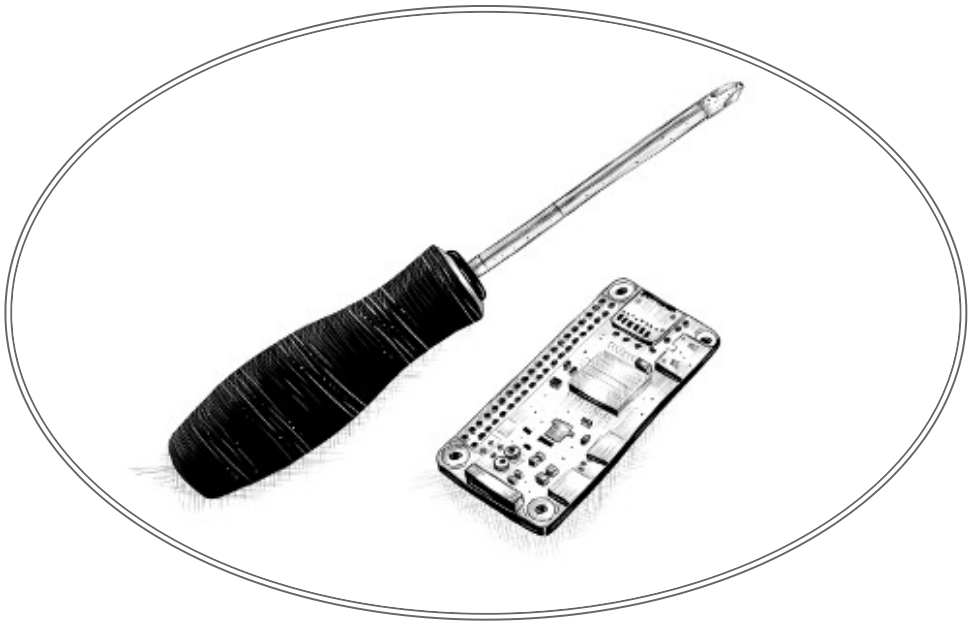
Introdução

> “Achamos que estamos criando o sistema para nossos próprios propósitos. Acreditamos que o estamos fazendo à nossa própria imagem... Mas o computador não é realmente como nós. É uma projeção de uma parte muito tênue de nós mesmos: aquela porção devotada à lógica, ordem, regra e clareza.”
>
> —Ellen Ullman, *Perto da Máquina: Tecnofilia e seus descontentamentos*



Este é um livro sobre instruir computadores. Computadores são tão comuns quanto chaves de fenda hoje em dia, mas são bem mais complexos, e fazê-los fazer o que você quer que eles façam nem sempre é fácil.

Se a tarefa que você tem para seu computador for comum e bem compreendida, como mostrar seu e-mail ou agir como uma calculadora, você pode abrir o aplicativo apropriado e começar a trabalhar. Mas para tarefas únicas ou abertas, geralmente não há um aplicativo apropriado.

É aí que a programação pode entrar. *Programação* é o ato de construir um *programa* — um conjunto de instruções precisas dizendo a um computador o que fazer. Como os computadores são bestas idiotas e pedantes, a programação é fundamentalmente tediosa e frustrante.

Felizmente, se você conseguir superar esse fato — e talvez até mesmo aproveitar o rigor de pensar em termos com os quais máquinas burras podem lidar — a programação pode ser recompensadora. Ela permite que você faça coisas em segundos que levariam *uma eternidade* manualmente. É uma maneira de fazer sua ferramenta de computador fazer coisas que ela não conseguia fazer antes. Além disso, ela cria um jogo maravilhoso de resolução de quebra-cabeças e pensamento abstrato.

A maior parte da programação é feita com linguagens de programação. Uma *linguagem de programação* é uma linguagem construída artificialmente usada para instruir computadores. É interessante que a maneira mais eficaz que encontramos de nos comunicar com um computador toma emprestado tanto da maneira como nos comunicamos uns com os outros. Assim como as linguagens humanas, as linguagens de computador permitem que palavras e frases sejam combinadas de novas maneiras, tornando possível expressar conceitos sempre novos.

Em um ponto, interfaces baseadas em linguagem, como os prompts BASIC e DOS das décadas de 1980 e 1990, eram o principal método de interação com computadores. Para uso rotineiro do computador, elas foram amplamente substituídas por interfaces visuais, que são mais fáceis de aprender, mas oferecem menos liberdade. Mas se você souber onde procurar, as linguagens ainda estão lá. Uma delas, *JavaScript* , é incorporada a todos os navegadores modernos da web — e, portanto, está disponível em quase todos os dispositivos.

Este livro tentará familiarizá-lo o suficiente com esta linguagem para que você possa fazer coisas úteis e divertidas com ela.

Sobre programação

Além de explicar JavaScript, apresentarei os princípios básicos da programação. Programar, ao que parece, é difícil. As regras fundamentais são simples e claras, mas programas construídos sobre essas regras tendem a se tornar complexos o suficiente para introduzir suas próprias regras e complexidade. Você está construindo seu próprio labirinto, de certa forma, e pode facilmente se perder nele.

Haverá momentos em que ler este livro parecerá terrivelmente frustrante. Se você é novo em programação, haverá muito material novo para digerir. Muito desse material será então *combinado* de maneiras que exigirão que você faça conexões adicionais.

Cabe a você fazer o esforço necessário. Quando estiver com dificuldades para seguir o livro, não tire conclusões precipitadas sobre suas próprias capacidades. Você está bem — você só precisa continuar. Faça uma pausa, releia algum material e certifique-se de ler e entender os programas de exemplo e exercícios. Aprender é um trabalho duro, mas tudo o que você aprende é seu e tornará o aprendizado posterior mais fácil.

> "Quando a ação se torna inútil, reúna informações; quando a informação se torna inútil, durma."
>
> —Ursula K. Le Guin, *A Mão Esquerda da Escuridão*

Um programa é muitas coisas. É um pedaço de texto digitado por um programador, é a força motriz que faz o computador fazer o que faz, são dados na memória do computador e, ao mesmo tempo, controla as ações realizadas nessa memória. Analogias que tentam comparar programas a objetos familiares tendem a falhar. Uma superficialmente adequada é comparar um programa a uma máquina — muitas partes separadas tendem a estar envolvidas e, para fazer a coisa toda funcionar, temos que considerar as maneiras pelas quais essas partes se interconectam e contribuem para a operação do todo.

Um computador é uma máquina física que atua como um host para essas máquinas imateriais. Os próprios computadores podem fazer apenas coisas estupidamente diretas. A razão pela qual eles são tão úteis é que eles fazem essas coisas em uma velocidade incrivelmente alta. Um programa pode combinar engenhosamente um número enorme dessas ações simples para fazer coisas muito complicadas.

Um programa é uma construção de pensamento. Não tem custo para construir, não tem peso e cresce facilmente sob nossas mãos digitadoras. Mas conforme um programa cresce, sua complexidade também cresce. A habilidade de programação é a habilidade de construir programas que não confundem o programador. Os melhores programas são aqueles que conseguem fazer algo interessante e ainda são fáceis de entender.

Alguns programadores acreditam que essa complexidade é melhor gerenciada usando apenas um pequeno conjunto de técnicas bem compreendidas em seus programas. Eles compuseram regras rígidas ("melhores práticas") prescrevendo a forma que os programas devem ter e permanecer cuidadosamente dentro de sua pequena zona segura.

Isso não é apenas chato — é ineficaz. Novos problemas geralmente exigem novas soluções. O campo da programação é jovem e ainda está se desenvolvendo rapidamente, e é variado o suficiente para ter espaço para abordagens totalmente diferentes. Há muitos erros terríveis a serem cometidos no design de programas, e você deve ir em frente e cometê-los pelo menos uma vez para entendê-los. Uma noção de como é um bom programa é desenvolvida com a prática, não aprendida com uma lista de regras.

Por que a linguagem é importante

No começo, no nascimento da computação, não havia linguagens de programação. Os programas pareciam algo assim:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

Este é um programa para somar os números de 1 a 10 e imprimir o resultado: . Ele poderia rodar em uma máquina hipotética simples. Para programar os primeiros computadores, era necessário colocar grandes conjuntos de interruptores na posição correta ou fazer furos em tiras de papelão e alimentá-los com eles no computador. Você pode imaginar o quão tedioso e propenso a erros esse procedimento era. Até mesmo escrever programas simples exigia muita inteligência e disciplina. Os complexos eram quase inconcebíveis.`1 + 2 + ... + 10 = 55`

Claro, inserir manualmente esses padrões arcanos de bits (os uns e zeros) deu ao programador uma profunda sensação de ser um mago poderoso. E isso tem que valer alguma coisa em termos de satisfação no trabalho.

Cada linha do programa anterior contém uma única instrução. Poderia ser escrito em inglês assim:

1. Armazene o número 0 no local de memória 0.

2. Armazene o número 1 no local de memória 1.
3. Armazene o valor do local de memória 1 no local de memória 2.
4. Subtraia o número 11 do valor no local de memória 2.
5. Se o valor no local de memória 2 for o número 0, continue com a instrução 9.
6. Adicione o valor do local de memória 1 ao local de memória 0.
7. Adicione o número 1 ao valor do local de memória 1.
8. Continue com a instrução 3.
9. Emite o valor do local de memória 0.

Embora isso já seja mais legível do que a sopa de bits, ainda é bastante obscuro. Usar nomes em vez de números para as instruções e locais de memória ajuda.

```
  Defina “total” como 0.
  Defina “contagem” como 1.
[laço]
  Defina “comparar” como “contar”.
  Subtraia 11 de “comparar”.
  Se “comparar” for 0, continue em [fim].
  Adicione “contagem” a “total”.
  Adicione 1 a “contagem”.
  Continue em [loop].
[fim]
  Saída “total”.
```

Você consegue ver como o programa funciona neste ponto? As duas primeiras linhas dão a dois locais de memória seus valores iniciais: `total` será usado para construir o resultado do cálculo e `count` manterá o controle do número que estamos olhando no momento. As linhas using `compare` são provavelmente as mais confusas. O programa quer ver se `count` é igual a 11 para decidir se pode parar de executar. Como nossa máquina hipotética é bastante primitiva, ela pode testar apenas se um número é zero e tomar uma decisão com base nisso. Portanto, ela usa o local de memória rotulado `compare` para calcular o valor de `count - 11` e toma uma decisão com base nesse valor. As próximas duas linhas adicionam o valor de `count` ao resultado e incrementam `count` em 1 toda vez que o programa decide que `count` ainda não é 11.

Aqui está o mesmo programa em JavaScript:

```
deixe  total = 0 , contagem = 1 ;
 enquanto (contagem <= 10 ) {
  total += contagem;
  contar += 1 ;
}
console.log(total);
// → 55
```

Esta versão nos dá mais algumas melhorias. Mais importante, não há mais necessidade de especificar a maneira como queremos que o programa salte para frente e para trás — a `while`construção cuida disso. Ele continua executando o bloco (envolto em chaves) abaixo dele enquanto a condição que lhe foi dada for mantida. Essa condição é `count <= 10`, que significa "a contagem é menor ou igual a 10". Não precisamos mais criar um valor temporário e compará-lo a zero, o que era apenas um detalhe desinteressante. Parte do poder das linguagens de programação é que elas podem cuidar de detalhes desinteressantes para nós.

No final do programa, após a `while`conclusão da construção, a `console.log`operação é usada para escrever o resultado.

Por fim, aqui está como o programa poderia ser se tivéssemos as operações convenientes `range`e `sum`disponíveis, que respectivamente criam uma coleção de números dentro de um intervalo e calculam a soma de uma coleção de números:

```
console.log(soma(intervalo( 1 , 10 )));
 // → 55
```

A moral desta história é que o mesmo programa pode ser expresso de formas longas e curtas, ilegíveis e legíveis. A primeira versão do programa era extremamente obscura, enquanto esta última é quase inglesa: `log`the `sum`of the `range`of números de 1 a 10. (Veremos em capítulos posteriores como definir operações como `sum`e `range`.)

Uma boa linguagem de programação ajuda o programador permitindo que ele fale sobre as ações que o computador tem que executar em um nível mais alto. Ela ajuda a omitir detalhes, fornece blocos de construção convenientes (como `while`e `console.log`), permite que você defina seus próprios blocos de construção (como `sum`e `range`), e torna esses blocos fáceis de compor.

O que é JavaScript?

O JavaScript foi introduzido em 1995 como uma forma de adicionar programas a páginas da web no navegador Netscape Navigator. Desde então, a linguagem foi adotada por todos os outros principais navegadores gráficos da web. Ela tornou possíveis os aplicativos da web modernos — ou seja, aplicativos com os quais você pode interagir diretamente sem precisar recarregar a página para cada ação. O JavaScript também é usado em sites mais tradicionais para fornecer várias formas de interatividade e inteligência.

É importante notar que JavaScript não tem quase nada a ver com a linguagem de programação chamada Java. O nome similar foi inspirado por considerações de marketing em vez de bom senso. Quando o JavaScript estava sendo introduzido, a linguagem Java estava sendo fortemente comercializada e estava ganhando popularidade. Alguém achou que seria uma boa ideia tentar aproveitar esse sucesso. Agora estamos presos ao nome.

Após sua adoção fora do Netscape, um documento padrão foi escrito para descrever a maneira como a linguagem JavaScript deveria funcionar para que os vários softwares que alegavam suportar JavaScript pudessem garantir que realmente fornecessem a mesma linguagem. Isso é chamado de padrão ECMAScript, em homenagem à organização Ecma International que conduziu a padronização. Na prática, os termos ECMAScript e JavaScript podem ser usados de forma intercambiável — são dois nomes para a mesma linguagem.

Há aqueles que dirão coisas *terríveis* sobre JavaScript. Muitas dessas coisas são verdadeiras. Quando precisei escrever algo em JavaScript pela primeira vez, rapidamente passei a desprezá-lo. Ele aceitava quase tudo que eu digitava, mas interpretava de uma forma completamente diferente do que eu queria dizer. Isso tinha muito a ver com o fato de que eu não tinha a mínima ideia do que estava fazendo, é claro, mas há um problema real aqui: JavaScript é ridiculamente liberal no que permite. A ideia por trás desse design era que ele tornaria a programação em JavaScript mais fácil para iniciantes. Na verdade, ele dificulta principalmente encontrar problemas em seus programas porque o sistema não os apontará para você.

Essa flexibilidade também tem suas vantagens, no entanto. Ela deixa espaço para técnicas que são impossíveis em linguagens mais rígidas e cria um estilo de programação agradável e informal. Depois de aprender a linguagem corretamente e trabalhar com ela por um tempo, comecei a realmente *gostar* de JavaScript.

Houve várias versões do JavaScript. A versão 3 do ECMAScript foi a versão amplamente suportada durante a ascensão do JavaScript ao domínio, aproximadamente entre 2000 e 2010. Durante esse tempo, o trabalho estava em andamento em uma versão ambiciosa 4, que planejava uma série de melhorias radicais e extensões para a linguagem. Mudar uma linguagem viva e amplamente usada de uma forma tão radical acabou sendo politicamente difícil, e o trabalho na versão 4 foi abandonado em 2008. Uma versão 5 muito menos ambiciosa, que fez apenas algumas melhorias não controversas, saiu em 2009. Em 2015, a versão 6 saiu, uma grande atualização que incluía algumas das ideias planejadas para a versão 4. Desde então, tivemos novas e pequenas atualizações a cada ano.

O fato de que o JavaScript está evoluindo significa que os navegadores precisam se manter constantemente atualizados. Se você estiver usando um navegador mais antigo, ele pode não suportar todos os recursos. Os designers da linguagem são cuidadosos para não fazer nenhuma alteração que possa quebrar programas existentes, então novos navegadores ainda podem executar programas antigos. Neste livro, estou usando a versão 2024 do JavaScript.

Os navegadores da Web não são as únicas plataformas nas quais o JavaScript é usado. Alguns bancos de dados, como MongoDB e CouchDB, usam JavaScript como sua linguagem de script e consulta. Várias plataformas para programação de desktop e servidor, mais notavelmente o projeto Node.js (o assunto do Capítulo 20), fornecem um ambiente para programação de JavaScript fora do navegador.

Código e o que fazer com ele

Código é o texto que compõe os programas. A maioria dos capítulos deste livro contém bastante código. Acredito que ler e escrever código são partes indispensáveis do aprendizado da programação. Tente não apenas dar uma olhada nos exemplos — leia-os atentamente e entenda-os. Isso pode ser lento e confuso no começo, mas prometo que você pegará o jeito rapidamente. O mesmo vale para os exercícios. Não presuma que você os entendeu até que tenha realmente escrito uma solução funcional.

Recomendo que você tente suas soluções para exercícios em um interpretador JavaScript real. Dessa forma, você terá um feedback imediato sobre se o que está fazendo está funcionando e, espero, você ficará tentado a experimentar e ir além dos exercícios.

Ao ler este livro no seu navegador, você pode editar (e executar) todos os programas de exemplo clicando neles.

Executar os programas definidos neste livro fora do site do livro requer algum cuidado. Muitos exemplos são independentes e devem funcionar em qualquer ambiente JavaScript. Mas o código em capítulos posteriores é frequentemente escrito para um ambiente específico (o navegador ou Node.js) e pode ser executado somente lá. Além disso, muitos capítulos definem programas maiores, e os pedaços de código que aparecem neles dependem uns dos outros ou de arquivos externos. O sandbox no site fornece links para arquivos ZIP contendo todos os scripts e arquivos de dados necessários para executar o código para um determinado capítulo.

Visão geral deste livro

Este livro contém aproximadamente três partes. Os primeiros 12 capítulos discutem a linguagem JavaScript. Os próximos sete capítulos são sobre navegadores da web e a maneira como o JavaScript é usado para programá-los. Finalmente, dois capítulos são dedicados ao Node.js, outro ambiente para programar JavaScript. Há cinco *capítulos de projeto* no livro que descrevem programas de exemplo maiores para lhe dar uma amostra da programação real.

A parte de linguagem do livro começa com quatro capítulos que introduzem a estrutura básica da linguagem JavaScript. Eles discutem estruturas de controle (como a `while`palavra que você viu nesta introdução), funções (escrever seus próprios blocos de construção) e estruturas de dados . Depois disso, você será capaz de escrever programas básicos. Em seguida, os Capítulos 5 e 6 introduzem técnicas para usar funções e objetos para escrever código mais *abstrato* e manter a complexidade sob controle.

Após um primeiro capítulo de projeto que constrói um robô de entrega bruto, a parte de linguagem do livro continua com capítulos sobre tratamento de erros e correção de bugs , expressões regulares (uma ferramenta importante para trabalhar com texto), modularidade (outra defesa contra a complexidade) e programação assíncrona (lidar com eventos que levam tempo). O segundo capítulo de projeto , onde implementamos uma linguagem de programação, conclui a primeira parte do livro.

A segunda parte do livro, Capítulos 13 a 19 , descreve as ferramentas às quais o JavaScript do navegador tem acesso. Você aprenderá a exibir coisas na tela (Capítulos

14 e 17), responder à entrada do usuário (Capítulo 15) e se comunicar pela rede (Capítulo 18). Há novamente dois capítulos de projeto nesta parte: construir um jogo de plataforma e um programa de pixel paint .

O Capítulo 20 descreve o Node.js, e o Capítulo 21 cria um pequeno site usando essa ferramenta.

Convenções tipográficas

Neste livro, o texto escrito em uma `monospaced`fonte representará elementos de programas. Às vezes, esses são fragmentos autossuficientes e, às vezes, eles apenas se referem a parte de um programa próximo. Programas (dos quais você já viu alguns) são escritos da seguinte forma:

```
função  fatorial ( n ) {
   se (n == 0 ) {
     retornar  1 ;
  } senão {
     retornar fatorial(n - 1 ) * n;
  }
}
```

Às vezes, para mostrar a saída que um programa produz, a saída esperada é escrita depois dele, com duas barras e uma seta na frente.

```
console.log(fatorial( 8 ));
 // → 40320
```

Boa sorte!

● ► ?