

Imersão Dev - Aula 6:

Rafaella: Olá, Dev! Descansou no final de semana? Eu espero que sim e que esteja tudo pronto para começarmos a nossa **segunda semana de Imersão Dev**. Na semana passada, nós estudamos muitos conteúdos sobre lógica de programação. Nessa semana, além de novos conceitos, também aprenderemos como aplicar tudo que já estudamos. Não sei se você já pesquisou a **#ImersãoDev** nas redes sociais, mas está "bombando" com o projeto e eu estou absurdamente orgulhosa.

Se você tiver qualquer dúvida nessa aula ou em qualquer outra, você pode nos procurar na nossa comunidade do Discord. Então, pegue o café, abra o CodePen e vamos para a **sexta aula**.

Paulo: Olá para você aluna/aluno da Imersão Dev que está encarando essa nossa sexta aula, que faz parte da segunda semana. Claro que cada pessoa está em um estágio diferente, cada um assistindo à aula em um momento diferente, mas eu sei que muitas pessoas estão, às oito da manhã, esperando para receber o nosso e-mail, para ver a Rafa e o Guilherme.

Nesse segundo momento da Imersão, nós estudaremos novos conceitos, novas ferramentas, novas instruções e comandos, estruturas lógicas, mas, você perceberá que nós aplicaremos tudo que aprendemos até aqui, desde a declaração de variáveis, alterar o HTML, até usar as listas, o *if*, usar a estrutura menos comum do *for*.

Isto é, nós vamos misturar a utilização de todos os conceitos para dar mais um passo e criar um projeto ainda mais interessante, que seja ainda mais vivo, no CodePen, para que você possa mostrar para as pessoas, criar o seu *portfolio*, nos mostrar no Discord, compartilhar no LinkedIn e para que você comece a fazer parte do **AluraVerso**, nós queremos você nessa comunidade.

Não só no Discord da Imersão Dev e não só se você é aluno da Alura, queremos você em todos os lugares, para que você participe, de fato, da comunidade de tecnologia brasileira. Acho que, hoje, o Guilherme e a Rafa têm algo de interesse para muitas pessoas, um sistema que faremos e que dá para aproveitar e usar em várias situações. Agora é com vocês!!

Rafaella: Exatamente, Paulo! Hoje, faremos um projeto de tabela de classificação. Então, será uma tabela para contarmos pontos de vitórias, de derrotas e empates também. Nós faremos esse cálculo e aprenderemos um conceito novo, conhecido como **Objeto**. Já aprenderemos como tudo isso funciona.

Guilherme: Vamos fazer o Fork?

Rafaella: Vamos fazer o Fork.

Guilherme: Já estou com o link do projeto, o template principal, e vamos dar uma "garfada" apertando o Fork, estou logado na minha conta e já temos o projeto com o HTML, o CSS e o Layout bem bonito para criarmos o nosso projeto. Rafa, quer explicar um pouco sobre qual é a ideia?

Paulo: E classificação de quê? No que dá para usar, Rafa? É o que mais interessa. Já gostei da foto das pessoas na Lan House, na minha época chamávamos assim, mas acho que deve ser só a imagem de um campeonato. Hoje em dia, as pessoas só se sentam lado a lado em campeonato oficial mesmo.



Rafaella: Essa tabela, sinceramente, nós conseguimos usar onde quisermos. Faremos a lógica pensando em uma quantidade de pontos. Por exemplo, uma vitória vale 3 pontos, mas, é possível adaptar para o jogo desejado, cartas, videogame, e colocar quantas colunas, linhas, pessoas que vão jogar, também. Olhando nosso HTML, temos uma tabela. Na linha 12, está escrito `table`.

```
<h1>Tabela de classificação</h1>

<table style="width:100%">
  <thead>
    <tr>
      <th>Nome</th>
      <th>Vitórias</th>
      <th>Empates</th>
      <th>Derrotas</th>
      <th>Pontos</th>
      <th colspan=3>Ações</th>
    </tr>
```

```
</thead>
<tbody id="tabelaJogadores">
<tr>
<td>Paulo</td>
```

Essa *tag* **table** é onde começamos a fazer a tabela, justamente essa tabela em que aparece Paulo, Rafa, as vitórias, empates, derrotas e os pontos também.

Tabela de Classificação

Nome	Vitórias	Empates	Derrotas	Pontos	Ações		
Paulo					Vitória	Empate	Derrota
Rafa					Vitória	Empate	Derrota

Primeiro, queremos preencher a nossa tabela com os pontos que cada pessoa for ganhando e temos, no HTML mesmo, algumas *tags* que vão preenchendo essa tabela. O **<thead>** é, de fato, o cabeçalho da nossa tabela, então, colocaremos os títulos de cada uma das colunas. Então, temos o **<th>**, que é o *header*, as primeiras linhas que terão uma atenção diferente das outras células dentro da tabela, que é: Nome; Vitórias; Empates; Derrotas; e Pontos.

Cada uma dessas *tags* simboliza uma das células de cabeçalho. Embaixo, temos o corpo da nossa tabela e cada uma das células também estarão dentro de uma **<td>**, uma *Table Data*, são os dados que queremos colocar na nossa tabela. O **<tr>** são as linhas. Portanto, dentro de cada *tag* **<tr>**, temos uma linha, por isso existe o **<tr>** Paulo, **<tr>** Rafa, logo, existirá uma **<td>** para cada uma dessas linhas, a **<td>** dos pontos de vitória, de empate, de derrotas e de pontos.

Todas elas estão, ainda, sem os pontos, nós preencheremos mais para frente. O cabeçalho é o **<th>** de *Theader*.

Paulo: Rafa, apenas confirmando, nós temos quatro "tds", o do Paulo, depois, quatro espaços, referente às células vitórias, empates, derrotas e pontos e, em seguida, três botões e, então, tudo de novo, dessa vez para Rafa.

Rafaella: Exatamente!

Paulo: E, então, são *tags* novas, o HTML tem infinitas *tags*. O nosso foco será no JavaScript, mas gostei de você ter feito essa passagem, porque, para quem quiser

explorar, não se assustar. Provavelmente, mudaremos o `innerHTML`, pegaremos o `value`, entre outras coisas.

Rafaella: Isso também podemos sempre pesquisar, por exemplo, "tabela no HTML", e encontraremos todas as `tags` já com a tradução do `td`, como eu estava comentando, mas, são essas que utilizaremos por formarem uma tabela de classificação. Poderíamos começar preenchendo, no próprio HTML, os pontos de cada pessoa. O que vocês acham? O que podemos jogar e pontuar?

Paulo: Pode ser **Warcraft III**, um jogo das antigas, RTS. Talvez o pessoal mais jovem não conheça, mas, está bom.

Guilherme: Vou colocar qualquer valor: vitória, 2; empate, 5; derrotas, 1; número de pontos, 24.

```
<td>Paulo</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>24</td>
```

Paulo: Só lembrando que a tela está piscando, porque, nesse CodePen, nós não ativamos os `settings` para não autocarregar. Repararam que o Guilherme foi digitando e a tela ficou piscando? É possível usar assim, conforme a preferência. Nós não estamos fazendo isso, porque, no meio da explicação, ficamos confusos, ele começa a dar erro. Enfim, para o curso, é mais interessante que isso não aconteça, mas fique a vontade para usar como preferir.

Guilherme: Vou aproveitar e já mudar. Para isso, apertaremos o botão "Settings". Na próxima tela, selecionaremos "Behavior". Nós não queremos o salvamento automático e a atualização automática, e, sim, a formatação, então, moveremos o botão de "Format on Save" para sim, "On". Depois, vamos em "Pen Details". Nas "Tags", adicionaremos as nossas três `tags`: `aluraflix`, `imersaodev`, `alura`. No `aluraflix`, colocaremos "tabela de jogos". Portanto, ficará: `tabela de jogos`, `imersaodev`, `alura`. Agora, basta salvar, "Save & Close".

Continuando, faremos os valores da Rafa. Vamos lá, Rafa! Quais são os seus valores?

Rafaella: Pode ser umas três vitórias.

Guilherme: Quantos empates?

Rafaella: Cinco empates, porque tem que ser igual.

Guilherme: Pode ser qualquer valor.

Paulo: É que não está fazendo sentido. Se ela teve três vitórias e eu tive uma derrota, está faltando um pedaço da tabela com outras pessoas. Mas, tudo bem, eu entendi, você está colocando números aleatórios, porque o cálculo de pontos vai mudar, enfim, nós ainda faremos muitas coisas.

Guilherme: Sim, é isso. Então, vou colocar qualquer valor: 6 e, no próximo, 2. Não faz o mínimo sentido, porque são valores aleatórios.

```
<td>Rafa</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2</td>
```

A Rafa teve três vitórias, mais que o Paulo, e está com 2 pontos, não tem lógica.

Tabela de Classificação

Nome	Vitórias	Empates	Derrotas	Pontos	Ações		
Paulo	2	5	1	24	Vitória	Empate	Derrota
Rafa	3	5	6	2	Vitória	Empate	Derrota

Nós temos os botões de "Vitória", "Empate" e "Derrota". Quando apertamos eles, não acontece nada na parte visual, mas recebemos uma mensagem no JS avisando que algo não está certo. Ao abrir o console, encontramos as ações desses botões e eles informam que `adicionarDerrota` não está definido, `"adicionarDerrota is not defined"`, `adicionarEmpate` também não está definido, `"adicionarEmpate is not defined"`, e `adicionarVitoria`, também não, `"adicionarVitoria is not defined"`. Faz sentido, pois ainda não criamos.

Seguindo, gostaria de destacar um ponto importante. A Rafa explicou muito bem que cada pessoa que for jogar, precisará da estrutura do HTML inteira. Para adicionar uma nova pessoa, no caso, se eu quiser jogar com vocês, precisarei copiar o HTML inteiro, dar um "Enter", colar ele embaixo, mudar o nome de "Rafa" para "Gui", salvar, "Save", e rodar, "Run".

```
<tr>
  <td>Gui</td>
  <td>3</td>
  <td>5</td>
```

```

<td>6</td>
<td>2</td>
<td><button onClick="adicionarVitoria()">Vitória</button></td>
<td><button onClick="adicionarEmpate()">Empate</button></td>
<td><button onClick="adicionarDerrota()">Derrota</button></td>
</tr>

```

Salvei, rodei e, agora, apareço na tabela, logo abaixo da Rafa.

Tabela de Classificação

Nome	Vitórias	Empates	Derrotas	Pontos	Ações		
Paulo	2	5	1	24	Vitória	Empate	Derrota
Rafa	3	5	6	2	Vitória	Empate	Derrota
Gui	3	5	6	2	Vitória	Empate	Derrota

Ficou bom, mas pense num jogo em que precisarmos cuidar dos pontos, das vitórias, dos empates e derrotas de 20 pessoas. Imagine como seria. O nosso HTML já está com 58 linhas, o que já é muita coisa, manter desse jeito é difícil. Como conseguimos guardar essas informações? O nome da jogadora ou jogador, vitórias, empates, derrotas e os pontos? No JavaScript, falamos: nós temos esses jogadores, mostre-os na tela. Vamos aprender uma técnica diferente para isso.

Primeiro, vou comentar a parte do `</tbody>` inteiro. O HTML do CodePen gosta de ficar “trollando”, então, vou comentar até o `<tr>` (de baixo para cima). Para comentar, eu apertei Ctrl + barra, e ele colocará uma *tag* para comentarmos no HTML `<!--`. O mesmo acontece no JavaScript, se apertarmos “Ctrl + barra”, já aparece a opção para comentarmos `//`.

Paulo: Já é possível perceber que, em cada linguagem, o mesmo tipo de comando tem uma aparência diferente.

Guilherme: Podemos até verificar no CSS também, ao apertar “Ctrl + barra”, já aparece de outra forma, `/* */`. Continuando, eu comentei as linhas e, agora, vamos criar, do lado do JavaScript, todas essas informações. Minha intenção é criar uma variável capaz de armazenar propriedades diferentes, nome, vitória, empates e derrotas, no JavaScript. A primeira coisa que faremos é criar uma nova variável.

Então, vou criar a `var rafa =`. Diferente das outras propriedades, se colocarmos apenas o nome “rafa”, isto é, `var rafa = "Rafa"`, teríamos uma variável que está armazenando uma palavra, e não é isso que queremos. Se colocamos uma lista, `var rafa = []`, vamos armazenar outras informações, mas queremos uma estrutura capaz de fazer coisas que

a lista não é capaz de fazer. Por isso, não usaremos colchetes e nem parênteses, mas, sim, chaves, `var rafa = {}`.

Todas as vezes, no JavaScript, que trabalharmos com chaves, nós queremos referenciar um tipo chamado de **objeto**. Vocês perceberão que a estrutura dele não é complexa, não é tão difícil assim. Um objeto é formado por propriedades de tipos diferentes. Por exemplo, nesse objeto, `var rafa = {}`, podemos ter uma propriedade chamada nome: e dizer que ela receberá, com os dois pontos, o nome "Rafa", ou seja, `var rafa = { nome: "Rafa" }`.

Agora, podemos colocar uma vírgula e adicionar outras propriedades, como vitórias: e atribuímos o valor 2, `var rafa = { nome: "Rafa", vitorias: 2 }`. Repare que estamos separando cada propriedade com a vírgula. Então, temos o número de vitórias, também os empates, `var rafa = { nome: "Rafa", vitorias: 2, empates: 1 }`, derrotas, `var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1 }` e, para finalizar, os pontos, `var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }`.

Deixei os pontos com valor zero, porque vamos calcular daqui a pouco. Isso é um objeto do JavaScript: `var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }`. Temos uma sequência de variáveis diferentes apontando para uma mesma referência que, no caso, é a `rafa`. Vamos criar o do Paulo também.

Paulo: Só um minuto, Guilherme. Antes de colocar o do Paulo, vamos entender e imprimir um só para conseguirmos analisar. Pelo que entendi, é como se fosse uma variável e, dentro dela, podemos guardar várias variáveis. Está realmente parecendo uma linha da nossa tabela. Uma só linha. E se mandarmos imprimir?

Guilherme: Vamos ver o que acontece? Colocarei um `console.log()` e, entre parênteses, "rafa": `console.log(rafa)`. Agora, basta salvar e executar. Nada será exibido na tela, nós só queremos ver no console. Ele nos mostra um tipo objeto, `// [object Object]`, e o `"nome": "Rafa"`, com os outros valores (vitórias, empates, derrotas e pontos).

Paulo: Algumas pessoas chamarão isso de **registro** ou de **estrutura**. No JavaScript, nós falaremos muito de objeto.

Guilherme: Vamos criar o seu, Paulo? Então, vamos criar mais uma variável chamada Paulo, portanto, mudaremos o nome para Paulo, ou seja: `var paulo = { nome: "Paulo", }` e adicionar o valor de vitórias, empates, nas derrotas, colocaremos uma a mais, só para fazer um pouco de sentido.

```
var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }
var paulo = { nome: "Paulo", vitorias: 1, empates: 1, derrotas: 1, pontos: 0 };

console.log(rafa);
```

Um ponto importante nesta parte é que nós temos o `console.log()` da Rafa e vamos fazer um do Paulo também, `console.log(paulo)`, salvar e rodar, para visualizarmos os dois objetos no nosso console. Então, temos dois objetos, eles são referenciados por chaves, e também todas as propriedades. Ficou bem interessante.

```
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 0
}

{
  "nome": "Paulo",
  "vitorias": 1,
  "empates": 1,
  "derrotas": 2,
  "pontos": 0
}
```

Rafa, o que podemos fazer para deixar nosso projeto ainda melhor?

Rafaella: Nós poderíamos mostrar como acessar uma informação dentro desse objeto. Assim como nas listas, onde conseguimos acessar um item da lista por meio do índice, que começava com zero e assim por diante. No objeto, como usamos chave e valor, então, a chave é sempre o que é aquele valor e em seguida o valor, ou seja, "nome", dois pontos e o valor dele, "vitória", dois pontos e o número de vitórias.

Paulo: A estrutura é chave, dois pontos, valor?

Rafaella: Isso! Chave, dois pontos, valor. Podemos acessá-los usando essas chaves. É justamente para isso que as utilizamos. No `console.log(rafa)`, é possível acessar, por exemplo, o número de vitórias que temos, então, para isso, colocamos o ponto vitórias, `console.log(rafa.vitorias)`, sempre chamamos a nossa chave para descobrir o valor dela. No Paulo, podemos tentar acessar o `.empates`. Vamos salvar e rodar.

Ele puxou diretamente o valor que colocamos à frente da chave que chamamos, o 1. Portanto, essa é uma forma de acessar os valores específicos. Pode acontecer de desejarmos fazer um cálculo apenas com as vitórias ou apenas com as derrotas e é dessa forma que conseguiremos. Não é necessário chamar o objeto inteiro, o `rafa` inteiro com todas as informações.

Então, Gui, primeiro, podemos calcular os nossos pontos. Assim como você tinha colocado que estamos com o ponto zero, acredito que o motivo é que queremos calcular os pontos de uma maneira personalizada. Por exemplo, para cada vitória, dar 3 pontos e, para cada empate, 1 ponto, e para cada derrota, não fazemos nada. O que você acha? Vitória vale 3, empate vale 1 e derrota não vale nada, não soma nada e também não diminui nenhum ponto.

Para isso, podemos criar uma função para calcular esses pontos, isto é, uma função `calculaPontos`. Vamos usar o `function`, o nome dela pode ser `calculaPontos`, seguida de parênteses e as duas chaves para colocarmos o que desejamos executar quando chamamos essa função.

```
function calculaPontos() {  
  
}
```

Primeiro, vamos calcular os pontos de um jogador ou jogadora específico. Portanto, precisamos mandar essa informação para a nossa função, enfim, mandar qual é o jogador ou jogadora que queremos alterar os pontos, não dá para mudar os pontos de todas as pessoas ao mesmo tempo, porque não faria sentido, alteraríamos de uma forma muito errada. Nós já aprendemos como mandar informações para dentro de uma função e a utilizá-la dentro, que é usando parênteses.

Então, dentro dos parênteses, vamos inserir a nossa variável jogador, já que mandaremos um jogador por vez para calcularmos o ponto dele, `function calculaPontos(jogador) {`. E dentro, faremos esse cálculo de três vitórias, uma vitória vale três pontos e o empate vale 1. Sendo assim, criamos uma nova variável para contarmos esses pontos, `var pontos =` e vamos passar os números de vitória e de empate.

Para acessarmos os números de vitórias desse jogador, faremos `jogador.vitorias`.

```
function calculaPontos(jogador) {  
  var pontos = jogador.vitorias  
  
}
```

Lembrando que, quando passamos para dentro da função alguma informação, é nos parênteses que indicaremos como vamos chamá-la. Podemos passar, por exemplo, objeto `rafa` para dentro da função, mas, quando ele entra na função, nos parênteses já aparece que o objeto `rafa` virou `jogador`, então, para acessarmos as informações da Rafa, utilizaremos o `jogador`, assim como utilizaríamos o `rafa`. Portanto, `var pontos = jogador.vitorias`.

Continuando, nós faremos vezes três, porque cada vitória valerá três, `var pontos = jogar.vitorias * 3`. E podemos deixar entre parênteses para garantir que primeiro faremos a multiplicação, `var pontos = (jogador.vitorias * 3)`, somaremos com o número de empates e para acessar o número de empates dessa pessoa, colocaremos `jogador.empates`, isto é, `var pontos = (jogador.vitorias * 3) + jogador.empates`.

Vamos dar um `console.log()` nesses pontos, só para verificarmos se o cálculo está funcionando da maneira que queremos, `console.log(pontos)`. Nós precisamos, também, chamar essa função.

Paulo: Rafa, mostre uma coisa para mim. E se não chamarmos? Acabamos de escrever essa função que calcula pontos de um jogador. Ela multiplica vitórias por 3 e soma 1 ponto para cada empate. Até poderíamos ter escrito vezes 1, mas é como o resultado é ele mesmo, nós não escrevemos, mas poderíamos ter escrito. Nós também logamos o ponto. Se rodarmos agora, ele calcula o ponto para quem?

Rafaella: É isso que queremos saber.

Guilherme: Vamos lá! Posso rodar? Rodei e ele não calculou e nem exibiu nada no console.

Paulo: Nem mensagem de erro, não é?

Guilherme: Nem mensagem de erro.

Paulo: Porque ele executou o que foi pedido. Toda vez que escrevemos uma `function`, estamos declarando um bloco de código que pode ser chamado, ou, invocado, outro termo usado para isso. Não tem ninguém dizendo: calcule agora os pontos para o jogador. Nós apenas declaramos.

Quando usamos a palavra-chave `function`, estamos criando a função, não estamos chamando. A Rafa já havia comentado, e, eu pensei que deveríamos testar assim. Agora falta pedir para calcular os pontos de algum dos jogadores que temos disponíveis.

Rafaella: Exatamente! Podemos fazer isso do lado de fora da função, que, de fato, será executado logo que rodarmos o nosso projeto. Para isso, como vamos executar a função? Colocaremos, antes da função, o `calculaPontos` e, entre parênteses, passar o objeto que será o jogador dentro da função. Por exemplo, pode ser "rafa", `calculaPontos(rafa)`.

```
calculaPontos(rafa);

function calculaPontos(jogador) {
  var pontos = jogador.vitorias * 3 + jogador.empates;
  console.log(pontos);
}
```

```
}
```

Paulo: A Rafa e o Guilherme estão sendo ousados e colocando o código `calculaPontos()` antes da declaração da função. Isso pode gerar certo estranhamento por pensarmos que ainda não existe a função. Mas, tanto no JavaScript, como, hoje em dia, na maioria absoluta das linguagens, essa ordem não importa, considerando que se trata de um bloco de declaração.

Nós aprenderemos que, por organização, colocaríamos isso em outro lugar, mas é interessante perceber que, neste caso, a ordem não importa, mas, em outras situações, sim.

Rafaella: É verdade. Então, vamos executar e verificar o nosso `console.log()`.

Guilherme: Salvei, rodei. Ele apresentou o valor 7.

Rafaella: Está certo, isso? Vamos analisar, 3 vezes 2 é igual a 6, mais 1, é igual a 7. Está certo! Nossa função está, de fato, fazendo corretamente o cálculo de pontos que queremos. Porém, estamos apenas imprimindo esses pontos.

Paulo: Rafa, posso colocar outra charada que eu fiquei pensando? Ao invés de ter criado o `var rafa` que tem, dentro, várias variáveis, campos, atributos, poderíamos ter feito `var rafa = nome`, `var rafa = vitórias`, `var rafa = empates`, `var rafa = pontos`.

Mas aí teríamos o mesmo problema que tivemos na lista. Começar a criar mil variáveis, começar a se perder e não conseguiríamos ter uma função que pega só a vitória e só o empate. Teríamos que passar vários parâmetros, dá pra passar parâmetros separados por vírgula, mais de um parâmetro, mais de um argumento para uma função.

Eu gostei desse modelo das variáveis. Só queria saber o que acontece, por exemplo, se digitarmos errado. Se ali na `var rafa` em vez de `empates: 1`, tirar a última letra "e" e deixar `"empats: 1"`. Sutil, né? Quando executarmos esse código, o que vai acontecer?

Ele retornou `NaN`. Então ele não achou. Na hora em que ele tentou pegar o `jogador.empates` veio provavelmente aquele `undefined`, ou alguma daquelas coisas de quando tentamos acessar uma variável que tem algumas formas - tem uma série de regras mais complexas por trás - e na hora em que ele somou com número, porque o `vitórias` ele achou, o 2x3 ele fez. Depois, 6 mais uma variável que não existe, ele devolveu esse "NaN" (*Not a Number*).

Ele falou "Você está fazendo uma conta com variáveis que não consigo usar". Se ainda fosse uma string. Lembra que somávamos e ele concatenava? Ainda daria para fazer alguma coisa. Mas aqui foi uma maluquice tão grande, que o JavaScript retornou isso. Lembrando que isso não é erro, não deu uma linha vermelha, o Codepen não falou que você escreveu errado. Ele retornou um "NaN" e às vezes pode ser o que você queria, não é o nosso caso.

Repare que nesse mecanismo dos objetos também temos que tomar cuidado e fazer corretamente. Garantir que todo jogador tem `empates` e `vitorias`, que esses objetos seguem a mesma interface, eles têm uma forma parecida de se comunicar. Não pode ter um jogador sem empates, se isso acontecer nosso cálculo de pontos não vai rolar. Claro que poderíamos nos defender disso, mas no âmbito geral é assim que vamos seguir.

Guilherme: Uma coisa legal é pesquisar os erros e coisas inesperadas que aparecem aqui. Por exemplo, se eu não conheço esse "NaN" posso pesquisar no Google "NaN JavaScript" e abrir esse resultado da pesquisa que é o [link da documentação do Mozilla](#). Vai ter uma explicação detalhada sobre essa propriedade.

Então, qualquer erro desconhecido que você tiver, dê uma pesquisada, na própria documentação ou na comunidade, no Stack Overflow e outros fóruns.

Rafaella: Já temos a nossa função pronta, porém estamos fazendo uma única coisa. Além de calcular os pontos, estamos só imprimindo esses pontos no console. E não é isso que queremos. O que nós queremos é mandar o resultado desses pontos para um outro lugar que não está dentro da função, porque futuramente ainda vamos imprimir esses pontos. Vamos juntar todos os pontos com as vitórias e com as derrotas e imprimir na tela futuramente com outra função que faremos adiante.

Mas como fazemos para enviar o resultado desses pontos da função para outro lugar?

Guilherme: Rafa, eu tenho uma pergunta. Se eu colocar, por exemplo, um `console.log(rafa)` antes da função e colocar outro `console.log(rafa)` depois da função, os pontos que já calculamos, 7 pontos, não entraram no JavaScript?

Rafaella: Não. Mas vamos ver isso.

Guilherme: Esqueci de chamar a função `calculaPontos(rafa)`, vou chamar. Então temos o `console.log()` que vai mostrar exatamente como está o objeto, chamamos o `calculaPontos` passando a `rafa` de jogador, e no fim tem mais um `console.log(rafa)`.

```
var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }
var paulo = { nome: "Paulo", vitorias: 1, empates: 1, derrotas: 1, pontos: 0 };

console.log(rafa);
calculaPontos(rafa);

function calculaPontos(jogador) {
  var pontos = jogador.vitorias * 3 + jogador.empates;
  console.log(pontos);
}

console.log(rafa);
```

Guilherme: Vamos salvar, limpar o console e rodar. Vejamos o que o console exibiu:

```
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 0
}

7

{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 0
}
```

Guilherme: Temos Rafa e as informações dos campos. Em seguida, apareceu os pontos: 7. Mas no último `console.log()` quando mostrei você de novo os pontos estão zerados, então esses 7 pontos não entraram.

Rafaella: Exatamente. É por isso que precisamos fazer uma alteração fora da função. Como podemos mandar o resultado da nossa variável de pontos dentro da função para fora dela?

Nós podemos retornar esse valor e atribuir ele a outra variável. No caso, podemos usar a própria variável de pontos que já existe. Vamos primeiro criar uma variável e depois atribuímos para a que já existe.

Vamos criar `var resultadoDosPontos`. E fazer o retorno da nossa função, que serão os pontos, serem atribuídos a essa variável. Vamos então fazer o retorno dela. Pode apagar o `console.log()`, não vamos precisar imprimir nada no console. E para retornar esses pontos vamos apenas escrever `return pontos`.

Então quando chamarmos essa função ela não vai ser executada. Ela vai, na verdade, retornar um valor.

Agora conseguimos atribuir o resultado dessa função para uma variável.

Ali você podem ver que colocamos na linha 7 o `calculaPontos(rafa)` porque só iríamos executá-la, mas agora que ela tem um retorno, tem um valor, vamos atribuir esse valor para uma nova variável, a `var resultadoDosPontos`, ela vai ser igual a `calculaPontos(rafa)`, que vai ser o retorno da função quando chamamos esse `calculaPontos` mandando o objeto `rafa` como jogador.

Lá embaixo vamos fazer um `console.log(resultadoDosPontos)` para ver se conseguimos retornar os pontos para essa variável.

```
var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }
var paulo = { nome: "Paulo", vitorias: 1, empates: 1, derrotas: 1, pontos: 0 };

console.log(rafa);
var resultadoDosPontos = calculaPontos(rafa);

function calculaPontos(jogador) {
  var pontos = jogador.vitorias * 3 + jogador.empates;
  console.log(pontos);
}

console.log(resultadoDosPontos);
```

Rafaella: Vamos salvar, limpar o console e executar.

```
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 0
}

7
```

Rafaella: Olha só. O console retornou o valor que tínhamos colocado lá dentro, que foi o cálculo com resultado de 7 pontos. Agora estamos atribuindo esse valor do retorno da função para uma variável nova, mas já temos uma variável que queremos alterar e já está pronta, que é o `pontos`, no objeto `rafa`.

Para isso podemos alterar diretamente essa variável. Pode apagar essa `var resultadoDosPontos` e no lugar dela escrever `rafa.pontos`, que é a forma como acessamos uma chave e alteramos ou chamamos seu valor.

```
rafa.Pontos = calculaPontos(rafa);
```

Rafaella: Agora vamos imprimir um `console.log(rafa)`. Vamos salvar e rodar para ver o que aparece no console:

```
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 0
}

{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 7
}
```

Guilherme: Agora sim!

Rafaella: Agora foi alterado. Primeiro nós imprimimos o primeiro `console.log(rafa)` antes de chamar a função e embaixo chamamos a função, trocamos o número de pontos no `rafa.pontos`, e lá embaixo imprimimos essa função já com o valor alterado.

Guilherme: Isso ficou bem legal. Agora conseguimos, por exemplo, calcular os pontos da Rafa e os pontos do Paulo. Só para ter certeza, vou calcular e imprimir também os pontos do Paulo já calculados. Vou apagar esse primeiro `console.log()` que não vamos mais usá-lo.

```
var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 }
var paulo = { nome: "Paulo", vitorias: 1, empates: 1, derrotas: 1, pontos: 0 };

rafa.pontos = calculaPontos(rafa);
paulo.pontos = calculaPontos(paulo);

function calculaPontos(jogador) {
  var pontos = jogador.vitorias * 3 + jogador.empates;
  return pontos;
}

console.log(rafa);
console.log(paulo);
```

Guilherme: Ao rodar o código, tanto o Paulo quanto a Rafa terão os seus pontos calculados:

```
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 7
}

{
  "nome": "Paulo",
  "vitorias": 1,
  "empates": 1,
```



```
"derrotas": 2,  
"pontos": 4  
}
```

Guilherme: Rafa está com 7 pontos e o Paulo com 4 pontos. Mas não faz muito sentido exibir isso só no console. Nós queremos exibir essas informações na Tabela de Classificação lá no nosso HTML. Para conseguirmos manipular esses objetos de forma mais simples, vou colocar ambos os objetos dentro de uma lista chamada "jogadores".

Paulo: Antes disso, vamos dar uma revisada nesse código. Já tem bastante coisa. Para ficar um pouco mais simples de lermos o que está acontecendo, mova a função para o bloco central, embaixo da linha de `var paulo`.

Vamos ler do começo ao fim para entender. Então, criamos uma variável chamada `rafa` que se referencia a um objeto que, por sua vez, é uma estrutura com campos e cada campo tem seu valor. Cada campo ou atributo, ou, como vocês falaram, cada chave. Dentro de `rafa` tem `nome`, `vitorias`, `empates`, `derrotas` e `pontos`.

Não por coincidência, o objeto que `paulo` se referencia também tem os mesmos campos, mas com valores diferentes. Em seguida, criamos uma função que sabe calcular pontos dado um parâmetro. Esse parâmetro se chama `jogador`, parâmetro ou argumento são sinônimos neste caso.

Então pegamos o valor de vitórias multiplicamos por 3 e soma ao valor de empates desse jogador. Em seguida, não só guardamos o resultado dentro de pontos, mas damos um `return`.

"Return" é uma palavra-chave nova, é a primeira que estamos vendo nessa aula além da estrutura de objeto. E ele retorna, é muito diferente de só jogar no `console.log()`. Estamos falando "JavaScript retorna isso que eu vou te passar para a parte do código que me chamou". Não estou falando de console, de tela, não tem nada a ver. É muito diferente de fazer `console.log()` ou `innerHTML`, porque o código vai ter acesso a isso.

E embaixo estamos fazendo atribuição de variáveis. Começa pelo lado direito para guardar no lado esquerdo. Estou mandando calcular os pontos da jogadora rafa, faz aquela conta maluca, pega o retorno e, por sua vez, guarda dentro da chave pontos do objeto `rafa`.

É mais ou menos isso. Está válida minha explicação?

Guilherme: Sim. Está legal.

Paulo: Pelo que você já deu spoiler aqui, você quer criar uma lista com esses vários objetos. Para, provavelmente, iterar nesses objetos e ir alterando linha por linha, talvez adicionando linha por linha.

Guilherme: Exatamente. O que acontece aqui, Paulo, se compararmos com o problema que tínhamos, no HTML cada jogador representava todas essas listas aqui, várias tags `<td>` com essas informações.

Nesse caso, do lado do JavaScript, se eu quero criar um novo jogador é só vir aqui, nas linhas de `var rafa` e `var paulo`, fazer uma cópia desse valor e passar as propriedades para essa nova variável, esse novo jogador.

Mas teríamos o seguinte problema, se eu crio um *inner*, que sabemos que é pegar algo lá do HTML, colocar dentro dele, só para `rafa` e depois só para `paulo`, só mudamos o problema de lugar, pois continuamos com o mesmo problema.

Então eu vou criar uma lista, vou chamá-la de "jogadores". E esse `var jogadores` vai ser uma lista com todos os jogadores que eu tenho. Nesse caso vai ser a `rafa` e o `paulo`.

```
var jogadores = [rafa, paulo]
```

Guilherme: Se eu tenho uma lista, o que eu posso fazer? Lembra que minimizamos a quantidade de `document.write` que tínhamos lá porque usamos um *for*? Quero fazer a mesma coisa agora, só que no lugar de uma lista simples eu tenho uma lista com alguns objetos. E podemos até usar o `console.log()` para ver como vai ficar. Antes estávamos exibindo os dois objetos separadamente e agora com um `console.log(jogadores)` só dessa minha lista jogadores, observe como ela vai ser exibida na tela. Vou executar e o console exibe:

```
// [object Array] (2)
{
  "nome": "Rafa",
  "vitorias": 2,
  "empates": 1,
  "derrotas": 1,
  "pontos": 7
}
```

```
}  
  
{  
  "nome": "Paulo",  
  "vitorias": 1,  
  "empates": 1,  
  "derrotas": 2,  
  "pontos": 4  
}
```

Guilherme: No console apareceu colchetes ([]) e até informa que temos 2 objetos aqui dentro, [object Array] [2]. Se eu quiser criar mais um jogador, eu o coloco na lista e depois faço um monte de coisa "mágica" para todos os jogadores de uma só vez. A ideia principal é essa.

Então vamos partir para a nossa função principal. Queremos exibir todas essas informações na tela da Tabela de Classificação.

A primeira coisa a fazer é criar uma variável chamada "exibeJogadoresNaTela". E o que essa função vai precisar receber? Apenas a nossa lista de jogadores.

```
function exibeJogadoresNaTela(jogadores) {  
  
}
```

Guilherme: Eu vou apagar esse `console.log(jogadores)`, não vamos mais usá-lo porque já sabemos que recebemos os dois objetos dessa lista.

E com essa nova função eu quero criar um grande elemento que contenha a linha de cada jogador que eu tenho dentro dessa lista. Então vou criar uma variável chamada "elemento". E vou começar esse elemento com uma string vazia, porque vamos montar essa estrutura. Faremos em etapas para que a ideia principal fique mais clara.

```
function exibeJogadoresNaTela(jogadores) {  
  var elemento = ""  
  
}
```

Guilherme: Lá no HTML, só para lembrarmos, cada jogador vai precisar ter todas essas linhas de `<td>`. Então cada jogador é uma `<tr>` nova e vem até onde fecha a `<tr>`. Eu vou copiar toda essa estrutura. Vai ficar um código muito maluco, eu copiei tudo aquilo e vou colocar aqui abaixo de `var elemento`.

```
function exibeJogadoresNatela(jogadores) {  
  var elemento = ""  
  <tr>  
    <td>Paulo</td>  
    <td>2</td>  
    <td>5</td>  
    <td>1</td>  
    <td>24</td>  
    <td><button onClick="adicionarVitoria()">Vitória</button></td>  
    <td><button onClick="adicionarEmpate()">Empate</button></td>  
    <td><button onClick="adicionarDerrota()">Derrota</button></td>  
  </tr>  
}
```

Guilherme: Todos os meus jogadores, sem exceção, vão precisar dessa estrutura. Eu vou colocar cada uma dessas linhas dentro da nossa variável `elemento`. Eu quero pegar o conteúdo que já tenho na minha variável `elemento`, que é uma string vazia, e falar: "pega essa string vazia e coloca também essa `<tr>`". Vou colocar um sinal de mais e igual, e ele vai colocar esse conteúdo dentro da nossa variável `elemento`.

```
function exibeJogadoresNatela(jogadores) {  
  var elemento = ""  
  elemento += <tr>  
    <td>Paulo</td>  
    <td>2</td>  
    <td>5</td>  
    <td>1</td>  
    <td>24</td>  
    <td><button onClick="adicionarVitoria()">Vitória</button></td>  
    <td><button onClick="adicionarEmpate()">Empate</button></td>  
    <td><button onClick="adicionarDerrota()">Derrota</button></td>  
  </tr>  
}
```

Guilherme: Dentro dessa `<tr>` temos o nome do jogador que vamos receber, mas lembra que já vamos receber a nossa lista jogadores nessa função? Os jogadores serão referenciados por conta de seu índice na lista, no índice 0 vai trazer informações da Rafa, no índice 1 informações do Paulo.

Então para conseguirmos pegar linha a linha de cada um dos jogadores, vou incluir todas essas linhas dentro de um `for`. Para pegar todas as informações e no lugar do nome "Paulo" ou do nome "Rafa" colocar o nome correto do jogador.

Vou criar um `for`, ele será formado pelo parênteses e depois as chaves, dentro desse `for` vamos colocar uma variável `i` que vai ser nossa inicialização, vou começar com o valor `0`. Vou colocar também até quando essa nossa estrutura vai ser executada. Será executada enquanto tiver jogador. Então enquanto `i` for menor que `jogadores.length` queremos que essa estrutura seja executada. E para finalizar vamos fazer aquela ação final de incrementar para passar para o próximo jogador. Vou inserir essa estrutura dentro do `for` e o `elemento +=` vai se repetir em todas as linhas dessa estrutura.

```
function exibeJogadoresNatela(jogadores) {
  var elemento = ""
  for (var i = 0; i < jogadores.length; i++) {
    elemento += "<tr><td>Paulo</td>"
    elemento += <td>2</td>
    elemento += <td>5</td>
    elemento += <td>1</td>
    elemento += <td>24</td>
    elemento += <td><button onClick="adicionarVitoria()">Vitória</button></td>
    elemento += <td><button onClick="adicionarEmpate()">Empate</button></td>
    elemento += <td><button onClick="adicionarDerrota()">Derrota</button></td>
    elemento += </tr>
  }
}
```

Paulo: Só me assustei um pouco com esse sinal mais e igual (`+=`), esse sinal é para não ficarmos repetindo que ele é igual ao que já tinha antes mais, é só uma forma de falar que ele soma nele mesmo. É uma sintaxe diferenciada um pouco mais curta. Mas quem quiser pode escrever `elemento = elemento + <tr><td>Paulo</td>`, por exemplo.

E está faltando as aspas aí, não é? Isso não vai funcionar, certo?

Guilherme: Não, ainda não.

Paulo: No JavaScript, não podemos misturar HTML e JavaScript dessa forma. Tudo o que temos de HTML através do nosso código tem que ser tratado como texto, precisa estar entre aspas. Você ainda vai mudar isso, ainda não está pronto.

Guilherme: Ainda não. Agora, note que nessa estrutura temos o nome Paulo. Não é o nome "Paulo" que queremos mostrar agora. Então vou fechar as aspas, colocar dois sinais de mais e vai ter alguma coisa aqui dentro, do lado do JavaScript, que queremos concatenar com essa estrutura.

```
elemento += "<tr><td>" + + "</td>"
```

Guilherme: E aqui eu quero exibir o nome do jogador no índice em que ele estiver no índice `i`. Então vou pegar nossa lista de jogadores e colocar no índice `i`, `jogadores[i]`. O que isso significa?

Vamos fazer um teste de mesa, vamos fazer pensando de forma manual. A Rafa está no índice 0. Então a primeira vez que esse código rodar ele vai fazer assim: a variável começa no índice 0, o índice 0 é menor que o tamanho da lista `jogadores`, que tem 2 jogadores? Sim, então continua. Ele vai criar esse primeiro elemento e reconhecer que o jogador no índice 0 é a Rafa.

Mas o que eu quero mostrar desse objeto `rafa` que está sendo trabalhado aqui? Quero mostrar o nome dele. Então coloco `jogadores[i].nome`.

```
elemento += "<tr><td>" + jogadores[i].nome + "</td>"
```

Guilherme: Dessa forma ele vai exibir o nome da Rafa. Depois é a mesma ideia para o valor de vitórias, `jogadores[i].vitorias`. E a mesma coisa para empates, `jogadores[i].empates`.

Paulo: Tentar ler essa sintaxe às vezes pode ser um pouco incômodo. Porque tem o `for`, depois colchetes passando `i`, e antes de terminar ainda tem um ponto. Mas se não fizer isso vem aquele bloco cheio de informação que vimos no console. Não queremos tudo aquilo. Queremos só o valor de uma das chaves, o valor de um dos campos do objeto. Por isso estamos inserindo ponto seguido dos campos que queremos.

Paulo: Por isso estamos fazendo ponto em cada um dos jogadores `i`, tá bem?

Guilherme: É isso aí. Então eu vou fazer os pontos primeiro. Esses botões que colocamos aqui no código Javascript são exatamente os mesmos que colocamos no HTML. Nesse primeiro momento, ainda não vamos trabalhar com esses valores, então vamos deixá-los sem qualquer ação. Como queremos que eles estejam visíveis na tela, vou colocá-los entre aspas.

```
function exibeJogadoresNaTela(jogadores) {
  var elemento = "";
  for (var i = 0; i < jogadores.length; i++) {
    elemento += "<tr><td>" + jogadores[i].nome + "</td>";
    elemento += "<td>" + jogadores[i].vitorias + "</td>";
    elemento += "<td>" + jogadores[i].empates + "</td>";
    elemento += "<td>" + jogadores[i].derrotas + "</td>";
    elemento += "<td>" + jogadores[i].pontos + "</td>";
    elemento +=
      "<td><button onClick='adicionarVitoria()'>Vitória</button></td>";
    elemento +=
      "<td><button onClick='adicionarEmpate()'>Empate</button></td>";
    elemento +=
      "<td><button onClick='adicionarDerrota()'>Derrota</button></td>";
    elemento += "</tr>";
  }
}
```

Guilherme: Nós fechamos aspas no `onClick="`, colocamos a chamada de uma função, como `adicionarVitoria()`, e depois abrimos aspas novamente. Isso pode parecer estranho - nós temos aspas dentro de aspas, e o Javascript não entende isso. Para solucionar esse problema, vamos utilizar as aspas simples dentro das aspas duplas. Feito isso, até mesmo a cor do `adicionarVitoria()` vai ser alterada, pois o Javascript vai entender que se trata de um texto.

```
function exibeJogadoresNaTela(jogadores) {
  var elemento = "";
  for (var i = 0; i < jogadores.length; i++) {
    elemento += "<tr><td>" + jogadores[i].nome + "</td>";
    elemento += "<td>" + jogadores[i].vitorias + "</td>";
    elemento += "<td>" + jogadores[i].empates + "</td>";
    elemento += "<td>" + jogadores[i].derrotas + "</td>";
    elemento += "<td>" + jogadores[i].pontos + "</td>";
    elemento +=
```

```

" <td><button onClick='adicionarVitoria()>Vitória</button></td>";
    elemento +=
" <td><button onClick='adicionarEmpate()>Empate</button></td>";
    elemento +=
" <td><button onClick='adicionarDerrota()>Derrota</button></td>";
    elemento += "</tr>";
}

```

Paulo: Para o HTML meio que tanto faz usar aspas simples ou duplas, mas no Javascript não podemos ter aspas duplas dentro de aspas duplas - a não com alguns truques que não convêm agora.

Guilherme: Já escrevemos bastante código e agora eu quero exibir essas informações na tela. No HTML temos um `id="tabelaJogadores"` referente à nossa tabela. Sendo assim, posso fazer algo que já estamos bem acostumados. No JS, criaremos uma variável `tabelaJogadores` recebendo `document.getElementById()`, passando como parâmetro a nossa id `tabelaJogadores`.

```

var tabelaJogadores = document.getElementById("tabelaJogadores");

```

Guilherme: E vamos fazer o `inner` para colocarmos essas informações dentro do elemento que trouxemos. Para isso, faremos com que `tabelaJogadores.innerHTML` receba o grande elemento que montamos de cada jogador.

```

var tabelaJogadores = document.getElementById("tabelaJogadores");
tabelaJogadores.innerHTML = elemento;

```

Guilherme: Fizemos a função `exibeJogadoresNaTela` e esperamos que ela receba uma lista jogadores, e montamos todo o elemento representando como será a representação da página na tela, mas ainda não executamos a nossa função - tanto que se salvamos e executarmos o código, não acontecerá nada na tela ou no console.

Portanto, precisamos chamar a função `exibeJogadoresNaTela()` passando a nossa lista `jogadores` como argumento.

```

exibeJogadoresNaTela(jogadores);

```


Guilherme: Após salvar e executar, conseguiremos visualizar os jogadores na tela. Isso ficou bem legal. Rafa, quer tocar o próximo passo?

Rafaella: Já estamos conseguindo adicionar todas as vitórias, empates e pontos, porém ele ainda está um pouco estático, já que estamos predefinindo no código quais são esses valores - exceto o cálculo de pontos, que está automático. Seria interessante utilizarmos os botões que já colocamos no design da página para adicionarmos essas vitórias, empates e derrotas. Assim, começaríamos com os valores zerados e adicionaríamos um a cada clique no botão.

Paulo: Está estático, mas já tem uma coisa legal: se alguém quiser adicionar mais um jogador, não precisa mexer em nada no código todo, basta colocar um objeto "guilherme" e adicioná-lo na variável `jogos`.

Guilherme: Vou criar aqui rapidinho para vermos se isso que o Paulo falou é verdade.

```
//      chave. valor.
var rafa = { nome: "Rafa", vitorias: 2, empates: 1, derrotas: 1, pontos: 0 };
var paulo = { nome: "Paulo", vitorias: 1, empates: 1, derrotas: 2, pontos: 0 };
var gui = { nome: "Gui", vitorias: 1, empates: 1, derrotas: 2, pontos: 0 };

function calculaPontos(jogador) {
  var pontos = jogador.vitorias * 3 + jogador.empates;
  return pontos;
}

rafa.pontos = calculaPontos(rafa);
paulo.pontos = calculaPontos(paulo);
gui.pontos = calculaPontos(gui);

var jogadores = [rafa, paulo, gui];
```

Guilherme: Após salvar e executar, meu nome também vai constar na lista. Muito bom. E ele calculou, eu estou com os mesmos pontos do Paulo, nós estamos empatados, ficou legal. Muito mais simples do que manter toda a estrutura do HTML manualmente.

Rafaella: Agora podemos começar a utilizar os botões que estão na tela. Vamos dar uma olhada e verificar o que tem no `onClick` do HTML quando criamos os botões. Lembrando que nosso HTML foi meio que "transferido" para o Javascript para que possamos exibir na tela por meio do próprio Javascript.

No JS, vamos verificar os botões que foram adicionados no elemento exibido na tela.

```
function exibeJogadoresNaTela(jogadores) {
  var elemento = "";
  for (var i = 0; i < jogadores.length; i++) {
    elemento += "<tr><td>" + jogadores[i].nome + "</td>";
    elemento += "<td>" + jogadores[i].vitorias + "</td>";
    elemento += "<td>" + jogadores[i].empates + "</td>";
    elemento += "<td>" + jogadores[i].derrotas + "</td>";
    elemento += "<td>" + jogadores[i].pontos + "</td>";
    elemento +=
      "<td><button onClick='adicionarVitoria()'>Vitória</button></td>";
    elemento +=
      "<td><button onClick='adicionarEmpate()'>Empate</button></td>";
    elemento +=
      "<td><button onClick='adicionarDerrota()'>Derrota</button></td>";
    elemento += "</tr>";
  }
}
```

Rafaella: Em cada linha do elemento temos três botões: `adicionarVitoria()`, `adicionarEmpate()` e `adicionarDerrota()`, que são executados no clique mas que ainda não escrevemos. Vamos começar definindo a função `adicionarVitoria()`, seguida das chaves dentre as quais escreveremos os blocos de código.

```
function adicionarVitoria() {
}
```

Paulo: Lembro que quando o Guilherme estava explicando os objetos ele comentou das chaves, mas nem sempre usamos para objetos, nós também usamos para declarar um bloco de código. Porém, quando temos `= {}`, estamos lidando com um objeto. Existem marcadores na sintaxe que são usados de maneiras diferentes dependendo do contexto. Os parênteses, por exemplo, nós usamos para receber parâmetros, mas também para indicar a ordem de uma conta.

Rafaella: Exato. Nós queremos pegar especificamente aquele o jogador da linha em que o clique foi feito. Sendo assim, precisamos passar essa informação para a função por meio do parênteses. Em `onClick='adicionarVitoria()'`, passaremos dentro do parênteses qual é o jogador que queremos calcular a vitória.

Da mesma forma que estávamos fazendo anteriormente, vamos concatenar, usando aspas duplas, o `+ i +` de modo a obter o mesmo jogador que estamos listando.

```
elemento += "<td><button onClick='adicionarVitoria(\" + i + \")>Vitória</button></td>";
```

Quando fazemos a estrutura de repetição, já estamos pegando as informações desse jogador `i`, e é exatamente isso que passaremos para a nossa função `adicionarVitoria()`.

Guilherme: Eu coloquei aspas simples para testar, mas elas não vão funcionar nesse caso, precisa ser aspas duplas.

Rafaella: Parece meio bizarro passar dentro do parênteses esse `" + i + "`, mas na verdade a única coisa que estamos passando é o próprio `i`. Dentro da função `adicionarVitoria()`, receberemos o índice `i` do jogador e criaremos uma variável `jogador`, que receberá a nossa lista `jogadores` no índice `i`. Dessa forma, conseguiremos descobrir qual é o objeto que estamos trabalhando.

```
function adicionarVitoria(i) {  
    var jogador = jogadores[i];  
  
}
```

Mas por que podemos utilizar esse `jogadores[i]` se não passamos essa lista como parâmetro? Nós precisamos dele para sabermos qual é o jogador que está sendo iterado dentro do `for`. Porém, a nossa lista `jogadores` está fora das funções, ela está no escopo global. Sendo assim, não precisamos passá-la para a função `adicionarVitoria()`, e é possível acessá-la normalmente.

Agora que já sabemos qual é o jogador `i` e guardamos essa informação na variável `jogador`, queremos aumentar a vitória em 1 ponto sempre que clicarmos no botão. Vamos acessar `jogador.vitorias` e incrementaremos com `++`, aumentando o valor dessa variável em 1.

```
function adicionarVitoria(i) {  
    var jogador = jogadores[i];  
    jogador.vitorias++;  
}
```

Paulo: Também funcionaria utilizar `jogador.vitorias++ = jogador.vitorias + 1`?

Rafella: Sim, só é mais comum fazer dessa forma. Abaixo, calcularemos os pontos novamente - afinal, nós adicionamos uma vitória e é necessário chamar a função `calculaPontos()` novamente para exibir o resultado na tela. Lembrando que essa função tem um retorno pontos que precisa ser atribuído a uma variável, nesse caso `jogador.pontos`.

```
function adicionarVitoria(i) {  
  var jogador = jogadores[i];  
  jogador.vitorias++;  
  jogador.pontos = calculaPontos(jogador);  
}
```

Rafaella: Adicionamos uma vitória, recalculamos os pontos e agora precisamos exibir o resultado na tela com a função `exibirJogadoresNaTela()`, recebendo a nossa lista `jogadores` como parâmetro.

```
function adicionarVitoria(i) {  
  var jogador = jogadores[i];  
  jogador.vitorias++;  
  jogador.pontos = calculaPontos(jogador);  
  exibeJogadoresNaTela(jogadores);  
}
```

Rafaella: Pode salvar e vamos rodar.

Guilherme: Quando eu clicar no botão "Vitória" do primeiro jogador, "Rafa", os valores em "Vitórias" e "Pontos" dessa linha devem ser alterados. Cliquei e... maravilha, deu tudo certo.

Rafaella: As vitórias que estavam com o valor 2 foram para 3, e os pontos que estavam com 7 foram para 10.

Guilherme: Vou colocar mais uma vitória para confirmar, e vai funcionar normalmente. Para os outros jogadores também vai funcionar, Rafa?

Rafaella: Vai funcionar também.

Guilherme: Vou testar aqui com o Paulo e comigo, e maravilha, funcionou para todos.

Paulo: Eu confesso que me perdi ali, de onde veio o `i` de `adicionarVitoria()`? Como ele sabe qual o jogador, qual a linha da tabela?

Rafaella: Lá em cima, na própria função que o Gui criou, `exibirJogadoresNaTela()`, estamos fazendo um `for`, rodando com o índice relacionado a cada uma das linhas.

Paulo: Ah, tá certo, não é mais vazio.

Rafaella: Exatamente, é o mesmo `i` que estamos usando para imprimir nome, vitórias, empates e assim por diante.

Guilherme: Uma sacada legal Rafa, partindo desse mesmo princípio, eu acho que podemos fazer o código do empate, certo? Ele vai ser muito parecido com essa estrutura. Vou adicionar o `+ i +` na linha do `onClick='adicionarEmpate()`.

```
elemento += "<td><button onClick='adicionarEmpate(" + i +  
<td><button></td>";
```

Guilherme: Só para lembrar, os pontos estão sendo calculados corretamente porque incluímos no elemento um botão com uma função `onClick` chamada `adicionarVitoria()`, e agora teremos a função `adicionarEmpate()`.

Nela teremos um comportamento parecido: vamos descobrir para qual jogador o valor será adicionado (`jogador = jogadores[i]`) e incrementaremos a variável `jogador.empates`. Em seguida, `jogador.pontos` receberá `calculaPontos(jogador)` para atualizarmos o valor, e mostraremos na tela com `exibeJogadoresNaTela(jogadores)`.

```
function adicionarEmpate(i) {  
  var jogador = jogadores[i];  
  jogador.empates++;  
  jogador.pontos = calculaPontos(jogador);  
  exibeJogadoresNaTela(jogadores);  
}
```

Guilherme: Será que é só isso?

Rafaella: Lembrando que o cálculo que estamos fazendo de multiplicar as vitórias por 3, por exemplo, estão todos em outra função. Por isso, quando adicionamos uma vitória em `adicionarVitoria()`, o cálculo dos pontos é feito corretamente. O número de empates é alterado da mesma forma.

Guilherme: Vou salvar e rodar novamente. Vou clicar em "Empate" na linha do "Gui", que está com 2 empates e 8 pontos. Ao clicar, passará a exibir 3 empates e 9 pontos. Certo.

Rafaella: Perfeito, e agora precisamos fazer as derrotas. Lembrando que as derrotas não somam nenhum ponto, então não precisamos recalcular. Passaremos o `+ i +` na linha do `onClick='adicionarDerrota()'` e criaremos a função `adicionarDerrota()` recebendo o `i` como argumento.

Iniciaremos a variável `jogador` recebendo a lista de jogadores no índice `i`, incrementaremos `jogador.derrotas`, aumentando o valor em 1, e exibiremos o resultado com `exibeJogadoresNaTela()`.

```
function adicionarDerrota(i) {  
  var jogador = jogadores[i];  
  jogador.derrotas++;  
  exibeJogadoresNaTela(jogadores);  
}
```

Paulo: Poderia recalcular os pontos, não é?

Guilherme: Poderia. Se você quiser, poderia até perder pontos por derrota. Vou executar aqui e testar os novos botões. Quando clico em "Derrota" na linha do "Gui", que está com 2 derrotas. Feito isso, o valor mudará para 3, sem alterar o número de pontos. Tudo certo

Rafaella: Perfeito. Então é isso, nós temos alguns desafios também para propor para vocês. Começando por uma coisa que me deixou muito incomodada: sempre que aumentamos o empate em alguém, não aumenta em outra pessoa. Seria interessante fazer essa lógica do empate também aumentar com os outros jogadores.

Outra coisa, quando você tem uma derrota os outros jogadores ganham uma vitória automaticamente, seria interessante.


Paulo: Como tem mais de dois jogadores, acho que seria legal pelo menos validar, verificar se a quantidade de empates está fazendo sentido, se a soma das vitórias está igual à soma das derrotas e assim por diante. Não faria sentido, por exemplo, uma pessoa ter 10 empates, outra ter 0 e outra 2.

Tem algumas regras que você pode fazer para validar essa lista de jogadores, é um exercício legal para treinar `for`, `if` e assim por diante. Você pode usar o console para mostrar esses resultados ou ainda usar o `innerHTML` e criar uma coluna nova mostrando "Válido" ou "Inválido".

Você também pode mostrar com um troféu quem é o campeão ou campeã, quem tem mais pontos naquele instante, alterando esse troféu conforme os pontos variam.

Guilherme: Eu tenho mais um desafio, veio uma ideia enquanto vocês estavam conversando. Nós vimos, com base nas aulas anteriores, que é possível trabalhar com imagens. E se colocássemos uma imagem para cada jogador também? É um exercício bacana.

E o que vocês acham de incluir um botão para zerar os pontos de todos os jogadores?

Rafaella: Muito legal. Outra coisa que dá para fazer por aqui é adicionar outro jogador por meio de um botão e um input, você coloca o nome e clica em "Adicionar jogador", criando uma linha nova (aquele  que a gente viu) com todas as informações zeradas. Assim você poderá adicionar quantas pessoas quiser, só incluindo o nome dela.

Paulo: Basicamente um PUSH na nossa lista, não é?

Rafaella: Exatamente.

Paulo: O desafio principal é você customizar essa tabela de classificação para o seu sistema preferido, para o seu jogo, seja League of Legends, Warcraft, Overwatch, truco, o que for, e que faça bastante sentido para você. Você pode levar isso muito longe, inclusive fazendo o jogo ser aqui dentro e gerando a tabela de classificação. O limite é a sua criatividade, e nós estamos esperando para ver o que vocês vão apresentar.

Hoje a gente misturou ainda mais o HTML com Javascript, o que às vezes gera alguma dificuldade. Tem muita coisa que dá para vocês praticarem usando Javascript puro, para quem tem mais medo dessas marcações de sintaxe. Para as próximas aulas vamos misturar ainda mais, traremos projetos maiores que vão envolver mais de uma aula e beiram uma situação real.

Realmente fica mais difícil, e é natural surgirem dúvidas, todo mundo passa por isso. Às vezes você vai precisar rever melhor o for, o objeto, o if, a variável, experimentar outras coisas, ver como fazer uma variável temporária e assim por diante.

Nas próximas aulas vamos nos aprofundar um pouco mais em relação a misturar essas coisas para criarmos projetos palpáveis, sem contar as últimas aulas, nas quais criaremos algo que você terá bastante orgulho de mostrar para outras pessoas, vai ser realmente a conclusão da nossa imersão.

Rafaella: Até mais, pessoal!

Paulo: Tchau, obrigado!