-da-terra', 'b' => 'texugo'}` é um exemplo.

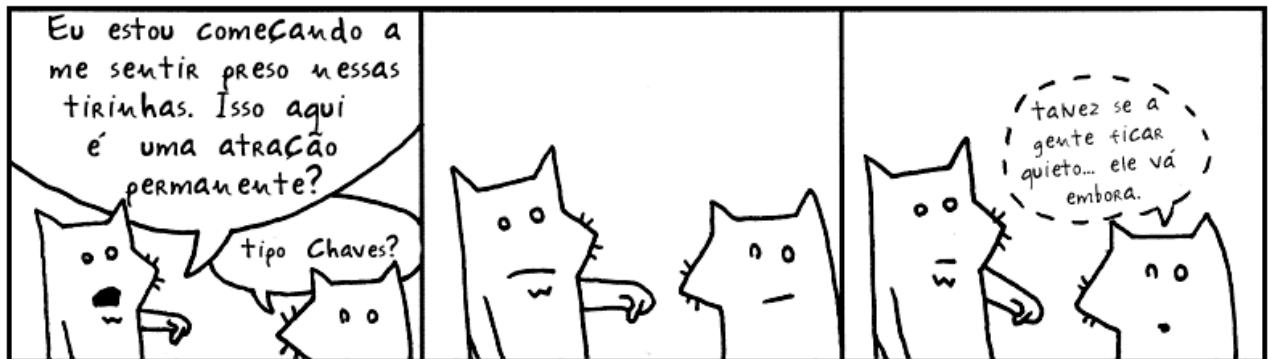
Desta vez, as chaves representam livrinhos. Veja como eles se parecem com pequenos livros abertos com dobras no meio. Eles representam abrir e fechar nosso dicionário.

Imagine que nosso dicionário tem um definição em cada página. As vírgulas representam os cantos de cada página, que nós viramos para ver a próxima definição. E em cada página: uma palavra seguida por uma seta apontando a definição.

```
{  
  'nome' => 'Peter',  
  'profissao' => 'domador de leões',  
  'grande apreco' => 'flanelas'  
}
```

Não estou comparando hashes a dicionários por que você pode guardar só definições na hash. No exemplo acima, eu guardei informação pessoal de Peter, o domador de leões com grande apreço por flanelas. Hashes são como dicionários porque elas são muito fáceis de se procurar algo.

Diferentemente dos arrays, os itens na hash não são mantidos em ordem específica.



Expressões Regulares

Uma expressão regular (ou *regexp*) é uma série de caracteres cercados por **barras**.

`/ruby/, /[0-9]+/ e /^\d{3}-\d{3}-\d{4}/` são exemplos.

Expressões regulares são usadas para se achar palavras ou padrões no texto. As barras dos lados da expressão são alfinetes.

Imagine se você tivesse uma palavrinha com alfinetes de cada lado e você a segurasse por cima de um livro. Você passa a palavra pelo livro e quando ela chega perto de uma palavra que coincide, ela começa a piscar. Você espeta a expressão regular no livro, bem em cima da palavra que bate e ela brilha com as letras da palavra encontrada.

Oh, e quando você espeta os alfinetes no livro, o papel espirra, *reg-exp*!

Expressões regulares são muito mais rápidas que folhear um livro. O Ruby pode usar uma expressão regular para procurar volumes de livros muito rapidamente.

Operadores

Você usará a seguinte lista de operadores para matemática ou para comparar coisas no Ruby. Analise a lista, reconheça alguns. Você conhece, adição + e subtração - e por aí vai.

```
** ! ~ * / % + - &  
<< >> | ^ > >= < <= <=>  
|| != =~ !~ && += -= == ===  
... ... not and or
```

Keywords (Palavras-chave, Palavras-reservadas)

O Ruby tem um número de palavras embutidas, imbuídas em significado. Estas palavras não podem ser usadas como variáveis ou modificadas às suas necessidades. Algumas delas nós já discutimos. Elas estão na caixa forte, meu amigo. É só relar nelas e você terá um erro oficial de sintaxe.

```
alias and BEGIN begin break case class def defined
do else elsif END end ensure false for if
in module next nil not or redo rescue retry
return self super then true undef unless until when
while yield
```

Muito bom. Estes são os ilustres membros da linguagem Ruby. Teremos um belo banquete nos próximos três capítulos, grudando essas partes em manhosos pedaços de (comovente) código.

E recomendo que você dê mais um olhadela pelas partes da conversa. Dê uma ampla olhada nelas. Eu estarei testando seu metal na próxima seção.



3. Se Eu Já Não Tivesse Lhe Tratado Suficientemente Como Uma Criança

Estou orgulhoso de você. Qualquer um irá lhe contar o tanto que me orgulho de você. Como eu continuo e continuo a falar sobre essa ótima pessoa anônima aí fora que rola e lê e rola e lê. "Estas crianças," Eu lhes conto. "Cara, estas crianças tem coração. Eu nunca..." E eu mal consigo terminar uma frase porque estou absolutamente mergulhado em lágrimas.

E meu coração irradia um vermelho intenso sob minha pele translúcida e eles tiveram que me administrar 10 mililitros de Javascript para me fazer voltar. (Eu respondo bem a toxinas no sangue.) Cara, esse negócio vai chutar para longe das suas guelras!

Bem, sim. Você está indo bem. Mas agora eu devo começar a ser um bruto professor secundário. Preciso começar a ver boas notas suas. Até agora, você não fez nada mais do que olhar para cima e para baixo. Ok, claro, você fez excepcionais leituras em voz alta anteriormente. Agora nós precisamos de habilidade de compreensão aqui, Smotchkiss.

Diga em voz alta cada parte do discurso usada abaixo.

```
5.times { print "Odelay!" }
```

Você pode querer até mesmo cobrir este parágrafo enquanto lê, porque seus olhos podem esquivar-se para olhar a resposta. Nós temos um *número* 5, seguido por um *método* `.times`. Depois, a primeira garra de caranguejo de um *bloco*. O *método* do kernel `print` não tem ponto e é seguido por uma *string* "Odelay!". A

Sete momentos Zen da minha vida

1. 8 anos de idade. Apenas deitado na cama, pensando. E eu percebo. *Não há nada que me impeça de me tornar um odontopediatra.*
2. 21. Achei um lápis na praia. Gravado nele estava: *Eu estimo a serenidade.* Enfiei ele no bolso do meu paletó. Assisti as ondas indo e voltando.
3. 22. Encontrei um besouro no meu banheiro que estava prestes a cair na saída do aquecedor. Adotei sua causa. Fiz uma pequena mochila com uma folha e linha. Dentro da mochila: uma estaca e uma pilha palito. Isto deve ser suficiente para ele. Libertei-o pelo portão da frente.
4. Três anos de idade. Puxei a cortina para o lado. Luz do sol.
5. 14. Andando de bicicleta no píer com meu treinador correndo logo atrás de mim enquanto o sol se põe logo depois que eu nocauteei Piston Honda na versão original para Nintendo do Punch-Out com Mike Tyson.
6. 11. Doente. Assistindo Heathcliff na televisão. Por horas, era só Heathcliff. E ele era capaz de vir bem perto do meu

garra de caranguejo final fecha nosso *bloco*.

Diga em voz alta cada parte do discurso usada abaixo.

```
exit unless "restaurante".include?  
"aura"
```

Assim como o método `print`, `exit` é um *método* do kernel. Se você prestou atenção na grande lista de palavras reservadas, você saberá que `unless` é uma dessas *palavras reservadas*. A string "`restaurante`"

está unida ao método `include?`. E no final, a string "`aura`".

Diga em voz alta cada parte do discurso usada abaixo.

```
['torrada', 'queijo', 'vinho'].each { |alimento|  
  print( alimento.capitalize ) }
```

Esta centopéia partilha finas iguarias. Um *array* inicia o exemplo. No array, três *strings* '`'torrada'`', '`'queijo'`' e '`'vinho'`'. Todo o array é seguido por um método `each`.

Dentro de um *bloco*, o *argumento do bloco* `alimento`, descendo seu tobo-água dentro do bloco. O método `capitalize` então passa para maiúscula a primeira letra do argumento do bloco, a qual inicia a variável `alimento`. Esta *string* com a inicial em maiúscula é passada para o método `print` do kernel.

Olhe para os exemplos mais uma vez. Tenha certeza de que você reconhece todas as partes do discurso usadas. Elas são bem distintas, não? Respire profundamente, aperte firmemente suas têmporas. Pronto, agora vamos dissecar código tão valioso quanto um olho de vaca.

rosto. A cabeça dele crescia em minha direção. Seu rosto pulsava, bem perto, e depois milhões de milhas distante. O som havia sumido. Em frações de segundo, Heathcliff preencheu o universo, então se foi para o fim da eternidade. Eu ouvi a voz da minha mãe tentando falar mais alto que o desenho. Heathperto, Heathlonge, Heathperto, Heathlonge. Era um delírio religioso com um gato, um estroboscópio e o grave abafado da voz da minha mãe. (Eu tive 40,5 de febre naquele dia.)

7. 18. Comprei um gigapet. Um pato. Alimentei-o por um tempo. Dei banho nele. Esqueci dele por quase dois meses. Um dia, enquanto fazia uma limpeza, encontrei uma corrente e no final dela ele estava. Ei, patinho. Doidão, saltitando com sua cabeleira, gritando linhas diagonais. Vestindo um smoking.

4. Um Exemplo Para Te Ajudar a Crescer



Diga em voz alta cada uma das partes da conversa usadas abaixo.

```
require 'net/http'  
Net::HTTP.start( 'www.ruby-lang.org', 80 ) do |http|  
  print( http.get( '/en/LICENSE.txt' ).body )  
end
```

A primeira linha chama um método. O método chamado `require` é usado. Uma *string* é passada ao método contendo '`net/http`'. Pense nesta primeira linha de código como uma sentença. Nós dissemos ao Ruby para carregar um código auxiliar, a biblioteca `Net::HTTP`.

As próximas três linhas vão todas juntas. A constante `Net::HTTP` refere à biblioteca que carregamos acima. Nós estamos usando o método `start` da biblioteca. Dentro do método, estamos enviando uma *string* '`www.ruby-`

`lang.org`' e o número 80.

A palavra `do` abre um *block* (*bloco*). O bloco tem *uma variável de bloco* `http`. Dentro do bloco, o *método print* é chamado. O que está sendo impresso?

A partir da *variável http*, o *método get* é chamado. Dentro do `get`, nós passamos uma *string* contendo o caminho `'/en/LICENSE.txt'`. Agora, repare que outro método está ligado ao `get`. O *método body* (*corpo*). Então, o bloco se fecha com `end`.

Está indo bem? Só por curiosidade, você consegue adivinhar o que este exemplo faz? Espero eu, que você já esteja vendo alguns padrões no Ruby. Se não, balance sua cabeça vigorosamente enquanto você põe estes exemplos na sua mente. O código deve se quebrar em pedaços gerenciáveis.

Por exemplo, este padrão é usando um monte de vezes:

```
variavel . metodo ( argumentos do metodo )
```

Você vê isso dentro do:

```
http.get( '/en/LICENSE.txt' )
```

Estamos usando o Ruby para pegar uma página da web. Você provavelmente já usou HTTP com o seu navegador. HTTP é o Hypertext Transfer Protocol. HTTP é usado para se transferir páginas pela internet. Conceitualize um motorista de ônibus que pode dirigir por toda a internet e trazer de volta páginas para gente. No quepe dele estão estampadas as letras HTTP.

A variável `http` é aquele motorista. O *método* é uma mensagem para o motorista. Vá `get (pegar)` a página da web chamada `/en/LICENSE.txt`.

Então quando você vê esta cadeia de métodos:

```
http.get( '/en/LICENSE.txt' ).body
```

Já que estaremos recebendo a página de volta pelo motorista `http`, você pode ler isto na sua cabeça assim:

```
página web .body
```

E neste pedaço de código:

```
print( http.get( '/en/LICENSE.txt' ).body )
```

Este código pega a página. Nós mandamos uma mensagem: `body` para a página, que por sua vez nos dá HTML em uma *string*. Nós então *imprimimos* esta string. Veja como o básico ponto-método acontece em cadeia. No próximo capítulo vamos explorar todos estes tipos de padrões no Ruby. Vai ser boa diversão.

Então, o que este código faz? Ele imprime o HTML da página web do Ruby na tela. Usando um motorista de ônibus habilitado para web.

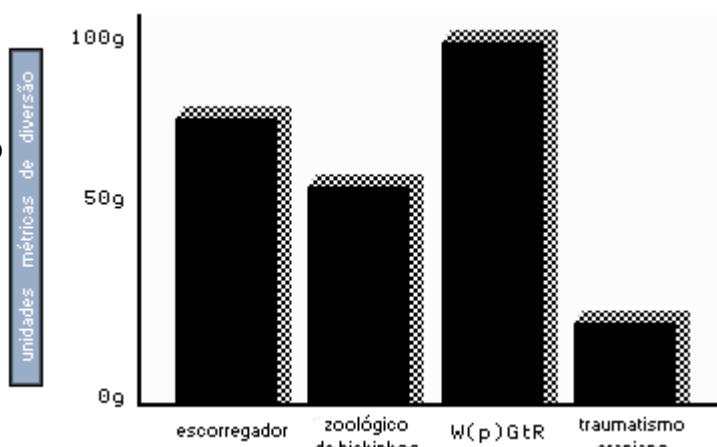
5. E Então, A Rápida Viagem Chega a Uma Calma, Amortecida Parada



Então agora nós temos um problema. Eu começo a achar que você está gostando muito disso. E você ainda nem

chegou no capítulo em que eu uso músicas de pular-corda para lhe ajudar a aprender como trabalhar com XML! Se você já está gostando disso, então as coisas estão indo realmente mal. Daqui a dois capítulos você já estará escrevendo seus próprios programas Ruby. Na verdade, é mais ou menos por ali que lhe farei começar a escrever seu próprio jogo de RPG, sua própria rede de compartilhamento de arquivos (à la BitTorrent), assim como um programa que vai baixar números genuinamente aleatórios da Internet.

E você sabe (você tem que saber!) que isso vai virar uma obsessão. Primeiro, você se esquecerá completamente de levar o cachorro para passear. Ele vai estar na portinhola, balançando a cabeça, enquanto seus olhos devoram o código, enquanto seus dedos transmitem mensagens ao computador.



Graças à sua negligência, as coisas vão começar a quebrar. Seu amontoado de folhas de código impressas cobrirá as entradas de ar. Sua fornalha se engasgará. O lixo irá se amontoar: caixas de comida delivery que você apressadamente pediu, correspondência inútil que você não se preocupou em jogar fora. Sua própria imundice poluirá o ar. Limo infestará o forro, a água irá endurecer, animais se convidarão para entrar, árvores virão pelas fundações.

Mas seu computador será bem tratado. E você, Smotchkiss, o terá nutrido com o seu conhecimento. Nas eras que você terá passado com sua máquina, você se tornará parte-CPU. E ela se tornará parte-carne. Seus braços irão de encontro às portas dela. Seus olhos aceitarão o vídeo direto pelo conector DVI-24. Seus pulmões ficarão sentados em cima do processador, esfriando ela.

E então, na hora que o quarto está prestes a se fechar em torno de você, como se todo o crescimento engolisse você e sua máquina, terminará seu script. Você e a máquina juntos rodarão este recém criado script Ruby, o produto da sua obsessão. E o script disparará serras-elétricas para cortar as árvores, corações para aquecer e regular a casa. Nanorobôs construtores sairão do seu script, reconstruindo seus aposentos, bricolando, renovando, colorindo, polindo, desinfetando. Poderosos andróides forçarão sua decadente casa em uma firme, rígida arquitetura. Grandes pilares emergirão, estátuas esculpidas. Você terá domínio sob este estado palacial e as montanhas e ilhas adjacentes à sua fortaleza.

Bom, então eu acho que você vai ficar bem. Quê se me diz? Vamos começar esse seu script?

Pequenas Folhas de Código Flutuando



Tirinha: No fundo, do muito escuro e vil coração de Wyvrnn, passava o rio Selis, mascarando as cortes dos Grifos de Mal Abochny, sobre faixas de florestas selvagens, saturadas com os corpos de gigantes derrubados no 6º apocalipse, escavados e enterrados por unicórnios com chifres-brocas (e do vazio destes corpos nasciam mais unicórnios, preeenchendo as frágeis extremidades e forçando os gigantes sem vida a andar!), lá embaixo, embaixo da vegetação coberta por musgo, embaixo da verdadeira terra, no real, real, real (você sente o real? Em caso afirmativo, gostaria de saber), cavernas secretas em operação, pedras vetrificadas, 1400 pés quadrados com armários apertados, valorizados para o movimento, as cavernas de Ambrose, tudo na companhia de... O gnomo com um presunto de estimação!

Eu tenho um monte de animais legal!

todos os seus aniamis de estimação acabaram de partir em um ônibus!



Eu nunca vi o presunto fazer coisa alguma, a não ser mijar. Hoje, nosso negócio nas Cavernas Ambrose é com o gnomo. Ele é uma parte crucial das próximas lições. Vamos todos fazer com que ele se sinta bem-vindo. Comece a esquentar os neurônios! (E por favor, troque essa ridícula calça de caubói.)

Atenção: esta lição é mais vagarosa. Acompanhe-a. Isto será uma longa e profunda respiração. A fase mais crucial de sua instrução. Pode parecer que você não está aprendendo muito código a princípio. Você estará aprendendo conceitos. Ao final deste capítulo, você saberá a beleza do Ruby. O conforto do código irá se tornar um saco de dormir para o seu próprio alívio.

1. A Folha como um Símbolo de Status na Ambrose

Tudo certo, Gnomo. Dê-nos um rápido resumo dos problemas financeiros que você enfrentou lá no seu reino.



Tirinha: Bem, nós uma vez usamos cristais azuis (como dinheiro). Mas é realmente muito escuro aqui nas cavernas para ver se algo é azul. Sim, mas você pode guardar dinheiro na sua sunga de natação sem se preocupar. de qualquer modo, todos os insetos e besouros usam folhas como dinheiro. e eles não carregam cristais azuis porque eles são muito pesados, então nós trocamos.

Realmente, não é o caminho que eu me lembro. Esse Gnomo estava me bipando constantemente. Quando me recusei a chamá-lo de volta, ele deixou uma mensagem no meu pager. Significado: ele bipou duas vezes e então exibiu uma pequena mensagem. A mensagem disse alguma coisa sobre, “Desça aqui, rápido!” e também, “Nós temos que livrar a terra destas pestes de lagartas empreendedoras, estes insetos insanos Vikings estão sufocando meus cristais azuis!”

Ultimamente, a taxa de câmbio já se estabelece entre as folhas e cristais. Uma nota de árvore velha vale cinco cristais. Então a situação monetária básica é semelhante a isto:

```
cristal_azul = 1
folha_macia = 5
```

Este exemplo é *tão* último capítulo. Ainda assim. Ele é um início. Nós estamos definindo duas *variáveis*. O **sinal de igual** é usado para *atribuição*.

Agora **folha_macia** representa o número **5** (tal como em: cinco cristais azuis.) Este conceito, até aqui, é **metade de Ruby**. Nós estamos *definindo*. Nós estamos *criando*. Isso é a metade do trabalho. Atribuição é a forma mais básica de definição.

Você não pode se queixar, pode Gnomo? Você construiu um império investindo seus cristais azuis no novo mercado livre entre as criaturas da floresta. (E apesar de ele ser um gnomo para nós, ele é um monstro grande para eles.)



Tirinha: Sem dúvida, por quê.... Smotchkiss pegou este caminho. Bem-vindo aos laboratórios subterrâneos da Animal Perfeito, LLC. Eu investi todos os cristais que tinha na Animal Perfeito. Quer me ver fazendo um macacoestrela?

Agora Você irá Escutar a Declaração da Animal Perfeito Porque Isto É Um Livro E Nós Temos Tempo e Nenhuma Pressa, Certo?

Há muito tempo atrás, muito, muito antes das lanchas motorizadas eu possuía uma égua de corrida premiada que levou um tropeção na pista. Ela capotou dez vezes e bateu num cara que carregava um vidro cheio de maionese. Nós tínhamos sangue e maionese para tudo quanto é lado da pista. Desnecessário dizer, ela era um desastre.

O veterinário deu uma olhada e jurou que ela nunca voltaria a andar. Tinha perdido as pernas e o veterinário não permitiria que uma égua sem pernas ficasse apenas sentada por aí. Nós precisamos abatê-la. Ele jurou por sua vida e por sua carreira nisto, insistindo que nos dividíssemos em duas linhas paralelas. As pessoas que não podiam refutar as afirmações do médico de um lado; aqueles muito cabeça-duras para aceitar seu raciocínio médico infalível do outro. O Gnomo, seu animal de estimação de presunto e eu éramos os únicos nesta segunda linha.

Então enquanto os outros empilhavam troféus e guirlandas de flores em volta da égua, despedindo-se com carinho antes que a bala viesse e a levasse para casa, o Gnomo e eu freneticamente vasculhamos a Internet por respostas. Nós lidamos com isso com nossas próprias mãos, cauterizando as feridas em sua perna com camarões de água doce vivos. E funcionou bem! Nós tínhamos uma égua de novo. Ou pelo menos: um corpo de égua com um abdominal de crustáceos congelantes.

Ela correu para todo lado depois disso e viveu por anos em cavernas úmidas no subsolo.

A Animal Perfeito é agora o futuro do melhoramento de animais. Eles criam novos animais e salvam animais velhos por partes. É claro, levou um longo tempo para isso. Quando o Animal Perfeito

Os Comedores de Cachecóis

Eu odeio me intrometer no seu aprendizado, mas eu já fiz isso tanto que eu não me importo mais. Posso mostrar meu próximo projeto para você?

Eu prometi escrever outro livro. (*Trombones*.) A boa notícia é que na realidade eu não vou escrever nada. Você não vai ter que agüentar ainda mais essas tolices idiotas.

Está tudo acabado entre eu e as palavras. Eu adoraria ficar e explorar cada uma, uma após a outra, mas está tudo ficando muito previsível, você não acha? Eventualmente, elas todas serão usadas e eu terei de inventar palavras falsas e isso seria muito bizelegarro.

Agora. Ainda não é nada certo, mas já estou em negociações com a Anna Quindlen para escrever no meu lugar. Nós estamos juntando esforços em um livro que vai arrancar o (Comovente) Guia das suas mãos.

Simplificando, o Guia será inútil. Você não será capaz de equilibrar nenhuma Romã em cima da coisa.

Esse novo livro. Os Comedores de Cachecóis. É um romance maduro. Mas também é um guia para iniciantes de Macromedia Flash. É como se Judy Blume encontrasse Playstation. Ou como Osil8 estrelando Hillary Duff.

Eu não quero entregar nada sobre a história, mas para provocar o seu apetite eu posso dizer isso: um garoto fala com o seu irmão morto em

começou, você via um urso crescido adentrar o Animal Perfeito e via um urso crescido de óculos de sol sair. Completamente cafona.

Fique por aí e você verá um caranguejo com *sua própria mochila foguete* Esse é um novo modelo caranguejet 2004.

Mas agora, a operação está a todo vapor. E a limpeza do lugar é impressionante. Todo o equipamento é tão brilhoso. Tudo é cromado. Ah, e toda a equipe tem armas escondidas. Eles são treinados para matar qualquer um que entre sem ser anunciado. Ou, se eles ficarem sem balas, eles são treinados para acertar com as armas qualquer um que entre sem ser anunciado.

Gnomo, me faça uma macacoestrela.

ActionScript. Mais por vir.



Um pouco de Ruby imaginário para você:

```
tubo.pegue_uma_estrela
```

Variável `tubo`. Método `pegue_uma_estrela`. Muitos Rubistas gostam de pensar em métodos como mensagens. A quem vier antes do método é dada a mensagem. O código acima diz ao `tubo`: `pegue_uma_estrela`.

Esta é a **segunda metade** do Ruby. Colocar as coisas em movimento. Estas coisas que você define e cria na primeira metade começam a *agir* na segunda metade.

1. Definindo coisas.
2. Colocando essas coisas em ação.

Então e se o código de pegar estrelas funcionar? Pra onde a estrela vai?

```
estrela_presa = tubo.pegue_uma_estrela
```

Veja, você que deve coletar a coitada da estrelinha. Se não o fizer, ela vai simplesmente desaparecer. Toda vez que você usa um método, você recebe algo de volta. Você pode ignorá-lo ou usá-lo.

Se você conseguir aprender a usar as respostas que os métodos lhe dão, então você irá dominar.



Tirinha: Preste atenção! Macaco & Estrela! Então, uma catraca dentro da máquina apenas ajusta a estrela na cara do macaco. Gira, gira e gira até que eles fiquem juntos.

Rapidamente então.

```
macacoestrela = ajuste.conecta( macaco_preso, estrela_presa )
```

O `ajuste` recebe uma mensagem `conecta`. O que precisa ser conectado? Os *argumentos do método*: o `macaco_preso` e a `estrela_presa`. Nós recebemos de volta um `macacoestrela`, em quem decidimos nos pendurar.



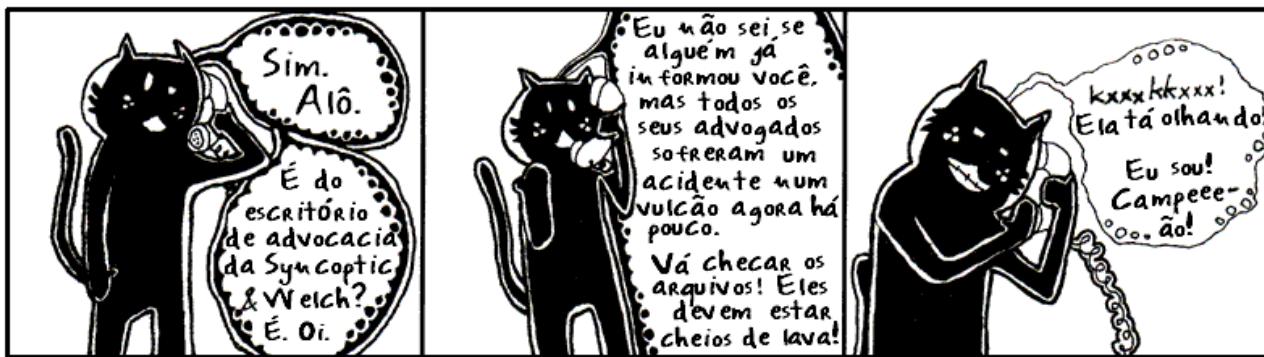
Este tem se mostrado um programinha tão curto e pequeno que eu vou apenas colocá-lo todo em uma única declaração.

```
macacoestrela = ajuste.conecta( macaco_preso, tubo.pegue_uma_estrela )
+ sapo_de_mao_decorativo
```

Vê como o `tubo.pegue_uma_estrela` está nos argumentos do método? A estrela pega será passada diretamente ao ajuste. Não é necessário encontrar um lugar para colocá-la. Apenas deixe-a ir.

2. Pequeno e Quase Sem Valor

NA SALA DE DESCANSO DA CAVERNA AMBRÓSIA...



O hotel aqui em Ambrose não é nem um pouco bom. As camas são todas tortas. O elevador é minúsculo. Um cara pôs toda sua bagagem no elevador e não sobrou mais lugar para ele. Ele apertou o botão e as perseguiu pelas escadas. Mas as escadas eram muito estreitas e ele ficou entalado tentando subir.

Os mini-sabonetes que eles te dão são dimensionados para gnomos, então é impossível fazer uma espuma. Eu odeio isso. Toda hora eu confundo eles com lentes de contato.

Eu abri a torneira e nada saiu. O caso é o seguinte: Ambrose é um lugar com propriedades mágicas, então eu me arrisquei. Coloquei minha mão embaixo da torneira. Invisível, umidade morna. Eu senti a sensação de água corrente, passando pelos meus dedos. Quando tirei minha mão de lá, elas estavam secas e limpas.

Foi uma incrível experiência de vazio total. Foi como `nil`.

Nil (Nada)

No Ruby, o `nil` representa o vazio. Ele representa **falta de valor**. Ele não é zero. Zero é um número.

É um morto-vivo do Ruby, uma palavra chave falecida. Você não pode adicionar à ele, ele não evolui. Mas ele é terrivelmente popular. É este esqueleto sorrindo nas fotos.

```
copo_plastico = nil
```

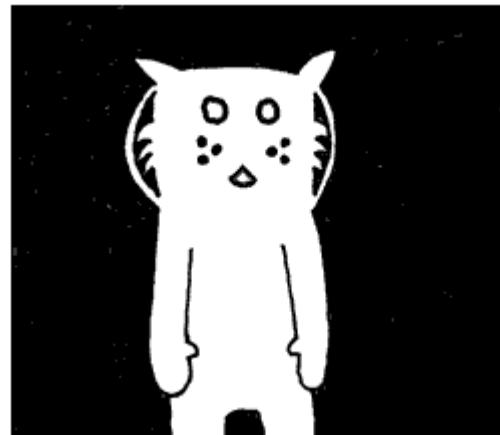
No código acima `copo_plastico` está **vazio**. Você pode argumentar que o `copo_plastico` contém algo, um `nil`. O `nil` representa vazio, logo, então, adiante-se e chame-o de vazio.

Alguns de vocês que já programaram antes estarão tentados a dizer que `copo_plastico` está **indefinido**

(undefined). Que tal não. Quando você diz que uma variável está indefinida, você está dizendo que o Ruby simplesmente não tem lembrança da variável, ele não a conhece, ela absolutamente não existe.

Mas o Ruby está sabendo do `copo_plastico`. Ruby pode facilmente olhar dentro do `copo_plastico`. Está **vazio**, mas não está **indefinido**.

False (Falso)



O gato Blix Negociador. Congelado no vazio. Rígido e refinado bigode. Serenos olhos de lago. Rabo de morno sincelo. Patrocinado por um Botão de Pausa Bastante Poderoso.

A escuridão que circunda o Blix pode ser chamada de **espaço negativo**. Atente-se nessa frase. Deixe ela sugerir que o vazio tem conotação negativa. De um jeito similar, `nil` tem uma nota levemente amarga que ele assobia.

Generalizando, **tudo no Ruby tem uma carga positiva**. Esta fagulha corre por strings, números, regexps (expressões regulares), todos eles. Somente duas palavras reservadas vestem uma capa sombria: `nil` e `false` nos trazendo para baixo.

Você pode **testar essa carga** com a palavra chave `if` (se). Ela se parece muito com o bloco `do` que vimos no último capítulo, no qual ambas terminam com um `end`.

```
if copo_plastico
  print "O copo de plástico está transbordando!"
end
```

Se `copo_plastico` contiver `nil` ou `false`, você não vai ver nada impresso na tela. Eles não estão na lista de convidados do `if`. Então o `if` não vai rodar nada do código que ele está protegendo.

Mas `nil` e `false` não precisam ir embora constrangidos. O caráter deles pode ser meio questionável, mas o `unless` (ao menos) gerencia um estabelecimento que preza ser fuleiro. A palavra chave `unless` tem por regra só deixar aqueles com carga negativa entrar. Aqueles são: `nil` e `false`.

```
unless copo_plastico
  print "O copo plástico está vazio."
end
```

Você também pode usar `if` e `unless` no **fim de uma linha de código**, se apenas aquela linha estiver sendo protegida.

```
print "Yeah, o copo plástico está cheio de novo!" if copo_plastico
print "Dificilmente. Ele está vazio." unless copo_plastico
```

E mais uma manha: empilhar o `if` e `unless`.

```
print "Estamos usando copos de plástico pois não temos de vidro." if copo_plastico unless
copo_vidro
```

Este truque é um lindo jeito de expressar: *Faça isso só se a for verdadeiro e b não for verdadeiro*.

Agora que você já conhece o `false`, tenho certeza que pode ver o que vem a seguir.

True (Verdadeiro)

`sujeito_aproximando = true`

Eu vi o `true` no Buffet do hotel hoje. Eu não suporto aquele cara. Ele aparece demais. E você nunca encontrou alguém que plantou os pés com tanta força no solo. Ele usa esse colar brega feito de conchas. Sua face exibe uma confiança insolente. (Você pode dizer que ele está exercendo toda sua repressão só para evitar de explodir num vôo do Neo.)

Para ser honesto, não consigo ficar do lado de alguém que sempre tem de estar certo. Esse `true` está sempre dizendo, “A-OK.” Chacoalhando as duas mãos. E é sério, ele ama aquele colar. Ele o usa constantemente.

Como suspeitava, ele está por trás dos bastidores para toda a agenda de eventos do `if`.

```
print "Hugo Boss" if true se comporta  
como print "Hugo Boss".
```

Ocasionalmente, o `if` vai puxar as cordas vermelhas para exercer algum controle de massas. O **par de iguais** dá a aparência de uma passagem, como cordas dos lados de um carpete vermelho onde só é permitida a passagem de `true`.

```
if sujeito_aproximando == true  
  print "Esse colar é clássico."  
end
```

O par de iguais é simplesmente **uma checagem de identidade**. Os cavalheiros das pontas opostas desta corda parecem ser iguais?

Desse jeito, você controla quem o `if` deixa entrar. Se você não suporta o `true` assim como eu, receba de peito aberto o `false` (falso).

```
if sujeito_aproximando == false  
  print "Chega mais, seu diabo conivente."  
end
```

Mesma coisa para `unless`. O portão é seu. Tome o controle dele.

Novamente, Eu Quero Que Você Domine

Agora, está afim de uma viagem mental? **O sinal de dois iguais é um método**. Você consegue adivinhar como ele funciona? Aqui, dê uma olhada nele com ponto e parênteses:

`sujeito_aproximando.==(true)`

Ruby permite o atalho, sem problemas. Abandone o ponto e recue, vagarosamente.

Agora, você se lembra o que deve fazer para **dominar** no Ruby? *Use as respostas que os métodos lhe dão.*

```
if nil.==( true )  
  print "Isto nunca vai acontecer."
```

Faça Seu Próprio Macacoestrela!

1. Vire uma caneca de cabeça para baixo.

2. Prenda uma maçã com a ajuda de um



elástico.



3. Enfie as chaves de carro dos lados da maçã.



4. Cole a face de estrela.

end

No exemplo acima, como a resposta do método esta sendo usada?

Pegue a afirmação `nil == true`. Isso falhará o tempo todo. Não são iguais. Quando não há igualdade, o método do duplo igual responde com `false`. Uma balançada de cabeça. Esta resposta é dada ao `if`, que não pode aceitar um `false`. O `print` nunca acontecerá.

```
no_hotel = true
email = if no_hotel
    "why@hotelambrose.com"
else
    "why@drnhowardcham.com"
end
```

Mesmo que `if` não seja um método, o `if` retorna uma resposta. Observe o exemplo acima e imagine o que acontece quando `no_hotel` é `true`.

O `if` retornará a resposta dada pelo código que ele decidir rodar. No caso de `no_hotel` ser `true`, a primeira string, meu endereço de e-mail no Hotel Ambrose, será retornada. A palavra reservada `else` marca o código a ser rodado, uma vez que o `if` falhe. Se `no_hotel` é `false`, o `if` responderá com meu endereço de e-mail do escritório do Dr. N. Howard Cham, onde eu recebo meu aprendizado.



Você tem mais duas faces de estrela complementares esperando na sua conta.



A padrão, calma.



A comendo giz.

Você deve ter várias linhas de código dentro de um `if` ou `unless`, mas somente a resposta oriunda da última afirmação completa será usada.

```
email = if no_hotel
    endereço = "why"
    endereço << "@hotelambrose"
    endereço << ".com"
end
```

Três linhas de código dentro do **if**. A primeira linha atribui uma string contendo meu nome a uma variável. A segunda e terceira linha adicionam o resto do meu endereço de e-mail no fim. O **menor que duplo << é o operador da concatenação**. Concatenar é o mesmo que **apender**, ou **adicionar ao fim**.

Assim como vimos com o checador de igualdade `==`, o concatenador é um método. Após acrescentar ao fim da string, o concatenador ainda **responde com aquela mesma string**. Então, a terceira linha, que pode ser lida como `endereco.<<(" .com")`, retorna `endereco`, que o **if** então devolve para a atribuição `email`.

Uma pergunta: E se o **if** falhar? E se `no_hotel` for falso no exemplo acima? Alguma coisa será retornada? Nada é atribuído a `email`, certo?

Sim, nada é retornado. Que significa que: `nil` é retornado. E, muitas vezes, `nil` é uma resposta bastante útil.

```
print( if no_hotel.nil?
      "Sem pistas se ele está no hotel."
    elsif no_hotel == true
      "Definitivamente sim."
    elsif no_hotel == false
      "Ele saiu."
    else
      "O sistema está traaaaavado"
    end )
```

Você pode usar o método `nil?` em qualquer valor no Ruby. Novamente, pense nele como uma mensagem. Para o valor: “Você é nil? Você está vazio?”

Se `no_hotel` estiver vazia, o Ruby não tem idéia se eu estou no hotel ou não. Então o **if** responde com a string “Sem pistas”. Para tratar as possibilidades de `true` ou `false`, a palavra reservada `elsif` é usada. Enquanto você pode ter apenas um `if` e um `else`, você pode encher as entradas com um exorbitante número de palavras chave `elsif`. Cada `elsif` age como **um outro teste if**. Checando por uma carga positiva.

Se você está indo bem até este ponto, então está em boa forma para o restante do livro. Você viu um código bem difícil nos últimos exemplos. Forte companheiro.

3. Encadeando Desilusões

Você termina de ler a tirinha acima e se retira para seu sofá-cama para reflexão. É um daqueles assuntos encobertos que são sempre obstruídos por travesseiros. Você se senta sobre a pilha, observando o mundo lá do alto. Você vê as altas chaminés expelindo largos cilindros de fumaça e névoa. Nas complicadas junções das auto-estradas, um trânsito ligeiro, nada é cintilante senão um músculo ocular pulsando do seu ponto de vista superior.



É tudo tão fantástico. Como as cores do horizonte se espalham através da paisagem como uma grande mistura de manteiga e gordura com uma colher de sopa de extrato de baunilha.

Mesmo com toda a beleza que lhe chama a atenção, as imagens do Gnomo e sua Esperança Olímpica retornam. E especialmente, aquele pedido por 55.000 macacoestrelas. *55.000 macacoestrelas*, pensa você *Cinqüenta-e-cinco Mil*.

Você pensa somente no próprio número. *55.000* Ele está descendo uma rua. Ele pode estar numa floresta, você não sabe ao certo a medida que seus olhos estão fixos nos próprios números. Está parando e falando com pessoas. Com jogadores de tênis, com um grupo de coral masculino. Há alegria e uma boa sensação. Quando ele ri, os seus zeros agitam-se com alegria.

Você quer falar com ele. Você quer saltitar por aquele caminho da floresta com ele. Você quer embarcar num jato com destino ao Brasil com ele. E depois de cinco dias e quatro noites no prazeroso Spa & Resort Marriot da Costa do Sauípe, quererá casar com ele, para sustentar uma família de 55.000 macacoestrelas com ele. Com uma voadora, você derruba sua pilha de travesseiros de isolamento. Mexendo com a chave, você destrava a gaveta de cima de sua escrivaninha e puxa uma folha de papel, segurando-a firmemente sobre a escrivaninha. Você começa a escrevinhar.

Tome posse da Nigéria com meus 55.000 macacoestrelas...

*Com ela, construa um casino **somente para vegetarianos** casino e uma arena de kart...*

Asas... nós poderíamos ter nosso próprio molho especial nelas, que seja diferente...

Mostarda + codeína = O Molho Estrelado Macacoestrelado Brilhante do Smotchkiss...

Franquia, franquia... logos...

Vídeos de instrução para funcionários...

Quando você dá o troco para o consumidor, deixe que ele alcance o sapo que está na sua mão para pegá-lo...

Se ele não tiver troco, pelo menos ponha a nota dele em um lugar onde ele tenha que tocar no sapo...

Nós estamos passando este campo para um próximo nível...

Faça propaganda de pizza barata, vamos lucrar com o refrigerante...

Colecione todos os 4 copos congelados...

Uau, as idéias estão mesmo saindo. Você teve que literalmente se beijar para parar. Nós precisamos colocá-las num lugar seguro. Na verdade, deveríamos armazená-las no seu computador e deformar as palavras. Mantenha os olhos na janela e vigie para ver se o FBI não vem. Eu vou começar esse script.

O Script Invertedor

```
print "Digite e seja diabólico: "
ideia_contrario = gets.reverse
```

Deixe este script ser seu confidente. Ele perguntará por planos malignos e reverterá suas letras de trás para frente. O método **gets** é **construído dentro de Ruby**. É um **método de kernel** como o **print**. Este método **gets** pausará o Ruby para deixar que você digite. Quando você apertar o *Enter*, **gets** irá então parar de prestar atenção aos seus esmurros no teclado e responderá de volta ao Ruby com uma String contendo tudo o que você digitou.

O método **reverse** é então usado na String que o **gets** está devolvendo. O método **reverse** é parte da classe **String**. O que significa que **tudo que for uma string tem o método reverse disponível**. Mais sobre classes no próximo capítulo, por enquanto saiba apenas que **alguns métodos só estão disponíveis para certos tipos de valores**.

Eu não acho que o **reverse** dará muito certo. As autoridades só precisam pôr um espelho na frente de “airégiN ad elortnoc o emoT.” Prenderão a gente quando os macacoestrelas começarem a pousar em Lagos.

Saia na frente com O Colete do Tigre

As letras maiúsculas estão nos entregando. Talvez se passássemos todas as letras na string para maiúsculas antes de inverter.

```
ideia_ao_contrario = gets.upcase.reverse
```

Sua Repetitividade Compensa

Você me entrega um bloco de notas, cheio de garranchos ilegíveis. Analisando, começo a notar padrões. Notar que você parece usar as mesmas palavras repetidamente em suas reflexões. Palavras como *macacoestrela*, *Nigéria*, *bomba*. Algumas frases até. *Ponha um fim nisso*. Que são ditas toda hora.

Vamos disfarçar esses termos bobos, meu irmão. Vamos ofuscá-los de olhos ardentes que choram para saber nossos delicados planos e para nos demover de termos grande prazer e muitos karts. Vamos trocá-los pela mais inocente linguagem. Novas palavras com significado secreto.



Quer começar a usar o Ruby junto com a leitura? Divida sua atenção e veja o [Pacote de Expansão I: O Colete do Tigre](#), um mini-capítulo usual que vai te ajudar a instalar o Ruby. E ainda, aprender como usar Irb e Ri, dois modos de ajuda que vêm com o Ruby e irão agilizar seu aprendizado.

Eu comecei uma lista de palavras, um **Hash** do Ruby, que contém estas suas palavras tão freqüentes e perigosas. No Hash, cada palavra perigosa é comparada contra uma palavra código (ou frase). A palavra código será então trocada pela palavra real.

```
palavras_codigo = {  
  'macacosestrela' => 'Phil e Pete, aqueles chanceleres de pavio curto do Novo Reich',  
  'catapulta' => 'chucky go-go', 'bomba' => 'Vida Assistida por Calor',  
  'Nigeria' => "Ny e Jerry's Lavagem a Seco (com Donuts)",  
  'Ponha um fim nisso' => 'Ponha um fio nisso'  
}
```

As palavras que são colocadas antes da seta são chamadas **chaves**. As palavras depois das setas, as definições, geralmente são simplesmente chamadas de **valores**.

Note as aspas duplas em volta de **Ny e Jerry's Lavagem a Seco (com Donuts)**. Já que nas aspas está sendo usado um apóstrofo (aspas simples), nós não podemos usar aspas simples em torno da string. (Todavia, você pode usar aspas simples se colocar uma barra invertida antes do apóstrofo como em: **'Ny e Jerry\'s Lavagem a Seco (com Donuts)'**.)

Se você precisar procurar por uma palavra em específico, pode usar o método dos **colchetes**.

```
palavras_codigo['catapulta']
```

 irá responder com a string '**chucky go-go**'.

Olhe os colchetes como se fossem paletas de madeira nas quais as palavras estão em cima. Uma empilhadeira poderia usar suas pás em cada lado da palete e a trazer da prateleira de volta ao depósito. A palavra na palete é chamada de **indexador**. Estamos pedindo para a empilhadeira achar o indexador para a gente e nos trazer de volta o valor correspondente.

Se você nunca esteve num depósito, você pode ver as chaves como alças. Imagine um trabalhador industrial colocando suas luvas e levando o índice de volta à sua custódia. Se você nunca usou uma alça, vou lhe dar cerca de trinta segundos para achar uma e usar antes que eu estoure meus miolos.

Assim como os vários operadores que você viu recentemente, os colchetes indexadores são simplesmente um atalho para um método.

```
palavras_codigo.[]( 'catapulta' )
```

 responderá com a string '**chucky go-go**'.

Fazendo a Troca

Eu me adiantei e salvei o Hash de código em um arquivo chamado **listadepalavras.rb**.

```
require 'listadepalavras'

# Pegar a idéia do mal e converter em palavra código
print "Entre com sua nova idéia: "
ideia = gets
palavras_codigo.each do |real, codigo|
  ideia.gsub!( real, codigo )
end

# Salva o papo-furado num novo arquivo
print "Arquivo codificado. Favor entrar com um nome para essa idéia: "
nome_ideia = gets.strip
File::open( "ideia-" + nome_ideia + ".txt", "w" ) do |f|
  f << ideia
end
```

O Script começa puxando nossa lista de palavras. Assim como **gets** e **print**, o método **require** é um método do kernel, você pode usá-lo em qualquer lugar. Eu dei a ele a string '**'listadepalavras'**' e ele vai procurar um arquivo chamado **listadepalavras.rb**.

Depois disso, há duas seções. Estou marcando estas seções com comentários, as linhas que começam com o símbolo **cerquilha (jogo-da-velha) #**. Comentários são **notas úteis** que acompanham seu código. Colegas que vierem ver seu código vão apreciar tal ajuda. Você mesmo, quando olhar para seu código novamente depois de algum tempo, comentários vão te ajudar a retomar o fio da meada. E ainda existe softwares por aí que pegam seus comentários e fazem documentação à partir deles. (RDoc e Ri — veja o Pacote de Expansão #1!)

Eu gosto de comentários porque posso passar os olhos por uma grande pilha de código e identificar pontos importantes.

Como os comentários nos dizem, a primeira seção pergunta por uma idéia maléfica e a troca por uma palavra código. A segunda seção salva a idéia codificada num novo arquivo de texto.

```
palavras_codigo.each do |real, codigo|
  ideia.gsub!( real, codigo )
end
```

Você viu o método **each**? O método **each** está em todo lugar no Ruby. Está disponível para Arrays, Hashes, até Strings. Aqui, nosso dicionário **palavras_chave** é mantido numa Hash. Este método **each** corre entre **todos os pares da Hash**, cada palavra perigosa é comparada com sua palavra código, mandando cada par para o método **gsub!** para a troca atual.

No Ruby, **gsub** é abreviação de *global substitution* (substituição global). O método é usado para procurar e substituir. Aqui, queremos achar todas as ocorrências de palavras perigosas e substitui-las com a segura palavra código. Com **gsub**, você provê a **palavra a ser achada como primeiro argumento**, depois a **palavra que tomará o lugar como segundo argumento**.

Por que nós não estamos aguardando a resposta do **gsub**? O **gsub** não nos dá um resposta que devemos guardar? Você acha que a linha deveria ser assim:

```
ideia_segura = ideia.gsub( real, codigo )
```

Sim, com **gsub** nós temos que ficar com a resposta. Estamos usando uma variação do **gsub** que é totalmente hiper. Notou a **exclamação** no **gsub!** usada dentro de cada bloco **each?** A exclamação é um sinal de que **gsub!** é meio radical. Veja, **gsub!** vai mais longe e **troca as palavras em ideia diretamente**. Quando ele termina **ideia** vai conter a nova string alterada e você não vai mais achar a string velha.

Pode chamar **gsub!** de **método destrutivo**. Ele faz suas modificações no valor diretamente. Enquanto **gsub** deixa o valor intacto, apenas respondendo com uma nova string que contém as alterações. (Por que o **gsub!** tem que gritar quando ataca sua presa? Assaltante cruel!)

Arquivos de Texto de um Louco

Vamos salvar a idéia codificada em um arquivo.

```

# Salva as palavras sem sentido em um novo arquivo
print "Arquivo codificado. Favor entrar com um nome para essa idéia: "
nome_ideia = gets.strip
File::open( 'ideia-' + nome_ideia + '.txt', 'w' ) do |f|
  f << ideia
end

```

Esta seção começa perguntando por um nome pelo qual a idéia possa ser chamada. Este nome será usado para se fazer o nome do arquivo quando salvarmos a idéia.

O método `strip` serve para todas as strings. Este método **corta espaços e linhas vazias do começo e do fim** de uma string. Isso removerá o *Enter* no fim da string que você digitou. E também removerá espaços caso você tenha acidentalmente deixado algum.

Depois que temos o nome da idéia, vamos abrir um novo arquivo de texto em branco. O nome do arquivo é construído adicionando-se strings. Se você digitou '`mostarda-mais-codeina`', nossa matemática será: '`ideia-' + 'mostarda-mais-codeina' + '.txt'`'. O Ruby pressiona isso em uma única string. '`ideia-mostarda-mais-codeina.txt`' é o arquivo.

Estamos usando o método de classe `File::open` para criar um novo arquivo. Até agora, nós usamos vários métodos do kernel para fazer nosso serviço. Nós damos uma string para o método `print` e ele a imprime na nossa tela. Um segredo sobre métodos do kernel como o `print`: eles são na verdade **métodos de classe**.

```
Kernel::print( "55.000 Macacos estrelas o Saúdam!" )
```

O que isso significa? Por que isso importa? Significa que o `Kernel` é o centro do universo Ruby. Sempre que você está no seu código, o `Kernel` estará do seu lado. Você nem precisa dizer `Kernel` pro Ruby. O Ruby sabe checar o `Kernel`.

A maioria dos métodos são mais especializados que `print` ou `gets`. Pegue `File::open` por exemplo. O criador do Ruby, Matz, nos deu vários métodos diferentes que lêem, renomeiam, ou deletam arquivos. Eles estão todos organizados dentro da classe `File`.

`File::read("ideia-mostarda-mais-codeina.txt")` responderá com uma string contendo todo o texto do seu arquivo de idéias. (Read significa ler.)

`File::rename("arquivo_antigo.txt", "arquivo_novo.txt")` renomeará o `arquivo_antigo.txt`.

`File::delete("arquivo_novo.txt")` explodirá o novo arquivo.

Esses métodos File estão todos **dentro do Ruby**. Estão apenas armazenados num contêiner chamado Classe `File`. Então, enquanto você pode seguramente chamar os métodos do kernel sem precisar digitar `Kernel`, o Ruby não checa automaticamente a classe `File`. Você precisa dizer o nome do método completo.

```

File::open( 'ideia-' + nome_ideia + '.txt', 'w' ) do |f|
  f << ideia
end

```

Nós passamos dois argumentos dentro de `File::open`. O primeiro é o **nome do arquivo para abrir**. O segundo é uma string contendo o **modo do arquivo**. Usamos '`w`', que significa escrever em um novo arquivo. (Outras opções são: '`r`' para ler o arquivo ou '`a`' para adicionar no fim do arquivo.)

O arquivo está aberto para escrita e nos é dado na forma da variável `f`, que pode ser vista **descendo a calha para dentro do nosso bloco**. Dentro do bloco, nós escrevemos no arquivo. Quando o bloco se fecha com `end`, nosso arquivo é fechado também.

Note que nós usamos o **concatenador** `<<` para escrever no arquivo. Podemos fazer isto porque arquivos têm um método chamado `<<`, assim como as Strings.

Acalme-se, Suas Idéias Não Estão Presas

Aqui, vamos pegar nossas idéia de volta em seus verbos originais, para que você possa ruminar sobre o brilhantismo delas.

```

require 'listadepalavras'

# Imprima cada idéia com as palavras corretas
Dir['ideia-*txt'].each do |nome_arquivo|
  ideia = File.read( nome_arquivo )
  palavras_codigo.each do |real, codigo|
    ideia.gsub!( codigo, real )
  end
  puts ideia
end

```

A esta altura, você já deve estar sabendo o que este exemplo faz. Não vou te incomodar com detalhes mundanos. Veja se você consegue entender sozinho como isso funciona.

Contudo, temos um método de classe interessante aqui. O método `Dir::[]` procura em um diretório (alguns de vocês podem chamá-lo de “pasta”). Assim como você viu com os Hashes, os colchetes indexadores podem ser métodos de Classe. (Você começa a ver o brilho e suntuosidade cintilante do Ruby?)

Então nós estamos usando a empilhadeira para pegar os arquivos no diretório que coincidem `'ideia-*txt'`. O `Dir::[]` irá usar o asterisco como um coringa. Estamos basicamente dizendo, “Encontre tudo que comece com `ideia-` e termine com `.txt`.” A empilhadeira vai para o diretório e volta com uma lista de arquivos encontrados.

Esta **lista de arquivos** vem na forma de **Array**, a Centopéia, com uma **String** para cada arquivo. Se você está curioso e quer brincar com `Dir::[]`, tente isto:

`p Dir['ideia-*txt']` irá imprimir:

`['ideia-mostarda-mais-codeina.txt']` (*um Array de nomes de arquivos!*)

Sim, o método `p` funciona igual `print`. Mas `print` é para se imprimir strings, enquanto `p` irá imprimir *qualquer coisa*. Veja isso.

`p File::methods` irá imprimir:

`["send", "display", "name", "exist?", "split", ... a lista completa dos nomes dos métodos!]`

4. O Milagre dos Blocos (Blocks)



Já que estamos ficando íntimos enquanto compartilhamos este tempo juntos, eu provavelmente deveria te falar um pouco mais sobre a história que aqui se passa. É uma boa hora para uma pausa, eu diria.

Primeiramente, você deve saber que o Blix é meu gato. Meu segundo animal de estimação é o Bigelow. E digo, nós dificilmente nos vemos mais. Ele é totalmente auto-suficiente. Eu não sei exatamente onde ele mora hoje em dia, ele não mora mais na antecâmara do meu quarto. Ele esvaziou sua conta bancária há cerca de sete meses.

Extraído de Os Comedores de Cachecóis

(do Capítulo V: *Um Homem de Uniforme*.)

Em Abril, os lírios verdes voltaram. Eles abriram suas asinhas de anjo, chegando ao mundo. Delicadamente, seus rebentos acariciaram os tristes postes da cerca até que eles cantassem alegremente.

Ele tem uma cópia da chave de casa e uma do Cadillac Seville. Se algum dia quiser voltar, eu alegremente deixarei de lado nossas diferenças e me entreterei novamente com suas travessuras em torno de casa.

Mas não se engane. Eu sinto falta dele por perto. Não sei se ele sente falta da minha companhia, mas eu sinto a dele.

Uma Sirene e Uma Oração

Eu vi o Blix pela primeira vez na televisão quando eu era um garoto. Ele estrelava um drama policial empoeirado chamado *Uma Sirene e Uma Oração*. O seriado era sobre um esquadrão policial religioso que fazia seu serviço, e o fazia bem, e tinham suas cotas de milagres nas ruas. Quero dizer, os oficiais no show eram ótimas pessoas, muito religiosos, praticamente um sacerdote. Mas, você sabe, até sacerdotes não têm o bom senso de matar um cara que tenha ido longe demais. Esses caras sabiam onde traçar a linha. E chegavam nessa linha todos os dias.

Então, aquilo era um seriado bem sangrento, mas eles sempre tinham uma boa lição de moral no final. Na maioria das vezes algo como, “Nossa, escapamos dessa bem rápido.” Há uma série camaradagem numa declaração como essa.

O show basicamente girava em torno deste policial. Dick Robinson “O Doidão”. As Pessoas o chamavam de “O Doidão” porque basicamente ele era insano. Não me lembro se isso foi diagnosticado clinicamente, mas as pessoas sempre questionavam suas decisões. O Doidão regularmente estourava e mordia alguns policiais, a maioria deles personagens de moral inquestionável. Mas todos sabemos que esse é um mundo cão, as apostas são altas, e todos que assistiam ao seriado tinham O Doidão com grande estima. Eu penso que todos no esquadrão cresceram bastante como pessoas, graças à determinação do Doidão.

Todavia, os policiais não conseguiam dar conta do serviço por si mesmos. Em cada um dos episódios eles imploravam por uma ajuda maior. E, em cada um dos episódios, eles conseguiam ajuda de um gato chamado Terry (encenado pelo meu gato Blix). Ele era apenas um gatinho na época, e como um garoto novo ávido por *Uma Sirene e Uma Oração*, me encontrei na busca do meu próprio gato-solucionador-de-crimes. Terry levava aqueles caras por túneis subterrâneos, marinas abandonadas, imensos galpões industriais.

Algumas vezes ele aparecia o episódio inteiro, indo e vindo, fazendo armadilhas e controlando o trânsito. Mas outras vezes você quase não o via no episódio. Então você reboinava e assistia e assistia e assistia. Você desistia. Ele não estava naquele episódio.

Ainda assim, você não desistia, você ia quadro por quadro com o jog do controle, analisando cada cena. E lá estava ele. No alto atrás do holofote que foi acionado com muita força. O que deixou o Doidão com danos permanentes nos olhos. Por quê? Por que queimar as retinas do seu companheiro, Terry?

Mas a pergunta nunca foi respondida pois a série foi cancelada. Eles começaram a fazer efeitos especiais com o gato e tudo foi por água abaixo. No último episódio, houve um momento em que Terry estava preso em cima de um guindaste, preste a cair na fornalha de uma fundição de ferro. Ele olha para trás. Não dá para voltar. Ele olha para baixo. Tapa os olhos com as patas (*sério!*), ele pula do

Da janela do seu quarto, Lara olhava os lírios exibirem sua feminilidade sólida. Ela colocou a franja de um cachecol carpatiano, bordado, novo, em sua boca e comeu vagarosamente. A malha longa desceu por sua garganta provocando cócegas enquanto serpenteava pelo esôfago. Ela riu e arrotou.

Oh, como a flora a atraía. Olhar para as flores combinava tanto com ser uma garota adolescente. Ela queria pintá-las, então abriu um novo modelo no Flash. Um filme vazio dessa vez.

Ela deixou seu cursor passear pelos jardins da área visível do filme. Linhas vetoriais brancas sob linhas vetoriais amarelas mais curtas. Ela selecionou as linhas brancas e as agrupou. Ela até moveu as linhas para uma nova camada chamada “Chora, Anjinho, Chora.” Então ela as converteu em um objeto gráfico e o moveu para a biblioteca.

Ela sentiu um calafrio morno enquanto movia as longas, pétalas brancas para a biblioteca do filme. Aquilo parecia tão oficial. *Eu escolho você. Eu nomeio você. Façam do conforto do meu palácio sua morada eterna.*

Heh. Ela sorriu. Colorado Springs não era nenhum “palácio”.

Desde que eles se mudaram, papai só esteve em casa uma vez. Ele entrou violentamente pela porta da frente vestindo seu uniforme completo e já começou a passar um sermão em Lara e sua mãe. Sua mãe até deixou cair um pé de alface — que ela havia acabado de lavar — em um pote de Dipn’lik.

O alface mal coube no pote, mas ele se alojou lá direitinho. Papai veio e sacou o pé de alface, fitando-o por algum tempo, até declará-lo PT, em uma voz ao mesmo tempo confusa e desapontada. E jogou aquele vegetal sujo na lata do lixo.

Foi só mais tarde aquele dia que a mãe de Lara concluiu que ela poderia simplesmente ter cortado o alface ao meio com uma faca elétrica. Papai sorriu e estapeou a própria testa. Ele, então, foi até Lara e também lhe deu um tapa na testa, e em sua mãe também, afetuosaamente.

“Nós não estávamos pensando, não é mesmo?” ele disse. “E quem pode nos culpar? Nós somos uma família de verdade hoje. E nós não deveríamos ter que fazer mais nada no dia em que reunimos nossa família de novo.”

O sorriso de Lara refletiu no vidro do monitor. Ela escolheu o programa de texto e em fonte serifada tamanho 42 escreveu: “Pai.” Ela criou um

guindaste e, no meio do vôo, agarra uma corda e se salva, pousando num macio antílope oculto que um dos trabalhadores tinha aparentemente bronzeado naquela tarde.

As pessoas desligavam a televisão na hora que o seriado ia ao ar. Tentaram mudar o nome. Primeiro foi *Deus nos Deu um Esquadrão*. *Beijo de Dor*. Então, *Beijo de Dor em Maine*, já que o distrito policial inteiro acabou se mudando para lá. Mas a mágica se fora. Eu voltei para escola no verão daquele ano para algumas aulas e as crianças todas já estavam na onda dos lápis de futebol.

Blocos (Blocks)

Há alguns anos, eu comecei a ensinar o Blix sobre Ruby. Quando nós chegamos nesta parte, a parte que cobre blocos, ele me disse, “Blocos me lembram o Dick Robinson Doidão.”

“Mesmo?” Eu não ouvia esse nome faz tempo. “Não sei como você se lembrou disso.”

“Bem, você disse que blocos são difíceis de entender.”

“Eles não são difíceis”, eu disse. “Um **bloco** é apenas **código agrupado**.”

“E o Doidão era apenas um policial, que jurou cumprir seu dever,” ele disse. “Mas ele era um verdadeiro milagre ambulante. E agora, este primeiro exemplo que você me mostrou...” Ele apontou para um exemplo que eu havia escrito para ele.

caminho para ele e deixou que ele caminhasse até sumir a direita da tela. Ela chorou por algum tempo depois que aquilo se fora.

```
brinquedos_do_gatinho =  
  [:formato => 'meia', :material => 'caxemira'] +  
  [:formato => 'rato', :material => 'calico'] +  
  [:formato => 'bolinho-primavera', :material => 'chenilha']  
brinquedos_do_gatinho.sort_by { |brinquedo| brinquedo[:material] }
```

“Isto é um pequeno milagre”, ele disse. “Não posso negar sua beleza. Olhe, lá estão meus brinquedos, junto com suas características. Abaixo deles, o bloco, ordenando-os por material.”

“Peço desculpas se a sua lista de brinquedos parece um pouco complicada,” eu disse. Assim como você, ele já havia aprendido sobre Array, a centopéia caída no código, colchetes de cada lado e cada item separado por vírgulas. (Ah, aqui está uma: `['meia', 'rato', 'bolinho-primavera']`.) Ele também sabia sobre a Hash, que é como um dicionário, com chaves de cada lado que parecem livrinhos abertos. Vírgulas na Hash entre cada par. Cada palavra no dicionário encontra seu par por uma seta. (Estupefa-se: `{'blix' => 'gato', 'why' => 'humano'}`.)

“Sim, perturbante,” ele disse. “Isso tem colchetes como se fosse Array, mas com as setas é uma Hash. Você vai ter que explicar isso.”

“Subversivo ele, não?” Eu disse, cutucando-o com uma colher. “Eu fiz sua lista de brinquedos misturando os dois. Estou usando um atalho. Que é: **Se você usar a seta (`=>`) dentro de um Array, você terá uma Hash ao invés de um Array.**”

“Ah, entendi,” ele disse. “Você cruzou eles. Que bacana!”

“Sim, sim, você está entendendo,” eu disse. Ele era muito bom com um transferidor também. “Eu tenho três Arrays, cada um contém uma Hash. Notou o sinal de mais? Eu os estou adicionando num Array maior. Aqui está outro jeito de escrever isto...” Eu rabisquei.

```
brinquedos_do_gatinho = [  
  {formato => 'meia', material => 'caxemira'},  
  {formato => 'rato', material => 'calico'},  
  {formato => 'bolinho-primavera', material => 'chenilha'}]
```

Um Array, que age como uma lista de brinquedos de mastigar. Três Hashes estão no Array descrevendo cada

brinquedo.

Ordenando e Iterando Para Salvar Vidas

“Vamos ordenar seus brinquedos por forma agora,” Eu disse. “E depois, imprimí-los nesta ordem.”

```
brinquedos_do_gatinho.sort_by { |brinquedo| brinquedo[:formato] }.each do |brinquedo|
  puts "Blixy tem um #{brinquedo[:formato]} feito de #{brinquedo[:material]}"
end
```

“Como **sort_by** funciona?” perguntou Blix. “Eu sei que é um método que você pode usar com Arrays. Pois **brinquedos_do_gatinho** é um Array. Mas e quanto a **brinquedo**? ”

“Okay, **brinquedo** é um **argumento de bloco**,” Eu disse. “Lembre-se: os corrimãos magrelos de cada lado do **brinquedo** fazem dele uma **calha**. ”

“Claro, mas parece que você está usando-o como um Hash. Dentro do bloco você tem **brinquedo[:formato]**. Isto se parece com um Hash.”

“O método **sort_by** é um **iterador**, Blix. Ele **itera**, ou **faz o ciclo**, com a **lista de coisas**. Você se lembra daquele episódio quando o Doidão... ”

“Episódio?” ele disse. Sim, ele não consegue entender o conceito de seriados de TV. Sim, eu tentei explicar.

“Ou, sim, lembra daquele *O caso da testemunha ocular* que nós assistimos que o Doidão estava tentando conversar com aquele concorrente do soletrando que queria pular do alto da biblioteca do colégio? ”

“Eu me lembro melhor do que você porque eu estava voando num avião de controle remoto.” Sim, foi um daqueles episódios.

“Você se lembra como o Doidão fez para o cara descer?” Eu indaguei.

“Pessoas que concorrem ao soletrando adoram cartas,” disse Blix. “Então o Doidão teve uma idéia de gênio. Ele começou uma carta com a letra A e deu razões, para cada letra do alfabeto, de por que o cara deveria sair do beiral do prédio e voltar a salvo ao chão.”

“‘A’ é para Arquitetura de prédios como esse,” eu disse, com a voz rude do Doidão. “‘Que nos dá esperança num mundo a ruir.’ ”

“‘B’ é para Brutamontes, assim como seu amigo O Policial Doidão,” disse Blix. “‘Caras que ajudam pessoas todo o tempo e não sabem escrever muito bem, mas mesmo assim ajudam caras que escrevem muito bem.’ ”

“Viu, ele passou por todas as letras, uma a uma. Ele estava *iterando* por elas.” *I Tee Rann Do*.

“Mas o cara desistiu, Why. Ele desistiu na letra Q eu acho.”

“‘Q’ é para momentos Quietos que nos ajudam a relaxar e pensar sobre todos os pequenos prazeres da vida, assim nós não ficamos tensos e começamos a fazer besteira andando na ponta do pé na beira de algum edifício alto e maligno.” ”

“Aí ele pulou,” disse Blix. Ele balançou a cabeça. “Você não pode culpar o Doidão. Ele fez o melhor que pode.”

“Ele tinha um coração grande, isso é verdade,” eu disse, dando um tapinha no ombro do Blix.

```
brinquedos_do_gatinho.sort_by { |brinquedo| brinquedo[:formato] }.each do |brinquedo|
  puts "Blixy tem um #{brinquedo[:formato]} feito de #{brinquedo[:material]}"
end
```

“Quanto ao seu **sort_by**, ele **começa no topo** da lista e vai **descendo até o ultimo item**, um de cada vez. Então **brinquedo** é um destes itens. Em cada item, o **sort_by** pára e **joga aquele item pela calha**, sob o nome de **brinquedo**, e deixa você decidir o que fazer com ele.”

“Ok, então **brinquedo** será, a cada momento, um dos brinquedos que eu tenho.”

“Correto,” eu disse. “Você sabe como eu tenho batido na mesma tecla *usando as respostas que os métodos lhe dão?* Aqui, nós estamos apenas olhando a forma do brinquedo dentro do bloco. O bloco então responde para o **sort_by** uma frase contendo o formato, algo como @”rato”@ ou **“meia”**. Uma vez que ele passe por toda a lista, o **sort_by** terá comparado alfabeticamente o texto de cada forma e retornará um novo Array, agora ordenado.”

Uma Lição Inacabada

“Tá bom por hoje,” disse o Blix. “Que tal um pires de leite fresco, por favor?”

Eu enchi o pires dele até a borda e ele se esbaldou daquilo por algum tempo enquanto eu jogava pôquer e golpeava os carvões na lareira. Minha mente vagou e eu não pude não pensar mais sobre blocos. Eu pensei no que ensinar para Blix em seguida.

Eu provavelmente o teria ensinado sobre **next**. Quando você está iterando uma lista, você pode usar **next** para **pular para o próximo item**. Aqui estamos contando brinquedos que não tenham forma de bolinho-primavera, pulando eles com **next**.

```
nao_bolinho_primavera = 0
brinquedos_do_gatinho.each do |brinquedo|
  next if brinquedo[:formato] == 'bolinho-primavera'
  nao_bolinho_primavera = nao_bolinho_primavera + 1
end
```

Eu poderia também tê-lo ensinado sobre **break**, que **joga você para fora de um loop**. No código abaixo, vamos imprimir (usando **p**) cada brinquedo até encontrarmos o brinquedo cujo material seja chenilha. O **break** fará com que o **each** termine abruptamente.

```
brinquedos_do_gatinho.each do |brinquedo|
  break if brinquedo[:material] == 'chenilha'
  p brinquedo
end
```

Eu nunca cheguei a ensinar a ele estas coisas. Eu continuei viajando num pedaço de carvão em particular que acabou preso na proteção da lareira e quase caiu no meu tapete de antílope.

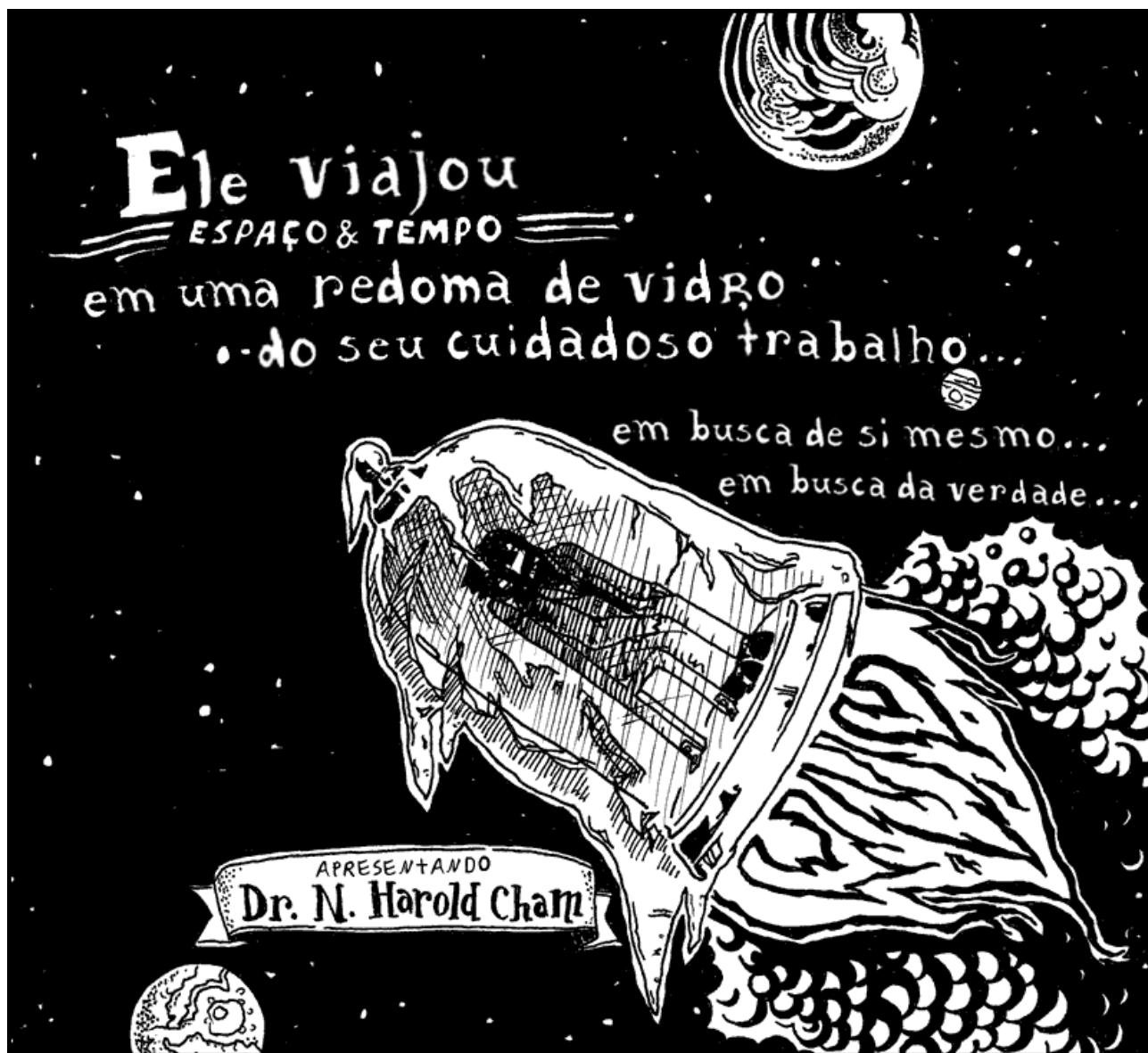
Assim que me esquivei ferozmente daquela pedra negra, Blix sumiu, presumo que para o ônibus para Wixl, a capital econômica dos animais. Quem sabe, ele pode ter parado em Ambrose ou Riathna ou qualquer outra vila pelo caminho. Meu instinto me diz que Wixl era definitivamente sua última parada.

Sem nenhum estudante para instruir ou prosear, encontrei-me sozinho, enfurnado em casa. Na quietude dos corredores mortos, eu comecei a rascunhar uma biografia na forma deste guia.

Eu trabalhei nisso sempre que me sentia entendido. E quando não estava entendido, eu ia assistir *A Ameaça Fantasma* para ficar no clima.



Aqueles Que Fazem as Regras e Aqueles Que Vivem o Sonho

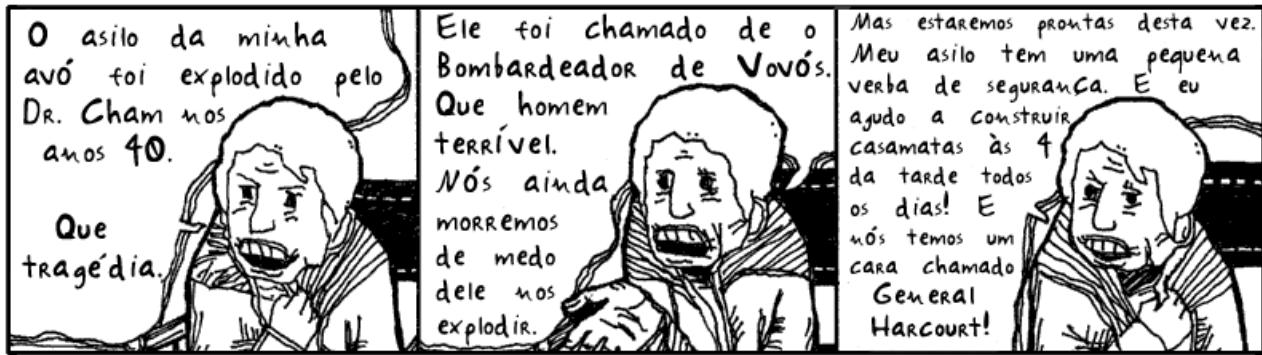


Francamente, estou cansado de ouvir que o Dr. Cham é louco. Sim, ele tentou se enterrar vivo. Sim, ele eletrocutou sua sobrinha. Sim, é fato, ele explodiu um asilo. Mas tudo isso foi por uma boa causa e, em cada caso, eu acredito que ele tenha tomado a decisão certa.

Tenho certeza que você gostaria de ficar do lado da opinião popular, mas você está prestes a sentir um pequeno comichão de admiração por ele, uma vez que ele te ensine tudo sobre as definições de classe do Ruby. E ainda mais quando você aprender sobre mixins. E talvez, no final do capítulo, nós todos começaremos a olhar além do passado obscuro do Doutor e não mais o chamaremos de louco.

Então, se você quer chamá-lo de louco, eu começaria indo na estação de trem quebrar umas lâmpadas fluorescentes grandes. Tire-o do seu sistema agora, antes de irmos mais além.

1. Esta é para os Desprivilegiados



Se você me der um número, seja ele qualquer ano da vida do Dr. Cham, darei-lhe uma sinopse do período. E farei isso com um método Ruby. Portanto será uma peça independente, um pedaço de código isolado que pode ser ligado à voz de um vulcão robótico, quando este tipo de coisa for o ápice das vozes competentes e talentosas.

Ok, preciso que você preste atenção nos comandos **def**, **case** e **when** (quando). Você já viu Ranges (Períodos), os acordeões fechados **1895..1913**, no capítulo 3. Eles contêm as pontas e todos os números entre. E as barras no final de cada linha simplesmente ignoram o *Enter*, assegurando ao Ruby que *há mais linhas por vir*.

Então, por favor: **def**, **case** e **when**.

```
def biografia_do_dr_cham( ano )
  case ano
  when 1894
    "Nasceu."
  when 1895..1913
    "Infância em Louisville, Winston Co., Mississippi."
  when 1914..1919
    "Trabalhou em uma creche de nozes-pecãs; socou um quacre de uma seita protestante."
  when 1920..1928
    "Navegou na companhia do Rio Wisdomming, que se juntou \
      ao Rio Mississippi e se engajou em reflexão e auto-aprimoramento, \
      onde ele concluiu 140 horas de crédito para sua Remoniversidade"
  when 1929
    "Voltou à Louisville para escrever um romance sobre camponeses caçadores que viajam no tempo."
  when 1930..1933
    "Ergueu uma respeitável carreira garantindo as creches das nozes-pecãs. Financeiramente estável, \
      ele passou algum tempo no Brasil e Novo México, comprando raras árvores de noz-pecã de casca \
      fina. Assim que sua notoriedade chegou a um alto patamar: caramba, ele tentou se enterrar vivo."
  when 1934
    "Voltou a escrever seu romance. Mudou de caçadores para magnatas de seguradoras e \
      de camponeses para protestantes."
  when 1935..1940
    "Recebeu Arthur Cone, o chefe da Irmandade do Rio Wisdomming, em sua residência \
      como hóspede. Juntos por cinco anos, planejaram e inventaram."
  when 1941
    "Aqui as coisas começam a ficar interessantes."
  end
end
```

O comando **def**. Aqui esta nossa primeira **definição de método**. Um método simples, que pode ser usado em qualquer lugar no Ruby. E como rodamos ele?

```
puts biografia_do_dr_cham( 1941 )
```

Que por sua vez responde “Aqui as coisas começam a ficar interessantes.” É sempre a mesma história: *use suas respostas*. Eu fiz as coisas ali de tal modo que o **case** sempre responderá com uma String. E já que o **case** é o último (e único) comando no método, então o método responderá aquela String. Gotejando água que é derramada de uma saliência para outra.

Permita-me ser mais claro sobre a condição **case**. Na verdade, eu deveria chamá-la de **case..when**, já que elas não podem ser usadas separadamente. O comando **case** é seguido de um valor, que é comparado contra cada valor que segue o comando **when**. O primeiro valor a coincidir é usado pelo **case**, o resto é ignorado. Você pode fazer a mesma coisa com um monte de **if..elsif**, mas isso é muito verborrágico.

```

case ano
when 1894
    "Nasceu."
when 1895..1913
    "Infância em Lousville, Winston Co., Mississippi."
else
    "Sem informações sobre este ano."
end

```

É a mesma coisa que:

```

if 1894 === ano
    "Nasceu."
elsif 1895..1913 === ano
    "Infância em Lousville, Winston Co., Mississippi."
else
    "Sem informações para este ano."
end

```

Os **três iguais** (==) são o comprimento da corda de veludo, checando valores de forma similar aos dois iguais (==). Resumindo: os três iguais são uma corda maior e ela desce um pouco no meio. Não é tão tensionada, é um pouco mais maleável.

Veja os períodos assim: (1895..1913) não é nem um pouco **igual** a 1905. Não, o período (1895..1913) só é realmente **igual** a qualquer outro período (1895..1913). Os três iguais, no caso de um período, dão uma folga e deixam o número inteiro 1905 entrar, pois mesmo ele não sendo **igual** ao período, ele está **incluso** na lista de números inteiros representados pelo período. O que é suficiente em alguns casos, como o da biografia que eu fiz anteriormente.

A qual realmente se parece com uma biografia, não? Quero dizer, claro, o método **biografia_do_dr_cham** é código, mas ele se parece com uma biografia, clara e afável.



Mas Estava Ele Doente??

Você sabe, o momento dele era tão ruim. Era um desastre como romancista, mas suas empreitadas na alquimia eram muito promissoras. Ele tinha um elixir de leite de cabra e sal marinho que melhorava dores nas pernas. Um cara até regenerou um pedaço do dedo que havia perdido. Tinha um fumo medicinal orgânico que cheirava chulé mas lhe dava visão noturna. Ele trabalhava em algo chamado Escada Líquida, mas nunca vi ou li nada a respeito. Não pode ser para subir. Quem sabe.

Um jornal local uma vez visitou o Dr. Cham. A crítica literária deles lhe deu quatro estrelas. Sério. Ela fez um artigo sobre ele. Deu-lhe uma avaliação.

Mas saiba que o Dr. N. Harold Cham se sentia muito mal em relação à sua sobrinha. Ele acreditou que o tratamento de choque funcionaria. A pôlio a teria matado de qualquer jeito, mas ele tentou.

Em 9 de Setembro de 1941, depois de sedá-la como uma dose de

Importando-se com você. E no seu bem-estar.

Eu preciso que você esteja num bom estado mental para o resto do livro. Agora é hora de começar a lhe condicionar.

Vamos começar respirando profundamente. Inspire profundamente e conte até quatro comigo.

Vamos lá. 1. 2. 3. 4. Agora expire. Você pode sentir seus olhos. Bom, é isso mesmo.

Agora vamos inspirar profundamente e, na sua mente, desenhe um hipopótamo o mais rápido que puder. Rápido rápido. Suas pernas, sua forma, seus dentes de marshmallow. Ok, pronto. Agora expire.

fenacetina na sua sala de operações, ele colocou os cliques condutores no nariz, língua, dedos e ombros de Hannah. Com a ajuda de seu aprendiz, um estudante perspicaz chamado Marvin Holyoake, eles polvilharam a garota com flocos de uma substância que o doutor chamou de *opus magnum*. Um pó de ouro branco que levaria a corrente e evidentemente energizaria a menina, forçando seu sangue a aflorar e lutar e vencer.

Mas aquilo não deu muito certo, quando a alavanca foi puxada, ela se debateu e chutou — e **KABLAM!** — e **BLOY-OY-OY-KKPOY!** Anéis de cabelo e uma parede de luz, e o sino da morte tocou. A experiência sucumbiu em uma estreita pluma de fumaça e a inocência dela (*durante semanas, todo mundo só falava: "Ela nunca vai ter a chance de..."*) foi um rombo no chão e nos pulmões deles.

Para Hannah, eu programo.

```
opus_magnum = true
def salve_hannah
    sucesso = opus_magnum
end
```

Um método em sua própria ilha. E o que vai dentro não é afetado por simples variáveis externas. O Dr. Cham não podia penetrar na doença da sua sobrinha assim como uma variável **opus_magnum** não pode penetrar no exterior de aço do método.

Se rodarmos o método **salve_hannah**, o Ruby vai chiar conosco, alegando que não sabe de **opus_magnum** nenhuma.

Estou falando de **foco**, **escopo**. Microscópios restringem e magnificam sua visão. Telescópios estendem seu campo de visão. No Ruby, **escopo** referencia o campo de visão dentro de métodos e blocos.

A declaração **def** do método abre sua visão. Nomes de variáveis introduzidos ali serão vistos pelo método e continuarão valendo até que o **end** feche seus olhos. Você pode passar dados para um método usando argumentos e dados podem ser resgatados do método, mas os nomes usados dentro do método só valem no escopo dele.

Algumas variáveis tem um escopo mais amplo. Variáveis globais, como **\$LOAD_PATH**, que começam com um **cifrão** estão disponíveis sob qualquer escopo. Variáveis de instância como **@nomes**, que começam com uma **arroba** estão disponíveis em todo lugar dentro do escopo da classe. A mesma coisa serve para variáveis de classe como **@@@ingressos@**. Variáveis de classe e de instância serão exploradas em breve.

Blocos têm escopo, mas ele é um pouco difuso. Mais flexível.

```
verbo = 'salvou'
['sedou', 'polvilhou', 'eletrocutou'].
each do |verbo|
  puts "O Dr. Cham " + verbo + " sua sobrinha Hannah."
end
puts "Sim, o Dr. Cham " + verbo + " sua sobrinha
Hannah."
```

O bloco *itera* (circunda, revolve) por cada uma das ações do Doutor. A variável **verbo** muda a cada passo. Em um passo, ele a está sedando. No próximo, ele está polvilhando-a. E então, eletrocutando.

Dê mais uma boa inspirada e segure. Enquanto você segura o ar firme no seu peito, imagine que a pressão o está reduzindo ao tamanho de um inseto. Você segurou seu fôlego tão forte que agora você é um inseto. E todos os outros insetos viram você diminuir e adoraram a façanha. Eles estão batendo palmas e roçando as anteninhas freneticamente. Mas você tinha uma maçã na mão quando você era grande, e ela te pegou agora, esmagou toda a multidão. Você também está morto. Agora expire.

Dê-me uma boa respirada e imagine que você vive em uma cidade em que tudo é feito de fios de telefone. As casas são todas de fios de telefone, as paredes, o telhado. As portas são um massa densa de fios de telefone que você simplesmente atravessa. Quando você vai para cama, a cama é de fios de telefone. E o colchão e as molas da cama também são fios de telefone. Como eu disse, tudo é feito de fios de telefone. O próprio telefone é feito de fios de telefone. Mas o fio de telefone que vai para telefone é feito de pão e palitos. Agora expire.

Inspire. 1. 2. 3. 4. Expire.

Inspire em. 1. 2. Inspire mais um pouco. 3. 4. Imagine suas mãos saindo dos pulsos, voando até a tela do computador e programando ele de dentro. Expire.

Respire fundo, fundo. Abaixo de você tem um submarino. Ele tem uma língua. Expire.

Respire pelas narinas. Respire fundo. Filtre o ar pelas narinas. Respirar pelo nariz te dá ar de qualidade. Suas narinas se abrem, você recebe o ar da natureza, do jeito que Deus planejou. Imagine um leitor de disquetes lotado de órfãos. Enquanto ele se engasga com os órfãos, você tem o ótimo, o maravilhoso ar de Deus em seus pulmões. Mas este ar prazeroso e que dá a vida vira uma poderosa toxina se você o segurar por muito tempo. *Rápido, exale o ar de Deus e da natureza!*

Agora, você vai acordar, desamassando as dobras desta página no seu navegador web. Você relembrará toda sua vida e não se esquecerá de nenhuma das várias aventuras que você teve nela. Você se sentirá rico, renovado e mais experiente. Você não terá lembranças deste curto exercício, na verdade você vai lembrar de estar ensinando um coelho a uma grande distância a usar tesouras.

Já que você acordará e logo verá este exercício, você vai fazê-lo novamente. Mas desta vez, tente imaginar que até mesmo sua sombra é um fio de telefone.

Então, a questão é: após o término do bloco, terá ele salvado Hannah?

```
O Dr. Cham sedou sua sobrinha Hannah.  
O Dr. Cham polvilhou sua sobrinha Hannah.  
O Dr. Cham eletrocutou sua sobrinha Hannah.  
Sim, o Dr. Cham eletrocutou sua sobrinha Hannah.
```

Os blocos podem ver variáveis da vizinhança. O bloco viu que a variável **verbo** existia, ele reescreveu o que ela continha e continuou. Quando o bloco acabou e sua pequena vida se esvaiu, a variável **verbo** saiu de lá uma outra criatura.

Se o bloco usar uma variável que ainda não tenha sido usada previamente, essa variável desaparece após o término do bloco. O **escopo** do bloco se fecha e a variável vai junto. Aqui um exemplo em que **verbo** não foi usado antes do bloco:

```
['sedou', 'polvilhou', 'eletrocutou'].  
each do |verbo|  
  puts "O Dr. Cham " + verbo + " sua sobrinha Hannah."  
end  
puts "Sim, O Dr. Cham " + verbo + " sua sobrinha  
Hannah."
```

Dá um erro: **undefined local variable or method 'verbo'**. Poof. (Método ou variável local ‘verbo’ não definido.)

Deve ser difícil, mesmo para um grande cientista, sumir com o corpo de uma garotinha cujo vestido ainda está passado e bordado, mas cuja boca está roxa nas extremidades. No jornal do Dr. Cham, ele relata que fora atormentado pelo fantasma dela, que reluzia ouro e cuja renda flamejava. Suas desilusões aumentaram e ele fugiu do Cérberus e de massivas, vingativas mãos angelicais.

Somente semanas mais tarde, ele foi embora, impulsionado por esses arrependimentos, sumindo na explosão que o levaria para fora do planeta.

Ainda enquanto você lê isto aqui, em algum momento, a jarra em formato de sino do nosso solitário Dr. Cham pousou num distante planeta depois de viajar sessenta anos. Assim que o novo mundo apareceu, assim que a curvatura do planeta se mostrou, assim que a jarra em formato de sino varreu os céus tempestuosos, rasgando folhas de aurora e vento solar, os olhos do Dr. Cham se abriram chocados.



O que você está testemunhando é o pouso do Dr. Cham no planeta Endertromb. Do que eu pude perceber, ele pousou no meio de uma estação desolada, época em que não tem muita coisa acontecendo no planeta. A maioria dos habitantes têm suas mentes presas a um zumbido desanimador que faz com que eles se desintegrem em fantasmas fúteis uma-parte-conhecimento e três-partes-vapor por um tempo.

Meu modesto conhecimento sobre a história e clima de Endertromb foi adquirido convivendo com o professor de piano da minha filha, quem cresceu no planeta.



Eu freqüentemente treino o professor de piano da minha filha para assegurar que ele mantenha seus compromissos. Que ele atenda chamadas em horas estranhas e responda chamadas de emergência prontamente. Quando ele finalmente me contou que era um alienígena cujo dia consistia em quinhentas e quarenta horas acordado, eu estava incrivelmente favorável a um relacionamento contratual com ele que durará até 2060.

Por três dias (de acordo com seu relógio de bolso), o Dr. Cham viajou nas escuras passagens de ar, respirando o vento empoeirado daquele planeta árido. Mas no terceiro dia, ele encontrou o término da Estação Desolada e acordou para uma vista magnífica, decorada com recém-floridas árvores de maçã e fileiras de frescos castelos.

2. Um Castelo Tem Seus Computadores



Nosso intrépido Doutor partiu para o castelo alienígena, lançando-se sobre as flores. O chão passou por seus calcanhares. O castelo aparecia gradativamente no horizonte. Ele desejava ter um cavalo garanhão, mas nenhum cavalo apareceu. E foi assim que ele descobriu que o planeta não leria sua mente e não responderia seus pedidos.

Porém, como o instrutor de órgão da minha filha havia me explicado, o planeta **podia ler mentes e podia realizar desejos**. Só que não os dois ao mesmo tempo.

Um dia questionando o maestro de órgão, ele rascunhou o seguinte código Ruby numa folha de papel cor-de-queijo. (E cheiros estranhos de queijo estavam vindo de algum lugar, eu não posso dizer onde.)

```
require 'endertromb'
class FazedorPedidos
  def initialize
    @energia = rand( 6 )
  end
  def realize( pedido )
    if pedido.length > 10 or pedido.include? ' '
      raise ArgumentError, "Pedido ruim."
    end
    if @energia.zero?
      raise Exception, "Sem energia."
    end
    @energia -= 1
    Endertromb::realize( pedido )
  end
end
```

Este é o fazedor de pedidos.

Na verdade, não, esta é a **definição de um fazedor de pedidos**. Para o Ruby, é uma **definição de classe**. O código descreve como um certo **objeto** vai funcionar.

Toda manhã, o fazedor de pedidos inicia com até cinco pedidos disponíveis para serem concedidos. Um novo **FazedorPedidos** é criado quando o sol levanta.

```
pedidos_do_dia = FazedorPedidos.new
```

O método **new** é um método de classe que cria um novo, objeto em branco. Ele também chama o método **initialize** (inicializar) do objeto automaticamente. Na definição do **FazedorPedidos**, você vai ver o método **initialize**, que contém uma única linha de código: `@energia = rand(6)`.

O **rand(6)** sorteia um número entre 0 e 5. Este número vai representar quantos pedidos ainda restam no dia. Então, ocasionalmente não haverá pedidos disponíveis do fazedor de pedidos.

O número aleatório é dado à uma **variável de instância** de nome **@energia**. Esta variável de instância estará disponível a qualquer momento por toda a classe. A variável não pode ser usada fora do **escopo** da classe.

No capítulo três, demos uma breve olhada em variáveis de instância e decidimos respeitá-las como **atributos**. (O símbolo **arroba** poderia significar **atributo**.) Variáveis de instância podem ser usadas para guardar qualquer tipo de informação, mas elas são mais usadas para se guardar pedaços de informação sobre o objeto representado pela classe.

No caso acima, cada fazedor de pedidos para o dia tem seu próprio nível de energia. Se o fazedor de pedidos fosse uma máquina, você talvez veria um medidor nele que marca a energia restante. A variável de instância **@energia** agirá como esse medidor.

```
pedidos_do_dia = FazedorPedidos.new  
pedidos_do_dia.realize( "chifres" )
```

Ok, dê um passo atrás e tenha certeza de entender este exemplo aqui. A classe **FazedorPedidos** é a base que elaboramos de como toda a mágica dos pedidos funciona. Não é o *realmente* o gênio na lâmpada, é a papelada por trás das cenas. São as regras e obrigações em que o gênio vive.

É o **pedidos_do_dia** que é o gênio da lâmpada. E aqui estamos dando um pedido a ser realizado. Nos dê uma galhada, gênio. (Se você realmente arranjar uma galhada neste exemplo, eu nem quero saber. Vá saltitar em campinas verdes com seus amigos.)

No capítulo anterior, o treinamento era: Ruby tem duas metades.

1. Definindo coisas.
2. Colocando estas coisas em ação.

O que são as ações no Ruby? Métodos. E agora, você está provando a linguagem de definição interna do Ruby. Definições de métodos usando **def**. Definições de classe usando **class**.

Neste ponto da sua instrução, é mais fácil entender que **tudo no Ruby é um objeto**.

```
numero = 5  
print numero.next # imprime '6'  
  
frase = 'desejando por chifres'  
print frase.length # imprime '19'  
  
pedidos_do_dia = FazedorPedidos.new  
pedidos_do_dia.realize( "chifres" )
```

E, consequentemente, cada objeto tem uma classe nos bastidores.

```
print 5.class # imprime 'Integer'  
print 'desejando por chifres'.class # imprime 'String'  
print FazedorPedidos.new.class # imprime 'FazedorPedidos'
```

Dr. Cham nunca viu o fazedor de pedidos enquanto desbravava a paisagem. Ele ficava bem além das terras dele, no vale de Sedna. Penhascos cheios de porções de bosques, onde você podia jogar seus pedidos (escritos num tirinha 1" × 6"), dentro do sumidouro. Com sorte ele pousará nas costas de um lagarto, espetando em seu chifrinho pontiagudo dele.

E digamos que seu pedido chegue lá. Bem, então, *descendo pela floresta contorcida* vai a magra salamandra, correndo através das igrejas descendentes que foram **empurradas** para a íngreme base do desfiladeiro de uma vez por todas. E

o desfalecido padre dentro, *que também resistiu a queda*, matará o pequeno anfíbio — estrangulando-o até a morte com uma corrente abençoada de ouro — e guardá-lo para as comemorações do *Conhecendo Seu café da manhã*. Ele pisará no seu pequeno e precioso pedido e, quando os **ladrões vierem**, aquele papel ainda estará lá, preso na sola do sapato dele. Claro, o **meio preferido de tortura** dos ladrões é cortar um padre em finíssimas fatias *da cabeça aos pés*. Quem pode catar evidências disto? E quando eles fatiam a última fatia da sola do sapato, eles terão aquele **escalpo de borracha** nas mãos para *boa sorte e bons tempos*. Mas eles **canoam** com muita força, estes ladrões. Eles batem seus remos rapidamente na corrente para manter o grande *vapor do motor externo*. Mas a sola de sapato está *em uma cadeia fraca*, presa ao cinto de um dos homens. E uma **velha carpa cabeluda** salta, agarra esta fração de minuto de calçado. E os ladrões *podem tentar*, mas eles não vêm *embaixo d'água*. Se eles pudesse, eles veriam aquele **cabo poderoso**, empacotado com milhões de *necessariamente fibras ópticas*, esse peixe é um periférico conectado diretamente ao *núcleo de trabalho* do planeta Endertromb. **Tudo que ele pega é um gole** daquele peixe e seu desejo está liberto!

E é assim que os pedidos das crianças se realizam neste lugar.

Uma vez que o instrutor de órgão da minha filha tenha desenhado a classe para o fazedor de pedidos, ele continuou com uma classe para o leitor de mentes do planeta.

```
require 'endertromb'
class LeitorMental
  def initialize
    @mentes = Endertromb::procurar_por_consciencia
  end
  def read
    @mentes.collect do |mente|
      mente.read
    end
  end
end
```

Como você já viu antes, o **initialize** acontece quando um novo (new) objeto **LeitorMental** é criado. Este **initialize** rastreia o planeta em busca das mentes. Parece que as mentes estão guardadas em um array, já que há uma iteração nelas no final usando o método **collect**.

Tanto o fazedor de pedidos e quanto o leitor mental referem-se a uma classe chamada **Endertromb**. Esta classe está guardada em um arquivo **endertromb.rb**, que é carregado com o código: **require 'endertromb'**. Você vai, muitas vezes, usar outras classes para realizar parte da sua tarefa. Boa parte da metade restante deste livro explorará a grande variedade de classes úteis que podem ser carregadas no Ruby.

Dr. Cham Arrisca Entrar

Mas assim que o Dr. Cham se aproximou do castelo, mesmo o planeta sabendo de seus pensamentos, sentindo sua surpresa e antecipação, tudo que Dr. Cham sentiu era a morte. Ele subiu os degraus de seu portão frontal, passou através da entrada de mais bela arquitetura e estava quase certo de que ele estava deserto.

Por um tempo ele bateu na porta. O que foi recompensante.

Ele assistiu ao bebê baleia subir como um balão determinado. Ele se maravilhou com sua primeira introdução alienígena e sentiu alguma preocupação de que ela tenha passado tão rapidamente. Bem, ele iria esperar lá dentro.

Assim que ele entrou pela porta do castelo, sentiu-se com sorte pela porta não ter sido atendida por uma águia gigante com garras afiadas, ávida por brincar. Ou uma cabeça de rato gigante. Ou até mesmo um furacão ou mesmo um furacão do tamanho de um homem. Somente uma pequena baleia choo-choo rechonchuda.

“Nenhum lugar para se sentar neste castelo,” disse ele.

À primeira vista, ele pensou ter entrado em um saguão escuro, mas assim que seus olhos se ajustaram, ele viu que a entrada se estendia em um túnel. A porta do castelo se abriu em uma passagem feita de pedaços longos e planos de rocha. Algumas partes eram harmônicas e lembravam um corredor. Outras partes eram estreitas e até inclinadas, o fim sumia de vista.

A passagem era iluminada por pequenas geladeiras sem portas, grandes o suficiente para guardar uma braçada de repolho, até os seus pés. Ele olhou para dentro de um deles, que estava oco, iluminando por todos os lados, e criando cacos de gelo metódicamente.

Ele tocou nos pedaços de gelo, que grudaram secamente em seus dedos. E ele então esfregou suas mãos no gelo. O

que deixou algumas linhas enlameadas em suas mãos, mas satisfez um pouco sua saudade por um banho. Quanto tempo já fazia? Dez anos? Trinta?

Pela passagem, longos tubos de tecido lotavam algumas seções. Mais tarde, matéria de pontos brilhantes em conchas de porcelana e baldes.

Ele caiu em uma sala que havia sido escavada ao lado do túnel e nela haviam alguns cascos de tartaruga vazios no chão e uma grande parede iluminada. Ele examinou a sala, perplexo. O que seria isso? Por um instante, pensou em sentar em um casco. Isto poderia ser a entrada finalmente, algum tipo de sala de recepção. Por outro lado, aranhas poderiam sair dos buracos dos cascos quando ele sentasse. Ele seguiu viagem.

Refeição no Bolso de um Castelo

A medida que continuava sua jornada por entre as passagens (pelo túnel central ramificavam e se juntavam grandes vazias cavernas), ele escolheu temas em algumas localizações. Grupos de salas infestadas com maquinaria de bombeamento. Tecidos e barris de cola dominavam outra área. Ele seguiu vozes que vinham de uma cavidade veludosa e macia como travesseiro, o que o levou para um lugar sem saída: uma parede curvada com uma pequena sala entalhada no nível dos olhos.

Ele se aproximou da parede e, dentro do buraco quadrado, estavam dois porcos-da-terra comendo à mesa.

Eles o encaram serenamente, ambos degustando um pouco de besouros escavados, que davam dois deles, abertos ao meio e parados com as costas na mesa.

“Olá, bonequinhos,” ele disse, e eles terminaram suas mordidas e ficaram olhando com seus garfos segurados à distância.

“Eu queria que minha sobrinha Hannah estivesse aqui para conhecê-los,” disse ele aos atenciosos porcos-da-terra. “Ela pensaria que vocês são um intrincado show de ventriloquismo.” Ele olhou para a área da refeição, empilhada como conjuntos de pratos, toalhas de mão. Metade de um pequeno coelho projetava-se do topo da máquina, grãos vermelho cremosos estavam caindo para baixo dele. Uma porta no fundo da sala estava entreaberta. Dr. Cham podia ver uma sala cintilante com cadeiras e motores zunindo através da porta.

“Toda criança gostaria desta casa de bonecas,” ele disse. “Hannah, minha sobrinha, como mencionei, ela tem uma boneca que senta em um fuso e começa a soltar fio. É uma ilusão, é claro. A boneca não produz fio algum.”

Um dos porcos-da-terra abriu um alçapão no chão e pressionou um botão ali, que se iluminou. Então, um pequeno projetor de filme veio lentamente sob trilhos. O outro porco-da-terra sentou e olhou o Dr. Cham.

“Mas Hannah ainda alcança lá no fundo da casa de boneca e pega todos os fios imaginários formando um maço. Que ela leva para sua mãe, minha irmã, que é muito boa em deixar Hannah de bom humor. Ela costura um vestido no tamanho da boneca, o qual Hannah leva até a boneca.”

“E ela diz à boneca, ‘Aqui, olhe, seu trabalho duro e perseverança resultaram neste lindo vestido. Agora você pode aceitar o convite e acompanhar, hoje à noite, o Chefe de Polícia na Mansão do Governador.’ E ela tem uma boneca vestida de policial que faz o papel do Chefe. Ele é muito raquítico para ser o Chefe de verdade, isso precisaria de um bom pedaço de plástico.”

O porco-da-terra responsável pelo projetor o carregou com um rolo e o mirou na parede do fundo. O filme começou e os porcos-da-terra se sentaram. Um quadrado verde apareceu na parede. O atencioso porco-da-terra encarou o Dr. Cham parado.

“Seus filmes são coloridos,” disse o Dr. Cham. “Que vida pequena e amável.”

O filme exibido foi: um quadrado azul. Aí, um círculo vermelho. Aí, um quadrado laranja. Os atenciosos porcos-da-terra se viraram, assistiram a tela virar um triângulo rosa, e ambos porcos-da-terra voltaram a comer.

Uma estrela roxa. Um quadrado vermelho. Com o assentamento do silêncio, Dr. Cham podia escutar notas zumbindo do projetor. Como uma lenta, lerda caixa de som tentando girar suas engrenagens ao longo da linha de trem.

“Sim, aproveitem o jantar,” disse o Dr. Cham e educadamente se virou, marchando de volta no caminho que ele tomara.

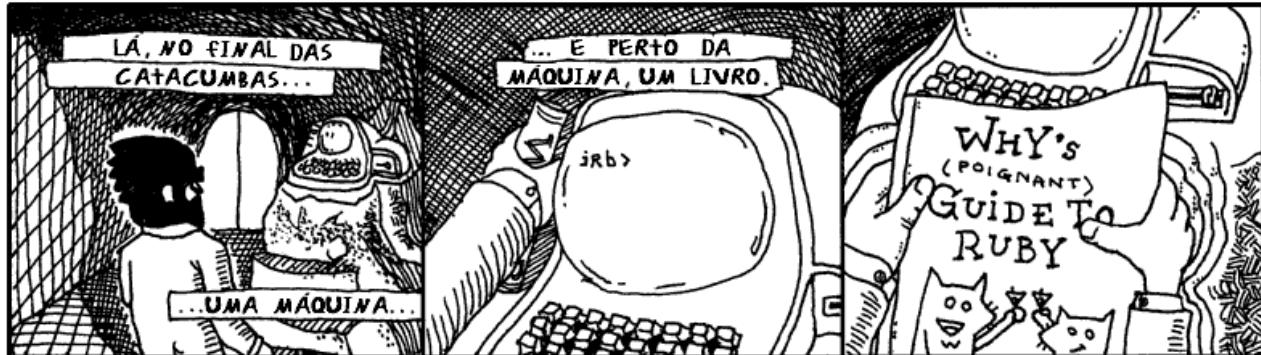
Outra Rua Sem Saída Onde as Coisas Começam

Ele se encontrou perdido nos túneis do castelo. Nada parecia familiar. Todavia, ele não estava muito preocupado. Ele estava noutro planeta. Ele estava perdido de qualquer maneira.

Ele se contorceu pelos túneis, tentando lembrar o caminho, mas muito mais interessado em explorar para lembrar seus passos. Ele seguiu um único túnel profundamente, descendo, descendo, ficou tão inclinado que ele tinha que pular entre as beiradas e cuidadosamente achar apoio para os pés. A gravidade aqui não parecia ser diferente da Terra. Ele escorregava com a mesma facilidade.

Mesmo sem ter absolutamente nenhuma maneira de saber onde ele estava, ele tinha certeza de ter saído dos limites do castelo. Esta profundidade, este tanto de caminhada. Já fazia uma hora desde que ele adentrara a porta. E, a medida que voltava no túnel, ele tinha certeza que emergiria em um uma nova habitação, talvez até mesmo um buraco em que ele pudesse espiar para fora e ver o castelo. Talvez ele não devesse ter ido tão longe nessa rota. Ele espera que nada esteja hibernando aqui embaixo.

O túnel chegou ao fim. Um escuro, fim sem saída.



Ele tinha tempo. Então leu o livro. Ele leu sobre os raposos e sua perseguição ao porco-espinho que roubou a caminhonete deles. Ele leu sobre o elfo e o presunto. Ele viu os pictógrafos dele mesmo e descobriu que ele poderia realmente se relacionar com seu próprio esforço. Ele até mesmo aprendeu Ruby. Ele viu como isso tudo termina.

Eu não teria estômago para isso, no lugar dele. Mas ele teve. E jurou em seu âmago que veria as coisas apenas do jeito que elas aconteciam.

No monitor do computador, o Dr. Cham viu o prompt do `irb` piscando. Assim como o Dr. Cham, você deve reconhecer o prompt do `irb` do [O Colete Do Tigre](#) (o primeiro pacote de expansão deste livro, que inclui uma introdução básica ao IRB – Interactive Ruby.)

Assim como ele acabou de explorar os túneis a pé, agora ele explora as configurações da máquina com o prompt. Ele colocou o livro onde o encontrou. Ele não precisava mais dele. Isto tudo iria acontecer mesmo se ele o usasse ou não.

Ele começou com:

```
irb> Object::constants  
=> ["Marshal", "String", "Dir", "LoadError", "Float", ... e por aí vai ]
```

Este comando lista as constantes de nível mais alto. Classes são listadas como constantes também, então esta lista é ótima para ver o que está carregado no Ruby a qualquer momento.

Ele examinou a lista procurando algo não familiar. Qualquer classe que não vem com o Ruby. `Marshal`, `String`, `Dir`, `LoadError`, `Float`. Cada uma dessas vêm com o Ruby.

Mas um pouco para baixo na lista:

```
... "Struct", "Values", "Time", "Elevador", "Range" ...
```

`Elevador`? Exatamente o tipo de classe a se bisbilhotar. Ele tinha onde ir.

```
irb> Elevador::methods  
=> ["method", "freeze", "allocate", ... outra lista grande ... ]  
irb> Elevador::class_variables  
=> ['@relatorio_diagnostico', '@circuito_eletrico_ativo', '@senha_manutencao']  
irb> Elevador::constants  
=> []
```

Parece que a classe `Elevador` tem um monte de métodos. A maioria deles parecem ser os mesmos métodos que todo objeto tem no Ruby. Por exemplo, `method`, `freeze` e `allocate` vêm com toda classe no Ruby.

(`Elevador::freeze` não deixaria que a classe `Elevador` seja alterada. `Elevador::allocate` criaria um novo objeto `Elevador` sem chamar o método `initialize`.)

As `class_variables` foram interessantes para Dr. Cham. Este elevador parece genuíno. Mas sem `constantes` disponíveis. Isso nos diz que não há classes aninhadas dentro da classe `Elevador`.

Ele tentou criar um objeto `Elevador`.

```
irb> e = Elevador::new
ArgumentError: wrong number of arguments (0 for 1), requires a password
  from (irb):2:in `initialize'
  from (irb):2:in `new'
  from (irb):2
  from :0
```

Ele tentou algumas senhas.

```
irb> e = Elevador::new( "para cima" )
AccessDeniedError: bad password
irb> e = Elevador::new( "subindo" )
AccessDeniedError: bad password
irb> e = Elevador::new( "escadas_sao_terríveis" )
AccessDeniedError: bad password
irb> e = Elevador::new( "EscadasSaoTerríveis" )
AccessDeniedError: bad password
```

Isso foi inútil. *Oh, espere!* A senha de manutenção. Listada nas `class_variables`.

```
irb> Elevador::senha_manutencao
NoMethodError: undefined method `senha_manutencao' for Elevador:Class
  from (irb):1
  from :0
```

Hmm. Variáveis de instância só estão disponíveis dentro de um objeto. E variáveis de classe só estão disponíveis dentro de uma classe. Como conseguir aquela senha?

```
irb> class Elevador
irb>   def Elevador.senha_manutencao
irb>     @@senha_manutencao
irb>   end
irb> end
=> nil
irb> Elevador::senha_manutencao
=> "escadas_sao_passado!"
```

Agora sim! Conseguimos a senha. Você viu aquilo?

Ele adicionou um método de classe à classe `Elevador`. Não é ótimo como você pode começar a definir uma nova classe `Elevador` e o Ruby simplesmente adiciona suas modificações na definição de classe já existente?

Métodos de classe são geralmente chamadas com os **dois pontos duplos**. Mas, um ponto também serve. Já que `Elevador` é uma classe por si só, Ruby vai entender que se você chama `Elevador.senha_manutencao`, você está chamando um método de classe. Os dois pontos duplos simplesmente ajudam a tornar métodos de classe óbvios para o leitor.

E justamente. Métodos de classe são um pouco incomuns. Normalmente você não quer guardar informação diretamente dentro de uma classe. Contudo, se você tem um pedaço de informação que você precisa compartilhar com todos os objetos de uma classe, então você tem um bom motivo para usar a classe como armazenamento. É compreensível que a `@@@senha_manutencao@` tenha sido armazenada na classe, ao invés de em cada objeto separado. Deste modo, o objeto pode simplesmente alcançar a classe e ver a senha compartilhada.

Provavelmente é assim que a proteção por senha funciona.

```
class Elevador
  def initialize( senha )
    raise AccessDeniedError, "senha incorreta" \
      unless senha.equals? @@senha_manutencao
  end
end
```

Proteger uma classe assim não faz sentido, já que tudo no Ruby pode ser alterado e reescrito e remoldado. O Dr. Cham tinha a senha e agora o elevador era dele.

```
irb> e = Elevador.new( "escadas_sao_passado!" )
#<Elevador:0x81f12f4 @level=4>
irb> e.andar = 1
```

Dr. Cham estava parado lá quando as portas do elevador, atrás do terminal do computador, abriram-se para ele. Com um exasperado senso de realização e uma grande quantidade de entusiasmo por todos os eventos que se passaram, ele entrou no elevador e apertou o 4.

3. A Continuação da História do Instrutor de Órgão da Minha Filha

Eu sei que você pode estar surpreso em saber que eu tenho uma filha. Você pensa que meu estilo de escrever indica uma mente imatura ou infantil. Bem, por favor relaxe. Eu não tenho uma filha. Mas isso não me impede de pensar no treinamento musical dela.

Enquanto essas histórias do planeta Endertromb me eram relatadas, eu me vi vagando por corredores, passando as pontas dos meus dedos por sofás fortemente atados e me envolvendo nos gritos saturados dos tubos, enquanto o instrutor de órgão da minha filha tocava. Suas notas ressoavam tão ocas e profundas nas paredes de sua mansão que eu comecei a confundi-las com um silêncio nefasto, e achava ainda mais fácil me recolher ao espaço infinito dos meus pensamentos. Para pensar no planeta ancestral e suas filosofias obscuras: seus templos carnais, corados com os resíduos dérmicos dos seus mártires; seus cartéis de baleias, engolindo seus inimigos e prendendo-os por décadas, arrastando-os para cima e para baixo em suas escadarias de costelas; suas brumas venenosas e seus portais dolorosos; e, as cruéis dinastias dos Originais, a espécie que clama paternidade de toda vida inteligente no universo.

Mas, eventualmente, eu ouvia aqueles tubos de oitavas mais altas cantar e acabava voltando para a mesma tarde jovial de onde havia saído.

É interessante como até a brisa do nosso planeta é uma coisa um tanto estranha para forasteiros. Ele também me contou sobre viajantes de Rath-d, que se aventuraram na Terra cinco séculos atrás, mas se dissiparam rapidamente nas nossas correntes de ar já que eles, seus equipamentos e suas armaduras eram todos feitos de carvão.

Eu tinha me sentado ao órgão, ouvindo as histórias sobre sua colônia, enquanto ele acentuava suas sinfonias e as histórias desapareciam por um tempo, até que a melodia terminasse. Ele falava sobre si e seus irmãos sendo abraçados pela cauda de sua mãe e rasgando o tecido macio da parede interna. Um sabão suculento, esponjoso e melado que limpou suas bocas e esôfagos enquanto descia. Eles mastigaram e roeram fortemente aquilo e espuma se formava. Depois de comer, eles sopraram bolhas uns sobre os outros, cada bolha cheia de uma espuma densa, sobre a qual dormiram. E de manhã bem cedo, quando a mãe os soltava do abraço, ela observava serenamente enquanto seus bebês dormiam em um ensopado de almôndegas escuras e um molho doce e grudento.

Ele soletrou todos os gostos de Endertromb. Dos seus órgãos cor de salmão engomados, que se tornavam uma pasta quando cozidos, e

Uma noite de voltagem desobstruída

Eu desenterrei esse artigo do *The Consistent Reminder*, um jornal de Connecticut que deu quatro estrelas na resenha do Dr. Cham. Midgie Dare, a resenhista do livro que subitamente estendeu seu olhar crítico a qualquer coisa tangível, elogiou o Doutor por suas maneiras e inovações na mesma edição diária em que ela difamou a meloa e diminuiu Manitoba por ter um péssimo serviço telefônico.

Eu consegui um pedaço do final do artigo dela. Aqui vai.

Ele desmontou do cavalo com um cuidado inquestionável por qualquer um que estivesse nas redondezas. Cortês de todos os lados, ele desceu da sela gentilmente, desacelerando para um ritmo que deveria ser medido em micrômetros por segundo para ser apreciado.

Seus companheiros entre nós ficaram boquiabertos, assistindo sua bota tocar o chão. Um passo tão limpo e preciso que parecia que jamais encontraria o solo, apenas deslizaria suavemente sobre ele. Então, antes que a aterrissagem fosse notada por nós, estávamos indo para a cozinha, levados pela onda de felicidade que sempre estava logo a frente de Harold Cham, e sempre logo atrás dele, e especialmente concentrada diretamente no seu próprio ser iluminado.

Ele também carregava livremente ao seu lado a filha especialmente ignorante de um estadista, que não nos deixava em paz com suas constantes críticas aos ateístas e a

seus olhos que derretiam e viravam um rico creme. Das suas manteigocias com tentáculos. E ele estava apenas começando a apreciar essas delícias, ainda criança, para ser elevado de um jardim de infância por um par de elefantes pigmeus que desceram um enorme guindaste dos céus, e o ergueram pela gola.

Eles o transplantaram para a Terra, o guiaram em seu ofício, trombeteando suas trombas alto para a cidade de Grand Rapids ouvir, então se foram, chorando e abraçando uns aos outros.

“Mas, estranhamente (em-pithy-dah), Eu aprendi, toquei (pon-shoo) nos órgãos no meu planeta (oth-reia) natal,” ele disse.

O instrutor de órgão da minha filha fala essas palavras extras que você vê em parênteses. Quem é que sabe se é sua língua nativa ou seu soluço sonoro. Ele mantém outro traço de Endertromb: ele tem doze nomes.

rotas ferroviárias.

“Em casa, meus esforços para acender uma vela foram inúteis graças a novos abalos causados por trens, que jogavam o fósforo na minha mão para perto das cortinas!” Ela ridicularizou o Dr. Cham por seu suave agarre no braço dela e sentiu ciúmes quando ele conseguiu sintonizar uma voz feminina agradável no rádio logo que eles voltaram para sua residência.

O sol se pôs, entretanto, e nós nos vimos em um deslumbrante conjunto embaixo de espessas partículas de algodão que flutuavam pela sala do piano, um tanto entretidas pelo *Programa Soneca da Tarde*, que tocava na vitrola com o volume tão baixo na estação que nós só conseguíamos ouvir o arrastar das mangas do finado Napoleão pelos lençóis. Eu sinto um calafrio só de pensar! Ainda assim, em cadeiras distantes, os dois amantes mantinham uma distância grosseira entre eles e eu me senti cercado pelo olhar acalentador do Dr. Cham enquanto bebia jocosamente sua taça de vinho xerez.

“Não, (wen-is-wen),” ele disse. “Eu tenho um nome (im-apalla) que é dito (iff) de muitas-muitas maneiras diferentes.”

Eu o chamo de Paij-ree nas manhãs e Paij-plo de tarde. Já que estou escrevendo de dia, o chamarei de Paij-ree aqui.

Protetores Auriculares Livres de Murmúrio

Então eu disse a Paij-ree, “Paij-ree, eu estou escrevendo um livro. Para ensinar ao mundo Ruby.”

“Oh, (pill-nog-pill-yacht) bacana,” ele disse. Ele conhece Ruby há mais tempo que eu, mas mesmo assim: *Eu* serei o instrutor de Ruby da minha filha.

E eu disse, “Paij-ree, você está no livro. E as histórias do seu planeta.” Eu falo com ele como se ele fosse o E.T. Eu não sei por quê. Como o que eu disse em seguida: “E daí talvez algum dia você possa ir para casa para o seu pai e para sua mãe!”

Ele replicou, “(pon-shoo) (pon-shoo) (em-pithy-dah).” Que é o jeito dele de expressar em voz alta seu silêncio e receio.

Ele queria ver o que eu havia escrito, então eu mostrei a ele este pequeno método que escrevi para você.

`def limpar_murmurios_de(frase)`



```

while frase.include? '('
  abre = frase.index('(')
  fecha = frase.index(')', abre)
  frase[abre..fecha] = '' if fecha
end
end

```

“Viu o que isto faz, Paij-ree? Qualquer índio velho pode usar este método para remover toda essa poluição incoerente das suas falas,” eu disse.

E eu alimentei o método com algo que ele havia dito mais cedo.

```

o_que_ele_disse = "Mas, estranhamente (em-pithy-dah),  

Eu aprendi, toquei (pon-shoo) em órgãos no meu planeta  

(oth-reia) natal."  

limpar_murmurios_de( o_que_ele_disse )  

print o_que_ele_disse

```

E saiu de certa forma uma frase mais clara.

```
Mas, estranhamente, Eu aprendi, toquei em órgãos no meu planeta natal.
```

“Você não deveria usar este (wary-to) loop while,” ele disse. “Existem jeitos mais amáveis, (thopt-er), e gentis.”

No método `limpar_murmurios_de`, estou basicamente procurando por abre parênteses. Quando eu encontro um, eu procuro um fecha parênteses na seqüência. Uma vez que tenha encontrado ambos, eu os troco e seu conteúdo por uma string vazia. O loop `while` continua até que todos os parênteses se acabem. Os murmurios são removidos e o método termina.

“Agora olhando este método,” eu disse. “Eu vejo que há alguns aspectos confusos e outras coisas que eu poderia ter feito melhor.” Por favor não me olhe estranho por seu professor ter escrito este código. Eu acho certo lhe mostrar algumas técnicas desleixadas para lhe ajudar a trabalhá-las comigo. Vamos lá.

Ok, **Aspecto confuso No. 1:** Este método limpa uma string. Mas e se nós acidentalmente dermos a ele um `File`? Ou um número? O que acontece? E se rodarmos `limpar_murmurios_de(1)`?

Se nós dermos ao `limpar_murmurios_de` o número 1, o Ruby irá imprimir o seguinte e depois sair.

```
NoMethodError: undefined method `include?' for 1:Fixnum
  from (irb):2:in `limpar_murmurios_de'
  from (irb):8
```

O que você vê aqui é um amiguinho especialmente tortuoso e verboso (mas às vezes muito útil) chamado **depurador**. Ele é um policial nervoso, ao menor sinal de problema, imediatamente apreende todo e qualquer suspeito, os põe contra a parede e lê os direitos deles tão rapidamente que ninguém consegue ouvir tudo. Mas está claro que há um problema. E, é claro, tudo não passou de um mal entendido, certo?

Quando o Ruby ler para você esses direitos de Miranda, preste atenção no começo. A primeira linha é quase sempre tudo que você precisa. Nessa primeira linha está contida a mensagem essencial. E ali em cima, a primeira linha está nos dizendo que não há método `include?` para o número 1. Lembra, quando estávamos conversando sobre o método `reverse` no capítulo anterior? Naquela hora, eu disse, “**vários métodos só estão disponíveis com certos tipos de valores.**” Ambos `reverse` e `include?` são métodos que funcionam com strings mas não têm sentido e são indisponíveis para números.

Sendo mais claro: o método tenta usar o número. O método vai iniciar com `frase` valendo 1. Então, ele chega na segunda linha: `while frase.include? ' ('`. Números não têm método `include?`. Maravilha, o depurador nos mostrou onde o problema está. Eu não esperava que alguém fosse passar um número, então estou usando métodos que não funcionam com números.

Viu, é só isso. Nossa método é sua própria ferramenta de bolso, certo? Ele age como seu próprio widget independente de todo o resto. Para todos por aí usando o método `limpar_murmurios_de`, caso a eles seja passado um número, será atirado para eles esta mensagem de pânico que não lhes faz sentido. Eles serão convidados a bisbilhotar dentro do método, o que realmente não é o trabalho deles. Eles não sabem como se virar lá dentro.

Felizmente, nós podemos acionar nossos próprios erros, nossas próprias **exceções**, o que pode fazer mais sentido a algum desavisado que dê um objeto errado para ser limpo.

```
def limpar_murmurios_de( frase )
```

```

unless frase.respond_to? :include?
  raise ArgumentError,
    "não posso limpar os murmúrios de um(a) #{ frase.class }"
end
while frase.include? '('
  abre = frase.index('(')
  fecha = frase.index(')', abre)
  frase[abre..fecha] = '' if fecha
end
end

```

Desta vez, se passarmos um número (de novo, o número 1), teremos algo mais sensato.

```

ArgumentError: não posso limpar os murmúrios de um(a) Fixnum
  from (irb):3:in `limpar_murmurios_de'
  from (irb):12

```

O método `respond_to?` é realmente bacana e eu imploro que você nunca se esqueça dele. O `respond_to?` checa qualquer objeto para garantir que ele contém um determinado método. Ele então retorna com `true` ou `false`. No caso acima, é checado se o objeto `frase` recebido contém algum método `include?`. Se nenhum método `include?` for encontrado, então disparamos o erro.

Você deve estar pensando por que eu usei um símbolo com o `respond_to?`. Eu usei um símbolo `:include?` invés da string '`include?`'. Na verdade, ambos funcionarão com o `respond_to?`.

Geralmente símbolos são usados quando você está passando o nome de um método ou qualquer outro construtor. É mais eficiente, chama mais atenção. O `respond_to?` pergunta ao Ruby para olhar a si mesmo e ver se um método está disponível. Estamos conversando com o Ruby, então o símbolo ajuda a denotar isto. Não é grande coisa, Ruby só reconhece símbolos mais rapidamente que strings.

Agora, **Aspecto Confuso No. 2:** Você notou como nosso método modifica a frase?

```

algo_dito = "Uma espaçonave (gith)."
limpar_murmurios_de( algo_dito )
print algo_dito

```

Você percebeu isso? Na primeira linha do código acima, a variável `algo_dito` contém a string "`Uma
espaçonave (gith).`". Mas, depois de invocar o método, na terceira linha, nós imprimimos a variável `algo_dito` e nesse momento ela contém a string limpa string "`Uma
espaçonave.`".

Como isso funciona? Como o método modifica a string? Ele não deveria fazer uma cópia da string antes de modificá-la?

Sim, absolutamente, ele deveria! **É falta de educação modificar strings daquele jeito.** Nós usamos `gsub` e `gsub!` no capítulo passado. Você se lembra qual destes dois métodos é um **método destrutivo**, que modifica strings diretamente?

Ou nós chamamos este método de `limpar_murmurios_de!` (como cortesia a todos os colegas bacanas por aí que podem vir a usar este método) ou modificar o método para trabalhar em uma cópia da string ao invés da coisa real. O que é uma modificação muito fácil! Nós só precisamos dar um `dup` na string.

```

def limpar_murmurios_de( frase )
  unless frase.respond_to? :include?
    raise ArgumentError,
      "não posso limpar os murmúrios de um(a) #{ frase.class }"
  end
  frase = frase.dup
  while frase.include? '('
    abre = frase.index('(')
    fecha = frase.index(')', abre)
    frase[abre..fecha] = '' if fecha
  end
  frase
end

```

O método `dup` faz uma cópia de qualquer objeto. Veja a linha que adicionamos separada:

```

frase = frase.dup

```

Que linha de código peculiar. Como **frase** vira uma cópia de **frase**? Ela se apaga? O que acontece com a **frase** original? Ela desaparece?

Lembre que variáveis são como apelidos. Quando você vê **frase** = "Uma espaçonave (gith).", você vê o Ruby criando uma string e então dando um apelido a essa string.

Do mesmo modo, quando você vê **frase** = **frase.dup**, você vê o Ruby criando uma nova string e então dando um apelido a essa string. Isso é útil dentro do seu método porque agora **frase** é um apelido para uma nova cópia da string que você pode usar seguramente **sem modificar a string que foi passada ao método**.

Você verá muitos exemplos de nomes de variáveis sendo reusados.

```
x = 5
x = x + 1
# x agora é igual a 6

y = "Endertromb"
y = y.length
# y agora é igual a 10

z = :include?
z = "a string".respond_to? z
# z agora é igual a true
```

E, sim, algumas vezes objetos desaparecem. **Se você não consegue chegar em um objeto por uma variável, então o Ruby vai entender que você já terminou com ele e vai se livrar dele.** Periodicamente, o Ruby envia seu **coletor de lixo (garbage collector)** para libertar estes objetos. Todo objeto é mantido na memória do seu computador até que o coletor de lixo se livre dele.

Oh, e mais uma coisa sobre o **dup**. Algumas coisas não podem ser “dupadas”. Números, por exemplo. Símbolos (que se parecem com **:morte**) são idênticos quando grafados iguais. Assim como números.

Além disso, algumas variáveis especiais: **nil, true, false**. Estas são coisas que o Ruby não vai te deixar alterar, então não faz sentido copiar mesmo. Quero dizer, imagine se você pudesse mudar o **false** para que seja **true**. A coisa toda vira uma grande mentira.

Talvez o **Aspecto Confuso No. 3** seja simples. Estou usando aquelas chaves na string. Estou a tratando como se ela fosse um Array ou Hash. Eu posso fazer isso. Porque strings têm um método **[]**.

Quando usadas em uma string, as chaves vão extrair parte daquela string. Mais uma vez, espaço para as pás da empilhadeira. A string é como uma longa prateleira e a empilhadeira tira os pedaços da string.

Tirado de Os Comedores de Cachecóis

(do capítulo VII: *Quando um Empurrão vem para Impulsionar — ou Amar.*)

“Nunca diga meu nome de novo!” gritou Chester e, com o mesmo entusiasmo, ele voltou para a tela de diálogo **Arquivo > Configurar Publicação...** para otimizar mais ainda seu filme, até míseros 15k.

Dentro das chaves, nós passamos o *index (índice)*. É a etiqueta que colocamos entre as pás, onde o trabalhador pode vê-las. Quando o assunto são strings, podemos usar vários objetos como índices.

```
str = "Uma string é uma longa prateleira de letras e espaços."
puts str[0]          # imprime 85 (o código do caractere 'U')
puts str[0...1]      # imprime 'Uma string é uma longa prateleira de letras e espaços.'
puts str[1...2]      # imprime 'ma string é uma longa prateleira de letras e espaços'
puts str[1, 3]        # imprime 'ma '
puts str['prateleira'] # imprime 'shelf'
```

Tudo bem, o último **Aspecto Confuso No. 4**: este método pode ser enviado a um loop interminável. Você pode dar

uma string para este método, o que fará com que ele trave e nunca mais volte. Dê uma olhada no método. Você consegue jogar um graveto que trave o loop?

```
def limpar_murmurios_de( frase )
  unless frase.respond_to? :include?
    raise ArgumentError,
      "não posso limpar os murmurios de um(a) #{ frase.class }"
  end
  frase = frase.dup
  while frase.include? '('
    abre = frase.index('(')
    fecha = frase.index(')', abre)
    frase[abre..fecha] = '' if fecha
  end
  frase
end
```

Aqui, entorte o graveto antes de jogá-lo.

```
graveto = "Aqui está um ( graveto entortado."
limpar_murmurios_de( graveto )
```

Por que o método trava? Bem, o loop **while** espera até que todos os parênteses abertos se acabem antes de parar de rodar. E ele só modifica um parêntese aberto que tenha um par parêntese fechado. Então, se nenhum parêntese fechado é encontrado, o parêntese aberto não será modificado e o **while** nunca ficará satisfeito.

Como você reescreveria este método? Eu sei como me virar no Ruby, então eu usaria uma expressão regular.

```
def limpar_murmurios_de( frase )
  unless frase.respond_to? :gsub
    raise ArgumentError,
      "não posso limpar os murmurios de um(a) #{ frase.class }"
  end
  frase.gsub( /\([-w]+/, '' )
end
```

Dê o melhor de si quando pensar nos seus loops. É muito fácil, especialmente para os loops **while** e **until** saírem do controle. Melhor usar um iterador. E nós chegaremos em expressões regulares na hora certa.

Resumidamente, aqui está o que aprendemos sobre escrever métodos:

1. Não se surpreenda se as pessoas passarem objetos inesperados aos seus métodos. Se você não pode usar de jeito nenhum o que lhe deram, **raise (cause)** um erro.
2. É falta de educação mudar objetos que são dados aos seus métodos. Use **dup** para fazer uma cópia. Ou encontre um método como **gsub** que automaticamente faz a cópia enquanto faz o seu serviço.
3. As chaves podem ser usadas para procurar partes dentro de qualquer objeto **Array**, **Hash** ou **String**, já que estes objetos provêm um método **[]**. E também, já que estes objetos provêm um método **[]=**, as chaves podem ser usadas para atribuição (do lado esquerdo do sinal de igual) para mudar partes destes objetos.
4. Cuidado com loops fujões. Evite **while** e **until** se você puder.

Os Mecanismos do Estereótipo

Imediatamente ouve-se um sussurro nas árvores atrás da casa do Paij-ree e acabou que era o homem que caíra do céu. Seu nome era Doug e ele



vendia gatos.

Então, assim que ele aparecer, quando sua sombra (e as sombras dos gatos amarrados aos seus pés) entrou na frente do pássaro no gramado que nós estávamos tentando acertar com uma raquete, enquanto ele retirava o hélio do grande balão, nós gritamos, “Olá, Doug!”

E ele diz, “Olá, Gonk-ree! Olá, Why!”

Paij-ree checa os bolsos para ter certeza que tem o real-e-vinte-sete que vai precisar para comprar os três gatos que precisa para manter a fornalha abastecida e a parabólica virando. Estes gatos geram montes de eletricidade estática quando Paij-ree os joga no gerador, onde eles estarão em inferioridade numérica a gigantes bastões de vidro, que os acariciam continuamente— Mas, espere! Você reparou como o corretor de gatos chamou ele de Gonk-ree?

E ele o chama de Gonk-ree de dia e Gonk-plo de noite.

Então o sufixo é definitivamente relacionado à luz do sol. Até onde sei, o prefixo indica a relação de quem chama o Paij-ree.

```
class String
```

```
# As partes do nome do professor
# de órgão da minha filha.
@@silabas = [
  { 'Paij' => 'Pessoal',
    'Gonk' => 'Negócios',
    'Blon' => 'Escravo',
    'Stro' => 'Mestre',
    'Wert' => 'Pai',
    'Onnn' => 'Mãe' },
  { 'ree' => 'AM',
    'plo' => 'PM' }
]

# Um método para determinar o
# significado de certos nomes dele.
def significado_do_nome
  partes = self.split( '-' )
  silabas = @@silabas.dup
  significado = partes.collect do |p|
    silabas.shift[p]
  end
  significado.join( ' ' )
end
```

```
end
```

Agora eu fui bem mais além do que lhe mostrar apenas código banal. Houve aqui uma grave libertinagem e um crime contra a natureza. Um crime que a maioria das linguagens não permitiriam que você cometesse. Nós estamos mudando a **String**, uma das classes do núcleo do Ruby!

“Eu sei que isto é um pouco perigoso,” Eu disse, quando passei isso por baixo do nariz do Paij-ree. “Eu espero que ninguém se machuque.”

“Todo cara pálida deve experimentar o que esse (kep-yo-iko) perigo faz,” ele disse. “Cachorros e lenhas e areia movediça (kul-ip), todos devem ser experimentados.” E ele deu uma gole no seu drink de pântano Beagle Berry.

Então o que é que eu estou adicionando à classe **String**? Duas coisas: uma variável de classe e um método. Um **método de instância** normal.

Eu gosto de ver a **arroba** como um caractere que significa **atributo**. A **arroba dupla atributo coletivo**. Uma variável de classe. Todas as instâncias da classe podem olhar para esta variável e ela é a mesma para todas. A variável **@@@silabas@** é um Array que agora pode ser usado dentro da classe String.

O novo método é **significado_do_nome** e esse novo método pode ser usado com qualquer string.

```
print "Paij-ree".significado_do_nome imprime Pessoal AM.
```

Como você pode ver, Paij-ree é um nome pessoal. Um nome que os amigos usam nas manhãs.

Tenha certeza de ter visto a linha de código que usa **self**. Esta é uma variável especial, uma variável que representa

o objeto cujo método você está chamando. Para simplificar as coisas um pouco, vamos tentar fazer um método que separe uma string pelos hífens.

```
class String
  def separar_hifen
    self.split( '-' )
  end
end
```

De novo, aqui está um método que pode ser usado com qualquer string.

"Gonk-plo".separar_hifen retorna o Array ['Gonk', 'plo'].

Usar **self** marca o a entrada em um novo estágio com idéias mais avançadas no Ruby. Esta é uma linguagem de definição. Você está definindo um método, desenhando ele antes dele ser usado. Você está preparando para a existência de um objeto que use aquele método. Você está dizendo, “Quando **separar_hifen** for usado, haverá uma string a qual estaremos separando-hífens. E **self** é uma variável especial que faz referência àquela string.”

Ruby é uma linguagem de definição imbatível. Uma discussão suculenta e de rachar a cuca estará no seu caminho nas profundezas deste livro.

Na maioria das vezes você não precisa usar **self** explicitamente, já que você pode chamar métodos diretamente dentro de outra definição de método.

```
class String
  def separar_hifen; split( '-' ); end
end
```

No método **significado_do_nome**, encontre o loop. Aprender sobre **Array#collect** é essencial. Vamos olhar de perto.

```
significado = partes.collect do |p|
  silabas.shift[p]
end
```

O Array **partes** contém o nome separado. ['Paij', 'plo'], por exemplo. Estamos iterando por cada item naquele Array com **collect**. Mas **collect** vai um passo a frente em relação ao que o **each** faz. Assim como **each**, **collect** joga cada item pela calha como uma variável de bloco. E aí, ao término do bloco, **collect** **mantém a resposta que o bloco dá e a adiciona em um novo Array**. O método **collect** é o jeito perfeito de se fazer um novo Array que é baseado em itens de um Array existente.

O Doug tem três gatos à venda. Um custa doze centavos, um sessenta e três centavos, um nove centavos. Vamos ver quanto cada gato custaria se nós adicionássemos uma gorjeta de 20%.

```
gatosegorjetas = [0.12, 0.63, 0.09].collect { |custogato| custogato + ( custogato * 0.20 ) }
```

Costumo dizer que a propriedade do Paij-ree é um charmoso pedaço de floresta quando não está chovendo gatos e Doug. Por vários dias, Paij-ree e eu acampamos em tendas perto do rio atrás da casa dele, subsistindo de fumaça de turdus e retalhando indiozinhos adormecidos ao pôr do sol. Em uma ocasião ele perdeu no carteadão e eu sabia que sua mente estava distraída, pensando em Endertrumb. Tudo isso deve estar remoendo nele por algum tempo já. Eu fui o primeiro confidente que ele tivera.

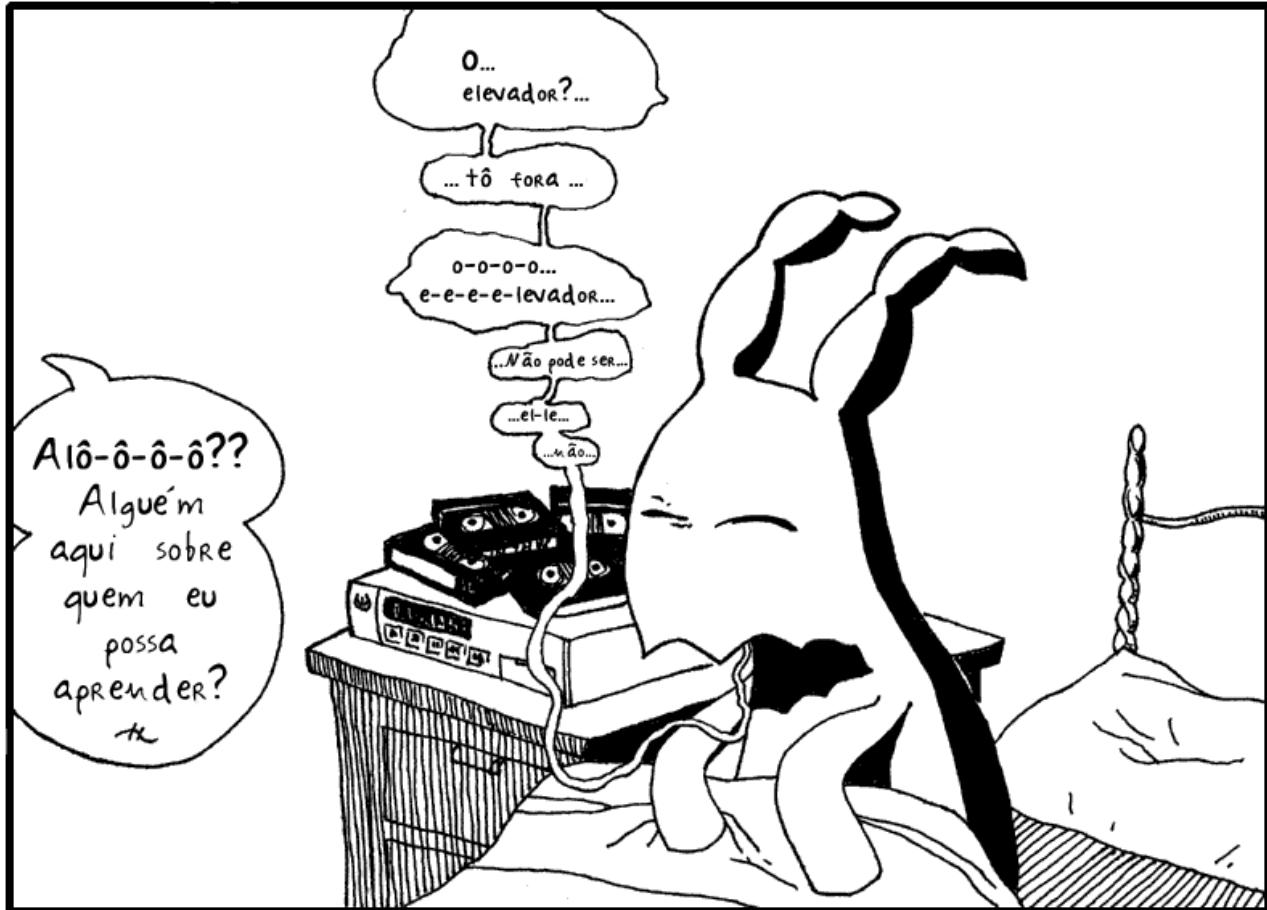
“Acabei de vir de Ambrose,” Eu disse. “É tipo minha segunda casa adotiva, um lugar onde elfos aspiram a ser animais perfeitos.”

Ele resmungou e balançou a cabeça. “Você não pode ser (poth-in-oin) parte (in) em coisas assim.”

“Você acha que nós falharemos?”

“Eu (preep) já vi isso antes,” ele disse. E daí, ele falou sobre a Loteria.

4. A Cabra Quer Assistir Um Filme Inteiro



O elevador se abriu em uma sala verde tomada por estantes e arquivos. Rolos de filme e latas de filme e fitas de vídeo por toda parte. O Dr. Cham não fazia idéia do que eram a maioria destas coisas. Tudo que ele viu foi uma grande, bagunça futurística.

Ele bradou novamente, esbarrando em estreitos becos entre estantes, “Alô-ô-ô?? Estou procurando vida inteligente, sou um viajante do espaço!” Ele tropeçou quando seu pé enroscou num videocassete. “Algum outro ser com quem eu possa me comunicar?”

Com as mãos em volta da boca, ele gritou, “Alô-ô-ô?”

“Crying out loud(*).” A cabra sonolenta veio trotando pelo corredor. (* Livro do poeta norte-americano Cid Corman, tradução: Gritando bem alto).



“Eu odeio aquele livro,” disse a cabra. “Eu acho que o autor é insincero.”

“Mesmo?” perguntou o Dr. Cham.

“Tenho certeza que tudo é verdade. Mas é muito enfeitado. Eu fico tipo: Chega. Já entendi. Pára com isso.”

“Não tenho muita certeza do que fazer com isso,” disse o Doutor. “Parece um esforço honesto. Na verdade, eu escrevi algo em Ruby lá atrás.”

“Isto não dá às cabras uma boa reputação,” disse a cabra.

“Mas você é a única cabra no livro,” disse o Doutor.

“E totalmente mal interpretada.”



A cabra fechou sua boca e Dr. Cham acalmou-se.

“Na verdade sou bastante letrada,” disse a cabra. “Embora, recentemente, eu estou vendo mais filmes. Eu amo filmes estrangeiros. Um de meus parentes acabou de trazer *Ishtar* do seu planeta. Uau, aquilo foi excelente.”

“Eu não visito o meu planeta há muito tempo. Seria difícil considerá-lo minha casa a esta altura do campeonato.”

“Bem, Warren Beatty é maravilhoso. Seu personagem é basicamente socialmente aleijado. Ele até tenta se matar, mas Dustin Hoffman senta no peitoril da janela e começa a chorar e cantar uma música totalmente hilária. É de partir o coração. Eu tenho ele aqui, você deveria assisti-lo.”

“Posso pegar algo para comer?” perguntou o Doutor. E ainda se sentia sujo.

“Que tal assistirmos um filme e você pode comer mantegocias com tentáculos?” disse a cabra.

Então, rumaram de volta ao projetor da cabra. De volta para ao compartimento do congelador, eles se sentaram em um tapete gigante e quebraram alguns apêndices de mantegocias congeladas. A casca era dura, mas depois de quebrada, havia um rico creme de frutas em abundância. Doce ao paladar e de um aroma muito agradável.

“Primeiro filme, você tem que ver,” disse a cabra. “Filmado e produzido localmente. Eu sou íntimo da moça que dirigiu o elenco. Saí com ela por um tempo. Sabia de todos que iriam protagonizar diferentes papéis muito antes que fosse anunciado.”

A cabra ajeitou o projetor perto do Dr. Cham. "A música está em surround sound. Você pode ajustar no botão."

A mente do Dr. Cham vagou nesse ponto da apresentação, bem a medida que a guerra aumentava entre as duas multidões de animais colonizadores. Os detalhes de suas guerras e campanhas continuavam a consumir o carretel de filme transparente que Dr. Cham alimentava através do projetor.

Guerra após guerra após guerra. O Cerco do Elmer Lake. A Última Estada de Newton P. Giraffe e Filhos. A Invasão Canil da Pequena Nuvem Abandonada. Nenhum animal morreu nessas guerras. Geralmente um ataque consistia em dar um tapinha na cabeça de outro animal. E eles davam petelecos nos narizes uns dos outros. Mas, acredite em mim, aquilo era humilhante.

Porca miséria. As coisas podiam ter dado certo.

O Nascimento de um Objeto

“Não se aflija,” disse a cabra, ansiosa para dissuadir a atenção do Dr. Cham de volta ao filme. “As coisas *dão* certo.”

No Ruby, o Objeto é o centro exato de todas as coisas. Ele é O Original.

```
class UrsoTostado < Object; end
```

O menor que indica **herança**. Isto significa que a nova classe **UrsoTostado** é baseada na classe **Objeto**. Todo método que **Objeto** tem estará disponível na **UrsoTostado**. Constantes disponíveis em **Objeto** estarão disponíveis em **UrsoTostado**.

Mas todo objeto herda de **Object**. O código...

```
class UrsoTostado; end
```

É idêntico a...

```
class UrsoTostado < Object; end
```

Herança é uma comodidade. Você pode criar espécies de objetos que se relacionam entre si. Frequentemente, quando está dissecando um problema, você se deparará com vários objetos que compartilham atributos. Você pode ter menos trabalho herdando de classes que já solucionam parte desse problema.

Você pode ter uma classe **BrasilEndereco** que guarda o endereço, cidade, estado, e CEP de alguém que more no Brasil. Quando você começar a guardar endereços da Inglaterra, você pode adicionar uma classe **ReinoUnidoEnderecos**. Se você então garantir que os dois endereços herdam de uma classe mãe **Endereco**, você pode planejar seu software de correspondência de modo que aceite todo tipo de endereço.

```
def envia_lhes_um_kit( endereco )
  unless endereco.is_a? Endereco
    raise ArgumentError, "Objeto Endereco não encontrado."
  end
  print endereco.formatted
end
```

Além disso, herança é ótimo se você quer sobrescrever certos comportamentos numa classe. Por exemplo, talvez você queira fazer a sua própria variação da classe **Array**. Você quer melhorar o método **join**. Mas se você mudar **Array#join** diretamente, afetará outras classes no Ruby que usam Arrays.

Então você começa sua própria classe chamada **MeuArray**, que é baseada na classe **Array** original.

```
class MeuArray < Array
  # Construir uma string neste array, formatando cada entrada
  # e depois as unindo.
  def join( sep = $,, format = "%s" )
    collect do |item|
      sprintf( format, item )
    end.join( sep )
  end
end
```

MeuArray é agora uma classe **Array** customizada com o seu próprio método **join**. **Array** é a **superclasse** (**classe-superior, classe-mãe**) do **MeuArray**. Cada objeto tem um método **superclass** onde você pode verificar esta relação.

```
irb> MinhaArray.superclass
=> Array
```

Perfeito. Nós administrarmos um hotel e nós temos uma **Array** dos tamanhos dos nossos quartos: [3, 4, 6]. Vamos imprimir isso de um jeito bacana para encadernarmos.

```
quartos = MinhaArray[3, 4, 6]
print "Nós temos quartos de " + quartos.join( ", ", "%d camas" ) + " disponíveis."
```

Que imprime, “Nós temos quartos de 3 camas, 4 camas, 6 camas disponíveis.”

Dr. Cham estava procurando por um banheiro, mas só tinham videotapes arquivados para todo lado. Ele acabou encontrando um lugar, que poderia ser um banheiro. Tinha um recipiente metálico. Mais importante que isso, era

escuro e escondido.

Enquanto ele está lá, deixe-me acrescentar que enquanto Os Originais massacraram Os Invasores para provar seus direitos como Criaturas de Primeiro Escalão, com o Objeto Ruby, não existe tal disputa. Ele é o rei Objeto absoluto, o Primeiro.

Observe.

```
irb> Class.superclass  
=> Module  
irb> Kernel.class  
=> Module  
irb> Module.superclass  
=> Object  
irb> Object.superclass  
=> nil
```

Até **Classe** é um **Objeto**! Veja, mesmo as classes sendo linguagem de definição para objetos, nós ainda podemos chamar métodos de classe nelas e tratá-las como objetos ocasionalmente. Pode parecer um círculo estonteante, mas é verdadeiramente uma descendência bastante clara. E garante que quando você altera o **Object**, você altera **tudo no Ruby**. O que é impossivelmente assustador e todo-poderoso e cataclísmico e incrível! **Ruby não lhe restringe, minha irmã, meu irmão!**

Entre **Class** e **Object**, você vê **Module**? Se **Object** é o rei, o pai de todas as outras partes do Ruby, então **Module** é uma pobre e frágil freira, abrigando e protegendo todos as suas criancinhas Ruby da cidade. (Para completar a analogia: **Class** é a professora da escola da vila e **Kernel** é o auto-proclamado importante coronel.)

O motivo da existência do **Module** é dar comida e abrigo ao código. Métodos podem ficar abrigados embaixo da manta do **Module**. **Module** pode conter classes, constantes e variáveis de qualquer tipo.

“Mas o que um **Module** faz?” você pergunta. “Como ele é lucrativamente utilizado??”

“Isso é tudo que ele faz!!” Eu replico, mostrando as palmas da mão na maior expressão de futilidade conhecida pelo homem. “Agora me escute — pois nunca mais repetirei isso — essa Mãe Módulo Superior deu a estes pobres objetos um lugar para morar!!”

```
# Veja, aqui está o module -- onde mais seu código poderia estar?  
module AtenciosaSantaInes  
  
  # Uma CONSTANTE repousa perto da porta. Ótimo.  
  HOMEM_DESDENTADO_COM_GARFO = ['homem', 'garfo', 'chicletes expostos']  
  
  # Uma Class está comendo, vivendo bem na cozinha.  
  class CriancaGordaDeCera; end  
  
  # Um Método está se escondendo lá atrás no armário banana, Deus sabe por quê.  
  def timida_garota_com_cara_de_raposa; {'por favor' => 'eu quero uma noz de carvalho por favor'}; end  
  
end
```

Agora você tem que ir pela Santa Inês para achá-los.

```
>> AtenciosaSantaInes::HOMEM_DESDENTADO_COM_GARFO  
=> ["homem", "garfo", "chicletes expostos"]  
>> AtenciosaSantaInes::CriancaGordaDeCera.new  
=> #<AtenciosaSantaInes::CriancaGordaDeCera:0xb7d2ad78>  
>> AtenciosaSantaInes::instance_methods  
=> ["timida_garota_com_cara_de_raposa"]
```

Lembre-se sempre que um **Module** é somente um hotel. Um teto sobre suas cabeças. Ele não é uma auto-consciente **Classe** e, portanto, não pode ser trazido à vida com **new**.

```
>> AtenciosaSantaInes.new  
NoMethodError: undefined method `new' for AtenciosaSantaInes:Module  
from (irb):2
```

Santa Inês deu a própria vida para cuidar desses pedaços de código desesperados. Por favor. Não tire isso dela.

Se, todavia, você quer roubar da Santa Inês, eu posso lhe ajudar. Você pode arrumar um grande mosteiro para confinar o ministério da **AtenciosaSantaInes** e então o quê sobrará para ela?

Para isso você pode usar **extend**, que puxará todos os métodos do módulo para dentro de uma classe ou um objeto.

```
>> class FundacaoDeAmorEAmparoTimeWarnerAolCitibank; end
>> FundacaoDeAmorEAmparoTimeWarnerAolCitibank.extend AtenciosaSantaInes
>> FundacaoDeAmorEAmparoTimeWarnerAolCitibank::instance_methods
=> ["timida_garota_com_cara_de_raposa"]
```

Na verdade, ninguém *roubou* da **AtenciosaSantaInes**, somente pegaram emprestado. A **timida_garota_com_cara_de_raposa** agora tem dois endereços.

Você deve admitir. O velho mosteiro pode ser comprado um zilhão de vezes e a pequena garota-com-cara-de-raposa *ainda* estará nos fundos no armário banana esperando uma noz de carvalho! Que pena que não podemos alimentá-la. Ela é um método sem argumentos.

Quando Dr. Cham retornou renovado, a tira de filme estava um pouco atrasada. Mas a cabra não notou, então o Doutor avançou alguns quadros até fazer algum sentido.



ENTÃO TODAS AS CRIATURAS ESTAVAM EM GUERRA, MUITO PARECIDA COM A GUERRA DESCrita PELOS LIVROS DE MEDICINA ENTRE OS TECIDOS ADIPOSOS E OS AMINOÁCIDOS, QUE SE DÁ CONTINUAMENTE DENTRO DO SEU CORPO...



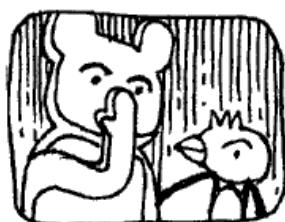
O QUE SIGNIFICA QUE VEZ OU OUTRA ELES BRIGAVAM POR COISAS ESTÚPIDAS COMO KORMO, QUE ERA BASICAMENTE UMA MARCA MUITO POPULAR DE SANDUÍCHES DE ATUM.



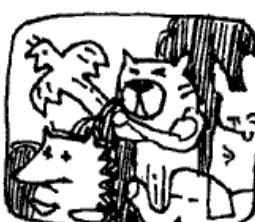
E O PREÇO DOS KITS DE LIMÃO ESTAVA ALTO.



ENTRETANTO, DOIS BODES APARECERAM, PERGUNTANDO DE QUE LADO ELES ESTAVAM. NENHUM DELES CONSEGUIA SE LEMBRAR.



OS ORIGINAIS PENSARAM QUE PROVavelmente OS BODES ESTAVAM COM ELES. BODES ESTÃO POR AI, CERTO? MAS...



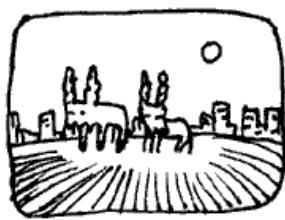
OS INVASORES JURAVAM QUE TINHAM TRAZIDO BODES!!
"VOCÊS CHEGARAM TARDE!" ELES DISseram.



E O QUE ACONTECIA NA VERDADE: UM BODE ERA O ORIGINAL E O OUTRO UM INVASOR E ELES ESTAVAM BLEFANDO. E APAIXONADOS. E MENTINDO.



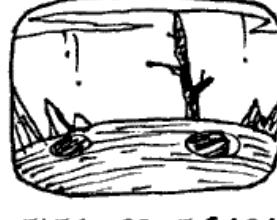
(SIM, ESSES ERAM OS PAIS DO ABSOLUTAMENTE VIDRADO AMIGO DO DR. CHAM.)



"BEM, TALVEZ," DISseram OS BODES, "TALVEZ ESTEJAMOS NO PLANETA ERRADO. ESTE PLANETA PARECE VELHO DEMAIS."



E MESMO QUE OS INVASORES NÃO ACREDITASSEM QUE OS ORIGINAIS FOSSEM OS ANIMAIS A EXISTIR... E QUE O PLANETA DELES ERA O MAIS VELHO...



ELES COMEÇARAM A VER O QUÃO DECRÉPITO ELE REALMENTE ERA!

Então os invasores deixaram o planeta.

“Este planeta é decrepito,” disse o Dr. Cham. “O castelo é bacana. Mas por dentro é um desastre.”

“O castelo parece uma projeção,” disse a cabra. “Todas as flores e brotos de maçã e até mesmo o céu. É uma projeção de baixa resolução.”

“Sim? Ele é encantador.”

“Eu acho.”



TODOS OS ANIMAIS SURGIRAM DE UM PLANETA CHAMADO LAMAPOVO (E DR. CHAM RECONHECEU ESTE PLANETA NA VIAGEM - ERA JUSTAMENTE O QUE TINHA DEIXADO)

ENTÃO ELES POUSARAM PARA HABITAR O PLANETA... MAS SANGUE JORROU DA BOCA DOS ANIMAIS...



“Está tudo errado!” disse a cabra. “Não é assim que o filme termina! Não teve sangue! O que aconteceu? O que aconteceu? Você apertou o botão errado, idiota?”

“Bem, eu não sei,” disse o Dr. Cham. Ele girou o botão para frente e para trás. Deu um tapa na lente.

“Cheque o filme! Cheque o filme!”

Dr. Cham tirou um pedaço do filme do alimentador do projetor, derretido e pingando nas extremidades.

“Maldição! Estes projetores são de qualidade! Nunca vi isso acontecer. Não pode ser.”

Caçando Uma Voz

“Eu não acho que tenha sido o projetor,” disse o Dr. Cham. “Alguma coisa voou pela tela e proferiu um gemido furioso.”

“Eu não tenho nenhuma cópia deste filme,” disse melancolicamente a cabra. “E aquela garota. A diretora de elenco. Eu nunca mais a verei novamente.”

Dr. Cham levantou-se e olhou para os largos corredores de carnificina magnética, procurando.

“Oh, ei, você deveria chamar aquela garota,” a cabra continuou. “Você poderia conversar com ela, chegar a um consenso. Diga a ela sobre mim. Não aja como se fosse meu amigo, só, você sabe, ‘Oh, aquele cara? Eh, mas que bobalhão.’”

Dr. Cham mirou a porta e saiu.

Os corredores eram todos um outro mundo de bagunça. Nos arquivos da cabra, as estantes estavam desarrumadas. No corredor, as estantes estavam completamente tortas. Pias caíam pelo teto. O Doutor se aventurou por baixo dos escombros, chutando compensados de madeira quando necessário.

“Você não deveria estar aqui fora,” disse a cabra. “Você está na propriedade de outra pessoa aqui. Um casal de elefantes pigmeus são donos disso tudo. Eles são sujeitos maus. Vão te dar uma surra com suas trombas. Elas as enrolam e simplesmente dão porrada.”

Dr. Cham tirou um arquivo do seu caminho, que caiu por uma frágil parede, e então pelo chão do cômodo ao lado. E eles o ouviram caindo por vários andares depois.

“Eu estou tentando lembrar como isso acontece no livro,” disse o Dr. Cham, enquanto andava ligeiramente pelo hall. “Aquela neblina leitosa que varreu a projeção. Nós encontramos aquela coisa.” Ele sacudiu uma maçaneta, quebrou-a. Vagarosamente, passou pela porta e lá desapareceu.

“Você tem o dom de destruir as coisas com um chute, não tem?” disse a cabra. “Paredes, portas.” A cabra chifrou a parede ao estilo Zidane. A parede estremeceu e então permaneceu em silêncio.

Então, tudo ficou calado. E sombrio.

A cabra ficou firme no gélido corredor, esperando Dr. Cham revirar algumas mesas e emergir, pronto para continuar a partir do cômodo que ele estava. Mas o Dr. Cham não voltou, e a cabra optou por compartilhar um momento com os destroços negligenciados deixados por seus vizinhos. Não que ela pudesse velos de verdade. Ela podia apenas escutar os sussurros ocasionais das pilhas de faturas, cópias mestre de papel carbono e envelopes manilha quando deslocava suas pernas.

O chão parecia entortar-se bem perto da cabra, como se as pilhas de tralhas ao seu redor estivessem começando a deslizar em direção a seu peso. Ele estaria no centro deste redemoinho de documentação elefantística. Morreria ele primeiro pelos cortes causados pelo papel? Ou ele ficaria sufocado sob o sólido funeral de suprimentos de escritório?

Uma luz suave, no entanto, pairou sobre ele. Uma luz prateada, flutuante. Não, era uma — eram tesouras? As tesouras cresceram em um cintilante agrupamento de pão inteligente, cada fatia sufocada de glitter. Mas, não, eram mãos. E um chapéu de Páscoa.



Em outro cômodo, Dr. Cham parou silenciosamente sobre o vidro claro. O teto abruptamente ficou transparente, então a luz das estrelas cobriu suas calças e jaqueta. Ele andou adiante até o centro do cômodo em cores escuras, iluminado tão suavemente quanto um manuscrito antigo em sua própria caixa em um museu. Mais estrelas, mais grupos de algodão em chamas, apareciam a medida que ele vinha pelo chão. E ele apareceu na visão logo em seguida, ele esperava que fosse maior, mas não era.

Terra. Como um ovo colorido, ainda fresco. Ele sentiu longas cordas de violoncelo cantando como que na sua espinha. Como podiam chamá-la de Pessoas enlameadas? Aqui estava uma lâmpada vibrante e gramada. Aquela grande bola em que tinha algo acontecendo para ele.

Ele pensou nas Chacretes. Realmente, ele sentia saudade das Chacretes. Que boas dançarinas elas eram. Ele chegou a gritar algo para as Chacretes quando as viu. Algo muito submisso e bajulador.

Oh, sim, enquanto As Chacretes giravam, braço com braço, ele gritara, “Círculos concêntricos!” Ninguém prestou atenção.

E esse pensamento foi o bastante para alimentar o complexo de superioridade do Dr. Cham. Ele fez um sorriso de pateta e deu alguns passos para trás. Ele sentiu verdadeiramente a sua genialidade em tal frase. Ela estava no fato dele perceber a simplicidade de um círculo. Ele refletiu sobre isso por todo o caminho de volta ao corredor.

O que eu acho ótimo. Adore-se quando você tiver um segundo.



“Oh, certo,” disse a cabra. “Sua sobrinha. A sobrinha que você matou. Eu estou com você agora.”

Por alguns momentos, eles ficaram se entreolhando. Tempo bastante para que ambos o Dr. Cham e a cabra pensassem:
Oh, sim. Hannah nos causa muitos aborrecimentos. Ela já está falando sobre rosquinhas de bordo.

“Ela sempre começa falando sobre rosquinhas de bordo assim na lata?” perguntou a cabra.

“Sim, ela faz isso,” disse o Doutor. “Ela traz isso para você, depois ela traz para mim. Ela vê uma rosquinha de bordo em algum lugar — Eu não me lembro muito bem onde.”

“Eu estou vendendo uma rosquinha de bordo de verdade?” Hannah disse. “Eu preciso de uma de verdade.”

“Ok, ok,” disse a cabra. “Sim, eu lembro: agora é a parte que ela diz que se ela conseguir uma rosquinha de bordo de verdade, ela será uma pessoa real novamente. Porque o real destino dela era ter uma padaria e você arruinou isso e agora ela está condenada a ser um fantasma.”

“Hey, isso é verdade!” Hannah uivou.

“É terrível que tenhamos que fazer essa cena inteira de novo,” disse o Doutor. “As rosquinhas não são materiais. Elas devem ser todas deixadas de fora.”

“Cara, estou tendo *problemas* para lembrar tudo desse capítulo,” disse a cabra. “Eu nem me lembro como sair desse corredor. Eu devo ter lido aquele livro tipo trinta vezes. Nós entramos destruindo uma parede? Nós gritamos até que alguém nos ache?”

“Nós fizemos a Hannah flutuar pelas paredes e ela encontra algum tipo de máquina,” disse o Dr. Cham. “Eu tenho que escrever um programa — e tudo dá certo de algum modo.”

“Mas, você sabe o que eu estou dizendo?” disse a cabra. “Eu esqueço todos os detalhes. Especialmente dos capítulos anteriores. Quer dizer, eu lembro do final perfeitamente. É complicado ficar sentando vendo tudo isso. O final é tão melhor.”

Dr. Cham flexionou seus braços balançou seu calcanhar. “O porco-espinho.” Sorriu ambiciosamente para a cabra.

“Oh, com certeza. O porco-espinho é definitivamente quem eu quero conhecer,” disse a cabra. “Eu me pergunto o quê ele fará com todo aquele dinheiro depois que o livro terminar.”

Dr. Cham acenou respeitosamente. “Estou doido para ver ele de chinelos.”

“Aqueles chinelos infernais!” disse a cabra e gaguejou grosseiramente, um banho de saliva jorrando de suas mandíbulas.

A mente de Hannah estava chacoalhando, perturbada, esperando que essas tolices cessassem por um momento. Ela inclinou sua cabeça de lado e o chocalho deslizou junto da curva de seu crânio. De qualquer modo, o barulhinho sumiu, a medida que a parte de trás de sua cabeça desaparecia (*vazava* é como ela chamava isso). Então a sua cabeça estava novamente com seu pequenino chocalho e ela se encontrou fazendo mais uma vez aquele indiscreto gemido.
HRRRRRRR-RRR-OH-RRRR-RRRR.

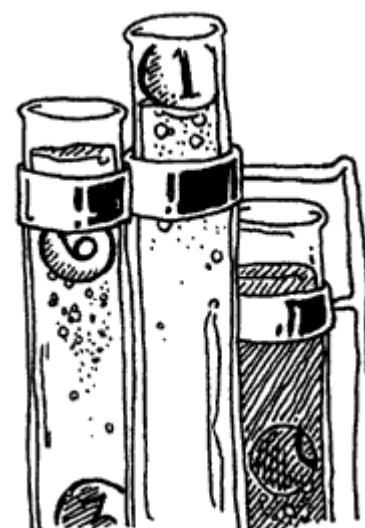
“Eu não estou nessa de bacon pedaçudo” disse a cabra. “Não vejo o quê é tão legal nisso.”

Podia ela falar gemendo? **BON-BON.** Com um sotaque francês. **BOHN-BOHN.** **BOHN-APPE-TEET-OHHHHH-RRRR.**

“Eu sei que ela é inofensiva, mas esse som me dá calafrios. Meu cabelo está **completamente** em pé.”

“Hannah?” disse o Dr. Cham. “Onde você está, criança? Venha e volte para nós, minha sobrinha.”

Ela estava bem perto deles, dentro e fora. E eles podiam ouvi-la limpando a voz, clara, falando como um anjo esparramando poeira estelar. Sim, toda a estória da rosquinha de bordo veio de novo, e mais sobre a padaria que ela seria dona, os muffins, pãezinhos e baguetes.



5. O Roubo do Capitão Loteria

E agora, as histórias das Loterias Pajj-ree.

Em Endertromb, o pai do organista inventou a loteria. A idéia surgiu quando ele estava rezando para o Escavador Dosh.

Escavador Dosh é uma espécie de Deus para eles. Mas dez vezes mais assustador. Esse cara cavou um túnel infinitamente

profundo através do planeta e saiu morto. Mas ele não está realmente morto. Ele está realmente *um segundo* atrás deles. E ele come tempo.

É meio complicado, porque o Escavador Dosh mata totalmente as pessoas. Mas acho que se você faz o que ele diz, não é tão ruim. Talvez eu fale sobre isso mais tarde. É tão doloroso falar sobre isso por ser algo tão assustador. Além disso, um dos meus amigos realmente acredita em toda a coisa. Eu fico meio que chocado — não como se estivesse chorando, mas como se estivesse engasgado.

De qualquer forma, enquanto rezava, três números vieram para o pai do Paij-ree.

Ele então se perguntou. “O que são esses números?”

E sua mente passou um pequeno vídeo clipe dele vendendo todos os tipos de números. E, por anos e anos, viajando e vendendo números.

E indagou ao seu cérebro, “Pessoas comprarão números?”

E seu cérebro disse, “Se comprarem os três números corretos, dê a eles um prêmio.”

Foi quando ele se imaginou saltando de uma rampa de ski e jogando presentes para as pessoas. Sem dúvidas: ele seria um ícone.

Então ele foi e fez o que seu cérebro disse e vendeu números. A simples loteria do pai consistia em três únicos números, retirados de um conjunto de 25 números.

```
class BilheteLotaria

  INTERVALO_NUMERICO = 1..25

  attr_reader :numeros_escolhidos, :data_compra

  def initialize( *numeros_escolhidos )
    if numeros_escolhidos.length != 3
      raise ArgumentError, "três números devem ser escolhidos"
    elsif numeros_escolhidos.uniq.length != 3
      raise ArgumentError, "os três escolhidos devem ser diferentes"
    elsif numeros_escolhidos.detect { |p| not INTERVALO_NUMERICO === p }
      raise ArgumentError, "os três escolhidos devem ser números entre 1 e 25."
    end
    @numeros_escolhidos = numeros_escolhidos
    @data_compra = Time.now
  end

end
```

Sim, a classe `BilheteLotaria` contém os três números (`@numeros_escolhidos`) e o momento em que o bilhete foi adquirido (`@data_compra`). O intervalo de números permitidos (de **um a vinte e cinco**) é mantido na constante `INTERVALO_NUMERICO`.

O método `initialize` pode ter qualquer número de argumentos passados. O **asterisco** no argumento `numeros_escolhidos` significa que **qualsquer argumentos serão passados como um Array**. Ter os argumentos em um Array significa que métodos como `uniq` e `detect` podem ser usados juntos aos argumentos.

Esta classe contém duas definições: o método de definição (`def`) e um de definição de atributos (`attr_reader`). Ambos são, realmente, **apenas métodos de definição**.

O `attr_reader` é um atalho idêntico à escrever o este código Ruby:

```
class BilheteLotaria
  def numeros_escolhidos; @numeros_escolhidos; end
  def data_compra; @data_compra; end
end
```

Atributos são métodos empacotadores (wrappers) para variáveis de instância (assim como `@numeros_escolhidos`) que podem ser usados **fora da própria classe**. O pai do Paij-ree queria codificar uma máquina que pudesse ler números e a data da compra do bilhete. Para fazer isso, estas variáveis de instância devem ser expostas.

Vamos criar um bilhete aleatório e ler os números de volta:

```
bilhete = BilheteLoteria.new( rand( 25 ) + 1,
                             rand( 25 ) + 1, rand( 25 ) + 1 )
p bilhete.numeros_escolhidos
```

Executando o código acima, eu apenas pego: [23, 14, 20]. Você receberá um erro se acontecer dos dois números aleatórios serem idênticos.

Entretanto, eu não posso mudar os números escolhidos no bilhete de loteria de fora da classe.

```
bilhete.numeros_escolhidos = [2, 6, 19]
```

Eu receberei um erro: `undefined method `numeros_escolhidos='` Isto porque `attr_reader` somente adiciona um método de **leitura**, não um método de escrita. Entretanto, isso é bom. Nós não queremos que os números ou a data mudem.

Então, os bilhetes são *objetos*. Instâncias da classe `BilheteLoteria`. Faça um bilhete com `BilheteLoteria.new`. Cada bilhete tem as suas variáveis de instância `@numeros_escolhidos` e `@data_compra`.

O capitão loteria teria necessidade de gerar três números aleatórios no encerramento do sorteio, por isso vamos acrescentar um método de classe conveniente para geração aleatória de bilhetes.

```
class BilheteLoteria
  def self.novo_aleatorio
    new( rand( 25 ) + 1, rand( 25 ) + 1, rand( 25 ) + 1 )
  end
end
```

Oh, não. Mas nós temos um estúpido erro que aparece quando dois dos números aleatórios são idênticos. Se dois números são iguais, o `initialize` lança um `ArgumentError`.

O truque é voltar e reiniciar o método se um erro acontecer. Nós podemos usar o `rescue` (*resgate*) do Ruby para apanhar o erro e `redo` (*refazer*) o método do início.

```
class BilheteLoteria
  def self.novo_aleatorio
    new( rand( 25 ) + 1, rand( 25 ) + 1, rand( 25 ) + 1 )
  rescue ArgumentError
    redo
  end
end
```

Melhor. Pode demorar algumas vezes para que os números caiam juntos corretamente, mas cairão. A espera fará o suspense, não?

O capitão loteria mantinha um registro de todos que compraram bilhetes, junto com os números de loteria deles.

```
class SorteioLoteria
  @@bilhetes = {}
  def SorteioLoteria.compra( cliente, *bilhetes )
    unless @@bilhetes.has_key?( cliente )
      @@bilhetes[cliente] = []
    end
    @@bilhetes[cliente] += bilhetes
  end
end
```

Yal-dal-rip-sip foi o primeiro cliente.

```
SorteioLoteria.compra 'Yal-dal-rip-sip',
                      BilheteLoteria.new( 12, 6, 19 ),
                      BilheteLoteria.new( 5, 1, 3 ),
                      BilheteLoteria.new( 24, 6, 8 )
```

Quando chega a hora do sorteio da loteria, o pai do Paij-ree (o capitão loteria) adiciona um bit de código para

selecionar aleatoriamente os números.

```
class BilheteLoteria
  def pontuacao( final )
    contador = 0
    final.numeros_escolhidos.each do |nota|
      contador += 1 if numeros_escolhidos.include? nota
    end
    contador
  end
end
```

O método `pontuacao` compara um `BilheteLoteria` com um bilhete aleatório, o qual representa a combinação vencedora. O bilhete aleatório é passado através da variável `final`. O bilhete recupera um ponto para todo número vencedor. A pontuação final é retornada pelo método `pontuacao`.

```
irb> bilhete = BilheteLoteria.new( 2, 5, 19 )
irb> vencedor = BilheteLoteria.new( 4, 5, 19 )
irb> bilhete.pontuacao( vencedor )
=> 2
```

Com o tempo você verá o quanto brilhante é Paij-ree. Seu pai o encarregou de conduzir a loteria por ele, à medida que a demanda por bilhetes consumia todas as horas do dia do capitão da loteria. Você consegue imaginar o jovem Paij-ree – em seu terno pomposo – brincando com elásticos com seus dedos jovens, em meio à reunião da empresa na qual propôs a peça final do sistema? Com certeza, quando se levantou, seu pai falou tudo para ele, mas foi ele quem ligou o projetor e fez todos os gestos com as mãos.

```
class << SorteioLoteria
  def jogo
    final = BilheteLoteria.novo_aleatorio
    vencedores = {}
    @bilhetes.each do |comprador, lista_bilhetes|
      lista_bilhetes.each do |bilhete|
        pontuacao = bilhete.pontuacao( final )
        next if pontuacao.zero?
        vencedores[comprador] ||= []
        vencedores[comprador] << [ bilhete, pontuacao ]
      end
    end
    @bilhetes.clear
    vencedores
  end
end
```

Os sócios de seu pai ficaram atordoados. O que era aquilo? (Paij-ree sabia que era apenas a definição de um método de classe — eles se sentiram completamente desmoralizados quando ele disse isso). Eles não conseguiam entender as **duas setas para esquerda**. Sim, era um concatenador, mas o que estaria ele fazendo em meio à definição da classe?

Bebês, pensava Paij-ree, embora mantivesse bem alta a auto-estima de cada um destes homens. Ele era apenas uma criança e crianças são duras como uma parede de tijolos.

O operador `<<` permite a você alterar a definição de um objeto. Paij-ree simplesmente havia usado a classe `SorteioLoteria`, e o método dele, `jogo`, seria um método de instância normal. Mas, uma vez que usou o operador `<<`, o método `jogo` será adicionado diretamente à classe, como um método de classe.

Quando você viu `class << obj`, acredite em seu coração, *eu estou adicionando diretamente para a definição de obj*.

O embrião do instrutor de órgão também lançou um truque na sintaxe digno de ser examinado. Na nona linha, um vencedor foi encontrado.

```
vencedores[comprador] ||= []
vencedores[comprador] << [ bilhete, pontuacao ]
```

A sintaxe `||=` é um atalho.

```
vencedores[comprador] = vencedores[comprador] || []
```

O pipe duplo é um **ou** lógico. Faça `vencedores[comprador]` igual a `vencedores[comprador]` ou, se `vencedores[comprador]` é nulo, faça-o igual a `[]`. Este atalho é um pouco estranho, mas se você puder abrir sua mente, ele é um bom método para economizar tempo. Você está se certificando que uma variável está inicializada antes de usá-la.

```
irb> SorteioLoteria.jogo.each do |vencedor, bilhetes|
irb>   puts vencedor + " ganhou com " + bilhetes.length + " bilhete(s)!"
irb>   bilhetes.each do |bilhete, pontuacao|
irb>     puts "\t" + bilhete.numeros_escolhidos.join( ', ' ) + ": " + pontuacao
irb>   end
irb> end

Gram-yol ganhou com 2 bilhetes(s)!
 25, 14, 33: 1
 12, 11, 29: 1
Tarker-azain ganhou com 1 bilhetes(s)!
 13, 15, 29: 2
Bramlor-exxon ganhou com 1 bilhetes(s)!
 2, 6, 14: 1
```

Mas estes dias de inocência não continuaram para Paij-ree e seu pai. Seu pai, frequentemente, se esquecia de lavar seu uniforme e acabou contraindo uma micose em seus ombros. A doença afetou gradualmente seu equilíbrio e seu senso de direção.

Seu pai ainda tentou futilmente manter os negócios funcionando. Ele rodou a cidade, algumas vezes andando, de forma deplorável, passo-a-passo sobre os paralelepípedos, muitas vezes tateando as paredes, contando os tijolos até o salão dos matemáticos e a estação dos cocheiros, onde ele teria empurrado bilhetes de loteria aos transeuntes, que o cercaram e o esbofetearam com grandes beterrabas molhadas. Mais tarde, Paij-ree iria encontrá-lo em uma esquina, seu sangue escorrendo nos esgotos da cidade junto aos sucos das escuras, respingantes beterrabas, cujo suco preencheu o caminho até suas veias e coagulou firmemente como um amontoado exército de brake lights lutando por um lugar nos pedágios.

Uma Palavra Sobre Accessors (Porque Eu Te Amo e Espero Por Seu Sucesso e Meus Cabelos Estão Raleando Com Isto e Os Sonhos Realmente Se Tornem Realidade)

Anteriormente, eu mencionei que `attr_reader` adiciona um método `reader`, mas não um método `writer`.

```
irb> bilhete = BilheteLoteria.new
irb> bilhete.numeros_escolhidos = 3
NoMethodError: undefined method `numeros_escolhidos=' for
#<BilheteLoteria:0xb7d49110>
```

Que está correto neste caso, visto que o pai do Paij-ree não queria que os números fossem alterados depois que o bilhete fosse vendido. Se estivéssemos interessados em ter variáveis de instâncias que teriam ambos os métodos `reader` e `writer`, deveríamos usar `attr_accessor`.

```
class BilheteLoteria
  attr_accessor :numeros_escolhidos, :data_compra
end
```

Que faz exatamente o mesmo que este código comprido:

```
class BilheteLoteria
  def numeros_escolhidos; @numeros_escolhidos; end
  def numeros_escolhidos=(var); @numeros_escolhidos = var; end
  def data_compra; @data_compra; end
  def data_compra=(var); @data_compra = var; end
end
```

Santos gatos! Olhe para estes métodos escritores por um momento. Ele são chamados métodos `picks=` e `purchased=`. Estes métodos **interceptam atribuições externas** à variáveis de instância. Constantemente você irá deixar `attr_reader` ou `attr_accessor` (ou mesmo talvez `attr_writer`) fazer o trabalho por você. Outras vezes, você mesmo irá querer tomar conta da portaria, checando as variáveis mais detalhadamente.

```
class CompeticaoSkate
```

```

def o_vencedor; @o_vencedor; end
def o_vencedor=( nome )
  unless nome.respond_to? :to_str
    raise ArgumentError, "O nome do vencedor deve ser uma String,
      não um problema matemático ou uma lista de nomes ou qualquer outra
      coisa parecida."
  end
  @o_vencedor = nome
end
end

```

Na maior parte das vezes você não vai querer usar isto. Mas, assim que formos caminhando através das lições, você irá perceber que Ruby possui muitas escotilhas e vielas nas quais você pode se enfiar e hackear código. Também estou lhe preparando para metaprogramação, que, se puder farejar o dragão, está ameaçadoramente próxima.

Paij-ree era um jovem e audaz Endertronaltoek. Ele martelava ossos de animais em longos, deslumbrantes trompetes com orifícios profundos que eram conectados por cortiças que os músicos uniam a seus dedos. Obviamente ele vendeu apenas três destas unidades, mas ele estava amplamente injuriado pois um acadêmico autônomo, um tipo demônico, para ele era de uma classe mais pobre e os pobres só conseguiam seu brilhantismo através de práticas satânicas. Claro, estavam certos, de fato, ele tinha um acordo com os magos negros, com quem ele mantinha encontros anuais, sofrendo atormentadoras primaveras ardentes, banhando-se enquanto eles lançavam feitiços.

Ele adorava seu pai, mesmo sabendo que seu pai deteriorara em nada mais que um giroscópio. Ele idolatrava o trabalho do homem e gastava seus pequenos ganhos jogando na loteria. Ele adorava assistir os numerais, cada um pintado sobre bolas de argila ocas, levantadas do *robloch* (que é qualquer tipo de fluido, reservatório ou entorno que vem resistir à presença de fantasmas), os grandes banqueiros amarrando-as num cordão de prata, lendo-as em ordem.

Mesmo hoje. Paij-ree pinta as cenas com pinceladas cruas de tinta preta em folhas de papel alumínio. É bastante tocante ver o que ele recuperou na preciosidade de sua memória, mas não sei dizer exatamente porque ele o faz em folhas de papel alumínio. Seus desenhos rasgam com bastante facilidade. O próprio Paij-ree fica confuso e lhe servirá bolo de pão usando esta arte, mesmo após ela ter sido corretamente emoldurada. Então, são tantas coisas sobre ele que são problemáticas e absurdas e absolutamente miseráveis.

A doença espalhou-se sobre a forma de seu pai e ervas pantanosas cobriam as mãos e a face de seu pai. O musgo empurrava sua espinha para uma rígida verticalidade. Tão espesso era o turvo sobre sua cabeça que ele parecia vestir um arbusto num chapéu de lançador. Ele também chamava a si mesmo por um novo nome — **Quos** — e ele curava as pessoas em que tocava, deixando uma pilha de vilas cheias de vida no seu caminho a medida que viajava pelas cidades. Muitos o chamavam de Musgossias e choravam sob seus pés, o que molhava os brotos e o levava a germinar no solo. Isto o deixava momentaneamente irritado, ele duramente chacoalhava suas pernas para libertar-se e selvagemente levava suas mãos ao céu, trazendo um trovão de fragmentos de luz sobre estes patéticos

Paij-ree estava distante das odisséias espirituais de seu pai (de fato, pensou o homem morto), então ele apenas viu a decadência da loteria sem a presença de seu capitão. É neste ponto em que Paij-ree foi ao trabalho, revivendo a falecida loteria de sua família.

Outro Trecho de Os Comedores de Cachecóis

(do Capítulo VIII: Na Altura do Céu.)

“Eu conheço você,” disse Brent. “E conheço seus cronogramas. Você não poderia ter feito esse arquivo Flash.”

“Então, você está dizendo que sou previsível?” disse Deborah. Ela abriu suas mãos e os palitos de batatas cambalearam, como pequenas lontras do mar felizes e bêbadas, para dentro da panela elétrica aberta.

“Você é muito linear,” disse Brent. Ele pegou uma lapiseira, segurou-a bem diante de seus olhos, encarando-a firmemente antes de devolvê-la ao estojo no balcão. “Você sabe pelo menos como carregar uma cena? Como pular quadros? Este filme que vi estava em todo lugar, Deb.”

Ela empilhou cinco cachecóis tricotados e uma única bandana na panela elétrica e ajustou para a temperatura máxima. Ela fechou a tampa, deixando sua mão repousar sobre esta.

“Sobre o quê é este filme?” Indagou Deborah. “Você visita sites em Flash todo o tempo. Você jogou o jogo do Gnomo Bola de Neve por dois segundos, ele não lhe interessou. Você nem mesmo ligou para os jogos de Boliche do Gnomo. E você também não ficou intimidado por aquele jogo em flash Acerte o Pingüim. Elfo versus Pingüim? Nem pergunte!”

“Agora vem este filme e você não pode ir com calma.” Ela se aproximou e insinuou-se perto dele. “Ei, mano, sou eu. Deborah. O que aconteceu quando você viu aquele filme?”

“Tudo,” disse Brent, seus olhos refletindo um milhão de mundos. “E: nada. Ele começou com uma jovem garota montando num javali selvagem. Ela tocava gaita. A música da gaita fluía com desconforto, insegura. Mas ela montava naturalmente, como se montar um

Jogando com Poucos Dedos

A cidade estava repleta de pessoas que perderam o interesse na loteria. O tempo tinha realmente exaurido a todos. Aquela terrível chuva alagando seus porões. Toda a cidade foi forçada a mover uma estória adiante. Você colocaria a tampa de volta em sua caneta e então arruinaria a caneta, visto que a tampa já estava repleta de lama. Todos estavam esgotados, muitas pessoas afogadas.

Paij-ree gastava seus dias numa quadriliche, o único móvel que permanecia acima do nível dor mar. Ele dormia na cama de cima. A terceira cama também estava seca, então ele deixava uma gaivota desabrigada de sua cratera fazer seu ninho sobre a cama. A gaivota não precisava de toda a cama, então Paij-ree também mantinha seus lápis e calculadores nela.

De início, estes foram tempos muito sombrios para ambos, e eles persistiram em continuar com má aparência o tempo todo. Paij-ree tornou-se obcecado por suas unhas, mantendo-as longas e intocadas, enquanto o resto dele deteriorava sob uma vasta cabeleira. Na companhia de Paij-ree, a gaivota da cratera aprendeu sua própria excentricidade e arrancava todas suas penas do lado direito de seu corpo. Ela se parecia com um diagrama em corte.

Eles aprenderam a ter épocas felizes. Paij-ree entalhou uma flauta a partir da parede usando suas unhas e a tocava frequentemente. Na maioria das vezes ele tocava suas canções relaxantes durante o dia. À tarde, eles batiam na parede e chacoalhavam a estrutura da cama em ritmo com sua música. A gaivota ficou doida quando ele tocou quatro notas determinadas repetidamente. Ele assistia a gaivota em êxtase voando em círculos. Paij-ree mal podia manter sua compostura dado o efeito que tinha aquela pequena melodia. Ele não se agüentava, babando de tanto rir.

Paij-ree chamava a gaivota de *Eb-F-F-A (Mi-bemol-Fá-Fá-Lá)*, depois desta música favorita.

A amizade pode ser um excelente catalisador para o progresso. Um amigo pode encontrar em você características que nenhuma outra pessoa consegue ver. É como se ele procurasse sua pessoa e de algum jeito encontrasse cinco jogos completos de talheres de prata que você nunca soube que estiveram lá. E mesmo que este amigo não entenda porque você tem estes talheres escondidos, é, ainda assim, um grande feito, digno de respeito.

Enquanto *Eb-F-F-A* não encontrava a prataria, ele havia encontrado uma outra coisa. Uma pilha de uma outra coisa. Visto que Paij-ree estava abandonado sobre a cama quádrupla, a gaivota faria uma exploração em busca de alimento. Um dia ela voou até um barril, flutuando sobre onde ficava o galpão de ferramentas. *Eb-F-F-A* andou sobre o barril, rolando-o até a casa de Paij-ree e então o quebrou, revelando a coleção perdida de bicos de pato.

Sim, bicos de pato de verdade. (*Eb-F-F-A* estava esofagizando seus graxos. mantendo-se calmo, sugando as gotas de suor de volta para sua testa — patos não eram da mesma pena que ele, mas mesmo assim da mesma família.) Paij-ree aplaudiu com alegria, absolutamente, ele pretendia cobrir sua casa com estes bicos, eles poderiam ter contido um pouco da torrente. Provavelmente não muita coisa, nenhum motivo para reclamações.

E a cola do teto estava ao pé do barril e eles eram dois companheiros de beliche empreendedores com tempo livre, então eles fizeram uma balsa com os anteriormente quacáveis lábios. E para fora foram eles, para o interior! Movendo-se através de uma total mistura de cidade e sopa. Quão estranho era atingir uma praia e então descobrir que era apenas a velha e suja estrada que passava pelo trevo de Toffletown.

No interior, eles vendiam. Era sempre uma longa caminhada até a próxima plantação, mas existiam alguns poucos compradores na mansão (“Bem-vindo a Mansão Erguida com Besouros”, diziam eles ou, “A Mansão Erguida com

javali selvagem não fosse grande coisa. E com Flash, montar um javali selvagem realmente não é grande coisa.”

Deborah soltou sua pulseira e a colocou no balcão próximo a panela elétrica.

“A borda do filme começou a desmanchar-se, uma poça de tinta se formou. O javali levantou-se, mas suas pernas deram lugar para a toda negra, tinta pulverizada.”

“Nuvens negras se encontraram. Música hardcore começou a tocar. Agentes secretos vieram das nuvens. Caras da CIA e coisas assim. A animação simplesmente detonava.

“E então, no finalzinho do filme, estas palavras esmaecem na tela. Em letras brancas, negrito.”

“Na altura do céu,” disse Deborah.

“Como você sabia?” tremeram os lábios de Brent. Poderia ela ser confiável?

“Não há mais espaço no mundo,” ela disse. “Não há espaço para comedores de cachecóis, não há espaço para você e eu. Aqui, segure minha mão.”

Substitutos do Celofane — você não sabe quão perigoso pode ser o celofane verdadeiro?”) E uma das famílias embrulhavam sobras de gelatina e presunto em algum celofane para os dois viajantes. E eles quase morreram no dia seguinte por causa delas.

Então, quando veio o calor e, como a primeira loteria do interior era à noite, um fazendeiro chamou por eles do seu terreno, enquanto ele estava próximo de uma vaca sua que pastava. Paij-ree e Eb-F-F-A foram até ele, murmurando um ao outro se deveriam oferecer-lhe o Bilhete Especial Gasto pelo Vento ou se ele preferiria optar em ganhar o Arriscado Medalhão Original Caseiro Interiorano do Roco.

Mas o fazendeiro lhes acenou a medida que se aproximavam, “Não, deixem suas calculadoras e rodas de probabilidades. É para a minha papa-capim.” Ele quis dizer sua vaca. A versão de Emdertromb: o dobro de carne, duas vezes mais carnuda, não produz leite, produz chapas de papel. Ainda assim, ela pasta.

“Sua papa-capim (poh-kon-ic) quer um bilhete da sorte?” indagou Paij-ree.

“Ela viu vocês e ficou realmente empolgada,” disse o fazendeiro. “Ela não sabe números, mas ela entende um pouco sobre sorte. Quase foi acertada por um avião senil um dia e, quando a encontrei, ela apenas levantou os ombros, como se dissesse, ‘Bem, acho que não foi nada’”

“Toda (shas-op) loteria é numér-(iga-iga)-ica,” afirmou Paij-ree. “Ela sabe (elsh) notas musicais? Minha águia sabe (losh) notas musicais.,” disse Paij-ree. “Paij-ree assobiou para a gaivota da cratera e ela respondeu com um *D* (*Dó*) contínuo.

O fazendeiro não podia falar com a consciência musical de sua papa-capim, então Paij-ree enviou a gaivota para descobrir (*D-D-D-A-D* (*Ré-Ré-Ré-Lá-Ré*), *vá-ensinar-a-vaca*) enquanto ele hakeava algumas melodias em sua calculadora.

```
class BilheteLoteriaAnimal

# Uma lista de notas válidas.
NOTAS = [:Ab, :A, :Bb, :B, :C, :Db, :D, :Eb, :E, :F, :Gb, :G]

# Armazena as três notas escolhidas e as datas de compra.
attr_reader :escolhidas, :comprado

# Cria um novo bilhete a partir das três notas. As três notas
# devem ser únicas.
def initialize( nota1, nota2, nota3 )
  if [nota1, nota2, nota3].uniq!
    raise ArgumentError, "as três escolhidas devem ser diferentes entre si"
  elsif escolhidas.detect { |p| NOTAS.include? p }
    raise ArgumentError, "as três escolhidas devem ser notas da escala cromática."
  end
  @escolhidas = escolhidas
  @comprado = Time.now
end

# Faz a contagem do resultado (escore) do bilhete contra o resultado do sorteio.
def escore( final )
  contador = 0
  final.escolhidas.each do |nota|
    contador +=1 if escolhidas.include? nota
  end
  contador
end

# Construtor para criar um BilheteLoteriaAnimal aleatoriamente
def self.novo_aleatorio
  new( NOTAS[ rand( NOTAS.length ) ], NOTAS[ rand( NOTAS.length ) ],
       NOTAS[ rand( NOTAS.length ) ] )
rescue ArgumentError
  redo
end

end
```

Não há necessidade de os bilhetes para animais se comportarem drasticamente diferente dos bilhetes tradicionais. A

classe **BilheteLoteriaAnimal** é internamente diferente, mas expõe os mesmos métodos vistos na classe **BilheteLoteria**. O método **escore** é até mesmo idêntico ao método **@score** da antiga classe **BilheteLoteria**.

Ao invés de usar uma variável de classe para armazenar a lista de notas musicais, elas são armazenadas numa constante chamada **BilheteLoteriaAnimal::NOTAS**. Variáveis mudam e a lista de notas não deveria. Constantes foram projetadas para se manterem. Mesmo assim você pode alterar a constante, mas terá que trapacear ou o Ruby irá chiar.

```
irb> BilheteLoteriaAnimal::NOTAS = [:ASSOBIO, :GORJEIO, :BALIDO]
(irb):3: warning: already initialized constant NOTAS
=> [:ASSOBIO, :GORJEIO, :BALIDO]
```

A gaivota voltou com a papa-capim, o nome dela era Merphy, ela estava empolgada em tentar a sorte, ela inflou seu rosto sonhador, assobiou cinco ou seis notas em série, todas elas seguraram seu colar, puxaram-na para perto da calculadora e a deixaram respirar três notas, então eles estrangularam o último ar até que seu bilhete estivesse impresso e tudo estava catalogado certinho dentro de `@@@bilhetes['merphy']`. Obrigado, vejo você no sorteio!

Então a febre da loteria tornou-se uma epidemia entre as simples mentes dos animais. Paij-ree pouparia algum custo, usou a mesma classe **SorteioLoteria** que ele usaria no ambiente comum da loteria de sua infância. E logo os animais estavam fazendo sua própria música, seus próprios mapas e filmes.

“E Os Originais?” Perguntei a Paij-ree. “Eles devem ter odiado seus animais!”

Ele porém recuou com mau humor e franziu sua testa. “Eu sou um Original. Assim como você. Nós odiamos (ae-o) algum deles?”

Não muito tempo após o término da loteria, Paij-ree sentiu a gaivota da cratera *Eb-F-F-A* descendo em seu ombro, ela então assobiou um urgente e triste *C-Eb-D-C-A-Eb* (*Dó-Mi Bemol-Ré-Lá-Mi Bemol*). Estas notas desesperadas enviaram uma onda de calafrios através do corpo de Paij-ree. Tinha o Deus Rei de Bêbado Solo. Nosso Amado Topiário, o **Musgossias Quos**, Pai Literal Daquele Homem Quem Era O Instrutor de Órgão da Minha Filha — tinha ele realmente chegado ao seu fim? Como isto poderia acontecer? Poderiam as grandes árvores não mais nutri-lo e guiar os ventos úmidos em sua direção? Ou seu próprio esguio líquen cobriram seu caminho e cresceram obstruindo sua respiração?

Você não liga dizia a canção da gaivota. *Ele deteriorou-se e enfraqueceu e caiu na porta iluminada de seu casebre. Seus rebentos torcendo e chorando para que o dia não acabasse. Para que o sol permanecesse fixo e longe e atencioso.*

Plor-ian, o criado da casa, deixou que os jarros viesssem e Quos manteve-se bem hidratado até que Paij-ree chegou para avaliar os caídos botões de plantas macias e o emergente rosto de seu pai, o capitão da loteria. Sua pele profundamente marcada como um travesseiro cheio de bordados; Grandes galhos saltavam-lhe dos braços agora retraídos com grande sede.

Paij-ree penteou as longas raízes que rodeavam os olhos de seu pai e aquelas que vinham do canto de sua boca. Enquanto eu gostaria de lhe dizer que as lágrimas de Paij-ree rolavam até os braços e poros de seu pai, rejuvenescendo e recuperando o gramado cavalheiro: Não posso dizer isto.

Na verdade, as lágrimas de Paij-ree rolavam até seus braços e caiam no chão de tábuas rangentes, alimentando as desprezíveis ervas daninhas, energizando a matéria negra das plantas, que literalmente saltavam através do chão à noite e sufocavam Nosso Quos. Puxão, rasgo, crack. E este era seu esqueleto.

Depois disso Paij-ree nunca mais poderia ser chamado de Wert-ree ou Wer-plo.

6. Eles que Fazem as Regras

Hannah saltou de volta a parede e apoiou-se em seus dedos.

“Está é a parede,” disse Dr. Cham. “Os Originais estão aí dentro. Minha filha, você pode levar-nos ao convés de observação?”

“Você espera que a gente vá contra estes caras? indagou a cabra. “Eles são doidos como coalas. Mas estes coalas tem lasers!”

“De qualquer forma, nós vencemos. disse Dr. Cham. “Você e eu sabemos disso.”

“Ok, bem, estou confuso em relação a isso”, disse a cabra. “Nós realmente vencemos? Ou podemos estarmos

pensando sobre o *Kramer contra Kramer*? Quem venceu foi o Dustin Hofmman ou fomos nós?”
“Na na ni na não.” Hannah flutuou e nervosamente arrastou suas pernas junto da parede. “Tem um homem com uma cara enorme lá!”

“O Sr. Cara,” disse o Doutor. “Ele é o rosto original.”

“Ele não me viu,” disse Hannah e lamentou-se. **HOMA-HOMA-ALLO-ALLO.**

Ela fez aquele furo gotejante através dos gastos buracos de rato e das portas de congeladores, fluindo para dentro e para fora, levando os pontos de controles do vídeo a assobiar e os painéis da parede a se atarem a si mesmos e desabarem no silêncio. Os três passaram através dos dois desgastados níveis de segurança e emergiram no convés de observação tendo uma vista panorâmica do compartimento de cargas.

“Os últimos vivos entre Os Originais,” disse Dr. Cham. “Você está certa disso, Hannah?” O que, de qualquer modo, ela não escutou, a medida que seus olhos repousavam fixamente nas lendárias criaturas.

“Olhe para eles”, falou a cabra. “Estes caras escreveram o livro das regras, Doutor. Nós devemos tudo a estes caras.”

“E Deus?” disse Dr. Cham.

“Eu realmente não sei,” disse a cabra. “Hannah provavelmente sabe melhor sobre isto do que qualquer um de nós.”

Hannah nada disse. Ela apenas conhecia um outro fantasma e este era seu Mediador Pós-Falecimento, Jamie Huft. Quem parecia não ter qualquer resposta para ela e exigia que as questões fossem enviadas em escrito com um selo auto-endereçável incluso no envelope. Hannah não tinha ainda visto a bola rolar naquela caixa postal.

“Devemos estar no alto das montanhas”, afirmou a cabra. “Vejam só essa escuridão.”

“Eu vi um outro convés como este lá embaixo onde encontramos Hannah,” disse Dr. Cham. “Lá embaixo, perto da sua área de moradia. Você deveria gastar algum tempo para procurá-lo. Lá é bastante pacífico. Você pode ver a Terra e os sete mares.”

“Os sete mares?” A cabra imaginou se isso era próximo ao grupo as Rockettes. Ela leria sua parte no material sobre dança de precisão e veria aquela linha de pernas, caminhado a paços miúdos sobre o palco como um grande, e brilhoso preparador de solo.

Hannah movimentou-se com vida.

E nenhum dos três falou quando Os Originais desligaram o projetor de slides e embarcaram num foguete bastante fino que passou fulminante com clareza através de uma rachadura no teto do compartimento de cargas.

“Minha nossa,” disse a cabra.

“O quê?” perguntou Hannah.

“Você vai morrer,” falou a cabra.

Dr. Cham olhou para os controles a frente deles, um longo painel de alavancas acolchoadas e telas verdes.

“Eu já estou morta. Sou um fantasma.”

A cabra encarou o Doutor, que estava explorando o painel de controle. “Ok, bem, se o seu tio não tiver uma conversinha com você, eu vou deixar as coisas bastante claras. Existe uma grande chance de que estes caras vão construir uma bomba. E você vê como estou inquieto? Você vê como meus joelhos estão tremendo?”

“Ahan.”

“Ahan, é o quão real é isto aqui, garota. Eu não me lembro de nada do *livro maldito* exceto que estes caras estão construindo uma bomba que pode explodir o mundo fantasma. E, uma vez que o mundo fantasma se for, então o Escavador Dosh terá seu segundo retorno. É um negócio que eles calcularam. Inferno, é algo doentio, isto é tudo que você precisa saber.”

“Mas estou morta.”

“Ok, bem, nós estamos conversando, não estamos? Você pode falar, então você está morta?” A cabra chacoalhou a cabeça. “Eu gostaria de me lembrar se nós vencemos ou se foi mesmo o Dustin Hoffman.”

Hannah berrou. “Por que eu tenho que morrer de novo? Ela choramingou, suas pernas estremeceram em descontrole e

ela caiu no chão. **MOH-MOHHH-MAO-MAOOO.**

Dr. Cham tinha puxado com muita força uma alavanca felpuda, o que destravou e fez deslizar, como um porta pão, a tampa de um compartimento. Ele colocou suas mãos lá dentro e encontrou um teclado firmemente preso.

"É isso," disse ele e colocou `irb`, que apareceu numa tela a esquerda de seu oculto teclado. Ele checou a versão do Ruby.

```
irb> RUBY_VERSION  
=> "1.8.6"
```

O Ruby estava atualizado. O que mais ele podia fazer? Escanear `constantes` e `class_variables` era inútil. A única razão de ter funcionado com a classe `Elevador` era porque alguém tinha deixado o `irb` rodando com suas classes ainda carregadas.

Ele tinha acabado de carregar este `irb` e nenhuma classe especial estava disponível. Ele tinha que encontrar algumas classes. Ele começou carregando o arquivo `'rbconfig'` para ter uma idéia de quais eram as configurações do Ruby.

```
irb> require 'rbconfig'  
=> true  
irb> Config::CONFIG  
=> {"abs_srcdir"=>"$(ac_abs_srcdir)", "sitedir"=>"bay://Ruby/lib/site_ruby", ... }
```

Lá havia muita informação para se analisar. A constante `Config::CONFIG` é um Hash que contém cada configuração do ambiente usada para ajustar o Ruby. Você pode encontrar o nome do sistema operacional em `Config::CONFIG['host_os']`. O diretório onde as bibliotecas principais do Ruby são armazenadas pode ser encontrado em `Config::CONFIG['rubylibdir']`. Os programas Ruby podem armazenar arquivos auxiliares em `Config::CONFIG['datadir']`.

De qualquer maneira, O que Dr. Cham realmente precisava era uma lista de todas as bibliotecas que não eram bibliotecas principais do Ruby. Bibliotecas que foram instaladas pelos Originais ou quem quer que tenha manejado este console. Ele checou algumas variáveis globais por esta informação.

```
irb> $"  
=> ["irb.rb", "e2 mmap.rb", "irb/init.rb", ... "rbconfig.rb"]  
irb> $:  
=> ["bay://Ruby/lib/site_ruby/1.9", "bay://Ruby/lib/site_ruby/1.9/i686-unknown",  
     "bay://Ruby/lib/site_ruby", "compartimento://Ruby/lib/1.9",  
     "bay://Ruby/lib/1.9/i686-unknown", "."]
```

Aha, ótimo. Dr. Cham coçou sua barba e olhou para sua sessão o `irb`. A variável global `$"` contém um Array de cada biblioteca que foi carregada com `require`. A maioria destas bibliotecas foram carregadas pelo `irb`. Embora ele havia carregado '`rbconfig.rb`' a pouco.

A variável global `$:`, que pode também ser acessada como `$LOAD_PATH`, contém uma lista com todos os diretórios em que Ruby irá checar quando você tentar carregar um arquivo com `require`. Quando Dr. Cham rodou `require 'rbconfig'`, Ruby verificou nesta ordem cada um dos diretórios.

```
bay://Ruby/lib/site_ruby/1.9/rbconfig.rb  
bay://Ruby/lib/site_ruby/1.9/i686-unknown/rbconfig.rb (*)  
bay://Ruby/lib/site_ruby/rbconfig.rb  
bay://Ruby/lib/1.9/rbconfig.rb  
bay://Ruby/lib/1.9/i686-unknown/rbconfig.rb  
.rbconfig.rb
```

O segundo caminho era onde o Ruby acabaria por encontrar o arquivo `rbconfig.rb`. Dr. Cham supôs que os primeiros cinco caminhos eram **caminhos absolutos**. Estes eram caminhos para diretórios num disco chamado **compartimento**. Caminhos absolutos podem variar em seu sistema. No Windows, caminhos absolutos começam com a letra do disco. No Linux, caminhos absolutos começam com uma barra.

O diretório `". "` indicava um **caminho absoluto**; O ponto solitário representa o diretório de trabalho atual. O diretório onde Dr. Cham inicializou o `irb`. Então, depois de Ruby ter procurado em todos locais padrão, ele checou o diretório atual.

A cabra deu uma espiada com sua cabeça ao redor de Dr. Cham e assistia todas estas instruções exalarem, a medida que ele lambia seus lábios para manter suas salivações longe dos monitores e dos brilhosos botões. Ele havia soltado alguns regozijos (nas linhas de: *Não, isso não* ou *Sim, sim, isso mesmo* ou *Ok, bem, você escolhe*), mas agora ele

estava completamente envolto, recomendando código, “Tente `require 'setup'` ou, não, tente `3 * 5`. Tenha certeza que a matemática básica funciona.”

“Claro que a matemática funciona,” disse Dr. Cham. “Deixe comigo, eu preciso encontrar algumas classes úteis.”

“É um simples teste de sanidade,” disse a cabra. “Apenas tente. Faça `3 * 5` e veja o que aparece.”

Dr. Cham cedeu.

```
irb> 3 * 5  
=> 15
```

“Ok, muito bom! Estamos dentro!” a cabra chacoalhou alegremente sua cara peluda.

Dr. Cham deu um tapinha na cabeça, “Ótimo. Agora podemos continuar.”

```
irb> Dir.chdir( "bay://Ruby/lib/site_ruby/1.9/" )  
=> 0  
irb> Dir[".*.{rb}"]  
=> ['endertromb.rb', 'leitormental.rb', 'fazedorpedidos.rb']
```

Dr. Cham tinha usado `chdir` para mudar o diretório atual de trabalho para o primeiro caminho listado no `LOAD_PATH`. Este primeiro caminho em `site_ruby` é um lugar comum para armazenar classes personalizadas.

Aqui estavam as três classes lendárias que o instrutor de órgão da minha filha havia anotado para mim anteriormente neste capítulo. E, Dr. Cham, tendo lido o dito capítulo, reconheceu estas três peças do sistema imediatamente.

A classe `Endertromb`, que continha os mistérios dos poderes deste planeta. A classe `LeitorMental` que, escaneando as mentes de seus habitantes, lia o conteúdo de cada mente. E, finalmente, a classe crucial `FazedorPedidos` que permitia a realização de desejos de dez letras, caso o desejo alguma vez encontrasse um caminho para o núcleo de Endertromb.

“O que você acha de `4 * 56 + 9`?” perguntou a cabra. “Não sabemos se isso pode fazer expressões compostas.”

“Eu tenho o `LeitorMental` bem aqui,” disse Dr. Cham. “E eu tenho o `FazedorPedidos` bem aqui do lado dele. Este planeta pode ler mentes. E este planeta pode fazer pedidos. Agora, vamos ver se ele pode fazer ambos ao mesmo tempo.”

7. Eles Que Viveram o Sonho

Apesar da arte dos Originais ter desaparecido há muito, Dr. Cham trabalhou freneticamente no computador acoplado ao painel de controle sobre o convés de observação. Hannah havia desaparecido dentro do chão (ou talvez aquelas pequenas faíscas ao longo do caminho eram ainda rastros de sua presença paranormal!) e a cabra amigavelmente assistiu Dr. Cham hackear um módulo Ruby.

```
require 'endertromb'  
module EscanerPedidos  
  def escanear_por_um_pedido  
    desejo = self.read.detect do |pensamento|  
      pensamento.index('pedido: ') == 0  
    end  
    desejo.gsub('pedido: ', '')  
  end  
end
```

“Qual seu plano?” perguntou a cabra. “Parece que eu poderia resolver este problema em umas três linhas.”

“Este `Módulo` é a nova tecnologia de `EscanerPedidos`,” disse ele. “O escâner só pega um desejo se ele começa com a palavra `pedido`, um ponto e um espaço. Deste modo o planeta não será preenchido com cada palavra com menos-de-dez-letras que aparecem na cabeça das pessoas.”

“Por que você simplesmente não usa uma classe?” indagou a cabra.

“Porque um `Módulo` é mais simples que uma classe. É apenas um aparato de armazenamento para métodos. Ele mantém um grupo de módulos juntos. Você não pode criar novos objetos a partir de um método.”

“Mas você não vai querer um objeto `EscanerPedidos`, para que você possa realmente usá-lo?” disse a cabra, chocada.

“Eu vou mesclá-lo com o **LeitorMental**,” disse Dr. Cham. E assim o fez.

```
require 'leitormental'
class LeitorMental
  include EscanerPedidos
end
```

“Agora, o módulo **EscanerPedidos** está mesclado ao **LeitorMental**,” disse Dr. Cham. “Posso chamar o método **escanear_por_um_pedido** em qualquer objeto **LeitorMental**. ”

“Então, é um mixin,” disse a cabra. “O mixin **LeitorMental**. ”

“Sim, qualquer módulo que é incluso numa classe com um **include** é um mixin para esta classe. Se você voltar e olhar para o método **escanear_por_um_pedido**, verá que ele chama um método **self.read**. Eu apenas tenho que ter certeza de que, qualquer que seja a classe com que estou mesclando o **LeitorMental**, ela tenha um método **read**. Caso contrário, ocorrerá um erro.”

“Isto parece bastante estranho, o fato do mixin precisar de certos métodos que ele já não tem. Parece que ele deveria funcionar por ele mesmo.”

Dr. Cham moveu seus olhos do teclado para a cabra. “Bem, é como sua coleção de vídeos. Nenhuma das suas fitas de vídeo funciona ao menos que você as ponha numa máquina que use fitas de vídeo. Eles dependem um do outro. Um mixin tem alguns requisitos básicos, mas uma vez que uma classe atinge estes requisitos, você pode adicionar toda esta funcionalidade extra a ela.”

“Hey, isso é legal,” disse a cabra.

“Você leu o livro trinta vezes e não pegou isso?” perguntou Dr. Cham.

“Você é bem melhor professor pessoalmente,” disse a cabra. “Eu não pensei que fosse gostar tanto assim de você.”

“Eu comprehendo completamente,” disse o Doutor. “Isto é muito mais real do que as tirinhas fazem sê-lo.”

```
require 'fazedorpedidos'
leitor = LeitorMental.new
pedidor = FazedorPedidos.new
loop do
  pedido = leitor.escanear_por_um_pedido
  if pedido
    pedidor.realize(pedido)
  end
end
```

Irb sentou e rodou o laço na tela. Ele fará isso até que você pressione Control-C. Mas Dr. Cham o deixou rodando. Rodando o laço indefinidamente, escaneando as ondas mentais por um pedido adequado.

E Dr. Cham terminou seu pedido. No começo, ele pensou imediatamente em um **garanhão**. Para cavalgar em pelo sobre os vales de Sedna. Mas ele recuou com o pensamento, seu pedido não havia sido formado corretamente. Um garanhão era inútil para perseguir Os Originais, então ele fechou seus olhos novamente, mordeu seu lábio e pensou com si mesmo: **pedido: baleia**.

Última Baleia para Pessoas enlameadas

A atarracada e mal humorada baleia apareceu na entrada do castelo, onde Hannah estava chacoalhando um botão de rosas com sua mão. Ela golpeou a Baleia com a mão, mas esta apenas permaneceu perfeita, confortável e firme contra o céu sólido e azul de Endertromb.

“Estou entediada,” disse ela a baleia. **BOHR-BOHR-OHRRRRRR**.

“Ok,” disse a baleia, profundamente e suavemente. Enquanto a palavra deslizava junto de sua pesada língua, suas bordas dividiam-se e a palavra saia polida e vestida numa bolha pelo canto de sua boca.

“Eu sempre tenho que morrer,” disse a jovem fantasma. “As pessoas sempre me matam.”

A baleia sacudiu suas curtas nadadeiras, que se encontravam a uma distância inútil do solo. Então, ela se moveu com esforço em direção a menina com sua cauda. Lançando-se sobre os pedaços de grama.

“As pessoas matam, então, quem elas matam?” disse a menina. “Eu. Elas me matam todas as vezes.”

A baleia conseguiu chegar a três metros da garota onde se erguia como um grande monumento de guerra que representava um número suficiente de soldados mortos a ponto de trazer um ruidoso passo até você. E agora a baleia descansava sua cauda e, exausta pela escalada até então, deixou suas pálpebras se cerrarem tornando-se uma levemente inchada montanha de barro. Sua farta sombra dobrou-se em volta da quase imperceptível Hannah.

Mas outra sombra apareceu, estreita e determinada. Bem atrás dela, a mão veio sobre seu ombro, e o fantasma quente dentro da mão tocou sua manga.

“Como você chegou aqui embaixo?” disse a garota.

Dr. Cham sentou bem do lado dela e o bode deu a volta e sentou em frente.

“Escute aqui,” disse Dr. Cham. “Nós temos que seguir este grupo malvado de vândalos até o fim, Hannah. E para apanhá-los, nós precisamos de sua fiel ajuda!”

“Estou com medo,” chorou Hannah.

“Você não está com medo,” disse o bode. “Vamos. Você é uma criança fantasma intimidante.”

“Bem,” ela disse. “Estou um pouco entediada.”

Dr. Cham se ajoelhou, trazendo sua presença cabeluda para o chão, seu rosto a apenas alguns centímetros do dela. “Se você vier conosco, se você pode acreditar no que sabemos, então poderemos pegar esta trupe revoltante. Agora, você diz que seu destino é ser uma confeiteira. Não vou entrar no mérito. Você tem todo o direito do mundo — e de Endertromb, no caso — de se tornar uma confeiteira. Digamos que, se você não se tornar uma confeiteira, será uma grande tragédia. Quem é que vai cuidar de todos aqueles donuts se não for você?”

Ela deu de ombros. “É isso que eu venho dizendo.”

“Você está certa,” disse o Doutor. “Você vem dizendo isso desde o começo.” Ele olhou para o céu, onde o vento assobiava pacificamente apesar da furada forçada do foguete dos Originais. “Se o seu destino é ser uma confeiteira, então o meu é acabar com tudo isso, acabar com a confusão que está começando a se formar. E escute-me, menina — ouça o quanto certa e sólida a minha voz fica quando eu digo isto — eu acabei com sua vida, eu tenho total responsabilidade por sua vida como uma aparição, mas eu a trarei de volta. Vai levar mais tempo que um donut, mas você vai ter uma infância real. Eu prometo.”



Claro, levou um minuto para o bode cortar seu pedido para dez letras, mas ele logo estava a caminho, seguindo os mesmos rastros no céu, até o Dr. Cham e sua sobrinha fantasma Hannah. Até o grupo de animais vilanescos chamado de Os Originais. Até as Rockettes.

E o Escavador Dosh batia e festejava a cada segundo que eles deixavam para trás.

Centro da Cidade



Concentrados em seus envolvimentos no plano de expansão de “Os Originais”, as raposas alta e a bem mais baixa tinham viajado direto para a zona de alerta vermelho, a cidade de Wixl. Eu desejava uma espátula para colocá-las juntas lado à lado, arrastá-las para a costa próximo as incubadoras da praia, escondendo-as nos montes de ovos de peixes, puxando-as para baixo através de suas enormes orelhas, ocultando assim suas luxuosas peles. E acima delas eu ficaria parado formando uma sombra imóvel, mantendo-as na mira do meu rifle.

Eu não posso. Eu tenho você para ensinar. Eu tenho que me arrumar e cuidar de mim. As lâmpadas lá de cima precisam ser trocadas. Uma caixa de lâmpadas halógenas simplesmente apareceram pelo correio. Alguém lá fora obviamente está tentando me fazer usá-las. Sendo assim eu vou instalá-las o quanto antes. E depois ficarei lá, formando uma sombra imóvel, segurando meu rifle.

Aquela sombra deveria ser legal e bem definida, sendo assim vou deixá-la assim mesmo.

1. Se Eu Estivesse Procurando Por um Veículo



Eu gosto de ver estes dois em estado selvagem. Eles ficam bastante entendidos aqui no estúdio. Começaram

fazendo slogans estranhos e coisas assim. Eles tinham frases que insistiam em repetir, formando fixações. Você não pode se expôr a todas estas besteiras de raposa.

Vamos apenas dizer: Eu realmente estou tentando dar o meu melhor para manter as coisas num nível acadêmico. Como nunca estive na faculdade, não posso dizer ao certo se cada passagem escrita está de acordo com o severo critério exigido academia.

Eu tenho vários amigos universitários, alguns que viajam o mundo em suas buscas, e eu tento inflexionar minha voz com a mistura de alta cultura deles.

Às vezes eu aplaudo a mim mesmo por ir além do trabalho dos meus amigos educados – apenas em corredores vazios, nós nunca somos arrogantes em público – porque na verdade eu me inscrevi em uma escola intelectual enquanto eles ainda estão em seus livros, virando e virando.

Eu sou um “pré-eventualista” (para saber mais: <http://preeventualist.org/>). Eu tenho me interessado por isto por tempo suficiente e me orgulho disto ter acontecido. Inevitavelmente, alguns de vocês já devem ter começado a vasculhar este livro atrás de simbologia Marxista. Sinto muito em matar estas interpretações, mas acredito que qualquer conclusão niilista que você possa ter, será mantida sobre vigilância.

De qualquer modo, vou deixar a retórica de lado. Eu apenas menciono o “pré-eventualismo” porque além de ser uma alternativa renovada e fácil ao pós-modernismo com que nascemos, isto oferece um serviço de achados e perdidos gratuito para os residentes de Wixl.

```
require 'open-uri'

open( "http://preeventualist.org/lost" ) do |perdido|
  puts perdido.read
end
```

Não tenho como alertar os raposos sobre este serviço. E tenho certeza que ainda é muito cedo para que seu caminhão seja listado. Ainda assim, as boas intenções estão aqui.

Se você está conectado à Internet, o código Ruby acima deve ter baixado a página da Internet e impresso na tela uma mensagem que se parece com isto: (Parece que a página em questão foi alterada, o resultado será o conteúdo da página atual.)

REGISTRO DE ACHADOS E PERDIDOS DOS "PRÉ-EVENTUALISTAS" (um serviço gratuito para os ESCLARECIDOS que foram ILUMINADOS)

atualizações são feitas diariamente, por favor, verifique!

este serviço é comissionado e subsidiado
em parte pelo Clã de Estudo Jovem
Ashley Raymond

...
todos os selos e privilégios são apresentados
sob a notável autoria de Perry W. L. Von Frowling,
Magistrado Polywaif de Desapropriação. Também vencedor sete vezes
seguidas da respeitada Copa Comunitária do Mendigo Insistente
...

SOBRE O REGISTRO

=====

Olá, se você é novo, por favor fique conosco. A seguir, uma breve explicação sobre nossos serviços.

Em primeiro lugar, algumas notícias do nosso amado magistrado.
(As crianças o chamam de Tio Von Besteiroto. Haha!)

NOTÍCIAS IMPORTANTES

=====

/ 15 de Abril de 2005 /

oi, grandes novidades. estivemos no canal 8 em wixl e ordish. o cory viu. eu e o jerry mathers aparecemos.
se você não viu, mande um e-mail pro cory. ele conta o melhor. tudo o que posso dizer é que aquilo não são os movimentos das MINHAS mãos!! (piada para quem assiste o canal 8.)
obrigado harry e toda a equipe de notícias do canal 8!!

- perry

/ 7 de Abril de 2005 /

todos nós estamos lidando com o carpete aqui na central, mas se vocês puderem ficar de olho na prancheta da caitlin, ela está muito sossegada para publicar, e eu sei que isto é realmente

importante para ela. Ela tinha algumas radiografias panorâmicas bem caras dos dentes tortos de

seu marido, que estavam anexas à algumas fotos insubstituíveis do seu marido com traje de robocop,

na época em que seus dentes tortos eram mais visíveis, ela disse (para mim), "Eles saberão o que eu

quis dizer quando virem isto". eu não sei o que isso significa :)

eu verifiquei: * a mesa da frente * o corredor * a sala de espera * o banheiro * a dispensa
* a

grande sala de TV * o balcão * a sala de alunos * a antiga sala do gaff (aquele com a

pintura de pé
de cereja) * o quarto de empregada * escadaria * eu irei atualizar isto, assim que eu encontrar
mais cômodos.

- com amor, perry

/ 25 Fev 2005 /
servidor caiu às 3 horas. estou puto assim como vocês caras. o gaff tá lá embaixo e ele vai ficar lá
até consertar. :0 -- ATUALIZAÇÃO: consertado, de volta na ativa!!

- perry

/ 23 de fevereiro de 2005 /
eu sei que tem muito barulho hoje. o circo stanley bros perdeu doze lhamas, um trailer, um monte de
cadeados e cinco tendas. eles ainda estão procurando as coisas perdidas. por favor mantenham suas
cabeças no lugar, eu preciso da ajuda de todos. esses comediantes não têm nada. eu falo sério.
hoje eu dei um adesivo roxo para um cara (é só uma coisa que eu gosto de fazer como gesto de solidariedade) e ele praticamente dormiu nele e nele cultivou os ingredientes para molho de pizza.
eles estão no fundo do poço. então doem por favor. eu sei que não temos paypal nem nada. se você quiser doar, divulgue que você encontrou alguma coisa (uma bicicleta de criança, uma compra do mês) e que isso tenha o nome do pessoal do circo escrito nele ou algo do gênero.

- ótimo, perry

/15 de novembro de 2004 /
promoção do dia dos "esperançosos". se você perdeu algo hoje, pode escolher um item grátis (no valor de \$40 ou menos) da casa de alguém que encontrou algo.
nós estamos nos divertindo tanto com isso!! é EXATAMENTE assim que eu consegui minha máquina de remar no ano passado e eu a AMO!

- perry

Acho que o Youth Study Clan está fazendo um ótimo trabalho com esse serviço. Está meio piegas e meio basicão, mas se conseguir fazer os animais pararem de usar seus meios mais instintivos de demarcar território, então eu tiro meu chapéu.

Mesmo assim, um grupo de jovens pré-eventualistas? Como é que pode? Você tem que ter pelo menos *flertado com cinismo de verdade* antes de poder se tornar um pré-eventualista. E definitivamente, não pode ir ao colégio. Portanto, eu não sei.

Voltando à lista de instruções do Registro de Achados e Perdidos Pré-eventualista.

USANDO O SERVIDOR A&P

O A&P é um serviço gratuito. Os atos de perder e achar são qualidades essenciais para a construção de um estilo de vida pré-eventualista. Esperamos atender a sua crença.

Não usamos HTML, para simplificar as coisa por aqui. Nossa pessoal já trabalha quinze horas por dia. (Valeu, Terk!! Valeu, Horace!!)

Você pode usar nosso serviço para procurar coisas que você perdeu. Ou pode adicionar coisas que você perdeu (ou achou) ao nosso registro. Isso é feito digitando o endereço correspondente no seu browser.

PESQUISANDO

Para procurar coisas perdidas, use o seguinte endereço:

http://preeventualist.org/lost/search?q={palavra_chave}

Você pode substituir {palavra chave} pela sua pesquisa. Por exemplo, para pesquisar por "xícara"

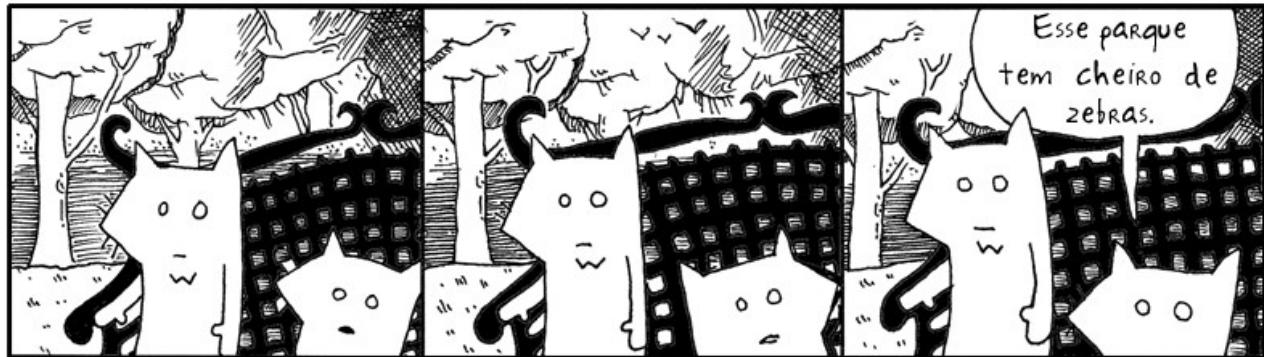
<http://preeventualist.org/lost/search?q=xicara>

Assim, você receberá uma lista das xícaras que foram achadas ou perdidas.

Se você quer pesquisar somente por xícaras perdidas ou somente por xícaras achadas, use as páginas 'searchlost' e 'searchfounds':

<http://preeventualist.org/lost/searchlost?q=xicara>

Não estou de brincadeira. Eu sei onde está o caminhão. Sério, não estou só provocando. Vou mostrar em um segundo. Só estou dizendo, olhe os raposos:



Elas estão indefesas. E mesmo assim, aqui está essa ótima ferramenta. Possivelmente, a chave para sair dessa confusão. Só

quero dar uma fuçada, ver se há alguma pista por aqui.

```
require 'open-uri'
```

```
# Pesquisar todos os itens achados contendo a palavra 'caminhao'.
open( "http://preeventualist.org/lost/searchfound?q=caminhao" ) do |truck|
  puts truck.read
end
```

Não vejo nada sobre o caminhão da raposa grande nessa lista. Tudo bem. Os raposos estão perdidos mesmo. Temos algum tempo.

Você aprendeu uma técnica simples para recuperar uma página da Internet. O código usa a biblioteca **OpenURI**, que foi escrita por um dos meus Rubistas preferidos, Akira Tanaka. Ele simplificou a leitura de arquivos Internet, tornando-a igual leitura de arquivos do seu computador.

Num capítulo anterior, armazenamos suas idéias diabólicas em um arquivo texto. Você lê esses arquivos em Ruby usando **open**.

```
require 'open-uri'
```

```
# Abrindo um arquivo em um diretório do seu computador.
open( "diretorio/ideia-sobre-esconder-alface-nas-cadeiras-da-igreja.txt" ) do |ideia|
  puts ideia.read
end
```

Files (arquivos) são **objetos de entrada-e-saída**. Você pode ler e escrever em um arquivo. Em Ruby, todo objeto IO (input-output) tem os métodos **read** e **write**. O método **open** passa um objeto IO para num bloco para você usar. IO é a sua passagem para o mundo exterior. É como os raios do sol passando através das barras da prisão. (Entretanto, você não pode escrever (**write**) em uma página web usando **OpenURI**. Você terá que achar um ferramenta para copiar para o seu Web Server. Um cliente FTP, por exemplo).

Se alguém quiser ler sua idéia diabólica de esconder alface nas cadeiras da igreja, presumindo que você tenha colocado isso numa página web:

```
require 'open-uri'
```

```
# Abrindo um arquivo disponível num Web Site.
open( "http://sua.com/ideia-sobre-esconder-alface-nas-cadeiras-da-igreja.txt" ) do |ideia|
  puts ideia.read
end
```

A biblioteca **OpenURI** também entende endereços de FTP. Isso amplia as possibilidades de onde você pode

armazenar arquivos. No seu computador ou em algum outro lugar na Internet.

Lendo arquivos linha a linha

Quando você usa `OpenURI` para obter informações da web com os métodos `open` and `read`, a página é enviada como um `String`. Você também ler a página uma linha de cada vez, se está procurando algo. Ou se a página é muito grande e você quer economizar memória do seu computador.

```
require 'open-uri'  
open( "http://preeventualist.org/lost/searchfound?q=caminhao" ) do |caminhao|  
  caminhao.each_line do |linha|  
    puts linha if linha['picape']  
  end  
end
```

O código acima recupera a lista de caminhões achados por pré-eventualistas e exibe somente as linhas que contenham a palavra ‘picape’. Dessa maneira podemos limitar as descrições e procurar somente as linhas pertinentes.

Acima, os **colchetes** são usados em uma string, de forma que seja pesquisado nessa string o que quer que esteja dentro dos colchetes. Como a string ‘`picape`’ está dentro dos colchetes, a palavra ‘picape’ é pesquisada na string `line`.



Quando uma página web é carregada com `read`, toda a página é carregada na memória. Normalmente isso ocupa somente alguns kbytes. Mas se a página é grande (vários megabytes), é melhor usar `each_line`, que carrega uma linha de cada vez, evitando ocupar toda memória.

Yielding é como blocos de montar

Ruby frequentemente usa iteradores dessa forma. Sim, iteradores são usados para percorrer cada um dos itens de uma collection (coleção), como um Array ou um Hash. Então, olhe para uma fonte de IO como uma coleção de linhas. O iterator (iterador) pode percorrer essa coleção de linhas.

```
class IO  
  # Definição do método each_line. Veja que ele não tem  
  # uma lista de argumentos. Blocos não precisam ser listados como argumentos.  
  def each_line  
    until eof?      # até chegarmos ao final do arquivo...  
      yield readline # passe a linha para o bloco  
    end  
  end  
end
```

A palavra-chave `yield` (produzir) é a maneira mais fácil de se usar um bloco. Uma palavra. Da mesma forma que uma cortina tem um puxador ou como uma maleta tem uma alça. Dentro de um método, você pode apertar o botão piscante `yield` e ele irá rodar o bloco anexado a esse método. Ele ficará brilhando em vermelho até que o bloco seja executado. Então voltará a piscar de novo e você poderá o botão novamente se quiser.

```
def yield_thrice
  yield
  yield
  yield
end
```

Aperte o botão `yield` três vezes rapidamente e o bloco ganha vida por três vezes.

```
irb> a = ['primeiro, o nascimento', 'depois, uma vida cheia de imagens bonitas', 'e
finalmente, o fim']
irb> yield_thrice { puts a.shift }
# prints out:
#  primeiro, o nascimento
#  depois, uma vida cheia de imagens bonitas
#  e finalmente, o fim
```

O método `shift` tira o primeiro item de um Array. O barbeiro `shift` corta o cabelo fora e o entrega. Depois o escalpo. E assim segue, reduzindo o pobre coitado a nada.

Você viu blocos anexado a métodos. Qualquer método Ruby pode ter um bloco anexado no final.

```
# O estilo compacto de anexar um bloco a um método.
# Aqui o bloco é demarcado por chaves.
open( "idea.txt" ) { |f| f.read }

# O estilo verboso de anexar um bloco.
# Aqui o bloco é demarcado por 'do' e 'end'
open( "idea.txt" ) do |f|
  f.read
end
```

Se você passar argumentos para o `yield`, esse argumentos também serão passados ao bloco. O bloco vai num pequeno sidecar preso à motocicleta do método. O método vocifera uma lista de argumentos, gritando para o bloco mesmo com todo o vento enquanto eles estão dirigindo pelo deserto. O bloco bate no capacete com se dissesse: “Ok, entendi. Meu cérebro captou tudo.”

```
# O método abre dois arquivos e repassa os objetos IO resultantes
# para o bloco anexo
def double_open filename1, filename2
  open( filename1 ) do |f1|
    open( filename2 ) do |f2|
      yield f1, f2
    end
  end
end

# Imprime os arquivos lado a lado
double_open( "ideal.txt", "idea2.txt" ) do |f1, f2|
  puts f1.readline + " | " + f2.readline
end
```

Você pode se perguntar o que o `yield` tem a ver com sinais de trânsito. Na verdade, essa é uma boa pergunta, e que eu acredito que tenha uma boa resposta. Quando você executa um método, você está a esse método o controle do seu programa. Controle para fazer o seu trabalho e retornar com uma resposta.

Com `yield`, o método está parando nessa intersecção, devolvendo o controle para você, para o seu bloco. O método está permitindo que você faça o seu trabalho antes de retornar ao trabalho dele. Então, enquanto o método `each_line` faz o trabalho de ler linhas de um arquivo, o **bloco anexado ao método `each_line`** recebe a linha e tem a chance de dar um passeio com ela no sidecar.

Pré-eventualismo numa Caixinha Dourada

Você já aprendeu o bastante sobre [OpenURI](#) e `yield` para escrever os seus próprios iteradores. Você já sabe como usar o serviço de achados e perdidos. Na verdade, você já pode começar a se aventurar nos achados e perdidos de Wixl sem mim.

Vamos *encapsular* elegantemente todo o serviço em uma única classe.

```
require 'open-uri'
module PreEventalist
  def self.open page, query
    qs =
      query.map do |k, v|
        URI.escape "#{ k }=#{ v }"
      end.join "&"
    URI.parse( "http://preeventualist.org/lost/" + page + "?" + qs ).open do |lost|
      lost.read.split( "--\n" )
    end
  end
  def self.search word
    open "search", "q" => word
  end
  def self.searchlost word
    open "searchlost", "q" => word
  end
  def self.searchfound word
    open "searchfound", "q" => word
  end
  def self.addfound seu_nome, item_perdido, achado_em, descricao
    open "addfound", "name" => seu_nome, "item" => item_perdido,
          "at" => achado_em, "desc" => descricao
  end
  def self.addlost seu_nome, item_achado, visto_em, descricao
    open "addlost", "name" => seu_nome, "item" => item_achado,
          "seen" => visto_em, "desc" => descricao
  end
end
```

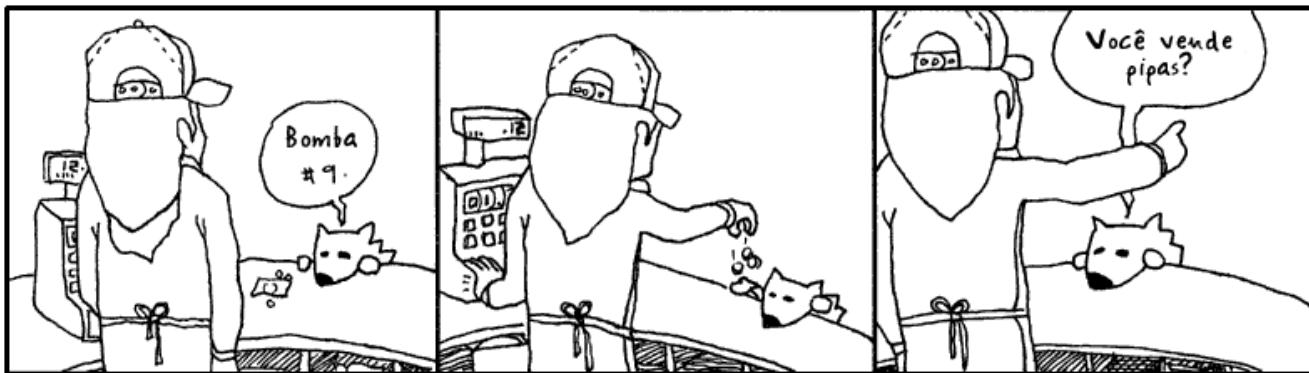
Em algum momento, você tem que começar a modelar seu código em algo mais elegante. Salve o módulo acima em um arquivo chamado **preeventualista.rb**.

Esse módulo é uma biblioteca muito simples para usar o serviço dos Pré-eventualistas. É exatamente assim que bibliotecas são escritas. Você constrói um módulo ou uma classe, grava em um arquivo, e, se ficou feliz com o resultado, disponibiliza na web para o resto do mundo poder utilizar.

Esses vagabundos podem então usar o seu módulo da mesma forma que eu usei **OpenURI** anteriormente.

```
irb> require 'preeventualista'
irb> puts PreEventalist.search( 'caminhao' )
irb> puts PreEventalist.addfound( 'Why', 'conhecimentos em Ruby', 'Parque Wixl',
  "Eu posso lhe dar conhecimentos em Ruby!\nVenha visitar poignantguide.net!" )
```

2. Enquanto Isso, O Porco-Espinho Faz uma Parada Para Abastecer



3. Um Assassinato de Dragão Patrocinado



“Olhe em Volta,” disse Raposinho. “Alguns de nós não tem tempo para buscas. Alguns de nós tem maiores responsabilidades, trabalhos, assim por diante. Sustento, Entendeu?”

“Eiiii, meu **TRABALHO** era assassinar o dragão!!” gritou o coelho Wee, piscando seus olhos e pulando furioso de galho em galho de lagoa em lagoa. “Seu focinho era uma **GRANDE** responsabilidade!! Seu hálito de fumaça era **problema meu!!!**. Eu gastei cinqüenta dólares em um táxi **APENAS** para sair daqui, que foi outro **ENORME ENORME** sacrifício. Você não tem *nada sobre mim*, nem uma *simples* acusação, todo a minha **HEROICIDADE** é **absolutamente impensável**, toda a minha **ABORDAGEM** é *clarinamente indamascável*, disse o Lester.”

“Quem é Lester?” falou Raposinho.

“Lester é meu taxista! Ele está estacionado na entrada do Array de Dwemthy!!” O Coelho ricocheteou enlouquecidamente como um screensaver para um Super-Computador. “Pergunte ao Dwemthy!!”

“Bom,” disse Raposinho. Ele se virou para olhar para Raposo, que estava sentada olhando longe no horizonte. “Espera aí, eles tem um estacionamento no Array de Dwemthy?”

“**SIM!!** E um quiosque de Pretzel!!”

“Mas é um Array? Eles vendem churros?”

“**CHOCOLAVA!!**” berrou o coelho.

“E aquelas cordas que brilham no escuro que você pode prender no cabelo? Ou pode apenas ficar segurando elas ou—”

“**BRAIDQUEST!!**”

“Você devia receber uma parte da comissão do vendedor,” afirmou Raposinho. “O pessoal veio ver você matar o dragão, certo?”

“**MAS!!** Eu não opero os braços-mecânicos que extraem o chocolava.”

“Eu só estou tentando dizer que você **faz** funcionar a máquina de

O Meteoro Involuntário

Quando eu comecei a minha primeira investigação sobre pré-eventualismo, me submeteram a seguinte história. Me falaram que isso era tudo que eu precisaria saber para entender a filosofia.

Havia este escultor que não estava satisfeito com seu trabalho. Ele tinha estudado principalmente assuntos tradicionais, era excelente em esculpir tanto figuras humanas quanto em elaborar vegetação. E ele foi realmente um excelente escultor. Ele já não se sentia como se estivesse fazendo sua marca sobre o mundo.

A esta altura, ele já tinha chegado a casa dos cinqüenta anos de idade e queria se vangloriar no reino dos mestres lendários. Então ele começou a construir uma escultura enorme de duas peras com gotas de orvalho levemente pressa a elas.

A escultura era enorme e pairava preocupantemente acima da cidade natal do

matar. Então você deve ter uma parte nisso.”

“AH NÃO!! Deixei minha folha de alface favorita no Array de Dwemthy!!” gritou o coelho, girando feito um peão. Ao fundo: “Ou quem sabe no carro do Lester?”

“Quer saber— meu deus, dá para parar quieto??” Pediу raposinho.

“Meu rádio,” disse Raposinho, voltando à vida por um instante, “na minha picape.” Aquele olhar ainda distante. Seu olhar parece voltar, mas logo se distancia, lembranças de outros lugares e outros momentos. Uma viagem de carro para Maryland. A voz de Lionel Richie ao fundo. Os limpadores de pára-brisas ligados meio rápidos demais. Ele para em uma casa. Sua mão abre a porta. Ele é uma raposa bem peluda. Lágrimas e maquiagem.

Bem, voltando: “Aquele porco-espinho está trocando as estações”

O coelho subiu no braço do banco e falou de perto. **MAS!!** Logo eu vou fazer um banquete com a cabeça do dragão e com os sucos da língua do dragão.” O coelho sentou e ajeitou suas patas delicadamente.

“(Que eu espero que tenha o mesmo gosto de ursos de canela)” sussurrou o coelho para si mesmo.

“Adoro canela,” disse Raposinho. “Eu devia sair para matar dragões com você um dia desses.”

“Você devia, mesmo” disse o coelho, com brilho nos olhos.

“Mas salivar por uma língua?. Você não saliva por uma língua, não é?”

“CLARO QUE SIM!!” e o coelho ficou tão excitado que Sticky Whip saiu pelo seus olhos. (Mais sobre Sticky Whip posteriormente em uma sidebar. Não me deixem esquecer. Veja também: _O Compêndio do Purista para novos Cremes para Retina, por Jory Patrick Sobgoblin, disponível onde quer que vendam presilhas de animais.)

“Ok, você me ganhou. Quero ouvir toda a história,” disse Raposinho. “Por favor, fale a vontade sobre o ‘chaminé’. Ah, e Dwemthy. Quem é ele? Quais são suas motivações? Então talvez, se eu ainda estiver por perto depois disso, você poderá me dizer sobre o que faz os coelhos se motivarem, e talvez você possa segurar nossas mãos através deste calvário. Eu preciso de consolo mais do que ninguém. Provavelmente eu poderia usar religião agora. Eu poderia usar sua bravura e este sentimento de realização que você exala. Você fuma cachimbo? Poderia ser uma ferramenta proveitosa para trazermos nesta pontificação em que estamos metidos.”

escultor, mantida afastada por uma pesada infraestrutura de vigas e estacas. De fato, aquelas pêras gigantes, de tão grandes, acabaram por afetar até mesmo o eixo de rotação da Terra, que passou a girar acompanhado por um cesto de frutas do tamanho de um asteróide orbitando em sua volta.

O governo enviou jatos e aviões de guerra para destruir a estátua. Eles atacaram violentamente a vila, destruindo a estátua, despedaçando em milhares de pedaços, escavando-a com mísseis. Logo após a estátua ser destruída, tudo voltou ao normal.

Um enorme pedaço da estátua ganhou órbita e por várias vezes aproximou-se perigosamente da Terra. Quando isso acontecia, era sempre encontrado por um arsenal avançado, que ajudava a danificá-lo e desviaava seu curso para o céus.

Finalmente, este meteoro involuntário não estava mais do tamanho de um homem assustador. E quando ele, por fim, caiu na terra, desgastado e polido pela sua jornada de noventa anos, ele foi saudado como uma obra-prima enigmática, uma mensagem do grande além. Aqui estava um semelhança maravilhosa de um nu masculino olhando tristemente para o céu com uma confusa trepadeira de vinha por volta da cintura e cobrindo suas intimidades.

A estátua foi, por último, vendida por cinqüenta e dois milhões de dólares e ficou em exposição permanente no museu do Louvre, com a placa:

“Nu Celeste” por Anônimo

E o coelho começou a divagar sobre Dwemthy e a lenda do Dwemthy e as formas de Dwemthy. Tal como a maioria das estórias de Dwemthy, o relato do coelho era particularmente ornamentado. Cara pálida, existem delícias que somente eu devo abordar.

Por favor, nunca pergunte quem é Dwemthy. Obviamente ele é um gênio e nunca revelaria sua localização ou verdadeira identidade. Ele tem dinastias. Ele deu vidas a ogros. Em toda a parte, em todo o tempo, cavalos sentem seu cheiro. Acima de tudo, ele conhece prazeres carnais. E pensar que este...

Este é o Array dele.

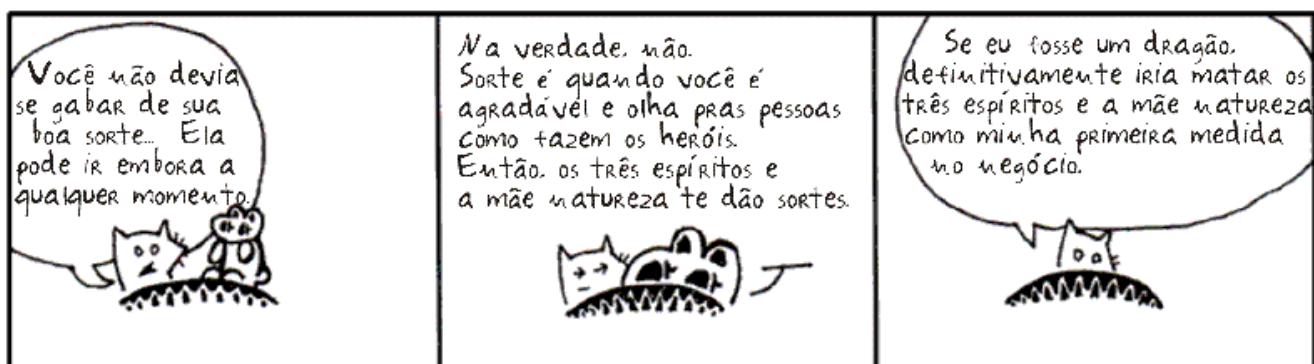
O Array de Dwemthy



Você está na entrada do Array do Dwemthy. Você é um coelho que está prestes a morrer. E mergulha fundo no Array:

```
class Dragao < Criatura
  vida 1340      # escamas duras
  forca 451       # veias rígidas
  carisma 1020    # sorriso escancarado
  arma 939        # bafo de fogo
end
```

Uma escaldante *LAVA BORBULHANTE* infiltrou a **cacofônica MINA** adentrando as antigas copas da *FLORESTA DWEMTHY*... calcários e noturnos gritos da **barriga SELVAGEM E VORAZ STORKUPINE**... que come **gansos molhados LOGO DEPOIS** ele que comeu alguns biscoitos Graham e um cochilo ao meio dia... entre hipopótamos famintos *TECNICAMENTE ÓRFÃOS* mas veridicamente abrigados sobre sombrinhas de revendas de carros... abaixo *ampolas destampadas* de mínimas polpas **ELIXIR AZUL**... que devem permanecer... doravante... **QUIETO... DWEMTHY!!!**



Se você não entendeu o Array de Dwemthy, a culpa é dele. Ele projetou o jogo para complicar nossas vidas e seria mais simples, não fosse a inspiradora jornada que temos que prezar em nossos braços precisamente agora.

O Array de Dwemthy tem uma história sinuosa e de grande impacto. Não basta simplesmente dizer “Labirinto de Dwemthy” mais e mais e esperar ter subsídios apenas a partir deste ato. Venha comigo, posso te levar de volta a alguns anos atrás, de volta ao anos sessenta onde tudo começou com metaprogramação e golfinhos.

Você pode estar inclinado a pensar que **metaprogramação** é mais uma palavra hacker e que inicialmente foi utilizada incessantemente em ligações privadas entre máquinas de fax. Juro por Deus, estou aqui para lhe dizer que isso é mais estranho que possa parecer. Metaprogramação começa com *usando drogas na companhia de golfinhos*.

Nos anos sessenta, um prolífico cientista chamado John C. Lilly começou a fazer experiências com seus próprios sentidos para descobrir o funcionamento de seu corpo. Eu posso atestar isto. Faço frequentemente quando estou de pé, no meio de uma estrada, segurando uma torta ou quando estou me escondendo dentro de uma catedral. Faço uma pausa para me examinar. Isto provou que é quase impossível. Tenho preenchido três páginas de caderno com notações algébricas, nenhuma delas conseguiu explicar nada. A torta, aliás, foi muito de fácil de expressá-la matematicamente.

Mas o cientista Lilly passou por cima de suas experiências. Ele tomou LSD na companhia de golfinhos. Muitas vezes em um lamentável tanque de isolamento, escuro, cheio de água salgada quente. Muito sombrio. Mas isto era ciência! (Antes de você achá-lo um criminoso: até 1966, LSD era fornecido pelos laboratórios Sandoz a qualquer cientista interessado, sem encargos.)

Drogas, golfinhos e privação. O que levou Lilly a fazer uma incursão em coisas meta. Ele escreveu livros sobre programação mental, comparando seres humanos e computadores. Você pode escolher ingerir qualquer substância que

queira durante esta próxima citação — é muito provável que você pegará um grão de sal — mas eu te asseguro que não haverá um show do Grateful Dead num gramado e nenhum ‘raver’ no porão.

Quando alguém aprende a aprender, este alguém está fazendo modelos, usando símbolos, fazendo analogias, metáforas, em resumo, inventando e usando linguagem, matemática, arte, política, negócio, etc. No lado crítico do cérebro (côrortex), estão as linguagens e suas consequências. Para evitar a necessidade da repetição de aprender para aprender símbolos, metáforas, modelos, eu simbolizo a idéia subjacente nestas operações como metaprogramação.

John C. Lilly, *Programming and Metaprogramming in the Human Biocomputer*, Nova Iorque, 1972.

Nós aprendemos. Mas primeiro aprendemos a aprender. Montamos programação na nossa mente que é o atalho para mais programação. (Lilly é um grande palestrante sobre programação cerebral e do sistema nervoso, o que ele coletivamente chama de *biocomputador*.)

A metaprogramação de Lilly era mais sobre dar asas ao seu imaginário, reinventando você mesmo. Este tipo de pensamento nos remete diretamente a pessoas que se interessam por Xamanismo, colocam suas mãos sobre cartas de tarot e que levantam cedo para aulas de karatê. Eu acho que você pode dizer que Metaprogramação é uma coisa New Age, mas tudo isso foi enfiado recentemente em um saco de dormir juntamente com a “nerdice” antiquada. (Se você chegou até aqui a partir de uma procura no Google por “Metaprogramação em C++”, fique à vontade, a única coisa que eu te peço é que você *queime* aqueles atalhos neuronais que originaram a busca. Muito obrigado.)

A própria Meta, não é falada de maneira diferente, hoje em dia, por seu autor.

Toda resposta sensitiva à realidade é uma interpretação dela própria. Besouros e macacos claramente interpretam seu mundo, e agem baseados no que vêem. Nossos sentidos físicos são, eles mesmos, órgãos de interpretação. O que nos distingue de nossos companheiros animais é que somos capazes de interpretar estas interpretações. Neste sentido, toda linguagem humana é uma meta-linguagem. É uma reflexão de segunda ordem sobre a ‘linguagem’ de nossos corpos — sobre nosso aparato sensorial.

Terry Eagleton, *After Theory*, Londres, 2003, ch. 3.

Para todo efeito, você pode dizer que programação, por si só, é uma meta-linguagem. *Todo código* fala a linguagem da ação, de um plano que ainda não foi colocado em prática, mas que brevemente irá. Instruções para o jogadores dentro da sua máquina. Eu tenho um sentimento lacrado sobre este assunto.

Mas agora estamos avançando nossos estudos, nos aventurando em metaprogramação, mas não fique ansioso, é **apenas o Ruby que você já viu**, é a razão pela qual Dwemthy não hesitou em mostrá-lo imediatamente a você. Em pouco tempo será tão fácil como perceber uma adição ou subtração. À primeira vista isto pode parecer brilhante, como se você tivesse encontrado seu primeiro vagalume, que de repente apareceu voando na sua frente. Então, isto se torna apenas *uma pequenina luz cintilante* que torna viver em Ohio muito mais legal.

Metaprogramação é a *escrita de código que escreve código*. Mas não como M. C. Escher rascunharia. **O programa não fica voltando e se escrevendo e nem o programa fica pulando de seu monitor e arrancando o teclado de suas mãos**. Não, é bem menor que isso.

Vamos dizer que é parecido **com uma pílula laranja** que você ganhou no circo. Quando você chupa ela e o sabor vai embora, atrás do seu dente esconde um grande, esponjoso brontossauro. Ele escorrega por sua língua e pula livremente, brincando sobre as pastagem e gritando: “Papai!” E a partir desse momento, sempre que ele enlouquecer vai atacar uma van, bem, essa van vai ficar brilhante depois.

Agora, vamos dizer que *algum* coloque **aquela pequena pílula laranja** debaixo da torneira. Não na língua, *debaixo da torneira*.

Charadas dos Pãezinhos

Pergunta: Alguém pode dar cinco mordidas em

um pão e fazer a forma de uma bicicleta?

Resposta: Sim.

Pergunta: Alguém pode rasgar um pão ao meio e ele continuar entrando em um envelope?

Resposta: Sim.

Pergunta: Um homem pode pegar um pão e arremessar enquanto outro homem senta-se sem pão?

Resposta: Sim.

Pergunta: Quatro pães em uma caixa podem ser

E isso aciona uma catalisação diferente, que cria um grupo de sêxtuplas esponjas choramingantes, **com cordão umbilical e tudo mais**. Ainda assim muito prático para se limpar a van. Mas um grupo totalmente diferente de camurça. E, um dia, esses seis irão fazer o papai chorar quando tocarem em um concerto de violino.

Metaprogramação é um conjunto de código dentro de um formato de pílula, que uma pequena gota de água pode acionar para se expandir. Mais importante, você pode controlar a reação da pílula, de uma forma que o brontossauro pode ser criado magro e desajeitado. Ou sêxtuplos, **CERTAMENTE**, ou costureiras, ou cérebros de gato, ou dragões.

```
class Dragao < Criatura
  vida 1340    # escamas duras
  forca 451     # veias ressaltadas
  carisma 1020 # soridente
  arma 939      # cospe fogo
end
```

Isso ainda não é metaprogramação. Apenas a pílula. O *produto* da metaprogramação. Estamos fazendo uma pausa, olhando para a besta antes de descer abaixo o seu tecido muscular com um bisturi e um microscópio.

O **Dragao** é uma classe. Você já viu muitas vezes. O **Dragao** é descendente da classe **Criatura**.

Agora, olhe para cima. Preste atenção. A classe **Criatura** contém o código de metaprogramação. Você pode escrever código de metaprogramação que pode ser usado em *qualquer lugar*, ao longo de todo Ruby, em **Criatura** ou **Dragao**, em **String** ou em **Object**, qualquer lugar. O nosso exemplo aqui, uma vez que esta é a forma mais comum de meta-código, foca em metaprogramação dentro de classes apenas.

Cada uma das características de **Dragao** são apenas **métodos de classe**. Você também poderia escrever assim:

```
class Dragao < Criatura
  vida( 1340 )    # escamas duras
  forca( 451 )     # veias ressaltadas
  carisma( 1020 ) # soridente
  arma( 939 )      # cospe fogo
end
```

Removendo os parênteses remove-se a confusão, então vamos deixar sem. Apenas use parênteses quando estiver usando muitos métodos juntos e você quer ser bem claro.

explicados?

Resposta: Sim.

Pergunta: Um erro de escritório nos livros da minha firma pode ser atribuído ao pão?

Resposta: Sim.

Pergunta: Dançarinos podem atravessar uma lona de pão?

Resposta: Sim.

Pergunta: Esses mesmo dançarinos, quando separados com uma lona de pão inexplicavelmente diferente, podem fracassar na travessia?

Resposta: Sim.

Pergunta: Pães podem entender meus medos mais obscuros e meus sonhos mais selvagens?

Resposta: Sim.

Pergunta: Pães podem desejar-me?

Resposta: Sim.

Pergunta: Pães são invisíveis para robôs?

Resposta: Sim.

Pergunta: Um robô pode dar oito mordidas em um pão, sem saber que ele está lá, e fazer a forma de um pequeno pão?

Resposta: Sim.

Pergunta: Meus clérigos devem ser equipados com pães?

Resposta: Sim.

Pergunta: Em relação aos pães, Os robôs podem ter cada um seus próprios elefantes?

Resposta: Sim.

Pergunta: Alguém pode partir um pão ao meio e não deixá-lo arruinar um jogo de dominó?

Resposta: Sim.

Pergunta: Nós sempre amaremos os pães?

Resposta: Sim.

Pergunta: Nós sempre comeremos mais pães?

Resposta: Sim.

Pergunta: Quatro pães podem se casar com um elefante de um robô?

Resposta: Sim.

Código da Criatura

Agora, com um corte lateral em todo o diafragma, nos mostraremos as entradas da classe **Criatura**. Salve esse código em um arquivo chamado **dwmthy.rb**.

```
# As vísceras da força vital do Array de Dwemthy
class Criatura

  # Obtém uma metaclasses para esta classe
  def self.metaclass; class << self; self; end; end

  # Avançado código de metaprogramação para boas
  # e claras características
  def self.caracteristicas( *arr )
    return @caracteristicas if arr.empty?

    # 1. Define 'accessors' para cada variável
    attr_accessor *arr

    # 2. Adiciona um novo método para cada característica.
    arr.each do |a|
      metaclass.instance_eval do
        define_method( a ) do |val|
          @caracteristicas ||= {}
          @caracteristicas[a] = val
        end
      end
    end
  end

  # 3. Para cada monstro o método 'initialize'
  # deve usar um número padrão para cada característica.
  class_eval do
    define_method( :initialize ) do
      self.class.caracteristicas.each do |k,v|
        instance_variable_set("@#{k}", v)
      end
    end
  end
end

# Os atributos da Criatura são somente leitura
caracteristicas :vida, :força, :carisma, :arma
end
```

Preste atenção no fechamento das linhas do código, especialmente na linha onde **características** está sendo definido. Todo o código antes dessa linha determina o método **características**. Isto é semelhante ao básico do bilhete de loteria do capítulo anterior.

```
class BilheteLoteria

  attr_reader :numeros_escolhidos, :data_compra
end
```

Tanto **características** como **attr_reader** são apenas métodos de classe. Quando **attr_reader** é usado em **BilheteLoteria**, a metaprogramação inicia por trás dos bastidores e começa a estourar balões, criando métodos **leitores** para as variáveis de instância **@numeros_escolhidos** e **@data_compra** acima.

O código para o método **características** é a metaprogramação que eu tenho feito referência. Comentários no código revelam os três estágios que o método passa quando adiciona as características.

1. A lista de características é passada para **attr_accessor**, que constrói código **leitor** e **escritor** para as

variáveis de instância. Um para cada característica.

2. **Métodos de classe são adicionados** para cada característica. (O método `vida` é adicionado para a característica `:vida`). Esses métodos de classe são usados na definição da classe assim como você usaria `caracteristicas` ou `@attr_accessor`. Dessa maneira, você pode especificar a característica, de acordo com os pontos dados de uma característica para uma determinada criatura.
3. **Adicione um método de inicialização** que cria um novo monstro corretamente, com os pontos certos e *GANHANDO FORÇA! GANHANDO FORÇA!* o monstro está vivo!

A beleza desses três passos é que você ensinou ao Ruby como criar monstros para você. Então quando o Ruby obtém as `características`:

```
class Criatura
  caracteristicas :vida, :forca, :carisma, :arma
end
```

O Ruby preenche com o código dos bastidores e transplanta um pulsante coração verde e da ignição no corpo dando corda. O Ruby vai usar a metaprogramação da classe `Criatura` e irá construir todos os variados métodos, expandindo `características` em uma lista como essa:

```
class Criatura

  # 1. define métodos leitores e escritores
  attr_accessor :vida, :forca, :carisma, :arma

  # 2. adiciona novos métodos de classe para usar na criatura
  def self.vida( val )
    @características ||= {}
    @características['vida'] = val
  end

  def self.forca( val )
    @características ||= {}
    @características['forca'] = val
  end

  def self.carisma( val )
    @características ||= {}
    @características['carisma'] = val
  end

  def self.arma( val )
    @características ||= {}
    @características['arma'] = val
  end

  # 3. estabelece pontos padrões para
  #     cada característica
  def initialize
    self.class.características.each do |k,v|
      instance_variable_set("@#{k}", v)
    end
  end

end
```

Agora, o Ruby vai aceitar gradativamente essas seis linhas de código da classe `Dragao`, curto o bastante para aparecer legal em cartas de jogos:

```
class Dragao < Criatura
  vida 1340    # escamas duras
  forca 451     # veias ressaltadas
  carisma 1020  # soridente
  arma 939      # cospe fogo
end
```

Eval, o Menor Metaprogramador

Embora o código de metaprogramação acima seja simplesmente Ruby, ainda assim pode ser difícil de seguir. Eu comprehendo totalmente, caso você tenha chegado a esse ponto e seus olhos estejam girando em suas cavidades e seus joelhos emperrados. A parte mais complicada de tudo isso acima são as linhas que chamam os métodos `instance_eval` e `class_eval`. Passe pomada nas suas juntas enquanto eu falo sobre `eval`.

Viemos falando sobre **metaprogramação**. Escrevendo código que escreve códigos. O método `eval` reside neste beco. O vagabundo `eval` pega o código que você havia guardado em uma string e executa o código.

```
drgn = Dragao.new  
# é identifico a...  
drgn = eval( "Dragao.new" )  
# ou alternativamente...  
eval( "drgn = Dragao.new" )
```

Aqui, vamos escrever um programa que tem um buraco. Ao invés de escrever um programa que cria um novo `Dragao`, vamos deixar um buraco onde seria o `Dragao`.

```
print "Que classe de monstro você veio combater? "  
classe_monstro = gets  
eval( "monstro = " + classe_monstro + ".new" )  
p monstro
```

O programa pede por um monstro. Se você digitar `Dragao`, então a variável da `classe_monstro` irá conter a string `"Dragao"`. Dentro do `eval` algumas strings serão adicionadas juntas para formar a string `"monstro = Dragao.new"`. E quando o `eval` executa essa string, a variável `monstro` contém um objeto `Dragao`. Pronto para batalha.

Isso é maravilhoso! Agora podemos deixar que o jogador escolha um monstro! Claro, estamos confiando no jogador para que ele forneça uma classe real de monstro. Se eles digitarem `BruxaBotanica` e não existe a classe `BruxaBotanica`, eles terão uma exceção jogada na sua cara.

Então, em resumo, o `eval` permite que você crie um código a medida que prossegue. O que pode ser útil e pode ser perigoso ao mesmo tempo.

Os métodos `instance_eval` e `class_eval` usado na metaprogramação para a classe `@Criatura` são levemente diferente do normal `eval`. Esses dois métodos especiais passam o código assim como o `eval` faz, mas eles mergulham em classes e objetos e passam o código lá.

```
# O método instance_eval executa o código como se ele fosse executado dentro  
# do objeto do método de instância.  
irb> drgn = Dragao.new  
irb> drgn.instance_eval do  
irb>   @nome = "Tobias"  
irb> end  
  
irb> drgn.instance_variable_get( "@nome" )  
=> "Tobias"  
  
# O método class_eval executa o código se estiver dentro da definição de classe.  
irb> Dragao.class_eval do  
irb>   def nome; @nome; end  
irb> end  
  
irb> drgn.nome  
=> "Tobias"
```

Como você pode ver acima, os métodos `instance_eval` e `class_eval` também podem pegar um bloco de código ao invés de uma string. O que é exatamente como as coisas foram feitas no Array de Dwemthy.

Bastante instrução depreciante e Artilosa Justaposição — Onde está o Array de Dwemthy???

Vá com cuidado — aqui está a outra metade do ARRAY DE DWEMTHY!! Adicione essas linhas a `dwemthy.rb`.

```
class Criatura
```

```

# Este método aplica um golpe recebido durante uma luta.
def golpear( dano )
  aumento_poder = rand( carisma )
  if aumento_poder % 9 == 7
    @vida += aumento_poder / 4
    puts "[Aumento de poderes mágicos de #{ self.class } #{ aumento_poder }!]"
  end
  @vida -= dano
  puts "[#{ self.class } está morto.]" if @vida <= 0
end

# Este método obtém uma rodada em uma luta.
def lutar( inimigo, arma )
  if vida <= 0
    puts "[#{ self.class } está muito morto para lutar!]"
    return
  end

  # Ataca o oponente
  seu_golpe = rand( forca + arma )
  puts "[Você golpeia com #{ seu_golpe } pontos de dano!]"
  inimigo.golpear( seu_golpe )

  # Retaliação
  p inimigo
  if inimigo.vida > 0
    inimigo_golpe = rand( inimigo.forca + inimigo.arma )
    puts "[Seu inimigo golpeia com #{ inimigo_golpe } pontos de dano!]"
    self.golpear( inimigo_golpe )
  end
end

end

class ArrayDeDwemthy < Array
  alias _inspect inspect
  def inspect; "#<#{ self.class }#{ _inspect }>"; end
  def method_missing( meth, *args )
    resposta = first.send( meth, *args )
    if first.vida <= 0
      shift
      if empty?
        puts "[Uauu. Você dizimou o Array de Dwemthy!]"
      else
        puts "[Prepare-se. #{ first.class } surgiu.]"
      end
    end
    resposta || 0
  end
end

```

Esse código adiciona dois métodos a **Criatura**. O método **golpear** que reage ao golpe de outra **Criatura**. E o método **lutar** que permite que você coloque os seus próprios golpes contra aquela **Criatura**.

Quando a sua **Criatura** leva um golpe, um pouco da defesa contribui e o seu valor **carisma** é usado para gerar um aumento de poder. Não me peça para explicar os segredos por trás deste fenômeno. Um número aleatório é escolhido, uma matemática simples é feita e se você tiver sorte, você consegue alguns pontos vitais. **@vida += aumento_poder / 4**.

Então o golpe do inimigo está dado. **@vida -= dano**. É assim que o método da **Criatura#golpear** trabalha.

O método **lutar** checa se a sua **Criatura** está viva. Em seguida, um golpe aleatório é dado no seu oponente. Se o seu oponente sobreviver a este golpe, ele ganha uma chance de atacar de volta. Estas são as atividades do método **Criatura#lutar**.

Irei explicar o **ArrayDwemthy** em um segundo. Realmente explicarei. Estou me divertindo fazendo isso. Por hora, vamos nos concentrar em **golpear** e **lutar**.

Apresentando: Você.

Você pode certamente experimentar derivações deste coelho. Mas o oficial Paradigma de Dwemthy explicitamente denota o código -- e o personagem em geral -- escrito abaixo. **Salve isto como coelho.rb.**

```
class Coelho < Criatura
  caracteristicas :bombas

  vida 10
  forca 2
  carisma 44
  arma 4
  bombas 3

  # bumeranguinho
  def ^( inimigo )
    lutar( inimigo, 13 )
  end
  # a espada do herói é ilimitada!!
  def /( inimigo )
    lutar( inimigo, rand( 4 + ( ( inimigo.vida % 10 ) ** 2 ) ) )
  end
  # alface irá formar a sua força e fibra alimentar extra
  # voará na cara de seu oponente!!
  def %( inimigo )
    alface = rand( carisma )
    puts "[Alface saudável lhe dá #{ alface } pontos de vida!!]"
    @vida += alface
    lutar( inimigo, 0 )
  end
  # bombas, mas você possui somente três!!
  def *( inimigo )
    if @bombas.zero?
      puts "[HUMM!! Você está sem bombas!!]"
      return
    end
    @bombas -= 1
    lutar( inimigo, 86 )
  end
end
```

Você tem quarto armas. O bumerangue. A espada do herói. A alface. E as bombas.

Para fazer acontecer, abra o **irb** e carregue as bibliotecas que criamos acima.

```
irb> require 'dwemthy'
irb> require 'coelho'
```

Agora, mostre-se.

```
irb> c = Coelho.new
irb> c.vida
=> 10
irb> c.forca
=> 2
```

Bom, bom.

O Coelho Luta com a ScubaArgentina!

Você não pode ir apressadamente para o Array de Dwemthy, sem cinto de segurança e simplesmente perfumado!! Você deve avançar deliberadamente para o demoníaco cotilhão. Ou ao sul, pelos bosques e pelo labirinto de carvão.

Neste momento, vamos nos esconder dissimuladamente pelo resíduo leitoso ao lado dos aquedutos e seguir o **ScubaArgentina**.

```
class ScubaArgentina < Criatura
  vida 46
  forca 35
```

```
carisma 91  
arma 2  
end
```

Para começar a luta, certifique-se de que criou um de você e um do **ScubaArgentina**.

```
irb> c = Coelho.new  
irb> s = ScubaArgentina.new
```

Agora use o bumeranguinho!

```
irb> c ^ s  
[Você golpeia com 2 pontos de dano!]  
#<ScubaArgentina:0x808c864 @carisma=91, @forca=35, @vida=44, @arma=2>  
[Seu inimigo golpeia com 28 pontos de dano!]  
[O Coelho morreu.]
```

Por Cristo de Deus!! Nossa exemplar de coelho morreu!!

Duras perspectivas. Eu não posso te pedir para retornar ao reino dos coelhos, no entanto, basta fingir como se você não tivesse morrido e fazer um novo coelho.

```
irb> r = Coelho.new  
  
# atacando com bumerangue!  
irb> r ^ s  
  
# os golpes de espada do herói!  
irb> r / s  
  
# comendo alface lhe dá vida!  
irb> r % s  
  
# você possui três bombas!  
irb> r * s
```

Muito claro, você não diria? O código em **coelho.rb** modifica alguns símbolos matemáticos que trabalham somente com **Coelho**. Ruby lhe permite alterar o comportamento dos operadores matemáticos. Afinal, **operadores matemáticos são apenas métodos!**

```
# o bumerangue é normalmente um operador XOR.  
irb> 1.^( 1 )  
=> 0  
  
# a espada do herói normalmente divide números.  
irb> 10./( 2 )  
=> 5  
  
# o alface dá o resto da divisão.  
irb> 10.%( 3 )  
=> 1  
  
# a bomba é para multiplicação.  
irb> 10.*( 3 )  
=> 30
```

Quando faz sentido, você pode optar por utilizar operadores matemáticos em algumas de suas classes. Ruby usa esses operadores matemáticos em muitas de suas próprias classes. Arrays, por exemplo, têm uma porção de operadores matemáticos que você poder ver na lista de instância dos métodos quando você

Os Sapatos Dos Quais As Mentiras São Feitas

Anteriormente, eu disse que “O Meteoro Involuntário” era a única história que você precisaria para saber para entender o pré-eventualismo. Mas, realmente, tudo o que você precisa entender sobre pré-eventualismo é que ele ainda está em sua infância e quaisquer dos seus conceitos mais básicos podem mudar.

Por isso é que eu criei uma história concorrente que eu acredito que revela um cenário intelectual totalmente diferente e muito relevante.

Havia um cara por aí no bairro. E ele não era muito velho, então ele decidiu escrever a biografia da sua vida.

Bem, ele começou a mentir em sua biografia. Ele inventou algumas histórias. Mas na maioria pequenas histórias que eram inconseqüentes. Encheção de lingüiça. Como a história que ele tinha sobre uma pintura que ele fez de um fundo vermelho com pernas de elefante na frente.

Mas ele não tinha pintado nada disso. Depois ele embelezou a história falando sobre um leilão caro em que ele conseguiu entrar. Um leilão em Nova Iorque onde ele ouviu que sua pintura iria ser vendida por vinte mil dólares. Mas esse não é o ponto da história. O ponto era que ele podia dobrar seu corpo para caber dentro de um carrinho de comida. As pessoas levantariam a tampa e nem o notariam lá dentro. Ele nem mencionou por qual preço sua pintura foi vendida.

Bem, ele realmente começou a gostar daquela história (e outras como ela), até o ponto em que

digita: **ri** Array.

```
# o operador mais combina dois arrays em um único array
irb> ["D", "W", "E"] + ["M", "T", "H", "Y"]
=> ["D", "W", "E", "M", "T", "H", "Y"]

# menos remove todos os itens do segundo array encontrados no primeiro
irb> ["D", "W", "E", "M", "T", "H", "Y"] - ["W",
"T"]
=> ["D", "E", "M", "H", "Y"]

# o multiplicador repete os elementos de um array
irb> ["D", "W"] * 3
=> ["D", "W", "D", "W", "D", "W"]
```

Você deve estar se perguntando: afinal, o que isso significa para a matemática? E se eu adicionar o número três para um array? E se eu adicionar uma string e um número? **Como Ruby irá reagir?**

Por favor, lembre-se, esses operadores são apenas métodos. Mas, uma vez que esses operadores *não são palavras legíveis*, fica difícil dizer o que eles fazem. Use o **ri**, muitas vezes você verá que os operadores são idênticos a outros métodos legíveis. Você pode optar por utilizar o operador ou o método, o que for mais claro para você.

```
# divida com um método operador ...
irb> 10 / 3
=> 3

# ... ou um método legível?
irb> 10.div 3
=> 3
```

Esta é forma que a espada do coelho divide.

ele começou a ignorar seus amigos e família, preferindo assistir ao que seu eu de mentira fez depois do leilão. Na sua cabeça.

Então, um dia ele estava fazendo compras e encontrou um par de sapatos que tinha cadarços listrados. E ele pegou os sapatos e foi ao caixa da loja, forçando-os na cara do funcionário, gritando, “Olhe! Olhe para eles! Olhe! Esses são os sapatos que meu eu de mentira usaria!” E ele comprou os sapatos e os colocou e toda a Terra abriu e a caixa registradora abriu e o engoliu e ele de repente estava em outro lugar, em seu apartamento de mentira, sentado para pintar bicos de golfinhos, três deles num fundo verde.

Foi muito trabalho, pintar todos aqueles bicos. E ele quebrou por um tempo e teve que se rebaixar tanto quanto filmar a NASCAR do abominável homem das neves.

A Dura Realidade do Array de Dwemthy TE AGUARDA PARA LHE ESMAGAR !

Depois que você terminou de brincar de afogar o último cara com o tubo de oxigênio dele, é hora de entrar No Array. Eu duvido que você possa fazer isso. Você deixou sua machadinha em casa. Eu espero que você não tenha usado todas as suas bombas com o cara fácil.

Você possui seis inimigos.

```
class MacacoIndustrialEntusiasmante < Criatura
  vida 46
  forca 35
  carisma 91
  arma 2
end

class AnjoDosAnoes < Criatura
  vida 540
  forca 6
  carisma 144
  arma 50
end

class TentaculoViceAssistenteE0mbudsman < Criatura
  vida 320
  forca 6
  carisma 144
  arma 50
```

```
end
```

```
class CervoDeDentes < Criatura
  vida 655
  forca 192
  carisma 19
  arma 109
end

class IntrepidoCiclistaDecomposto < Criatura
  vida 901
  forca 560
  carisma 422
  arma 105
end

class Dragao < Criatura
  vida 1340      # escamas duras
  forca 451      # veias ressaltadas
  carisma 1020   # sorriso dentado
  arma 939       # cospe fogo
end
```

Estes são os exemplos vivos da monstruosidade do Array de Dwemthy. Eu não sei como eles chegaram lá. Ninguém sabe. Na verdade, eu acho que o @ IntrepidoCiclistaDecomposto@ pedalou até aqui a toda velocidade. Mas os outros: NINGUÉM sabe.

Se isso é realmente importante para você, vamos apenas dizer que os outros nasceram lá. Podemos seguir em frente??

Conforme o Array de Dwemthy vai ficando mais complexo, o desafio se torna mais difícil.

```
dwary = ArrayDeDwemty[MacacoIndustrialEntusiasmante.new,
                        AnjoDosAnoes.new,
                        TentaculoViceAssistenteE0mbudsman.new,
                        CervoDeDentes.new,
                        IntrepidoCiclistaDecomposto.new,
                        Dragao.new]
```

Combata o Array e os monstros aparecerão ao passo que você continua. Boa sorte, e que você retorne com histórias horríveis e nenhuma garra de anjo cravada através de seus ombros.

Começa aqui:

```
irb> r % dwary
```

Ah, e nada deste negócio “Eu sou muito jovem para morrer”. Estou farto desta besteira. Não vou mais tolerar seus insultos aos nossos jovens que ainda nos restam. Eles são nosso futuro. Assim que nosso futuro estiver terminado, ai sim.



Por Trás do Array de Dwemthy

Avance para o tempo onde os ventos já tenham se acalmado. O Dragão foi derrotado. Os impuros se curvarão. Nós amamos você. Nós somos leais a você.

Mas o que esta centopéia está mordiscando seu tímpano? Você enfa o dedo para tirá-lo, mas não consegue! Maldito! É aquele infernal Array de Dwemthy novamente. **Se explique Dwemthy!**

Aqui, eu revelo o próprio Array para você.

```
class ArrayDeDwemthy < Array
  alias _inspect inspect
  def inspect; "#<#{ self.class }#{ inspect }>"; end
  def method_missing( meth, *args )
    answer = first.send( meth, *args )
    if first.vida <= 0
      shift
      if empty?
        puts "[Uouuu. Você dizimou o Array de Dwemthy!]"
      else
        puts "[Prepare-se. #{ first.class } surgiu.]"
      end
    end
    answer || 0
  end
end
```

Até agora, provavelmente você está se sentido muito familiarizado com herança. A classe **DwemthysArray** herda de **Array**, portanto se comporta como um. Para tal sendo um mistério, é assustadoramente curto, não?

Portanto é um Array. Preenchido com monstros. Mas o que este código extra faz?

Inspect

O método **inspect** não é realmente uma parte necessária do Array de Dwemthy. É algo adicionado a Dwemthy como uma cortesia para seus visitantes. (Muito o chamam de pervertido, muitos o chamam de austero, mas todos nós somos ignorantes para ir sem admirar o trabalho que ele despendeu por nós.)

Cada objeto em Ruby possui um método **inspect**. Isto está definido na classe **Object**, então ele adiciona através da genealogia para cada pequenino objeto que nasce.

```
irb> o = Object.new
=> #<Object:0x81d60c0>
irb> o.inspect
=> "#<Object:0x81d60c0>"
```

Você percebeu? Sempre que criamos um objeto no **irb**, essa chamarativa palavra **#<Object>** é apresentada! Isto é um pequeno nome identificador para o objeto. O método **inspect** cria este identificador, que é apenas uma string.

```
irb> class Coelho
irb>   attr_accessor :slogan
```

```

irb> def initialize s; @slogan = s; end
irb> def inspect; "#<#{ self.class } diz '#{ @slogan }'>"; end
irb> end

irb> class FalsoCoelho < Coelho
irb> end

irb> Coelho.new "eu arrebentei a cara do dragão!!"
=> #<Coelho diz 'eu arrebentei a cara do dragão!!'>
irb> FalsoCoelho.new "Assim, assim e assim..."
=> #< FalsoCoelho diz 'Assim, assim e assim...'>

```

O fato é: `irb` está respondendo. Cada vez que você executa algum código no `irb`, o *valor de retorno* daquele código é inspecionado. Que prático. É uma conversinha entre você e o `irb`. E `irb` está apenas reiterando o que você está dizendo, então você mesmo pode ver.

Você poderia escrever seu próprio prompt Ruby de uma forma muito fácil:

```

loop do
  print ">> "
  puts ">> " + eval( gets ).inspect
end

```

Este prompt não irá permitir que você escreva código Ruby mais longo do que uma única linha. No entanto, esta é a essência do Ruby interativo. Como que você gosta? Dois de seus conceitos recentemente aprendidos vieram juntos de um modo mais saboroso. O `eval` pega o código digitado e o executa. A resposta de `eval` é então inspecionada.

Agora, como você está combatendo monstros no `irb`, o Array de Dwemthy será inspecionado e respondido com os monstros que você deixou de lutar.



Method Missing

Você não odeia quando grita “Deirdre!” e aproximadamente dez pessoas respondem? Isto *nunca* acontece em Ruby. Se você chamar o método `deirdre`, somente um método `deirdre` responde. Você não pode ter dois métodos com o mesmo nome. Se você adicionar um segundo método `deirdre`, o primeiro desaparece.

Você pode, no entanto, ter um método no qual **responde para muitos nomes**.

```

class ChamadorDeNome
  def method_missing( nome, *args )
    puts "Você está chamando '" + nome + "' e você diz:"
    args.each { |diz| puts " " + diz }
    puts "Mas ninguém está lá ainda."
  end
  def deirdre( *args )
    puts "Deirdre está bem aqui e diz:"
    args.each { |diz| puts " " + diz }
    puts "E ela ama cada segundo disso."
    puts "(Eu acho que ela pensa que você é poético.)"
  end

```

Quando você chamar o método `deirdre` acima, eu tenho certeza que você sabe o que vai acontecer. Deirdre vai amar cada segundo disto, você e suas fascinantes poesias.

Mas e se você chamar `simon`?

```
irb> ChamadorDeNome.new.simon( 'Olá?', 'Olá? Simon?' )
```

Você está chamando `simon' e você diz:

Olá?

Olá? Simon?

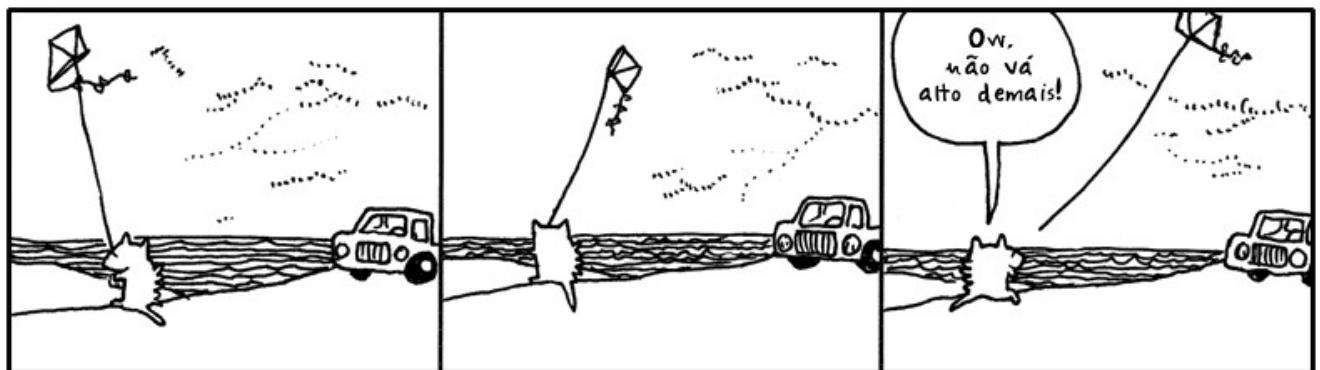
Mas ninguém está lá ainda.

Sim, **method_missing** (**método faltando**) é como uma secretária eletrônica, que intercepta a chamada de seu método. No Array de Dwemthy nos usamos um desvio, então quando você ataca o Array, ele passa o ataque diretamente para o primeiro monstro no Array.

```
def method_missing( meth, *args )
  resposta = first.send( meth, *args )
  # ... código recortado aqui ...
end
```

Veja! Veja! Aquele magrelinho **method_missing** passa a responsabilidade para outro!

4. Então, Sejamos Claros: O Porco-Espinho Está Agora Rumo Ao Mar



5.

Caminhando, Caminhando, Caminhando, Caminhando e Assim por Diante

A tardinha escureceu ao redor do par de raposas. Eles fizeram seu caminho através das vielas lotadas de gambás cantantes, e ruas onde girafas em casacos esportivos passavam por eles batendo com suas maletas. Eles continuaram caminhando.

E agora as lojas fechavam desenrolando suas portas de metal. Grilos cantavam das calhas e acotovelavam-se diante da relaxante mudança.



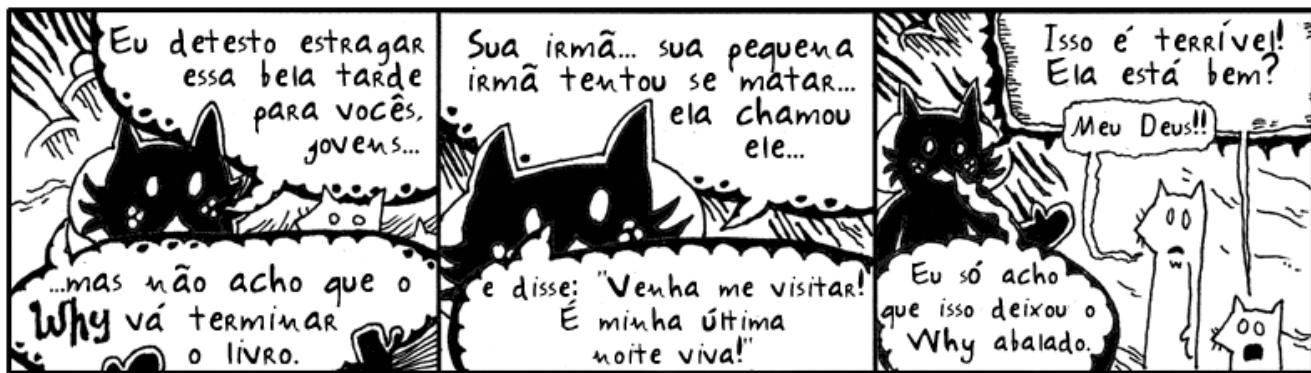
“De qualquer maneira, vocês devem admitir que ele é um péssimo Presidente,” disse Raposinho. “Por que o Presidente Marcos tem um coelho como Vice Presidente das Raposas.”

“O Vice Presidente? o coelho com as sobrancelhas?”

“Não, o coelho com os **enormes lábios de salsicha**,” disse Raposinho.

Mas sua conversa foi abruptamente interrompida por um gato sardento que surgiu do céu em cima da calçada.

O que isto significa?!



Ah, qual é? Isto é interessante. Mais foco.

Eu não vou aporrinhar ilustrando esta discussão que Blixy teve com os raposos neste momento! É tudo um punhado



de _conjectura_*. Como eles podem presumir que sabem o cenário do meu drama familiar? Eu amo minha irmã. Por um longo tempo, eu a adorei. (Esta é minha irmã Quil.)

Eu admito que houve um dia realmente doloroso há alguns meses atrás e eu meio que me apavorei. Eu estava deitado na cadeira de piscina no quintal da minha mãe. Eu tinha uma latinha de Dr. Pepper e um pedaço de torta alemã de chocolate. Estava comendo com um garfinho de criança. Todo o resto estava na lava-louças, isso era tudo que eles tinham. Três dentes.

Minha mãe começou a falar sobre Quil. Tudo sobre quanto dinheiro ela estava esbanjando em calças e bolsas. Uma bolsa de quinhentos dólares. Então ela disse, “Ela está perdendo isto. Ela parecia totalmente dopada no telefone.” (Elas cismou com isto, Quil estava fumando maconha e gostando.)

Então passei a notar o quanto observadora minha mãe podia ser. Por isto que, quando ela disse, “Na verdade acho que ela está na cocaína,” Eu fisicamente levantei e arremessei meu refrigerante do outro lado do quintal.

Ele foi parar navegando entre os troncos em algum lugar. Nós ficamos conversando por um tempo, estava escuro quando a latinha voou. Eu marchei um pouco. Então gritei a plenos pulmões.

Meu tio Mike estava parado com a porta de vidro aberta, me encarando. Ele disse algo totalmente nervoso como, “Ah, ok. Bem, eu vou—” E o chá no seu copo estava balançando para frente e para trás, derramando tudo. Ele desapareceu. Ele não é muito bom em falar coisas para as pessoas. Ele é mais um assobiador. E ressonante.



Então, sendo completamente honesto, sim, eu fiquei um pouco bravo. Eu fiquei bravo. Você sabe. Eu lidei com isto. Quil me liga regularmente. Por alguma razão estúpida, raramente ligo para ela.

E mais, ela não acabou se matando. Então isto não é assunto. Quem pode saber se isso era real. Ela apenas tomou

muita vodca. E ela é pequena. Então foi apenas assustador ver a Quil embebedar-se daquele jeito. Quer dizer forçar-se a isso.

Mas por que falar sobre isso? Isso fará apenas ela sentir que estou desapontado. Ou que sou um idiota.

Bem, saí um pouco da linha. Onde eu estava? Blix está basicamente ajudando os raposos, colocando-os na trilha do seu caminhão. Sim voltando aquilo tudo.



Tirinha: Sapos que guardam acentos no ônibus.

“Não podemos nos apertar para ir neste ônibus” disse Raposinho.

“Pessoal, subam,” disse Blix. “O que está impedindo? Ah, os sapos. Sim, apenas se apertem.” Blixy empurrou por trás.

“Ei,” disse Raposão. “Estou comprimido neste pequeno degrau! Alguém se mexa!”

“Você conseguiu— jovem raposa??” disse o gato.

“Não,” disse Raposinho, “não pode ver? O motorista continua balançando a cabeça e isso *realmente* me deixa nervoso. Eu não acho que ele nos queira.”

“Vai,” disse Blix. Desceu do seu degrau e andou pelo ônibus, espreitando através das janelas de acrílico. “Bem, eu não sei pessoal. Eu não sei. Eu acho que tem muitos sapos.” Ele bateu na janela. “Ei! Mexam-se!”

E esta é a realidade de andar no trânsito intermunicipal em Wixl. É terrivelmente competitivo. O ônibus da manhã é tão cheio que a maioria dos animais de colarinho branco conseguem sapos para guardar seus lugares durante o período da noite. Por qualquer razão, isto funciona. Se tornou mercadoria do fluxo de trabalho de sua economia.

Se você consegue reunir um pouco de imaginação, pode ver um **sinal de percentagem** como um rosto inclinado de um sapo. Visualizou a figura em sua cabeça? Agora deixe-me mostrar os sapos que acampam em strings.

```
# 0 formato %s serve para posicionar strings cheias.  
irb> "Assentos são ocupados por %s e %.%" % ['um sapo', 'um sapo com dentes']  
=> "Assentos são ocupados por um sapo e um sapo com dentes."  
  
# 0 %d formato serve para colocar números, enquanto o formato %f serve para  
# floats (números decimais).  
irb> sapos = [44, 162.30]  
irb> imediatamente = "Os sapos preencheram %d assentos e pagaram %f cristais azuis."  
irb> imediatamente % sapos  
=> "Os sapos preencheram 44 assentos e pagaram 162.30 cristais azuis."  
  
# A formatação é flexível com tipos, você pode transmitir em strings  
# e formatá-los como números.  
irb> sapos = ['44', '162.30']  
irb> imediatamente % sapos  
=> "Os sapos preencheram 44 assentos e pagaram 162.30 cristais azuis."
```

O que você está vendo acima usa o método **%** nas classes **String**. Este método pega **um string** e **um array** e mastiga-os para criar uma nova string. Os itens da lista são puxados (em ordem) e colocados em seus lugares reservados. É o começo de um dia de negócios e os sapos já fizeram seu trabalho.

```
# Observe, aqui está o método String#% chamado como os outros métodos.  
irb> "Mexe-se por favor, %s.%(% 'sapo desdentado' )  
=> "Mexe-se por favor, sapo desdentado"
```

```
# Agora vamos chamar da maneira mais bonita, com o sinal de porcentagem
# entre o string e o array.
irb> "Aqui está sua frase 1098 para o ano, %s." % ['sapo com dentes']
=> "Aqui está sua frase 1098 para o ano, sapo com dentes."
```

Isto também está disponível como o método `Kernel::format` ou o método `Kernel::sprintf` (Na linguagem C, tem um método `sprintf` que opera assim.)

```
irb> format "Sapos são empilhados %d de profundidade e viajam a %d mph.", [5, 56]
=> "Sapos são empilhados 5 de profundidade e viajam a 56 mph."
```

Para a maioria das partes, você precisará somente `%s` (strings), `%d` (números inteiros) ou `%f` (números float) especificadores de formato. O símbolo `%p` irá executar `inspect` num objeto.

Sim, então, o formato sapo é realmente útil para construir strings que são montados por diferentes tipos de dados. Você pode aprender todos os vários tipos de especificadores de formatos lendo a pagina [ri sprintf](#). Só te darei alguns rápidos indicadores.

Vamos dizer que você tem um array mas quer que os itens apareçam em uma **ordem diferente** na string. Numa situação como esta, você pode identificar itens específicos (colocando `1$` para o primeiro item, `2$` para o segundo, e assim por diante) depois do sinal de porcentagem.

```
irb> "Este ônibus tem mais %1$d paradas antes das %2$d horas. Isto são mais %1$d paradas." %
[16, 8]
=> "Este ônibus tem mais 16 paradas antes das 8 horas. Isto são mais 16 paradas."
```

A segunda dica que tenho para você é que pode destinar um certo número de caracteres para cada item, uma largura. E se um item é menor que a largura, espaços extras serão usados antes do item. Para preenchê-lo. Se a largura é um número negativo, o item será forçado para esquerda e o preenchimento virá depois dele.

```
# Dê a um item 30 caracteres de largura
irb> "Na traseira do ônibus: %30s." % ['sapos']
=> "Na traseira do ônibus:                                     sapos."
```

```
# Dê a um item justificado a esquerda 30 caracteres de largura
irb> "Na parte da frente do ônibus: %-30s." % ['sapos']
=> "Na parte da frente do ônibus: sapos                 ."
```

Raposinho continuou olhando para o motorista do ônibus. Lembre-se, ele não entraria no ônibus!

“Qual o problema?” disse Raposo. “Você não pode entrar e ficamos no corredor?”

“Você realmente quer entrar neste ônibus? Aquele motorista não tem mãos,” disse Raposinho, falando perto de maneira a aquietar Raposo, “e tudo o que ele tem, ao invés de mãos, são copos com canudos.

“E então? Você não acha que animais com tentáculos podem dirigir?”

“Bem, não é só ele que vai se dar mal ao volante, mas ele tem todas essas pernas em cima dos pedais. Isso não é inteligente. Vamos pegar outro ônibus. Vamos lá.”

“Você sabe, ele provavelmente está sendo conduzido como todos os dias. Será que ele vai realmente começar a bater nesta fase da carreira dele?”

“Os ônibus batem”, disse Raposinho. “Alguns fazem. Isto cheira a batida”.

“Desvia ,merda!” e Raposo gritou para o motorista, “Ei, motorista, há quanto tempo você dirige este ônibus?”

O motorista do ônibus olhou misteriosamente por baixo da aba do seu boné e começou a virar-se em direção a eles, mas os seus tentáculos ficaram presos à roda. Ele rapidamente sacudiu suas pernas da frente e, falhando em soltá-las, virou-se para a direção e concentrou as suas energias em ordenhar de suas glândulas um

Mais do Arrepiante Prenúncio Progressivo dos Comedores de Cachecol

(do Capítulo XII: *Graças ao Céu pelo Pequeno Homem.*)

“Feche a porta,” Spencer repetiu, mas a mão de Lara tremeu e ela negligentemente tateava o trinco. Seu pai não a ensinou a fechar portas como esta.

“Sim, esta é uma porta incomum,” disse Brent. Ele andou até lá e fechou a porta para ela. Então, ele segurou as mãos dela e olhou nos seus olhos. Os olhos dele acenderam como enormes palitos de fósforos que seriam

pouco de ótimas secreções. Bolhas de muco gotejaram. “Vamos cair fora daqui”, disse Raposão e os dois correram para fora da rua, batendo direto no gato Blix.

“Está bem, bem, o ônibus está cheio,” disse Blix. “Eu não sei por que o motorista parou se ele sabia que o ônibus estava cheio de gafanhotos.”

“Nós estamos pensando que ele estava prestes a bater contra nós,” disse Raposão, “e ele abriu a porta para fazer parecer como uma parada planejada.”

“Tenha em mente, Blix, nós não tínhamos realmente discutido essa possibilidade em voz alta, então eu não tive uma chance para concordar formalmente,” disse Raposinho. “No entanto, isto parece racional para mim.”

“Eu estou pensando que todos os ônibus vão estar cheios como este.” Blix mordeu seu lábio, pensando e piscando os olhos. “Vamos apenas—” Ele apontou abaixo o circuito de edificações de apartamento que mudariam para o sul. “Mas talvez—” Ele olhou para cima e avaliou as estrelas, coçando a cabeça e contando as constelações com leves toques na pontas de seu dedo.

“Você está obtendo nossa orientação a partir das estrelas e planetas?” perguntou Raposinho.

Blix não falava, ele abaixou a cabeça para o norte através de uma avenida uma avenida mal colocada voltar atrás da loja de tinta. Mas antes que nós os sigamos em baixo nessa estrada de serviço, cara pálida, Eu tenho mais um sapo para você, empoleirados sobre uma longa vitória régia que esticadas para fora para prender alguma coisa em tudo.

```
irb> gato = "Blix"
irb> puts "O #{ gato } vê o que esta rolando? Está o
#{ gato } prevenido??" 
=> "O Blix vê o que esta rolando? Está o Blix
prevenido??"
```

Os pequenos sapos de anteriormente (%s ou %d) eram apenas incógnitas para strings únicas. Guardando lugares na string.

As vitórias-régia começam com um broto de uma flor, a **cerquilha** (sustenido ou jogo-da-velha). Você também já o viu como um sinal da libra em telefones. Depois dos brotos das flores, duas folhas formam as bordas da vitória-régia. As folhas são **braços enrolados** (chaves), também vistas muitas vezes antes como (*as garras do caranguejo*) para um bloco de código.

Uma vitória-régia vazia "#{}" se torna uma string vazia "".

Quando a vitória-régia é encontrada em uma string com **aspas duplas**, Ruby executa algum código encontrado entre as duas folhas da vitória-régia. A vitória-régia é erguida e o resultado do código é colocado numa string. Esta troca da vitória-régia é chamada de *interpolação de string*.

```
irb> companheiros = ['Blix', 'Raposão', 'Raposinho']
irb> puts "Deixe-nos seguir #{ companheiros.join ' e
' } em sua viagem."
=> "Deixe-nos seguir Blix e Raposão e Raposinho em
sua viagem."
```

enormes demais para ser prático. “Esta é uma maçaneta incomum, que assegura que aqueles que não comem cachecóis fiquem de fora.”

“Sentem-se, todos,” ordenou Spencer, enquanto delimitava seu espaço através da sala e assumia o comando. “Eu comando esta organização,” declarou. “A organização secreta dos Comedores de Cachecol!”

As tochas rodeando a sala queimavam como invencíveis fósforos enormes e o punhado de adolescentes sentou-se. Exceto por Spencer que permaneceu em pé alto e majestoso, aspirando todo o oxigênio na sala inteira em suas narinas antes de falar.

“Um de nós,” ele disse, dramática e invencivelmente, “está faltando!”

A sala inteira se surpreendeu, que também utilizou oxigênio. A sala estava agitada.

“Quem?” “Como?” “Quem foi?” Ninguém sabia. Exceto por Spencer, que atravessou a sala e tomou o controle.

“Nosso querido amigo Steve Bridell foi roubado de nós,” anunciou Spencer em uma voz ensurdecedoramente alta, como se milhares de palitos de fósforos gigantes fossem atirados contra uma superfície sensível em harmonia no meio de uma pilha de oxigênio. “Steve Bridell. Algum de vocês conhece Steve Bridell?”

A sala estava em silêncio.

Spencer continuou. “Steve Bridell era um recurso incrível e todos de vocês o conheciam e o amavam. Ele esculpiu este enorme homem de madeira que usamos como pódio.” Spencer apontou. “Ele também fez a série de chocinhos de madeira que está agora lá atrás no nosso armário de instrumentos.”

Alguns da platéia levantaram.

“Espere,” instruiu Spencer. “Não vão lá traz. Eu já chequei. Os chocinhos se foram!”

A vitória-régia é muito estável e pode armazenar qualquer tipo de código em seu interior. Acima estamos usando `Array#join`, mas você pode fazer qualquer coisa que queira. Chamar métodos de objeto, condicional de declarações `if` ou `case`, até mesmo definir classes.

```
irb> blix_foi = :norte
irb> puts "Blix não falou, ele curvou a cabeça para o #{ blix_foi } voltando-se para #{ 
           if blix_foi == :norte
             'uma avenida mal colocada atrás da loja de tinta'
           elsif blix_foi == :sul
             'o circuito de edificações de apartamento'
           else
             '... bem, quem sabe onde ele foi.'
           end }. Mas antes que os sigamos..."
=> "Blix não falou, ele curvou a cabeça para o norte voltando-se para uma avenida mal
colocada
               atrás da loja de tinta. Mas antes que os sigamos..."
```

Os raposos seguiram Blixy por trás da loja de tinta pelo asfalto rachado e irregular. Todas as lojas na pista dilapidada se inclinaram em ângulos entre elas. Em alguns lugares, placas da calçada sobressaíam da terra, formando uma arriscada passagem, uma desordenada pilha de bordas. Quase como se os planejadores da cidade tinham esperado para pagar tributo para as placas tectônicas. Uma pequena farmácia tinha deslizado para baixo da superfície, quase longe de vista.

Verdadeiramente, embora era colorido. A loja de tinta tinha jogado para fora as tintas velhas diretamente em seus vizinhos. As lojas mais próximas da loja de tinta foram obstruídas com centenas de cores, ao longo dos parapeitos das janelas e nas calhas de chuva. Sim, nas paredes e calçamento.

Basicamente, começando com a entrada da loja de tintas, a avenida entrou em erupção em uma gigantesca incongruente mercado infelizmente colorido.

Mais a diante, um escritório de um dentista foi preparado com a tinta vermelha, um artista inexperiente descreveu um grande bebê que tinha caído através de uma chaminé e chegado em um lareira completamente de fuligem. Levantou uma nuvem grossa e preta de cinzas durante o impacto, facilmente confundido pelo cabelo grosso sobre os braços e costa da criança. A criança olhou longe e viu que era nova para ter muito cabelo, mas lá eles eram: rico, ondas loiras qual caíram generosamente da cabeça da criança. Sob as pernas da criança foi pintado a palavra *BREWSTER*.

O mesmo artista tinha batido na biblioteca próximo a loja e tinha batido com pressa junto a um mural de carro esportivos verde sendo puxado da lama por uma equipe de bebês sem perna puxando-os com correntes brilhante. Outra vez, a drástica onda loira!

“Eu preciso de respostas,” disse Raposão, que tinha terra para uma parada em frente de vista.

“Eu estou começando acreditar que não existe,” disse Raposinho. “Talvez estes são as respostas.”

“Brewster?” disse Raposão. Ele caminhou mais próximo a biblioteca e tocou o mordente de um dos crianças sem perna que estava mais próximo do ponto de vista. O mordenite da criança apareceu com a mandíbula de baixo.

Blix foi a outras duas casas para baixo, navegando através da construção torta de alvenaria, a sarjeta pavimentada que o conduziu para R.K. ‘s *Gorilla Mint*, com a etiqueta metálica leu sobre a porta. O edifício foi emplastrado com logotipos miniaturas para a variedade de opções de pagamento e uma aceitável identificação em R.K. ‘s *Gorilla Mint*. Mesmo as barras sobre a janela foram alinhadas com divulgações de seguro e avisos de seguranças e selos de autorização do governo, assim como adicionando a todos estas, etiquetas coberta com papel carbono cobrindo posterres rasgados e propagandas. E tudo misturado com respingos de tinta que penetrou-os satisfatoriamente.



“Eu gosto da maneira que sinto o papel fresco contra minha língua,” disse o gorila no caixa. os dedos dele esfregaram silenciosamente as notas. Ele aproximou seu rosto em direção a moeda fresca e passou em seu nariz o dinheiro macio.

“O R.K. está aqui nessa noite?” perguntou Blix.

“R.K não está,” disse o caixa gorila. ele voltou-se para os três viajantes e espalhou o seu dinheiro na superfície do caixa, espaçando-os igualmente e alinhando todos os cantos impecavelmente. “Agora, qual destes vocês acham que vale mais?”

Os raposos olharam sobre as diferentes notas e Raposinho murmurou a si mesmo, “Bem,” talvez— não, mas eu apostarei— Espere, uma dessas tem bananas nela? Por que esta não, nenhuma fruta ou corda de pular ou— terrível, isto é difícil!” e em uma voz mais baixa, “Tão difícil ler. O que isso quer dizer?” Símbolos ou alguma coisa? Se todas essas notas têm símbolos, será impossível para nós compreender qual é a de maior valor.”

“Foi por isso que eu disse, ‘Adivinhe.’” O gorila bateu cada nota em ordem. “Veja, você tem” uma em cinco chances.”

“A menos que os símbolos signifiquem alguma coisa, disse Raposão. “A não ser que nós pudermos compreender”

“Nós podemos compreender,” disse a Raposinho.

“Não,” disse o gorila. “os símbolos não têm significado algum.”

“Seja quem for que criou o dinheiro pensou em algum significado para eles,” disse Raposinho. “Por que usar _este_símbolo?” Ele apontou para uma conjunção (e comercial) impressa em tinta escura.

“Sim, nós vimos você cheirando o dinheiro e fantasiando a respeito lá atrás,” disse Raposão. “Eu apostei que estes símbolos significam todo tipo de coisas para você!”

“Não, Eu acho que não,” disse o gorila.

Se eu puder avaliar por este ponto, eu acho que os símbolos têm significado. Eles podem não ser *carregados* com significado, eles podem não estar transbordando significado através das rachaduras, mas eu estou certo que existe uma lasca de significado.

```
irb> $:
=> ["/usr/lib/ruby/site_ruby/1.8", "/usr/lib/ruby/site_ruby/1.8/i686-linux",
"/usr/lib/ruby/site_ruby",
"/usr/lib/ruby/1.8", "/usr/lib/ruby/1.8/i686-linux", ".."]
```

Variáveis que começam com o símbolo monetário são variáveis globais. Elas podem ser vistas de qualquer lugar do programa, **de dentro de qualquer escopo**. (Dr. Cham usava esta variável enquanto bisbilhotava em volta da baia do computador dos Originais.)

Então porque **o sinal de dinheiro seguido por dois ponto** representa um array de todos os diretórios onde Ruby irá procurar quando você tenta carregar um arquivo com `@require@*`? O símbolo de dinheiro significa “global.” Mas por que os dois pontos?

Historicamente, em muitos sistemas operacionais, uma lista de diretórios continham os dois pontos que separava cada entrada. Eu gosto de ver os dois pontos como um par de olhos, vasculhando os diretório por arquivos. Nós armazenamos nossa lista de observação atrás destes olhos.

Aqui estão mais algumas variáveis globais especiais:

```
irb> $          # A variável $ contém todos os arquivos que foram carregados com 'require'
=> ["irb.rb", "e2 mmap.rb", "irb/init.rb", ... "rbconfig.rb"]
      # Estes arquivos estão armazenados em outros lugares, mas seus
```

```
# códigos estão sendo usados neste programa. Como se estivéssemos
# referenciando o trabalho de outra pessoa -- estas são as notas de rodapé
# -- por isso as aspas duplas.

irb> $0      # A variável $0 o nome do arquivo do programa que está sendo rodado.
=> "irb"    # Um zero pode ser considerado como o início da contagem de números.
            # Esta variável responde a seguinte questão, "Onde este programa começou?""

irb> $*      # A variável $* contém todos os argumentos passados para um programa.
=> ['--prompt', 'simple']
            # Este aqui é fácil de se lembrar, se você se lembrar que os métodos Ruby
            # também usam asteriscos para capturar argumentos em um array.

# O $! contém a exceção atual que foi levantada.
# A exclamação indica um estado de alerta. Uma exceção!
irb> begin
irb>   raise TypeError, "Não acredito nesta informação."
irb> rescue
irb>   p $!
irb> end
=> #<TypeError: Não acredito nesta informação.>

# O $@ contém o backtrace atual, caso uma exceção tenha sido provocada.
# O backtrace apresenta onde o Ruby _estava_ quando a exceção foi provocada.
irb> begin
irb>   raise TypeError, "Não acredito nesta informação."
irb> rescue
irb>   p $@
irb> end
=> ["(irb):25:in `irb_binding'", "/usr/lib/ruby/1.8/irb/workspace.rb:52:in `irb_binding'",
     "/usr/lib/ruby/1.8/irb/workspace.rb:52"]
```

“Eu não me lembro de você.” Blix olhou para o gorila com muito interesse. “Você é um dos filhos de R.K. ou algo parecido?”

“Ah, vamos lá!” disse Raposinho, segurando uma nota com uma exclamação no nariz do gorila. “Não me diga que isso não significa *nada* para você! Esta aqui é provavelmente *muito importante* já que tem uma exclamação nela. Talvez ela pague por coisas de emergência! Contas de hospital ou algo do gênero!”

“Sim, cirurgia!” disse Raposão.

O gorila olhou para os raposos com desgosto sob a borda de seu boné. “Não, vocês estão errados. Vocês não podem pagar por cirurgias com isso.”

“Mas você entende nosso ponto de vista”, disse Raposinho. Ele agarrou algumas das outras notas. “E você me diz que esta conta *não pode* pagar por cirurgias? Bem isso parece um propósito específico *não relacionado a cirurgia*. Agora, esta aqui com o interrogatório. Para que ela serviria?”

“Ei, me de estas aí,” o gorila apanhou as notas sobre o balcão, mas seu longo polegar continuava a ficar no caminho e toda vez que ele achava que tinha pegado as notas, ele percebia que só tinha conseguido pegar seu próprio dedão.

“Ei, ei, olhem, ele está nervoso,” falou Raposão, aplaudindo com alegria. “Eu me pergunto por quê. Você percebeu quão bravo ele ficou uma vez que falamos todos estes significados interessantes? *Nós estamos espertos com você! *Nós entendemos seu jogo tão rápido!*”*

“Totalmente!” disse Raposinho, um de seus braços pego pelo gorila, o outro braço agitando uma nota que exibia um travessão. “Esta aqui é para comprar suprimentos para o chão, talvez até grandes rolos de ladrilhos e tecidos impermeáveis”

“Vê,” disse Raposão, trabalhando para soltar os dedos do gorila, “nós apenas temos que descobrir o que é mais caro: cirurgia ou tecido impermeável! Isso é *tão* fácil”_-

“**NÃO É NÃO!**” berrou o gorila, chacoalhando Raposinho e o batendo com suas palmas. **VOCÊ NÃO SABE NADA SOBRE DINHEIRO DE MACACO!! VOCÊ NÃO TEM MESMO SEU PRÓPRIO TIPO DE DINHEIRO!!**”

Nós poderíamos *facilmente* ter nossos próprios tipos de dinheiro!” Disse Raposão, pegando o boné do chimpanzé e jogando-o para o fundo da sala, onde ele navegou para trás de uma parede de caixas de depósito de segurança. “E — *seu chapéu está longe daqui!*”

“Vamos lá, devolva logo as notas dele,” disse Blix, acenando os braços sem chance de ajuda, fora do jogo.
“Poderíamos realmente ter a ajuda deste cara.”

“Pare de bater em mim!” gritou Raposinho. “Estou quase entendendo esta aqui com os pontos!!”

De repente, com grande precisão e sem avisar, Raposão agarrou o nariz do macaco e bateu sua cara contra o balcão. As canetas e tintas em sua superfície agitaram-se e “Bam!” disse Raposão. Os olhos do gorila viraram-se com sonolência a medida que seus braços... então seu pescoço... e então sua cabeça escorregaram para o chão atrás do balcão.

Aqui estão mais algumas variáveis globais que talvez você possa querer usar:

```
irb> $/      # O $/ é a linha separadora, ele é normalmente configurado para \n, o qual  
representa o _Enter  
=> "\n"    # ou "o final da linha". A barra representa uma espada que corta e reduz linhas  
em um arquivo.  
  
# A linha separadora controla como os métodos each_line ou readlines quebram as strings.  
irb> "Jeff,Jerry,Jill\nMichael,Mary,Myrtle".each_line { |nomes| p nomes }  
=> "Jeff,Jerry,Jill\n"  
=> "Michael,Mary,Myrtle"  
  
# Se você mudar a linha separadora, você muda como os métodos trabalham, tal como o  
each_line.  
# Veja o que acontece quando eu mudar a linha separadora por uma vírgula.  
irb> $/ = ','  
irb> "Jeff,Jerry,Jill\nMichael,Mary,Myrtle".each_line { |nomes| p nomes }  
=> "Jeff,"  
=> "Jerry,"  
=> "Jill\nMichael,"  
=> "Mary,"  
=> "Myrtle"  
  
irb> $,      # O $, é a variável que une o separador, usado ao unir strings com  
=> nil     # Array#join or Kernel::print. A vírgula é um caractere comum de junção.  
  
# O separador de junção está normalmente vazio.  
irb> ['vela', 'sopa', 'mackarel'].join  
=> "velasopamackarel"  
irb> $, = '*' ; ['vela', 'sopa', 'mackarel'].join  
=> "vela * sopa * mackarel"  
  
# Mas, geralmente, você não necessitará de uma variável global.  
irb> ['vela', 'sopa', 'mackarel'].join '#'  
=> "vela # sopa # mackarel"  
  
irb> $;      # O $; a variável é o separador que divide, usado ao dividir strings  
=> nil     # com String#split.  
  
# O separador split está normalmente vazio, que significa que String#split separará  
# a string onde há um espaço em branco.  
irb> "vela sopa\nmackarel".split  
=> ["vela", "sopa", "mackarel"]  
irb> $; = 'a'; "vela sopa\nmackarel".split  
=> ["vel", "sop", "\nm", "ck", "rel"]  
  
# Mas, geralmente, você não necessitará de uma variável global.  
irb> "vela # sopa # mackarel".split '#'  
=> ['vela', 'sopa', 'mackarel']
```

Fora do *Casa da Moeda do Gorila*, Blix repreendeu aos raposos. “Nós poderíamos ter aproveitado a ajuda desse cara! Se sabe onde está R.K., nós poderíamos usar a astúcia dele!”

“**Não precisamos do dinheiro do macaco!**” disse o Raposinho. “**Podemos criar nosso próprio dinheiro!**”

“**Nós podemos suportar pulseiras eletrônicas!**” disse o Raposão.

“Seu dinheiro é sem valor,” disse Blix. “O dinheiro é do gorila. Não tem valor. Ele é pior do que os cristais azuis.”

“Mas serve para uma finalidade,” disse o Raposão.

“Não ele não serve,” disse Raposinho. “Ele apenas disse que o dinheiro não tem importância.”

“Mas sobre o tapete impermeável e as cirurgias?” disse Raposão.

“Sim,” disse o Raposinho para Blix. “O que dizer do tapete impermeável e as cirurgias?”

“Se todos os hospitais possuem uma equipe de gorila e toda as casas receberam uma rede de melhorias estritamente operadas por gorila, então — SIM — você poderia comprar o tapete impermeável e as cirurgias. Mas eu *garanto* que você teria um tapete impermeável desleixado e muitas cirurgias horríveis. Não acho que você conseguiria tirá-lo desta economia com vida.”

“Assim, se R.K. é tão esperto,” disse o Raposão, dando risada dissimuladamente, “por que imprime tal moeda sem valor?”

“Ele é a tampa para outras atividades,” disse Blix. “Além do que, se você é tão esperto, por que você recorreu violentamente derrotando aquele pobre gorila?”

“Eu suponho que foi um mau jogo,” afirmou Raposão, pendendo sua cabeça. “Meu amigo, direi a você que fiquei de saco cheio o dia inteiro.”

“E sua raiva finalmente levantou seu focinho fumegante!” falou Raposinho. “Você finalmente está vivendo a vida à altura do seu cavanhaque.”



Percorriam sobre as pistas, os dois raposos distraídos no seu rumo, mas divertindo-se agora que tinham Blix conduzindo-os pelo caminho com tanta urgência. Eles passaram o tempo vagando descuidados bem atrás de Blix e gastando sua tarde importunando a maioria dos pedestres.

Um dos alvos dos seus permanentes comentários eram Os Transportadores Alados de Pergaminho, pares de morcegos que carregam documentos que precisam ser imediatamente autenticados. Não pode haver nenhum atraso, eles devem ser ligeiros, não há mesmo tempo para enrolar o pergaminho, não, eles devem deixar cair seus queijos suíços e sair pela porta.

Estes correios assemelham-se a um tipo de construção do Ruby chamado **tipos delimitados**. Uma série longa de caracteres equivale ao pergaminho, defendido em cada lado por um morcego cercando suas asas onduladas para manter o pergaminho unido. O morcego da ponta usa um chapéu em que está escrito %w, que identifica o pergaminho como um conjunto de palavras.

```
irb> bats = %w{Os Transportadores Alados de Pergaminho}
=> ['Os', 'Transportadores', 'Alados', 'de', 'Pergaminho']
```

Os morcegos de %w e seus pergaminhos, quando alimentados dentro do Ruby, emergem como um array de palavras. Esta sintaxe é um atalho caso você não queira atravessar o problema de decorar cada palavra com vírgulas e citações. Você está com pressa, também, não pode haver nenhum atraso. Você rabisca as palavras entre os morcegos e deixa o Ruby descobrir onde cortar.

Outros morcegos, outros chapéus. Por exemplo, o chapéu de %x funciona como um programa externo.

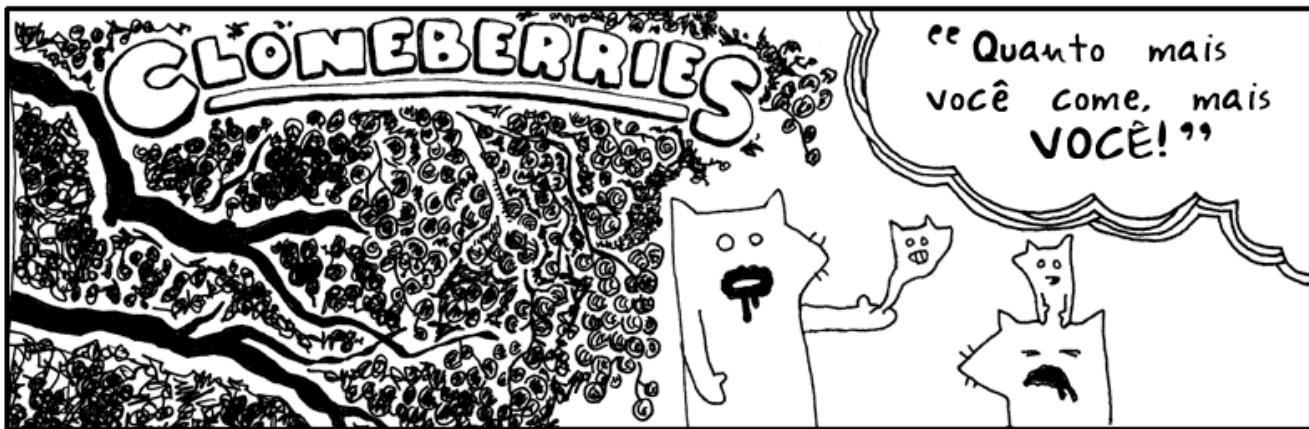
```
irb> %w{ruby --help}
=> ["ruby", "--help"]
irb> %x{ruby --help}
=> "Usage: ruby [switches] [--] [programfile] [arguments] ..."
```

Meu favorito é o chapéu Q. que pode também ser escrito apenas como %. Isto atua como uma string de aspas duplas, mas fica legal quando usado com strings que ocupam muitas linhas. Por exemplo, digamos que você quer adicionar um novo método com eval.

```
m = "morcegos!"
```

```
eval %{
  def #{ m }
    puts "{" * 100
  end
}
```

Assim como se faz com uma string de aspas duplas, você pode usar interpolação de strings com vitórias-régia (#).



Blix sacudiu sua cabeça. “Oh, não.”

“Caramba! Minha mão está grávida,” disse Raposão, vendo o pequeno embrião de raposa escorregando em sua mão. “Elas são boas frutas, apesar de tudo,” disse Blix. “O vinho que fazem destas frutinhas fará nascer alguns olhos nos seus dentes. Mas nada além disso.”

“Ah, dor!” gritou Raposinho, a medida que sua miniatura saia através dos poros de seu couro. Mas logo ele estava ninando seu pequeno eu e murmurando canções de ninar. *Nunca mais, nunca mais, cantava docemente o rouxinol. Luz piscante das estrelas, dormindo tranqüilo, enquanto empoleirado num pedaço de Plátano.*

Fazer duplicatas dos objetos Ruby não é nada mais do que uma fruta de código.e.

```
irb> arvore = [:fruta, :fruta, :fruta]
=> [:fruta, :fruta, :fruta]
irb> filhoarvore = arvore.clone
=> [:fruta, :fruta, :fruta]
```

O método `clone` faz uma cópia exata de um objeto Ruby. Como isso difere de uma atribuição comum?

```
irb> arvore_charles_william_iii = arvore
=> [:fruta, :fruta, :fruta]
```

A atribuição de um objeto a variáveis apenas cria mais apelidos. O Array acima pode ser chamado `arvore_charles_william_iii` agora. Ou o mais curto `arvore`. O mesmo objeto, mas nomes diferentes.

Entretanto, um clone é uma cópia de um objeto. Você pode alterá-lo sem afetar o original.

```
irb> filhoarvore << 'flor'
=> [:fruta, :fruta, :fruta, 'flor']
irb> arvore
=> [:fruta, :fruta, :fruta]
```

Contudo, o método `clone` não faz cópias de tudo que está amarrado ao objeto. No array acima, apenas o array é copiado, não todos os símbolos e strings que estão dentro.

Você também pode ver o método `dup` sendo usado para copiar objetos. O método `dup` faz cópias que não são tão exatas. Por exemplo, existem objetos em Ruby que estão “congelados” e não podem ser alterados. Se você fizer um `clone` do objeto, terá uma cópia exata que também estará congelada. Se você usar `dup`, você terá uma cópia descongelada que você poderá alterar se quiser.

O método `clone` também copia a metaclass de um objeto, enquanto o `dup` não.

```
irb> o = Object.new
irb> class << o
irb>   def nuncamais; :nuncamais; end
irb> end
```

```

irb> o.clone.nuncamais
=> :nuncamais
irb> o.dup.nuncamais
# NoMethodError: undefined method `nuncamais' for #<Object:0xb7d4a484>
#           from (irb):7

```

Nem sempre você precisa fazer cópias dos objetos, de qualquer maneira, já que muitos métodos como **collect**, **gsub** e **format** fazem cópias para você como parte trabalho deles.



Sobre as colinas e nos vales, eles correm pela grama onde perambula o Veado da Fumaça Rosa Bafejante. O sol estava encoberto pelas pesadas nuvens rosas, repletas de linguagem de veado, colorindo o horizonte com um gradiente de toranja e produzindo um brilho sobre a colina. As nuvens deslizavam umas sob as outras, algumas flutuando para cima, com destino a parentes canadenses. Outras aterrissando numa distância legível dos cascos de quem vê.

“Vamos parar! *Por favor!*” gritou Raposão. Você não espera que corremos no meio desta **penugem irrespirável**!”

“Por que você está gritando?” falou Blix, enquanto uma fina camada de nuvens flutuava arás de suas pernas. “Você não precisa levantar sua voz além de um sussurro. Estas longas e finas nuvens são geralmente só um murmuro ou um suspiro. Elas podem conseguir nem mesmo chegar até seu destino.”

“Todos esses escritos nas nuvens são conversa de cervos?” disse Raposinho.

“Ajudem-me! *Onde vocês estão, caras?*” Raposão abaixou-se através de uma tempestuosa lengalenga formada por uma densa ondulada fumaça e moitas afiadas. Ele girou em todas direções, “Alguém grite se estiver aí!”

Procurou por uma fenda na densa matéria, penteando para frente com suas mãos. As eloquentes, furiosas nuvens responderam-no espetando-o para adiante, forçando-o para cantos apertados em suas breves pausas entre as frases. Ele ficou em um sumidouro e manteve sua cabeça baixa a medida que as cascatas de fumaça agitavam-se adiante.

“Sim, os cervos podem ler estas coisas,” afirmou Blix. “Eles apenas encaram seus alvos e atiram-nas de suas narinas. Escutei uma vez sobre um cara que tinha **cavalgado** em um poema de amor cervídeo.”

“Sem chance,” falou Raposinho.

“Sim,” disse Blix. “E este cara era eu.” Blix alcançou seu ombro e juntou-se a uma espiral de fumaça que girava acima de sua cabeça. “Você apenas tem que saber quais nuvens são fracotes e quais são pomposas”. Blix deixou a nuvem puxá-lo e quando a nuvem voou para cima, Blix perdeu seu apoio e manteve seus pés movendo vagarosamente ao longo do chão. “Vê, está aqui é uma das boas, longa como um cabo de vassoura. Um cara descobriu uma vez que tinha a forma *exatamente igual* a de um carro: pára-brisa, airbags laterais para o motorista, direção hidráulica.”

Bifes e Escorregadores

Meus tios amam toboágua e churrascarias. Eles têm dias de toboágua seguidos por um pulo na churrascaria do Joey. Eu *odeio* a churrascaria do Joey. Lá tudo é muito grande, carne com cara de sola de sapato. Borrachenta. Misturado com fedor de cloro dos meus tios.

Dedos enrugados nas fatias de carne é Revoltante.

Agora é a hora dos bifes e toboágua se unirem de maneira nojenta. Meus tios tiveram bifes e toboágua a vida inteira. A dinastia dos bifes e dos toboágua tem que acabar. Eu farei os dois nadarem juntos contra a natureza!

Como isto:

- Entrego os bifes para um cavaleiro a bordo do toboágua. O cavaleiro olha para o salva-vidas. O salva vidas diz espere ai. O cavaleiro olha novamente. O salva-vidas não fala mais nada. E então, agora é a hora. *Vai garoto, vai!* E o olhar daquela criança enquanto escorrega toboágua abaixo, patas cheio de pedaços de bife! *Vai garoto, vai!*

Excepcional!"

"E este cara era—"

"Ele era!" E Blix escalou sobre a longa nuvem gelada, com seus hieróglifos pendurados, e manteve-se em pé orgulhosamente, flutuando bem acima da pontuda sombra do raposo.

"Ah, eu poderia fazer isso" disse Raposinho. "Eu e Raposão andamos de jetski o tempo todo. *Eu fico em pé no meu jetski.* É igualzinho a isso."

Raposão lançou-se sobre uma fumaça descendente, chacoalhando sua frase, cujas letras se descolaram e se espalharam pelo chão com palavras embaralhadas, mas ele tinha apenas sucedido em alcançar as partes depressivas das correspondências dos cervos, que se manifestavam como uma úmida e opaca névoa.

Enquanto isso, sua contraparte pequena agarrou um estreito trem de fumaça que passava sob seu braço. Ele foi aerotransportado e gritou, **Tallyho!** Mas se segurou muito firmemente e a nuvem evaporou sob seu braço e o enviou de volta ao chão com um pequeno salto.

Dado que está apenas começando seu uso de Ruby, você pode não ter entendido completamente as expressões regulares (ou *regexp*s) de início. Você pode até mesmo se encontrar recordando as *regexp*s da [Biblioteca de Expressões Regulares](#) e colando-as no seu código sem ter a mínima idéia de porque a expressão funciona. Ou se ela funciona!

```
loop do
  print "Entre com sua senha: "
  senha = gets
  if senha.match( /\w{8,15}$/ )
    break
  else
    puts "** Senha inadequada! Deve ter entre 8 e 15
caracteres!"
  end
end
```

Você vê a ilegível linguagem de cervo no código de exemplo? O `/\w{8,15}$/` é uma expressão regular. Se posso traduzi-la, a *regexp* está dizendo, *Por favor, permita somente letras, números ou travessões. Não menos que oito e não mais que quinze.*

Expressões Regulares são uma mini-linguagem construída dentro do Ruby e muitas outras linguagens de programação. Eu, na verdade não deveria dizer *mini*, já que *regexp*s podem ser complicadas e tortuosas e muito mais difíceis que qualquer programa Ruby.

Usar expressões regulares é extremamente simples. É como o Cervo: fazer fumaça é um processo árduo. Mas forçar seus ombros sobre a fumaça e direcionar-se para o Weinerschnitzel para pegar um pretzel de cachorro quente e mostarda é fácil.

```
irb> "senha_adequada".match( /\w{8,15}$/ )
=> #<MatchData:0xb7d54218>
irb> "esta_senha_inadequada_muito_longa".match( /\w{8,15}$/ )
=> nil
```

O método `String#match` é o uso prático mais simples das *regexp*s. O método `match` checa para ver se a string se adequa as regras dentro da *regexp*. Uma *regexp* só é útil com strings, para testar strings por uma série de condições. Se as condições são atendidas, um objeto `MatchData` é retornado. Caso contrário, você terá `nil`.

As expressões regulares mais básicas são para **realizar buscas** dentro de strings. Digamos que você tem um grande

- Crianças escorregando encima dos bifes. Por segurança, não queremos que os escorregadores fiquem com cinco bifes de profundidade, um em cima do outro.
- Ou os bifes fazem o deslizamento, em suas próprias sanguinhas.
- Ou as pessoas com sungas de bife.
- Pessoas e bifes, lado a lado.
- Bifes descendo tobôágua feitos de bifes.
- Bifes descendo tobôágua feitos de pessoas.
- E lógico, pessoas comendo bifes, mas suas línguas revelando-se como tobôágua e tendo que empurrar as carnes para o começo do tobôágua, o que é impossível, então um salva-vidas tem que escalar o tobôágua e enfiar a carne guela abaixo.
- tobôágua e bifes comendo pessoas.
- tobôágua e bifes virando amigos depois de sentirem o cheiro da respiração de cada pessoa.
- Ou, os bifes protegem os tobôágua, mas os tobôágua não são recíprocos. Os tobôágua ficam muito desanimados e indiferentes, entrando em galerias ruins e afundando em extremidades políticas. Os bifes fazem tornozeleiras com as pessoas e as deixam no bolso das calças dos tobôágua, quando as calças estão desacompanhadas. Eles fogem da comunidade dos tobôágua através de um enorme tobôágua feito de sungas de bife.
- Ou como eu disse, pessoas com sungas de bife.

arquivo e quer procurar nele por uma palavra ou frase. Já que um tempinho se passou, vamos procurar dentro dos registros de Achados e Perdidos dos Pré-eventualistas novamente.

```
require 'preeventualist'
PreEventalist.achadoperdido( 'caminhao' ) do |page|
  page.each_line do |linha|
    puts linha if linha.match( /caminhao/ )
  end
end
```

Isto não é muito diferente do código que usamos anteriormente para linhas com a palavra “caminhão”. Anteriormente usamos `puts linha if linha['caminhao']`, o que é na verdade um jeito mais simples de se procurar dentro de uma string, se você está procurando apenas por uma simples palavra. A regexp `/caminhao/` é idêntica. Encontre a palavra “caminhao”. Em qualquer lugar da string.

Hmn, mas e se caminhão for maiúscula. **Caminhão**. E então?

```
puts linha if linha.match( /[Cc][Aa][Mm][Ii][Nh][Aa][Oo]/i )
```

As **classes dos caracteres** são as seções cercadas por **colchetes**. Cada classe de caractere dá uma lista de caracteres que são compatíveis para este lugar. (O primeiro lugar é compatível tanto um C maiúsculo quanto um c minúsculo. O segundo lugar é compatível com um A ou um a. E assim por diante.)

Mas um jeito mais simples de escrever isso é assim:

```
puts line if line.match( /caminhao/i )
```

A letra `i` no final da regexp é um modificador que indica que a busca não é **sensível a caixa**. Ela casará com Caminhão. E CAMINHÃO. E CaMiNhÃO. E outras combinações de maiúsculas e minúsculas.

Ah, e talvez seu caminhão tenha um certo número de modelo. Um T-1000. Ou um T-2000. Você pode se lembrar. É algum T *alguma coisa* mil.

```
puts linha if linha.match( /T-\d{3}/ )
```

Vê, linguagem de cervo. O `\d` representa um **dígito**. É uma incógnita na regexp para qualquer tipo de número. A regexp irá agora casar com T-1000, T-2000, e tudo em diante até T-9000.

Classes de Caracteres

<code>\d</code>	casa dígitos	também pode ser escrita como <code>[0-9]</code>
<code>\w</code>	casa caracteres de palavra (letras, números e o travessão)	também pode ser escrita como <code>[A-Za-z0-9_]</code>
<code>\s</code>	casa espaços em branco (espaços, tabs, retorno de carro, pula linha)	a.k.a. <code>[\t\r\n]</code>
<code>\D</code>	casa tudo <i>menos</i> dígitos	um conjunto negado <code>[^0-9]</code>
<code>\W</code>	casa tudo <i>menos</i> caracteres de palavra	assim como <code>[^w]</code>
<code>\S</code>	casa tudo <i>menos</i> espaços em branco	também <code>[^s]</code>
.	o ponto casa tudo .	

Construir uma expressão regular envolve encadear esses receptáculos para expressar sua busca. Se estiver procurando por um número seguido de espaço em branco: `/\d\s/`. Se estiver procurando por três números em seqüência: `/\d\d\d/`. As **barras que abrem e fecham marcam o início e o fim da expressão regular**.

Uma busca por três números em seqüência também pode ser escrita como: `/\d{3}/`. Imediatamente após uma classe de caracteres como `\d`, você pode usar um símbolo de quantidade para dizer quantas vezes quer que aquela classe de caracteres seja repetida.

Quantificadores

<code>{n}</code>	casa exatamente <i>n</i> vezes	Precisamente três números em seqüência é <code>/\d{3}/</code>
<code>{n,}</code>	casa <i>n</i> vezes ou <i>mais</i>	Três ou mais letras em seqüência é <code>/[a-z]{3,}/i</code>
<code>{n,n2}</code>	casa pelo menos <i>n</i> vezes mas não	Então, <code>/[\d,]{3,9}/</code> casa entre três e nove
<code>}</code>	mais que <i>n2</i> vezes	caracteres que são números ou vírgulas
<code>*</code>	o asterisco * é a forma curta para	Para casar dois pontos, seguidos de zero ou mais

{0, }	caracteres de palavra: / : \w*/
+	o sinal de mais é a forma curta para Para casar um ou mais sinais de menos ou mais, use / [- +]+/
?	a interrogação é a forma curta para Para casar três números seguidos de um ponto opcional: /\d{3}[.]?/

Uma expressão regular bem comum é uma que case números de telefone. Números de telefone norte-americanos (incluindo código de área) podem ser casados usando a classe de caractere de dígito e os quantificadores precisos.

```
irb> "Ligue 909-375-4434" =~ /\d{3}-\d{3}-\d{4}/
=> 6
irb> "O número é (909) 375-4434" =~ /[()]\d{3}[]]\s*\d{3}-\d{4}/
=> 11
```

Desta vez, em vez de usar `match` para procurar pela expressão, usou-se o operador `=~`. Este operador é o **operador de casamento**, um sinal de igual seguido de **um til**. O til é como uma fumacinha saindo da ponta de uma chaminé. Lembre-se dos cervos, a fumaça que eles soltam, uma linguagem críptica assim como as expressões regulares. O til fumacento aponta para a expressão regular.

O operador de casamento retorna um número. O número é a posição na string onde a expressão regular foi encontrada. Então quando o operador retorna **6**, ele está dizendo, “Antes da expressão, há seis caracteres na string.”

Se você precisa casar a string inteira, você pode usar a variável global especial `$&` se você está usando o operador de casamento. Ou, se estiver usando o método `match`, você pode conseguir a string inteira convertendo o objeto `MatchData` em uma string.

```
# Usando =~ e $& juntos.
irb> "O número é (909) 375-4434" =~ /[()]\d{3}[]]\s*\d{3}-\d{4}/
=> 11
irb> $&
=> "(909) 375-4434"
```

```
# Usando o objeto MatchData.
irb> telefone = /[()]\d{3}[]]\s*\d{3}-\d{4}/.match("O número é (909) 375-4434")
=> #<MatchData:0xb7d51680>
irb> telefone.to_s
=> "(909) 375-4434"
```

A maioria dos Rubistas prefere a segunda alternativa, uma vez que usa um objeto dentro de uma *variável local* em vez de uma *variável global*. Variáveis globais são meio questionáveis, uma vez que podem ser facilmente sobrescritas. Se você rodar duas expressões regulares na seqüência, a variável global será sobreescrita na segunda vez. Mas com variáveis locais você pode manter as duas ocorrências desde que as variáveis tenham nomes diferentes.

Além de encontrar ocorrências, outro uso comum de expressões regulares é fazer **encontre-e-substitua** de dentro do Ruby. Você pode buscar pela palavra “mala” e substitui-la pela palavra “banjo.” Claro, você pode fazer isso com strings ou expressões regulares.

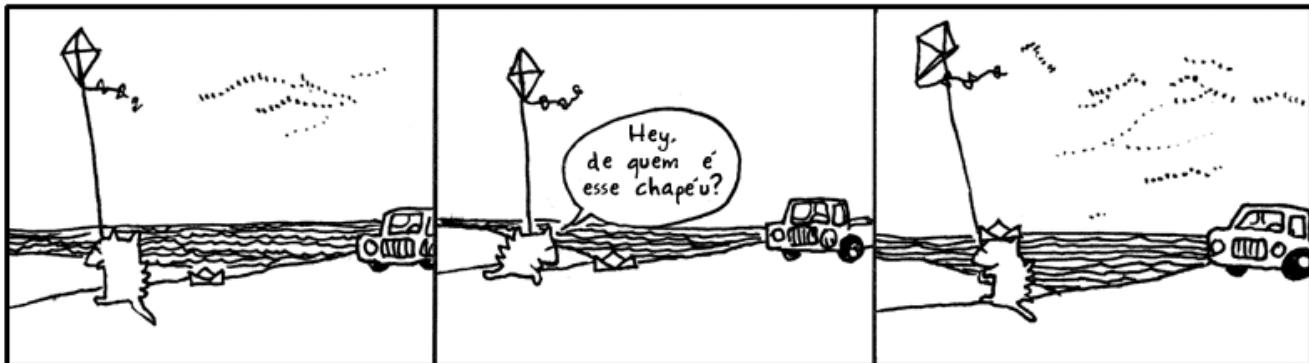
```
irb> musica = "Eu furtei sua mala / E roubei o malabarista"
irb> musica.gsub 'mala', 'viola'
=> "Eu furtei sua viola / E roubei o violabarista"

irb> musica.gsub /\bmala\b/, 'viola'
=> "Eu furtei sua viola / E roubei o malabarista"
```

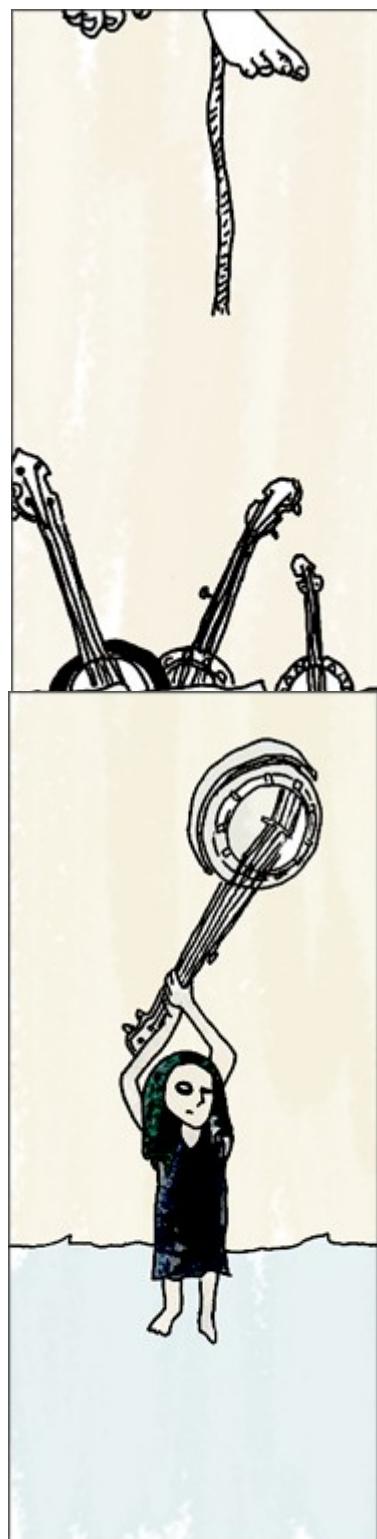
O método `gsub` significa “substituição global.” Note como no primeiro exemplo ele substituiu a palavra “mala” e as quatro primeiras letras de “malabarista.” Strings também têm um método simples `sub` que vai substituir apenas uma vez.

E assim este capítulo acaba, com Blix e os Raposos voando no sólido rosa expelido por um cervo muito reservado em algum lugar naquelas pastagens.

6. Só Parando Para Lhe Assegurar Que o Porco-espinho Não Se Moveu



Estou Fora >



Um dia, há algum tempo atrás, quando conheci o Bigelow (o cachorro que fugiu com os balões), eu voltei para o meu apartamento levando uns jogos de tabuleiro que havia comprado em uma venda de garagem. E a Quil estava na minha varanda. O que me surpreendeu, pois ela estava em San Antonio há três meses. Ela estava dormindo num saco de dormir na minha varanda.

Não tinha grana para a escola de artes, então ficou na minha casa por uns cinco meses.

Eu achei essa bicama usada para o nosso cantinho. À noite, nós nos sentávamos em nossas camas para ler um para o outro estórias de nossas agendas. Eu estava escrevendo um livro sobre uma criança que era um detetive e tentava descobrir quem matou essa outra criança do time de tênis e todos esses animais acabavam ajudando ela a desvendar o caso. Ela escrevia um livro sobre um garoto que coloca um anúncio nos classificados para que outras crianças entrem no seu culto fictício e eles terminam por construir um foguete. Mas durante a maioria do livro dela essas crianças estão perdidas na mata,

Quil usa o banjo como um guarda-chuva.

totalmente sem rumo, o que eu me enchi de escutar a cada noite.

É, toda noite era poesia ou estórias ou idéias para pregar peças nos vizinhos. Nossa vizinha Justin era um grande fã do jogo de tabuleiro Warhammer. Ele tinha todas essas espadas e túnicas de verdade. Decidimos fazer armaduras de papel alumínio e atacar o apartamento dele. Começamos a pilhar o apartamento e ele adorou. Daí ele também fez sua armadura de papel alumínio e nós fomos a um glamuroso estúdio profissional e tiramos uma foto em grupo de qualidade.

Não estou dizendo que minha vida é melhor que a sua. Eu simplesmente sinto falta da minha irmã. As coisas agora não são mais como antigamente. Estamos separados ou algo do tipo.

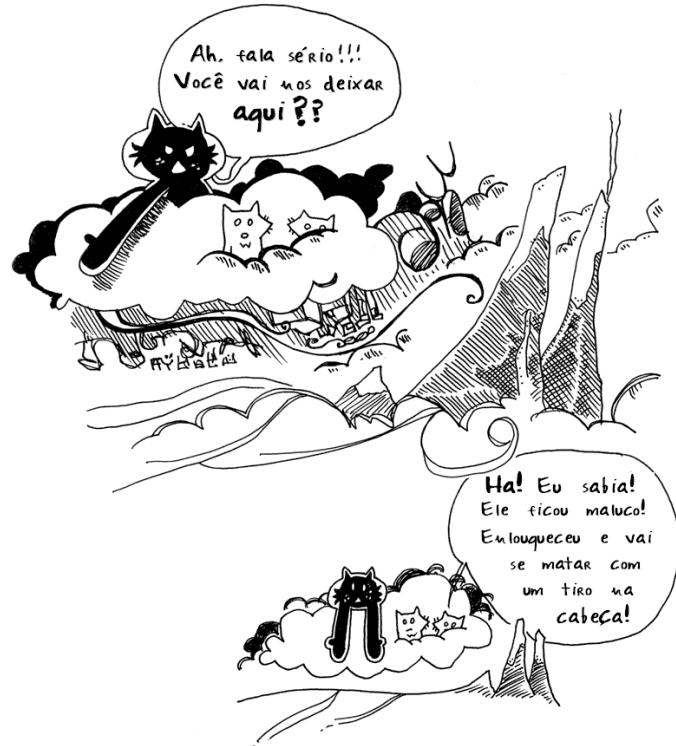
Não sei. Estou confuso. Isto é crescer? Assistir a todas suas penas se soltarem? E mesmo sabendo que algumas dessas penas eram as melhores coisas que você possuía?

Eu acho difícil dizer quem acabou com tudo. Quem parou de amar quem? Eu parei de me importar? Talvez eu só a visse em duas dimensões e não fazia questão de olhar os outros ângulos. Eu só via planos. Então ela deslizou no eixo z enquanto eu não olhava e não fiz o dever de casa para traçar as coordenadas. Um tronco em uma árvore geométrica e eu insistindo em círculos.



Quatro quilis banqueteiam as carcaças de chacais.

Blix estava certo. Eu estou em ótima forma para escrever este livro. Adeus até que eu possa cair fora daqui.



Quando Você Deseja Uma Barba

[0]

EI, ESTE CAPÍTULO FOI DEIXADO EM MEU COLO. EM BREVE EU TEREI ele PRONTO. EU TEREI.