Trechos de: Jon Duckett. "JavaScript and JQuery: Interactive Front-End Web Development". Apple Books.

1. COMPUTADORES CRIAM MODELOS DO MUNDO USANDO DADOS

Aqui está um modelo de hotel, junto com algumas árvores modelo, modelos de pessoas e carros modelo. Para um ser humano, é claro que tipo de objeto do mundo real cada um representa.

Um computador não tem um conceito predefinido do que é um hotel ou carro. Ele não sabe para que são usados. Seu laptop ou telefone não terá uma marca de carro favorita, nem saberá qual é a classificação por estrelas de seu hotel.

Então, como usamos computadores para criar aplicativos de reserva de hotel ou videogames onde os jogadores podem correr um carro? A resposta é que os programadores criam um tipo muito diferente de modelo, especialmente para computadores.

Os programadores fazem esses modelos usando dados. Isso não é tão estranho ou assustador quanto parece, porque os dados são tudo que o computador precisa para seguir as instruções que você lhe der para realizar suas tarefas.

1.1. OBJETOS E PROPRIEDADES

Se você não pudesse ver a foto do hotel e dos carros, os dados nas caixas de informações por si só ainda diriam muito sobre essa cena.

1.1.1 OBJETOS (COISAS)

Na programação de computadores, cada coisa física no mundo pode ser representada como um **objeto**. Existem dois **tipos** diferentes de objetos aqui: um hotel e um carro. Os programadores podem dizer que há uma **instância** do objeto hotel e duas instâncias do objeto carro.

Os objetos agrupam um conjunto de variáveis e funções para criar um modelo de algo que você reconheceria do mundo real. Em um objeto, variáveis e funções assumem novos nomes. Cada objeto pode ter seu próprio:

- Propriedades
- Eventos
- Métodos

Juntos, eles criam um modelo de trabalho desse objeto.

1.1.2 PROPRIEDADES (CARACTERÍSTICAS)

Ambos os carros compartilham características comuns. Na verdade, todos os carros têm uma marca, uma cor e um tamanho de motor. Você pode até determinar sua velocidade atual. Os programadores chamam essas características de **propriedades** de um objeto.

Cada propriedade tem um **nome** e um **valor**, e cada um desses pares nome / valor informa algo sobre cada instância individual do objeto. A propriedade mais óbvia deste hotel é o seu nome. O valor dessa propriedade é **Quay**. Você pode dizer o número de quartos que o hotel possui observando o valor ao lado da propriedade dos quartos.

A ideia de pares nome / valor é usada em HTML e CSS. Em HTML, um atributo é como uma propriedade; atributos diferentes têm nomes diferentes e cada atributo pode ter um valor. Da mesma forma, em CSS você pode alterar a cor de um título criando uma regra que dê à propriedade color um valor específico, ou você pode alterar o tipo de letra em que está escrito atribuindo à propriedade **font-family** um valor específico. Os pares nome / valor são muito usados na programação.

OBJETO HOTEL

O objeto hotel usa nomes e valores de propriedade para informá-lo sobre este hotel específico, como o nome do hotel, sua classificação, o número de quartos que possui e quantos deles estão reservados. Você também pode saber se este hotel tem ou não certas instalações.

OBJETOS CAR

Os objetos carro compartilham as mesmas propriedades, mas cada um tem valores diferentes para essas propriedades. Eles informam a marca do carro, a que velocidade cada carro está viajando no momento, de que cor ele é e que tipo de combustível necessita.

1.2. EVENTOS

No mundo real, as pessoas interagem com os objetos. Essas interações podem alterar os valores das propriedades desses objetos.

1.1.1. O QUE É UM EVENTO?

Existem maneiras comuns pelas quais as pessoas interagem com cada tipo de objeto. Por exemplo, em um carro, um motorista normalmente usa pelo menos dois pedais. O carro foi projetado para responder de forma diferente quando o motorista interage com cada um dos diferentes pedais:

- O acelerador faz o carro andar mais rápido
- O freio diminui a velocidade

Da mesma forma, os programas são projetados para fazer coisas diferentes quando os usuários interagem com o computador de maneiras diferentes. Por exemplo, clicar em um link de contato em uma página da web pode abrir um formulário de contato e inserir texto em uma caixa de pesquisa pode acionar automaticamente a funcionalidade de pesquisa. Um evento é a maneira que o computador tem de levantar a mão para dizer: “Ei, isso acabou de acontecer!”

1.1.2. O QUE FAZ UM EVENTO?

Os programadores escolhem a quais eventos eles respondem. Quando um evento específico acontece, esse evento pode ser usado para acionar uma seção específica do código.

Os scripts geralmente usam eventos diferentes para acionar diferentes tipos de funcionalidade.

Portanto, um script indicará a quais eventos o programador deseja responder e que parte do script deve ser executado quando cada um desses eventos ocorrer.

OBJETO DO HOTEL

Um hotel fará regularmente reservas de quartos. Cada vez que uma sala é reservada, um evento chamado *book* pode ser usado para acionar o código que aumentará o valor da propriedade de reservas. Da mesma forma, um evento de cancelamento pode acionar um código que diminui o valor da propriedade *bookings*.

OBJETOS DO CARRO

Um motorista irá acelerar e frear durante qualquer viagem de carro. Um evento de aceleração pode acionar o código para aumentar o valor da propriedade *currentSpeed* e um evento de freio pode

acionar o código para diminuí-lo. Você aprenderá sobre o código que responde aos eventos e altera essas propriedades na próxima página.

1.3. MÉTODOS

Métodos representam coisas que as pessoas precisam fazer com objetos. Eles podem recuperar ou atualizar os valores das propriedades de um objeto.

1.1.3 O QUE É UM MÉTODO?

Os métodos geralmente representam como as pessoas (ou outras coisas) interagem com um objeto no mundo real.

Eles são como perguntas e instruções que:

- Diga algo sobre esse objeto (usando informações armazenadas em suas propriedades)
- Altere o valor de uma ou mais propriedades desse objeto

1.1.4 O QUE FAZ UM MÉTODO?

O código de um método pode conter muitas instruções que juntas representam uma tarefa.

Quando você usa um método, nem sempre precisa saber como ele realiza sua tarefa; você só precisa saber como fazer a pergunta e como interpretar as respostas que ela lhe der.

OBJETO DO HOTEL

Normalmente, será perguntado aos hotéis se os quartos são gratuitos. Para responder a esta pergunta, um método pode ser escrito que subtrai o número de reservas do número total de quartos. Métodos também podem ser usados para aumentar e diminuir o valor da propriedade de reservas quando os quartos são reservados ou cancelados.

OBJETOS DO CARRO

O valor da propriedade *currentSpeed* precisa aumentar e diminuir conforme o motorista acelera e freia. O código para aumentar ou diminuir o valor da propriedade *currentSpeed* pode ser escrito em um método, e esse método pode ser chamado de *changeSpeed*().

1.4. CONSIDERAÇÕES FINAIS

Os computadores usam dados para criar modelos de coisas no mundo real. Os eventos, métodos e propriedades de um objeto se relacionam entre si: os eventos podem acionar métodos e os métodos podem recuperar ou atualizar as propriedades de um objeto.

OBJETO HOTEL

1. Quando uma reserva é feita, o evento do agendamento (book) é disparado.
2. O evento book aciona o método **makeBooking**(), que aumenta o valor da propriedade bookings.
3. O valor da propriedade de reservas é alterado para refletir quantos quartos o hotel tem disponíveis.

OBJETOS DO CARRO

1. Conforme o motorista acelera, o evento de aceleração é disparado.
2. O evento de aceleração chama o método **changeSpeed**(), que por sua vez aumenta o valor da propriedade **currentSpeed**.
3. O valor da propriedade **currentSpeed** reflete a velocidade com que o carro está se movendo.

Representando um objeto em JavaScript

Objetos em Javascript são estruturas de dados chave-valor, representadas entre chaves. Atributos e métodos podem ser representados com uma chave e um valor associado. Por exemplo, imagine o objeto pessoa, com os atributos nome, gênero e o método estudar(). Para ilustrar uma instância deste objeto, consideremos uma pessoa chamada Ana, de gênero feminino. O código Javascript para a criação de Ana segue abaixo:

```
var pessoa = {nome: "Ana"
  , genero: "Feminino"
  , estudar: function(){ //código para executar a atividade aqui
```

```
        }
}
```

3.2 EXERCÍCIOS

Crie um objeto hotel e um objeto carro que representem as instâncias mencionadas no exemplo acima.

2. COMO OS NAVEGADORES USAM OBJETOS

Você viu como os dados podem ser usados para criar um modelo de um hotel ou um carro. Os navegadores da web criam modelos semelhantes da página da web que estão exibindo e da janela do navegador em que a página está sendo exibida.

OBJETO WINDOW

Na página à direita você pode ver um modelo de um computador com um navegador aberto na tela. O navegador representa cada janela ou guia usando um objeto de janela. A propriedade de localização do objeto janela informará a URL da página atual.

OBJETO DOCUMENT

A página da web atual carregada em cada janela é modelada usando um objeto de documento.

A propriedade title do objeto de documento informa o que está entre a tag de abertura **<title>** e de fechamento **</title>** para aquela página da web, e a propriedade **lastModified** do objeto de documento informa a data em que esta página foi atualizada pela última vez.

2.1 COMO O NAVEGADOR VÊ UMA WEB PAGE

Para entender como você pode alterar o conteúdo de uma página HTML usando JavaScript, você precisa saber como um navegador interpreta o código HTML e aplica o estilo a ele.

1: RECEBER UMA PÁGINA COMO CÓDIGO HTML

Cada página de um site pode ser vista como um documento separado. Portanto, a web consiste em muitos sites, cada um composto por um ou mais documentos.

2: CRIE UM MODELO DA PÁGINA E ARMAZENE-O NA MEMÓRIA

O modelo mostrado na página à direita é uma representação de uma página muito básica. Sua estrutura é uma reminiscência de uma árvore genealógica. No topo do modelo está um objeto de documento, que representa todo o documento.

Abaixo do objeto de documento, cada caixa é chamada de nó. Cada um desses nós é outro objeto. Este exemplo apresenta três tipos de nós que representam elementos, texto dentro dos elementos e atributo.

3: USE UM MOTOR DE RENDERIZAÇÃO PARA MOSTRAR A PÁGINA NA TELA

Se não houver CSS, o mecanismo de renderização aplicará estilos padrão aos elementos HTML. No entanto, o código HTML deste exemplo está vinculado a uma folha de estilo CSS, de modo que o navegador solicita esse arquivo e exibe a página de acordo.

Quando o navegador recebe regras CSS, o mecanismo de renderização as processa e aplica cada regra a seus elementos correspondentes. É assim que o navegador posiciona os elementos no lugar correto, com as cores, fontes e assim por diante.

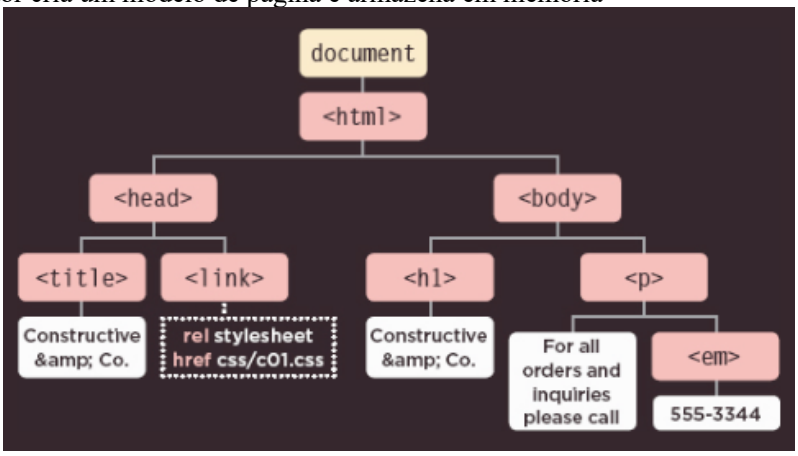
Todos os principais navegadores usam um intérprete JavaScript para traduzir suas instruções (em JavaScript) em instruções que o computador pode seguir. Quando você usa JavaScript no navegador, há uma parte do navegador que é chamada de intérprete (ou mecanismo de script).

O interpretador pega suas instruções (em JavaScript) e as traduz em instruções que o navegador pode usar para realizar as tarefas que você deseja executar. Em uma linguagem de programação interpretada, como JavaScript, cada linha de código é traduzida uma por uma conforme o script é executado

1. O navegador recebe uma página HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <p>For all orders and inquiries please call
      <em>555-3344</em></p>
  </body>
</html>
```

2. O navegador cria um modelo de página e armazena em memória



3. O navegador mostra a página na tela usando um mecanismo de renderização.

3. COMO HTML, CSS E JAVASCRIPT SE ENCAIXAM NESSE QUEBRA-CABEÇAS?

Para utilizar Javascript de maneira mais efetiva, você precisa saber como ela se encaixará com o HTML e CSS em suas páginas da web. Os desenvolvedores da web geralmente falam sobre três linguagens que são usadas para criar páginas da web: HTML, CSS e JavaScript.

3.1 Camada de conteúdo - arquivos HTML

É aqui que reside o conteúdo da página. O HTML fornece a estrutura da página e adiciona semântica. Sempre que possível, tente manter as três linguagens em arquivos separados, com a página HTML com links para arquivos CSS e JavaScript.

3.2 Camada de apresentação - arquivos CSS

O CSS aprimora a página HTML com regras que definem como o conteúdo HTML é apresentado (planos de fundo, bordas, dimensões da caixa, cores, fontes, etc.). Cada linguagem forma uma camada separada com uma finalidade diferente. Cada camada, da esquerda para a direita, se baseia na anterior.

3.3 Camada comportamental - arquivos JavaScript

É aqui que podemos mudar o comportamento da página, adicionando interatividade. Nosso objetivo é manter o máximo possível de nosso JavaScript em arquivos separados. Os programadores costumam se referir a isso como uma separação de interesses.

MELHORIA PROGRESSIVA

Essas três camadas formam a base de uma abordagem popular para a construção de páginas da Web chamada de aprimoramento progressivo. À medida que mais e mais dispositivos habilitados para web chegam ao mercado, esse conceito está se tornando mais amplamente adotado

APENAS HTML

Começar com a camada HTML permite que você se concentre no que há de mais importante em seu site: seu conteúdo. Sendo HTML puro, esta camada deve funcionar em todos os tipos de dispositivos, ser acessível a todos os usuários e carregar rapidamente em conexões lentas. Não são apenas os tamanhos de tela que variam - as velocidades de conexão e os recursos de cada dispositivo também podem ser diferentes

HTML + CSS

Adicionar as regras CSS em um arquivo separado mantém as regras sobre como a página parece diferente do conteúdo em si. Você pode usar a mesma folha de estilo com todo o seu site, tornando-os mais rápidos de carregar e fáceis de manter. Ou você pode usar diferentes folhas de estilo

com o mesmo conteúdo para criar diferentes visualizações dos mesmos dados. Além disso, algumas pessoas navegam com o JavaScript desativado, então você precisa ter certeza de que a página ainda funciona para elas.

HTML + CSS + JAVASCRIPT

O JavaScript é adicionado por último e melhora a usabilidade da página ou a experiência de interação com o site. Mantê-lo separado significa que a página ainda funciona se o usuário não puder carregar ou executar o JavaScript. Você também pode reutilizar o código em várias páginas (tornando o site mais rápido de carregar e mais fácil de manter).

4. ESPECIFICIDADES DA LINGUAGEM JAVASCRIPT

Ao longo do primeiro trimestre, vimos a sintaxe e em que contexto Javascript foi desenvolvida. Sabemos que se trata de uma linguagem multiparadigma com suporte à orientação a objetos e linguagem funcional. Esse caráter multipropósito pode trazer causar alguns estranhamentos. Discutiremos a seguir funções, escopo e uso de memória.

1.1. FUNÇÕES

Funções são blocos de comandos agrupados para realizar uma tarefa específica. Se partes diferentes de um script repetem a mesma tarefa, você pode reutilizar a função (em vez de repetir o mesmo conjunto de instruções). Agrupar as instruções necessárias para responder a uma pergunta ou executar uma tarefa ajuda a organizar seu código. Além disso, as instruções em uma função nem sempre são executadas quando uma página é carregada, portanto, as funções também oferecem uma maneira de armazenar as etapas necessárias para realizar uma tarefa.

O script pode então solicitar que a função execute todas essas etapas como e quando forem necessárias. Por exemplo, você pode ter uma tarefa que deseja executar apenas se o usuário clicar em um elemento específico da página. Se você vai pedir à função para realizar sua tarefa mais tarde, você precisa dar um nome à sua função. Esse nome deve descrever a tarefa que está executando.

Quando você pede que ele execute sua tarefa, é conhecido como chamar a função. As etapas que a função precisa realizar para realizar sua tarefa são empacotadas em um bloco de código. Um bloco de código consiste em uma ou mais instruções contidas entre chaves.

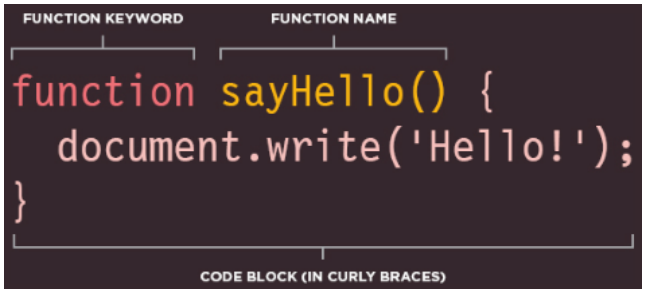
Algumas funções precisam ser fornecidas com informações para realizar uma determinada tarefa. Por exemplo, uma função para calcular a área de uma caixa precisaria saber sua largura e altura. As informações passadas para uma função são conhecidas como **parâmetros**. Quando você escreve uma função e espera que ela forneça uma resposta, a resposta é conhecida como um **valor de retorno**. Segue abaixo um exemplo de uma função no arquivo JavaScript. É chamado updateMessage(). A função tem um nome, updateMessage, e o valor é o bloco de código (que consiste em instruções). Quando você chama a função por seu nome, essas instruções serão executadas. Você também pode ter **funções anônimas**. Eles não têm um nome, então não podem ser chamados. Em vez disso, eles são executados assim que o interpretador os encontra.

```
var msg = “Sign up to receive our newsletter for 10% off!!!”;
function updateMessage(){
  var el = document.getElementById(‘message’);
  el.textContent = msg;
}
updateMessage();
```

DECLARANDO UMA FUNÇÃO

Para criar uma função, você dá a ela um nome e, em seguida, escreve as instruções necessárias para realizar sua tarefa entre as chaves. Isso é conhecido como declaração de função. Você declara

uma função usando a palavra-chave **function**. Você dá à função um nome (às vezes chamado de identificador) seguido por parênteses. As instruções que executam a tarefa ficam em um bloco de código (estão entre chaves).

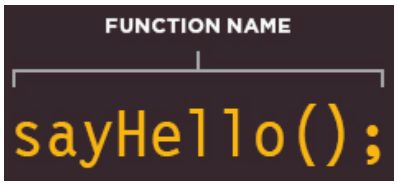


Esta função é muito básica (contém apenas uma instrução), mas ilustra como escrever uma função. A maioria das funções que você verá ou escreverá provavelmente consistirá em mais instruções. O ponto a ser lembrado é que as funções armazenam o código necessário para realizar uma tarefa específica e que o script pode solicitar que a função execute essa tarefa sempre que necessário. Se partes diferentes de um script precisam executar a mesma tarefa, você não precisa repetir as mesmas instruções várias vezes - você usa uma função para fazer isso (e reutiliza o mesmo código).

CHAMANDO UMA FUNÇÃO

Tendo declarado a função, você pode executar todas as instruções entre suas chaves com apenas uma linha de código. Isso é conhecido como **chamar a função**.

Para executar o código na função, você usa o nome da função seguido por parênteses. Na linguagem do programador, você diria que esse código chama uma função. Você pode chamar a mesma função quantas vezes quiser no mesmo arquivo JavaScript.



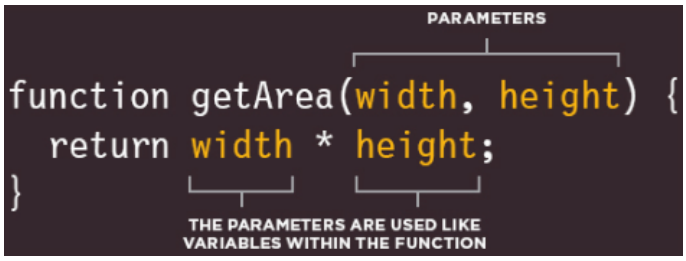
1. A função pode armazenar as instruções para uma tarefa específica.
2. Quando precisar do script para realizar essa tarefa, você chama a função.
3. A função executa o código nesse bloco de código.
4. Quando terminar, o código continua a ser executado a partir do ponto em que foi inicialmente chamado.

Às vezes, você verá uma função chamada *antes* de ser declarada. Isso ainda funciona porque o interpretador executa um script antes de executar cada instrução, portanto, ele saberá que uma declaração de função aparecerá posteriormente no script. Mas, por enquanto, declararemos a função antes de chamá-la.

DECLARAÇÃO DE FUNÇÕES QUE PRECISAM DE INFORMAÇÕES

Às vezes, uma função precisa de informações específicas para realizar sua tarefa. Nesses casos, quando você declara a função, você dá a ela parâmetros. Dentro da função, os parâmetros atuam como variáveis.

Se uma função precisa de informações para funcionar, você indica o que ela precisa saber entre parênteses após o nome da função. Os itens que aparecem dentro desses parênteses são conhecidos como parâmetros da função. Dentro da função, essas palavras agem como nomes de variáveis.



Esta função irá calcular e retornar a área de um retângulo. Para fazer isso, ele precisa da largura e da altura do retângulo. Cada vez que você chama a função, esses valores podem ser diferentes. Isso demonstra como o código pode executar uma tarefa sem saber os detalhes exatos com antecedência, desde que tenha regras que pode seguir para realizar a tarefa.

Portanto, ao projetar um script, é necessário observar as informações que a função exigirá para realizar sua tarefa. Se você olhar dentro da função, os nomes dos parâmetros são usados da mesma forma que você usaria as variáveis. Aqui, os nomes dos parâmetros largura e altura representam a largura e a altura da parede.

FUNÇÕES DE CHAMADA QUE PRECISAM DE INFORMAÇÕES

Ao chamar uma função que possui parâmetros, você especifica os valores que ela deve usar entre os parênteses após seu nome. Os valores são chamados de argumentos e podem ser fornecidos como valores ou variáveis.

ARGUMENTOS COMO VALORES

Quando a função abaixo for chamada, o número 3 será usado para a largura da parede, e 5 será usado para a altura.

```
getArea(3, 5);
```

ARGUMENTOS COMO VARIÁVEIS

Você não tem que especificar valores reais ao chamar uma função - você pode usar variáveis em seu lugar. Portanto, o seguinte faz a mesma coisa.

```
wallWidth = 3;
wallHeight = 5;
getArea(wallWidth, wallHeight = 5);
```

PARÂMETROS VS ARGUMENTOS

As pessoas costumam usar os termos parâmetro e argumento alternadamente, mas há uma diferença sutil. Na página esquerda, quando a função é declarada, você pode ver as palavras largura e altura usadas (entre parênteses na primeira linha). Dentro das chaves da função, essas palavras agem como variáveis. Esses nomes são os parâmetros.

Nesta página, você pode ver que a função getArea () está sendo chamada e o código especifica os números reais que serão usados para realizar o cálculo (ou variáveis que contêm números reais). Esses valores que você passa para o código (as informações de que ele precisa para calcular o tamanho dessa parede específica) são chamados de argumentos.

OBTENDO UM ÚNICO VALOR DE UMA FUNÇÃO

Algumas funções retornam informações ao código que as chamou. Por exemplo, quando eles realizam um cálculo, eles retornam o resultado.

Esta função **calcularArea**() retorna a área de um retângulo para o código que o chamou. Dentro da função, uma variável chamada área é criada. Ele contém a área calculada da caixa. A palavra-chave **return** é usada para retornar um valor para o código que chamou a função.

```
function calcularArea(width, height) {
  var area = width * height;
  return area;
}
var wallOne = calcularArea(3, 5);
var wallTwo = calcularArea(8, 5);
```

Observe que o interpretador deixa a função quando o retorno é usado. Isso remonta à declaração que o chamou. Se houvesse quaisquer instruções subsequentes nesta função, elas não seriam processadas. A variável wallOne contém o valor 15, que foi calculado pela função calcularArea(). A variável wallTwo contém o valor 40, que foi calculado pela mesma função calcularArea (). Isso também demonstra como a mesma função pode ser usada para realizar as mesmas etapas com valores diferentes.

OBTENDO VÁRIOS VALORES ATRAVÉS DE UMA FUNÇÃO

As funções podem retornar mais de um valor usando uma matriz. Por exemplo, esta função calcula a área e o volume de uma caixa. Primeiro, uma nova função é criada chamada getSize (). A área da caixa é calculada e armazenada em uma variável chamada área. O volume é calculado e armazenado em uma variável chamada volume. Ambos são então colocados em um vetor chamado de *sizes*. Esse vetor é então retornado ao código que chamou a função getSize (), permitindo que os valores sejam usados.

```
function getSize(width, height, depth) {
  var area = width * height;
  var volume = width * height * depth;
  var sizes = [area, volume];
  return sizes;
}
var areaOne = getSize(3, 2, 3)[0];
var volumeOne = getSize(3, 2, 3)[1];
```

A variável areaOne armazena a área de uma caixa 3 x 2. A área é o primeiro valor no array sizes. A variável volumeOne guarda o volume da caixa 3 x 2 x 3. O volume é o segundo valor no array sizes.

ESCOPO VARIÁVEL

O local onde você declara uma variável afetará onde ela pode ser usada em seu código. Se você declara dentro de uma função, ele só pode ser usado dentro dessa função. Isso é conhecido como escopo da variável.

VARIÁVEIS LOCAIS

Quando uma variável é criada **dentro** de uma função usando a palavra-chave **var**, ela só pode ser usada nessa função. É chamada de **variável local** ou **variável de nível de função**. Diz-se que tem **escopo local** ou **escopo em nível de função**. **Ela não pode ser acessada fora da função** na qual foi declarado. Abaixo, **area** é uma variável local. O interpretador cria variáveis locais quando a função é executada e as remove assim que a função termina sua tarefa. Isso significa que:

- Se a função for executada duas vezes, a variável pode ter valores diferentes a cada vez.
- Duas funções diferentes podem usar variáveis com o mesmo nome sem qualquer tipo de conflito de nomenclatura.

VARIÁVEIS GLOBAIS

Se você criar uma variável fora de uma função, ela poderá ser usada em qualquer lugar do script. É chamada de **variável global** e tem escopo **global**. No exemplo mostrado, **wallSize** é uma variável global. Variáveis globais são armazenadas na memória enquanto a página da web é carregada no navegador da web. Isso significa que elas ocupam mais memória do que as variáveis locais e também aumenta o risco de conflitos de nomenclatura (consulte a próxima página). Por esses motivos, você deve usar variáveis locais sempre que possível.

Se você esquecer de declarar uma variável usando a palavra-chave **var**, a variável funcionará, **mas será tratada como uma variável global** (isso é considerado uma prática inadequada).

ALOCAÇÃO DE MEMÓRIA PARA VARIÁVEIS

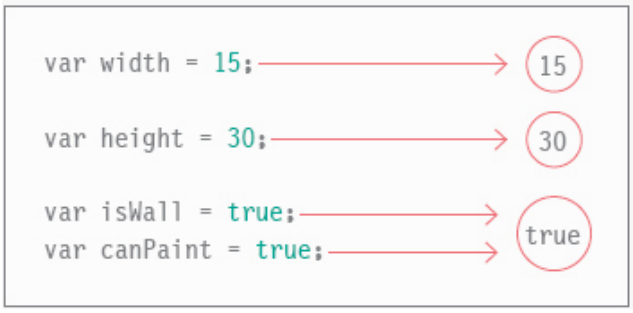
Variáveis globais **usam mais memória**. O navegador precisa se lembrar deles enquanto a página da web que os usa estiver carregada. Variáveis locais são lembradas apenas durante o período de tempo em que uma função está sendo executada.

CRIANDO AS VARIÁVEIS EM CÓDIGO

Cada variável que você declara ocupa memória. Quanto mais variáveis um navegador precisa lembrar, mais memória seu script requer para ser executado. Scripts que requerem muita memória podem ter um desempenho mais lento, o que faz com que sua página da web demore mais para responder ao usuário.

```
var width = 15;
var height = 30;
var isWall = true;
var canPaint = true;
```

Na verdade, uma variável faz referência a um valor armazenado na memória. O mesmo valor pode ser usado com mais de uma variável:



Aqui, os valores de largura e altura da parede são armazenados separadamente, mas o mesmo valor de true pode ser usado para isWall e canPaint.

COLISÃO DE NOMES

Você pode pensar que evitaria colisões de nomes; afinal você sabe quais variáveis está usando. Mas muitos sites usam scripts escritos por várias pessoas. Se uma página HTML usar dois arquivos JavaScript e ambos tiverem uma variável global com o mesmo nome, isso pode causar erros. Imagine uma página usando estes dois scripts:

```
// Show size of the building plot
function showPlotSize(){
  var width = 3;
  var height = 2;
  return 'Area: " + (width * height);
}
var msg = showArea()
```

```
// Show size of the garden
function showGardenSize() {
  var width = 12;
  var height = 25;
  return width * height;
}
var msg = showGardenSize();
```

- Variables in global scope: have naming conflicts.
- Variables in function scope: there is no conflict between them.