Comunique-se com o usuário

O `alert` é nossa primeira forma de comunicação com o usuário. Como podemos fazer para enviar duas mensagens? Basta executarmos duas vezes essa instrução. Crie um novo arquivo, que será gravado`comecando_javascript.html`, e coloque o seguinte conteúdo:

```
<script>
alert("olá mundo!");
alert("esse é meu segundo programa");
</script>
```

Você poderia ter utilizado o mesmo arquivo da lição anterior, mas é bom aqui criarmos um novo para poder acompanhar nosso aprendizado. Lembre-se de salvá-lo e depois abri-lo no seu navegador.

Verifique o resultado. Pode ser um pouco trabalhoso ter de ficar clicando no botão de *OK* a cada novo`alert`. Imagine se tivéssemos 15 mensagens para serem mostradas? Você teria que dar 15 cliques no botão *OK*. Chato, não?
Há várias formas de se comunicar com o usuário através do JavaScript e uma delas é através do`document.write`. Crie um novo arquivo, o `programa2.html` com o seguinte código:

```
<script>
document.write("olá mundo!");
document.write("esse é meu segundo programa");
</script>
```

Acesse a página e verifique o resultado.

Para não aparecer tudo numa única linha, você pode usar a tag `<code><br></code>`, que já conhecemos, para quebrá-la. Em vez de fazer `document.write("olá mundo!");`, faça:

```
document.write("olá mundo!<strong><br></strong> ");
```

Salve o arquivo e atualize a página. Entendeu a função da tag `br`?
Por enquanto só trabalhamos com mensagens fixas, estáticas. Trabalhamos com **sequências de caracteres**definidas entre os sinais das aspas. `"Casa do código"` é uma sequência de caracteres. `"Olá Mundo"`é uma outra sequência de caracteres, assim como `"Eu tenho 25 anos"` . Mas podemos também trabalhar com números:

```
<script>
document.write("Minha idade é:");
document.write(25);
</script>
```

Apenas as aspas se foram na segunda linha. Se estiver usando o *Notepad++*, vai ver que ele colore o número de forma diferente do que está entre aspas. Será então que não precisaríamos dela para mostrar as sequências de caracteres? Faça o teste sem elas e verá que, se não for apenas números, teremos um erro.

Mas por que utilizar um número sem as aspas? Afinal, poderíamos muito bem ter feito assim:

```
document.write("25");
```

Qual é a diferença de `25` para `"25"` ? Quando utilizamos as aspas, estamos dizendo ao JavaScript que queremos que isto seja encarado como uma sequência de caracteres, não como um simples número. A grande diferença estará na forma que a linguagem tratará cada um deles. Experimente fazer uma conta com números:

```
1.
2. document.write(25 + 25);
```

Agora vamos fazer algo muito parecido, utilizando o mesmo operador, porém com duas sequências de caracteres:

```
1.
2. document.write("25" + "25");
```

Esse teste é fundamental para você entender a diferença dos dois conceitos. No segundo caso, o operador `+`junta as duas sequências de caracteres. Esse processo de juntar sequências de caracteres é chamado de**concatenação**.
Em muitas linguagens, assim como no JavaScript, uma sequência de caracteres entre aspas é chamada de**`string`**.
Dizemos então que o `+`, além de somar números, concatena `strings`.
Você vai trabalhar com números ou sequências de caracteres (`string`)? Depende do que quer fazer. Com números, podemos trabalhar as operações matemáticas. Para saber uma uma estimativa do ano em que você nasceu, subtraímos o ano atual desse valor:

```
1.
2. document.write("Eu nasci em: ");
3. document.write(2012 - 25);
```

E se em vez de ter feito `document.write(2012 - 25)`, tivéssemos colocado `2012 - 25` todo entre aspas, fazendo `document.write("2012 - 25")`? Qual é o resultado?
Apenas por uma questão de concisão, às vezes vamos omitir a tag `script` dos próximos programas, como acabamos de fazer. Você deve sim utilizá-las. Aliás, o que aconteceria com nosso programa no caso de não colocarmos essas instruções dentro da tag `script`? Faça o teste.
Além do operador de subtração `-`, há o de soma `+`, multiplicação `*` e divisão `/`. Você pode somar a sua idade a dos autores. Paulo tem 32 anos e Adriano tem 26:

```
1.
2. document.write("A soma das nossas idades é: ");
3. document.write(25 + 32 + 26);
```

Para calcular a média, basta dividirmos o resultado da soma por 3:

```
1.
2. document.write("A média das nossas idades é: ");
3. document.write(25 + 32 + 26 / 3);
```

Verifique o resultado. Não é o esperado! A conta de divisão é calculada antes da soma, como na matemática da escola, logo o primeiro valor a ser calculado é `26 / 3`. Podemos utilizar parênteses para forçar a ordem desejada do cálculo, realizando primeiramente as somas:

```
1.
2. document.write((25 + 32 + 26) / 3);
```

Os parênteses são utilizados mesmo quando a precedência dos operadores já trabalha conforme esperamos, pois pode facilitar a legibilidade do que queremos fazer.

Também é possível misturar números com `strings`, mas sempre com cuidado. O que acontece ao somá-los?

```
1.
2. document.write("Minha idade é: " + 25);
```

A sequência de caracteres `"Minha idade é"` vai aparecer junta ao número `25`, isto é, serão concatenadas! Repare também que precisamos tomar cuidado com os parênteses. Vamos misturar strings e números mais uma vez:

```
1.
2. document.write("A média das nossas idades é: " + ((25 + 32 + 26) / 3));
```

Sem esses parênteses, a string seria dividida por um número, e o JavaScript geraria um valor que não nos tem muita serventia. Faça o teste!

Crie um arquivo `testando_idades.html` e vamos revisar o que já aprendemos. Coloque o código que calcula a média das idades:

```
1.
2. <script>
3. document.write("Minha idade é: " + 25);
4. document.write("A soma das nossas idades é: ");
5. document.write(25 + 32 + 26);
6. document.write("A média das nossas idades é: " + ((25 + 32 + 26) / 3));
7. </script>
```

Cada revisão dessas sempre oferecerá exercícios importantes para que você pratique e fixe o que vimos. Não deixe de ir além e realizar seus próprios testes. Sua curiosidade será importante para seu aprendizado.

Podemos imprimir o ano do nascimento de cada um de nós três, utilizando o recurso de juntar (concatenar) uma sequência de caracteres (uma *string*) com números.

```
1.
2. document.write("Eu nasci em : " + (2012 - 25) + "<br>");
3. document.write("Adriano nasceu em : " + (2012 - 26) + "<br>");
4. document.write("Paulo nasceu em : " + (2012 - 32) + "<br>");
```

Além desse que apareceu muitas vezes, tanto aqui quanto na seção anterior, o número `2012` é bastante repetido. O que acontecerá quando precisarmos atualizar esse número para `2013`? Ou quando descobrirmos que Paulo tem na verdade uma idade diferente? Precisaríamos substituir todos esses valores, um a um. Mesmo utilizando um atalho do seu editor para procurar/substituir, essa não é uma opção tão elegante. Além disso, esses números 2012, 25, 26 e 32 aparecem sem um sentido muito claro no seu código: quem lê-los provavelmente terá de se esforçar bastante para compreender o que você desejava expressas através deles.
Como então facilitar a mudança desses números e também tornar nosso código mais compreensível? O ideal seria poder ter uma forma de dizer 2012, sem precisar repeti-lo. Podemos fazer isso **atribuindo** o valor`2012` a, digamos, `ano`:

```
1.
2. var ano = 2012;
3. document.write("Eu nasci em : " + (ano - 25) + "<br>");
4. document.write("Adriano nasceu em : " + (ano - 26) + "<br>");
5. document.write("Paulo nasceu em : " + (ano - 32) + "<br>");
```

O que faz o trecho de código `var ano = 2012`? Ele **atribui** `2012` a `ano`. Chamamos `ano` de **variável**. Uma variável pode guardar praticamente o que você quiser: um número, uma string, um outro pedaço de código. Podemos fazer o mesmo com a soma das idades:

```
1.
2. var eu = 25;
3. var adriano = 26;
4. var paulo = 32;
5.
6. var total = eu + adriano + paulo;
7. document.write("A soma das idades é: " + total);
```

`var` é uma palavra especial no JavaScript. Chamamos esse tipo de palavras de **palavras-chave** de uma linguagem. Ela tem um tratamento diferenciado, nesse caso criando uma variável. Não se preocupe com a sintaxe, com essa forma diferente de escrever. Ficará mais claro no decorrer das lições.

O operador igual (=) não funciona exatamente como na matemática. Por exemplo, `2012 = ano` não funciona, não é o mesmo que `ano = 2012`. Dizemos que o operador = **atribui** o valor `2012` à variável, que fica a esquerda do =. Repare que o uso das aspas define o que será impresso. Se não há aspas dentro dos parenteses do`document.write(...)`, o JavaScript buscará o valor daquela variável. Caso contrário, utilizará o que está dentro das aspas apenas como uma sequência de caracteres (`string`), como vimos anteriormente. É muito importante você mesmo testar e ver essa diferença:

```
1.
2. var ano = 2012;
3. document.write("ano");
4. document.write(ano);
```

O que acontece?

Você utilizará variáveis o tempo inteiro. Vamos praticá-las!

Com essas mudanças no seu código, você pode calcular a média de idade da sua família de uma forma mais organizada, sem copiar os números pra dentro do `document.write`:

```
1.
2. document.write((eu + adriano + paulo) / 3);
```

Uma outra forma seria quebrar esse processo em passos. É bastante comum criar algumas variáveis a mais para ajudar a legibilidade. Um programador costuma trabalhar em uma equipe, onde outros colegas estarão sempre lendo, modificando e trabalhando com as mesmas linhas de código. A ideia aqui seria criar uma variável para a soma e outra para a média:

```
1.
2. var total = eu + adriano + paulo;
3. var media = total / 3;
4. document.write(media);
```

Na seção anterior, escrevemos nosso código de forma mais legível. Caso queira, crie o arquivo`testando_idades_com_variaveis.html` para praticarmos esse código uma última vez, depois passaremos para um outro problema. Inicialmente coloque a declaração da idade de cada pessoa:

```
1.
2. <script>
3. var eu = 25;
4. var adriano = 26;
5. var paulo = 32;
```

Depois calculamos os dados que precisamos: o total e a média.

```
1.
2. var total = eu + adriano + paulo;
3. var media = total / 3;
```

Por último imprimimos a média e fechamos a tag de `script`:

```
1.
2. document.write("A média de idade é " + media);
3. </script>
```

Seu código está funcionando? Cuidado com os nomes das variáveis. Você precisa utilizá-las da mesma forma como as declarou. O JavaScript diferencia, inclusive, maiúsculas de minúsculas. Erre o nome de uma variável para você ver qual é a mensagem de erro que aparecerá no JavaScript Console. Por exemplo, mostre `med`em vez de `media`, como havia sido declarado:

```
1.
2. document.write("A média de idade é " + med);
```

É comum digitarmos o nome de uma variável errada. Fique atento e habitue-se a utilizar o JavaScript Console que vimos no fim da lição passada. As mensagens de erro serão em inglês, e algumas vezes não serão tão específicas quanto gostaríamos, mas sempre dizendo com exatidão em que linha do código houve o problema.

Você está cansado de ver um número com tantas casas decimais? Você pode arrendondá-lo com o`Math.round(numero)`. Altere seu código:

```
1.
2. document.write("A média de idade é " + Math.round(media));
```

O `Math.round` pega o valor que está dentro dos parênteses e o arredonda, utilizando esse novo valor para juntar (concatenar) com o restante da frase que queremos mostrar.

Agora, vamos fazer um exercício completamente novo:

Vimos que utilizar variáveis já ajudou de duas formas: não precisamos mais copiar e colar tanta informação e também alguns trechos ficaram mais legíveis. As variáveis podem ir além, economizando na repetição de linhas de código.

Atenção! Nesta seção, não precisa alterar seu código, apenas acompanhe o que poderia ser feito. Você fará tudo que há de novo aqui durante a revisão, que virá logo a seguir.

Repare como ficou aquele nosso código que mostra as idades de cada pessoa:

```
1.
2. var ano = 2012;
3. document.write("Eu nasci em : " + (ano - 25) + "<br>");
4. document.write("Adriano nasceu em : " + (ano - 26) + "<br>");
5. document.write("Paulo nasceu em : " + (ano - 32) + "<br>");
```

Poderíamos organizá-lo um pouco mais, removendo o "br" para uma outra chamada do `document.write`.

```
1.
2. var ano = 2012;
3. document.write("Eu nasci em : " + (ano - 25));
4. document.write("<br>");
5. document.write("Adriano nasceu em : " + (ano - 26));
6. document.write("<br>");
7. document.write("Paulo nasceu em : " + (ano - 32));
8. document.write("<br>");
```

Já é um passo. Mas e se quiséssemos, em vez de pular uma única linha entre cada resposta, passar um traço? Ou pular duas linhas? Teríamos de modificar nosso código em **todos** os pontos que há`document.write("");`. Em vez de ter todo esse trabalho, podemos **colocar esse código dentro de uma variável**. É o que chamamos de **função**. Vamos criar uma função que executa isso e guardá-la dentro de uma variável `pulaLinha`:

```
1.
2. var pulaLinha = function() {
3.     document.write("<br>");
4. };
```

Opa! Agora apareceram muitas coisas novas. Temos um `function()`, temos as chaves (`{` e `}`), além do`document.write` estar estranhamente um pouco mais para direita. Não se preocupe em entender todos os detalhes agora. O `function()` indica que queremos criar um novo procedimento para não ter mais de copiar e colar

código. As chaves indicam o começo e o fim desse procedimento: tudo que está dentro delas faz parte dessa função. O `document.write` está mais a direita por uma questão fundamental de legibilidade. Todo programador escreverá o código dessa forma, para deixar claro que esse trecho está dentro da função que declaramos. Você pode e deve fazer isso com o `TAB` do seu teclado.
Mas como utilizar essa nova função, que está dentro da variável `pulaLinha`? Diferente de uma variável que guardava um número ou uma string, queremos **chamar** esta função, para que o código que está dentro dela seja executado. Para isso, escreveremos `pulaLinha();`, isto é, com parênteses, indicando que é para aquele código, de pular linha, ser executado. Nosso código ficaria então:

```
1.
2. var ano = 2012;
3. document.write("Eu nasci em : " + (ano - 25));
4. pulaLinha();
5. document.write("Adriano nasceu em : " + (ano - 26));
6. pulaLinha();
7. document.write("Paulo nasceu em : " + (ano - 32));
8. pulaLinha();
```

Lembrando que a **declaração** do `pulaLinha` (isto é, onde fizemos `var pulaLinha = ...`) terá de ficar em cima desse seu código.

O código está melhor? Parece até que ficou mais comprido. Vamos ver o que ganhamos com essa abordagem, e onde mais podemos melhorar.

Em vez de pular uma linha, podemos usar um efeito visual mais interessante. Que tal colocar uma linha que cruza o navegador de lado a lado? Para isso temos a tag `hr` no HTML. Como fazer com que nossa`pulaLinha` utilize essa tag em vez de `br`? Bastaria alterar suas declaração:

```
1.
2. var pulaLinha = function() {
3.     document.write( " <hr> " );
4. };
```

O que mais precisamos mudar? Nada! É exatamente essa a grande vantagem. Com as funções, conseguimos deixar um código em um único ponto, sem ter de ficar alterando muitos lugares para obter um resultado diferente do anterior.

Vamos escrever nosso código. Crie um novo arquivo, que será o `mostra_idades.html`.
Começamos com a tag de `script` e depois declarando a função `pulaLinha`. A forma de escrever (sintaxe) pode parecer estranha no começo:

```
1.
2. <script>
3.
4. var pulaLinha = function() {
5.     document.write("<br>");
6. };
```

Pronto. A variável `pulaLinha` agora se refere a uma função que pode ser chamada. Como chamá-la? Da mesma forma que você já fez com `alert`, por exemplo. Isto é, usando os parênteses após o seu nome:

```
1.
2. var ano = 2012;
3. document.write("Eu nasci em : " + (ano - 25));
4. pulaLinha();
5. document.write("Adriano nasceu em : " + (ano - 26));
6. pulaLinha();
7. document.write("Paulo nasceu em : " + (ano - 32));
```

```
8. pulaLinha();
```
Toda vez que aparece `pulaLinha();` o navegador vai executar o código da função `pulaLinha`. Dessa forma começamos a evitar código duplicado. Isso é apenas o início, vamos utilizar mais das funções para facilitar o nosso código.

Vamos novamente nos concentrar no texto, na revisão faremos o código que está sendo descrito aqui.

Já é possível enxergar bem onde ganhamos: podemos mudar o comportamento do nosso programa alterando apenas um único lugar: a função que criamos para a `pulaLinha`. Porém nosso código continua um pouco grande, e toda hora temos de chamar essa nossa função.
Quando percebemos que estamos sendo muito repetitivos, sempre podemos considerar a criação de uma nova função. Nesse caso, está fácil enxergar que toda vez que jogamos uma `frase` para o navegador com`document.write`, logo em seguida pulamos uma linha.Podemos unificar isso em um único lugar? Isso é possível criando mais uma função, uma que mostra uma frase e põe também o . Por exemplo:
```
1.
2. var mostra = function() {
3.     document.write("alguma frase<br/>");
4. };
```
Mas essa função não é tão útil: ela sempre mostra a mesma frase. Não serve para o que a gente quer, pois a frase que desejamos mostrar depende do momento. Não se desespere, há sim como resolver esse problema.
Quando **declaramos** (criamos) uma função, podemos fazer de tal forma para que recebamos algo a mais, alguma informação que seja importante para nós. No caso da função que mostra alguma frase, o que seria esse valor importante? A `frase` que queremos mostrar! Fazemos isso declarando dentro dos parênteses, como `function(frase)`, e depois utilizamos `frase` normalmente como as variáveis que já conhecemos:
```
1.
2. var mostra = function(frase) {
3.     document.write(frase + "<br/>");
4. };
```
Pronto. Mas como o `mostra` saberá que `frase` deve ser colocada no navegador? Isso será feito durante a chamada da função. Diferente do `pulaLinha`, que chamamos utilizando os parênteses sem nada dentro, o`mostra` será invocado com a `frase` dentro deles:
```
1.
2. mostra("Usando funções para melhorar o código");
```
Essa string que queremos mostrar vai ser **passada** para a função que criamos, e lá dentro será a nossa variável `frase`. Variáveis que são passadas para funções são frequentemente chamadas de **parâmetros** ou**argumentos**.
Podemos mudar um pouco a nossa função `mostra` para que ela também se aproveite da nossa velha`pulaLinha`, em vez de concatenar o por si só:
```
1.
2. var mostra = function(frase) {
3.     document.write(frase);
4.     pulaLinha();
5. };
```
É isso mesmo: uma função pode chamar outra função. É algo que ocorre com muita frequência. Qual é a vantagem aqui? Agora o nosso `mostra` também acompanhará as mudanças do `pulaLinha`. Se quisermos pular linha de uma forma mais visual, seja com `br`, `hr` ou outro recurso do HTML, a função `mostra` vai se beneficiar disso, sem nem mesmo precisar ser modificada!
Fizemos bastante durante essa lição. Vamos revisar como está seu arquivo agora. Organize-o, criando o arquivo `mostra_idades2.html`. Logo no começo, temos a definição das nossas funções. Começamos pela`pulaLinha`:
```
1.
```

```
2. <script>
3.
4. var pulaLinha = function() {
5.     document.write("<br>");
6. };
```

Logo abaixo vamos ter nossa segunda função, a `mostra`, que por sua vez faz uso da `pulaLinha`. Diferente da anterior, ela recebe um **parâmetro**, que será a frase a ser apresentada no navegador:

```
1.
2. var mostra = function(frase) {
3.     document.write(frase);
4.     pulaLinha();
5. };
```

Lembre-se de colocar o código dentro de uma função sempre mais a direita, usando o `TAB` do seu teclado. Esse é o processo de **identar** o código (neologismo do inglês *indent*). É importante que sua **indentação** esteja correta para facilitar a leitura do programa.

Após as duas funções declaradas, vamos utilizá-las no nosso código para imprimir quantos anos tem cada um dos envolvidos:

```
1.
2. var ano = 2012;
3. mostra("Eu nasci em : " + (ano - 25));
4. mostra("Adriano nasceu em : " + (ano - 26));
5. mostra("Paulo nasceu em : " + (ano - 32));
6. </script>
```

Vamos a alguns exercícios, começando por uns baseados nesse código.

O `alert` joga uma mensagem dentro de um popup. Utilizá-lo extensivamente pode acabar com a paciência do usuário, que precisará clicar em *OK* a cada nova mensagem. O `document.write` é menos intrusivo, mas você já reparou que as mensagens são jogadas diretamente na página, sem nem mesmo um espaçamento entre as linhas. Isso porque o próprio documento html é alterado. Se você quiser pular uma linha através do`document.write`, precisará utilizar tags html, como o , fazendo, por exemplo.

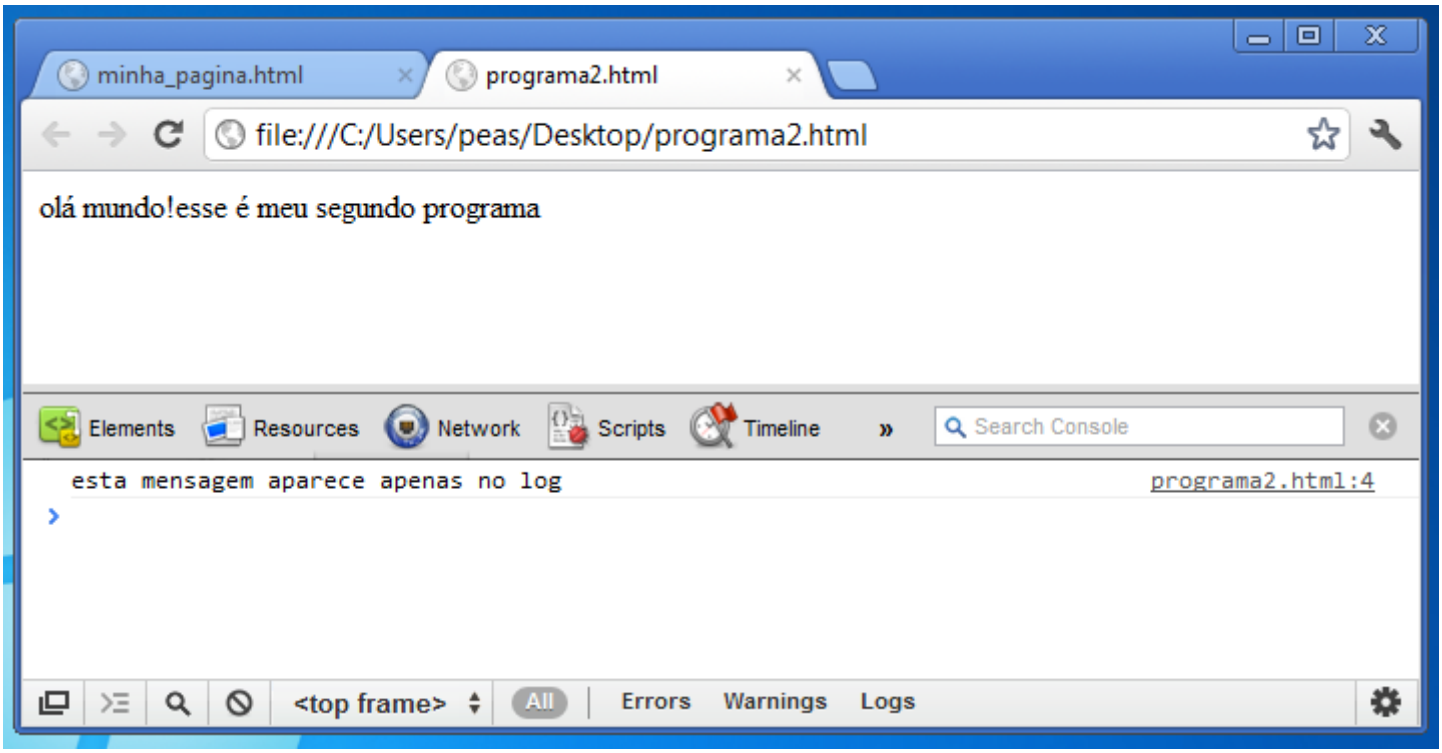
```
1. document.write("olá mundo!<br>");
```

Mesmo colocando o `document.write` em uma função, muitas vezes queremos que algumas mensagens não apareçam para o usuário, porém gostaríamos de poder vê-las durante o desenvolvimento de nosso programa. Isto é, uma mensagem que de alguma forma fosse visível apenas para você, programador. Isso é muito útil para descobrir erros (o que chamamos de **bug**), aprender novos truques e testar recursos. Guardamos esses dados em **logs**. É comum usar o neologismo **logar**, assim mesmo, em português. Para logar informações com JavaScript, há a função `console.log`. Faça um teste:

```
1.
2. <script>
3. document.write("olá mundo!");
4. document.write("esse é meu segundo programa");
5. console.log("esta mensagem aparece apenas no log");
6. </script>
```

Qual é o resultado?

A mensagem passada ao `console.log` não apareceu! Quando utilizamos essa função, o navegador guarda todas as mensagens em um local especial, longe da vista do usuário comum. Para ver o resultado precisamos habilitar a visualização do console, exatamente como fizemos na lição anterior para verificar erros.
No Chrome, você faz isso clicando no pequeno ícone de ferramentas/menus, escolhe a opção *Ferramentas*(*Tools*, se estiver em inglês) e depois *Console JavaScript*. É o mesmo console que você usou para ver as mensagens de erro do seu código:



Há a tecla de atalho *CTRL+SHIFT+J* no Windows e no Linux para abrir essa aba (*Command+Option+J* no Mac) e depois clicar no Console, caso outra opção esteja selecionada. Ele realmente é importante e você estará visitando-o com frequência.
Você pode combinar as três funções da melhor forma que encontrar: `alert` para destacar uma mensagem,`document.write` para adicionar informações dentro da própria página e o `console.log` para mostrar dados apenas a nós, programadores.
Também veremos, no decorrer do nosso aprendizado, outras formas e técnicas que nos auxiliam a descobrir problemas no nosso código. Maneiras de remover os **bugs**, isto é, como **debugar** o nosso código.
É também comum querermos colocar uma frase dentro do código que sirva apenas como referência para os programadores. Em JavaScript podemos fazer isso usando `//`. Tudo que vier após o `//` vai ser ignorado pelo navegador. Repare:

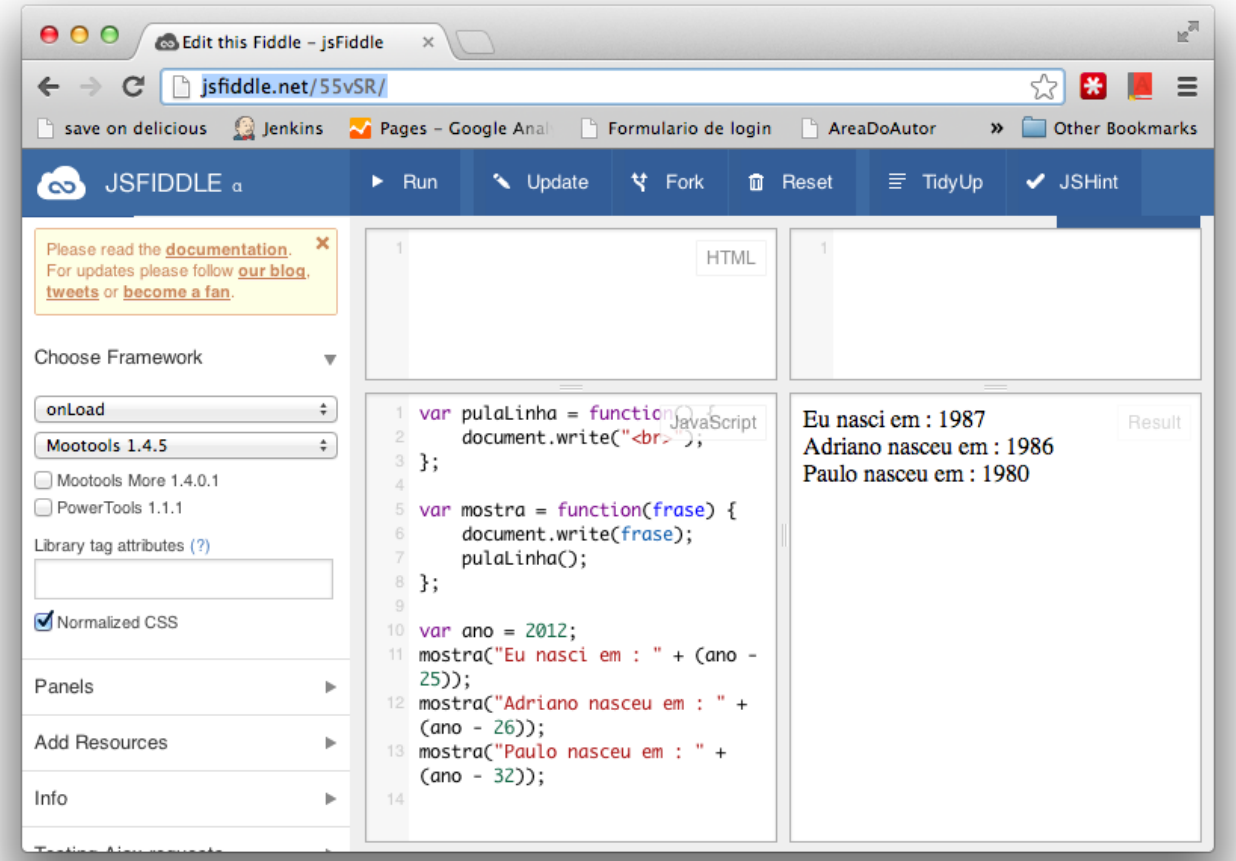
```
1.
2. // esta função mostra uma frase no navegador e pula uma linha
3. var mostra = function(frase) {
4.     document.write(frase);
5.     pulaLinha();
6. };
7.
8. // agora vamos colocar no navegador o ano em que nasci:
9. var ano = 2012;
10.mostra("Eu nasci em : " + (ano - 25));
```

Os comentários podem ajudá-lo a organizar melhor o código. De qualquer maneira, é sempre mais importante ter um código bem escrito, com nomes de variáveis expressivas que façam bastante sentido, do que ter de usar muitas linhas de comentários.

Há também a opção de colocar comentários entre `/*` e `*/`. Dessa forma, tudo que estiver entre esses dois identificadores será ignorado pelo navegador, inclusive se houver quebras de linha.
Até agora você rodou seus programas no seu próprio navegador. E se quisesse que um amigo pudesse ver o que está fazendo? Caso você já conheça um pouco mais de internet, poderia colocar seus arquivos `.html`em um servidor web. Mas há uma forma bem fácil de compartilhar seus exercícios e mostrar suas recém adquiridas habilidades de programador.

Alguns sites permitem que você escreva código HTML e JavaScript dentro de formulários e veja rapidamente o resultado. Qual é a vantagem de escrever dentro de um site em vez de no nosso próprio computador? É que esses sites permitem compartilhamento dos programas! Acesse nosso programa que calcula idade aqui:

http://jsfiddle.net/55vSR/
Esse site vai apresentar 4 diferentes espaços: HTML, CSS, JavaScript e o resultado. O JavaScript é o que nos interessa. Como essa é uma página especial, você não precisa (nem deve) usar a tag `script` dentro desse formulário. Você pode clicar em *Run* (no menu superior) ou pressionar `CTRL+ENTER` para rodar o código.



Você pode editar esse mesmo código e clicar em *Save*. O site JSFiddle vai gerar um novo endereço que você pode compartilhar com seus amigos e familiares. É uma forma interessante de mostrar sua evolução na programação para todos os conhecidos!

Também é possível criar uma conta para que você tenha todos os seus códigos organizados.

Compartilhe seus códigos e desafios na nossa lista de discussão! Não deixe de participar, seja tirando dúvida ou mostrando suas conquistas:

https://groups.google.com/group/comece-a-programar/
Há outras alternativas, talvez um pouco mais complexas e completas, como o http://tinkerbin.com/, ohttp://playground.html5rocks.com e o http://codepen.io/.