

Imersão Dev - Aula 07

Rafaella: Olá, Dev! Antes de começarmos, gostaria de dar os parabéns por vocês terem chegado até aqui. Eu sei que não é fácil tirar tempo para focar, estudar e desenvolver e só de você já ter feito isso, demonstra a sua força de vontade.

Ontem fizemos um projeto muito legal e eu vi nas redes sociais alguns projetos personalizados com outras cores, outras fontes e achei isso muito legal! Então se você quiser ver como essas pessoas tem estilizado os projetos, participe da nossa comunidade no Discord, porque está todo mundo postando o que tem feito por lá. Agora pegue o café, abra o CodePen e bora para a sétima aula.

Paulo: Olá! Estamos aqui em mais uma aula da imersão Dev com Rafaella e o Guilherme que vocês já conhecem muito bem e estamos indo para a sétima aula. Você já deve ter percebido que muito conteúdo que vimos nas aulas anteriores sempre usamos na aula atual e a partir de hoje isso vai começar a acontecer com mais frequência. Vamos, inclusive, começar a ter mais interação entre o JavaScript, o HTML e CSS.

Sabemos que é bastante informação, mas já queremos mostrar como essas linguagens se misturam! Faço agora um convite para você entrar no site e ver que estamos estruturados em sete escolas. Estamos usando bastante o Front End, que é um das escolas onde temos muita gente trabalhando, a Rafaella, por exemplo, está bem próxima dessa escola. Eu, por outro lado, estava mais próximo da escola de Back End, o Guilherme está mais próximo à escola de Back End, mas também tem um pé no Front e em desenvolvimento de jogos. O que quero dizer é que cada uma dessas escolas e cada uma das coisas que estamos aprendendo aqui envolvem outras áreas desde o design de um site até colocá-lo no ar.

Por essa razão na Alura temos um manifesto chamado **Dev em <T>** onde falamos desse fenômeno que não acontece apenas em tecnologia, em que as áreas se misturam. Neste manifesto temos diversos artigos falando dessa pluralidade e multidisciplinaridade que ajuda a cruzar informações para resolvermos problemas

complexos. Atualmente, quando você trabalha em uma empresa de tecnologia, você está o tempo inteiro resolvendo problemas complexos que misturam pessoas e disciplinas e áreas do conhecimento. Eu recomendo, é uma leitura fácil e agradável e além de você conhecer com profundidade sua área, você também vai conhecer essas outras. Vai ser interessante para a sua carreira.

A proposta da Alura é que você seja esse profissional, essa pessoa que é generalista mas tem sua especialidade. Batemos muito nessa tecla e você vai ver que dentro da Alura, dentro da nossa plataforma, é como enxergamos desenvolvimento de software. A partir dessa aula misturamos mais as disciplinas e ficará mais claro essa nossa abordagem, e o modo como encaramos o mercado de trabalho.

Agora vou deixar os universitários mostrarem o que faremos nessa aula.

Rafaella: Hoje vamos aprender a fazer um Super Trunfo! Neste jogo onde temos um baralho de um determinado tema e cada carta desse baralho possui uma lista de atributos com seus respectivos valores. Esses atributos podem ser, por exemplo, força, ataque, defesa ou magia. E temos que jogar com uma pessoa. Aliás, é possível jogar com mais de duas pessoas? Eu sempre joguei com duas pessoas.


Paulo: Eu acredito que sim.

Rafaella: Eu sempre joguei com duas. Mas funciona assim: escolhemos um desses atributos, pegamos uma carta do baralho e comparamos com o atributo da carta do nosso adversário. Aquele que tiver o atributo com o maior valor, vence e fica com a carta do outro.

Existe também a carta Super Trunfo que uma carta imbatível que vence de todas em todos os atributos, mas em nosso projeto vamos fazer de uma maneira um pouco mais simples ignorando essa carta.

Além disso, nós não vamos jogar contra uma outra pessoa, mas contra o computador. Sortearemos e criaremos três cartas. Para começar, vamos fazer um “fork” no projeto que está na conta da Alura. Estamos na conta do Gui, mas ele fará um fork do projeto que está na Imersão Dev Alura e vamos começar. Já estamos com o HTML e o CSS

prontos como de costume. Temos o `onClick()` temos o botão "Sortear Carta" e também o botão "Jogar", que agora está desabilitado. Vamos ver essas coisinhas aos poucos.

Nosso primeiro passo será criar as cartas. Mas não queremos criar, por exemplo, as cartas invariáveis diferentes, não queremos enviar carta 1, carta 2 e carta 3 e como nós já vimos conseguimos criar listas, arrays, quando queremos criar um conjunto de variáveis do mesmo tipo. Só que nesse caso, como vamos utilizar cartas e precisamos de atributos, precisamos do nome do personagem da carta. Vamos criar aquele objeto que também já vimos nas aulas anteriores que é .

Vamos criar uma lista com o mesmo formato que já vimos nos nossos arrays com colchetes, `[]`. Então, Gui, vamos criar a nossa variável `var cartas` e nela criamos o array com o colchetes.

Guilherme: Rafa, antes eu vou só colocar as tags da imersão, que serão "supertrunfo, imersãodev, alura". Quem quiser, pode colocar outras tags também. Salvei! Agora vamos lá!

```
var cartas = []
```

Rafaella: Dentro desses colchetes colocamos cada um dos objetos que vamos criar. Então damos um "Enter" e vamos colocar os nossos objetos um em cada espaço diferente.

Paulo: Rafa, você está querendo colocar vários objetos dentro de uma lista, de uma array, é isso?

Rafaella: Isso.

Paulo: Posso pedir para vocês dois universitários irem um pouco mais devagar e primeiro criarem uma ou três cartas separadas e ali dentro nós colocamos as cartas?

Rafaella: Vamos, então.

Paulo: Eu to pedindo para a Rafa e para o Guilherme porque eles já iam fazer de uma maneira muito comum no dia a dia da programação que é dentro do colchetes. Como ela quer colocar várias cartas ali dentro e cada carta por sua vez vai ter várias informações, acho melhor que nesse momento criarmos as cartas separadas para facilitar a visualização.

Rafaella: Perfeito.

Paulo: Só para ficar visível. Mas vocês viram que a Rafa já foi no modo de programação normal. Porém, logo de primeira pode parecer estranho. Depois eu peço para a Rafa também fazer do jeito *hardcore*.

Rafaella: Perfeito, beleza. Vamos criar então três cartas, acho que vai ser legal cada um de nós escolher um personagem. Eu vou escolher o primeiro personagem e será o Bulbassauo, que é do Pokemon. Então vamos dar um enter para ficar um pouco mais claro. E vamos criar a nossa primeira chave valor que será o nome do personagem. Então colocamos o `nome` e entre aspas `bulbassauo`.

Além do nome, também vamos adicionar todos os atributos, eles serão ataque, defesa e magia e serão englobados em `atributos` e criados separadamente dentro dele como `ataque`, `defesa` e `magia`. Seus valores são `7`, `8` e `6`, respectivamente.

```
var carta1 = {  
  nome: "Bulbassauo",  
  atributos: {  
    ataque: 7,  
    defesa: 8,  
    magia: 6  
  }  
};
```

Rafaella: Perfeito. Já temos nossa primeira carta para podermos jogar no Super Trunfo e agora vamos criar as outras duas chaves também. Paulo, qual personagem você quer que participe aqui no Super Trunfo.

Paulo: Rafa, antes deixa eu perguntar uma coisa. Isso aí é meio como se tivesse um objeto dentro de um objeto? Na verdade é uma referência, né? Esse atributos são como um objeto dentro do objeto carta, né?

Rafaella: Exato.

Paulo: Sabemos que não é exatamente isso, mas é isso. Então agora se eu quisesse pegar o ataque da primeira carta para mostrar na tela, eu só quero mostrar o ataque da primeira carta, com eu acesso, faço `console.log()` em que? Se eu fizer só `carta1` vai aparecer o bloco inteiro. Se eu tentar `carta1.ataque`, funciona?

```
console.log(cartas[0].ataque);
```

Guilherme: Vamos ver? Vamos executar e aparece `undefined`.

Paulo: É, não é assim.

Rafaella: Não é assim. Vamos tentar então o `atributos` em vez de `ataque`.

```
console.log(cartas[0].atributos);
```

Guilherme: Salvei e rodei. E aparece todos os atributos.

```
// [object Object]
{
  "ataque": 7
  "defesa": 8
  "magia": 6
}
```

Paulo: Não era o que eu queria, mas está mais próximo.

Rafaella: Agora põe um `.ataque` depois de `atributos`.

```
console.log(carta1.atributos.ataque);
```

Guilherme: Salvei, rodei mais uma vez. E conseguimos! Aparece `7`.

Paulo: Entendi! Usar dois `.`. É algo que não tínhamos visto dessa forma. Toda vez que damos o ponto vamos navegando neste monte de “flechinhas” que um objeto pode ter para outro objeto chamados grafos de referências. Para mim está ótimo.

Rafaella: Perfeito, exatamente. Então esse foi o formato que usamos para criar nossa primeira carta, agora vamos replicá-lo para criar as próximas duas cartas. Paulo, qual o personagem que você queria colocar no baralho?

Paulo: Rafa, deixa o Guilherme responder antes, eu estou pensando aqui eu queria lembrar de um personagem muito incrível, estou aqui nessa missão. Eu sabia que essa aula era sobre isso, mas eu não me preparei para as escolhas.

Guilherme: Certo, eu vou pensar aqui numa carta, vou colocar um personagem que eu gosto muito que é o Darth Vader, tá bom? Vou colocar o ataque dele muito forte com `9` a defesa também bem forte com `8` e a magia bem fraquinha com `2`.

```
var carta2 = {  
  nome: "Darth Vader",  
  atributos: {  
    ataque: 9,  
    defesa: 8,  
    magia: 2  
  }  
};
```

Guilherme: E enquanto o Paulo vai pensando vou dar “Ctrl C + Ctrl V” de toda a estrutura e ir fazendo os atributos da **carta3**. Posso colocar aleatório, Paulo?

Paulo: Não, coloca a magia alta. Magia e defesa altas e ataque médio. Eu vou escolher de novo o Shiryu de dragão, eu ia escolher uma menina porque ninguém colocou uma personagem feminina, mas vai o Shiryu de dragão.

Guilherme: Paulo, eu não faço ideia de como se escreve isso.

Paulo: S-h-i-r-y-u. Não teve infância não, né?

Guilherme: Não tive, esse é muito específico.

Paulo: Não é muito específico não, tem 200 episódios, minhas filhas de 5 anos assistiram todos.

Guilherme: É, eu não faço ideia do que seja isso.

```
var carta1 = {  
  nome: "Bulbassau",  
  atributos: {  
    ataque: 7,  
    defesa: 8,  
    magia: 6  
  }  
};
```

```
var carta2 = {  
  nome: "Darth Vader",  
  atributos: {  
    ataque: 9,  
    defesa: 8,  
    magia: 2  
  }  
};
```

```
    }  
};  
  
var carta3 = {  
    nome: "Shiryu de dragão",  
    atributos: {  
        ataque: 5,  
        defesa: 9,  
        magia: 10  
    }  
};
```

Guilherme: Já colocamos aqui as pontuações para as três cartas. Aqui temos algo interessante, caso a pessoa que está nos assistindo queira criar mais cartas, ela pode?

Rafaella: Pode, com certeza. Vocês vão criar com os personagens que vocês quiserem, gente. E também os atributos se vocês acharem mais legais.

Paulo: Para o pessoal entender como eles vão chegar nas cartas, eu vou pedir aqui para a edição colocar bem grandão na imagem como vai ficar esse nosso projeto e as cartas lindas de trunfo para as pessoas terem ideia de como dá para ficar.

Guilhermes: Temos com temas de carro, da Disney, de super heróis, de cobras...

Rafaella: Tem de tudo.

Guilherme: Pessoal, pode usar a criatividade.

Paulo: Vocês pode fazer os personagens favoritos do Aluraverso. Temos nós, tem o Jovem Nerd, tem o Leon e a Nilce, tem o Guga Mafra, tem o pessoal do Manual do Mundo, tem o Átila e a Camila do Nerdologia, tem um monte de gente aí envolvida. Tem o Mario, o Davi Coutinho, vocês podem fazer com seu professor preferidos e fazer as cartas com os atributos como linguagens de programação!

Rafaella: Nossa, legal.

Guilherme: Um ponto que seria interessante de fazermos é o seguinte: temos três cartas aqui e que no Mentalista ele escolhia um número aleatório de 0 a 10, certo? Nós poderíamos criar duas variáveis: uma carta da máquina e uma carta do jogador ou da jogadora e pedir para que o próprio computador sortearse as cartas que estariam em jogo. Vamos fazer isso?

Rafaella: Vamos.

Guilherme: Para começar, eu vou guardar todas essas nossas cartas dentro de uma lista. Vou chamar uma nova variável `cartas` e vou falar que ela vai ser uma lista com todas as nossas cartas.

```
var cartas = [carta1, carta2, carta3];
```

Guilherme: E se exibirmos no console essa nossa lista de cartas, vamos ver como ele vai aparecer.

```
console.log(cartas);
```

Guilherme: Ele traz todas as cartas. Ele fala que é um array com três objetos.

```
// [object Array] (3)
[// [object Object]
{
  "nome": "Bulbasauo",
  "atributos":{
    "ataque": 7,
    "defesa": 8,
    "magia": 6
  }
}
```

```
}, // [object Object]
{
  "nome": "Darth Vader",
  "atributos": {
    "ataque": 9,
    "defesa": 8,
    "magia": 2
  }
}, // [object Object]
{
  "nome": "Shiryu de dragação",
  "atributos": {
    "ataque": 5,
    "defesa": 9,
    "magia": 10
  }
}
}]
```

Guilherme: Então temos o Bulbassauo, o Darth Vader e o Shiryu de dragão, que eu não tenho a menor ideia do que seja, mas vou pesquisar depois.

Paulo: Para vocês terem ideia, a Rafa e o Guilherme iam fazer de uma forma que iam pular esse `carta1`, `carta2` e `carta3`, eles iam copiar e colar aquele “bloco” exatamente onde está escrito dentro da array. Então essas variáveis que chamamos de temporárias não existiriam, ele iria direto colocar o objeto aqui, o que realmente é comum.

Se fossemos inicializar algumas variáveis, e está meio claro o que estamos fazendo isso, não teríamos esse passo a passo tão estruturado. Fizemos dessa maneira para ficar mais claro cada etapa.

Guilherme: É isso aí! Então, já temos nosso baralho. O que vou fazer agora é criar uma função que vai sortear essas cartas para mim, vou criar uma sequência de código que no final vai sortear as cartas do jogo por questão de legibilidade e de compreensão do nosso código.

Para manter nosso código organizado, criarei mais duas variáveis aqui fora que vou chamar de `var cartaMaquina` que será a carta da máquina e para ela vou passar valor nenhum, vou passar com valor de `0`. E criar também um `var cartaJogador` e um valor de `0` também.

```
var cartaMaquina = 0  
var cartaJogador = 0
```

Guilherme: Preciso passar esse valor de zero na hora que estou criando essa nossa carta? Não, eu poderia deixá-lo sem nada. O que significa? Significa que dentro da memória do nosso computador vai existir um espacinho onde vamos guardar alguma coisa que vamos apontar e chamará de `cartaMaquina` e `cartaJogador`.

```
var cartas = [carta1, carta2, carta3];  
var cartaMaquina  
var cartaJogador
```

Rafaella: Por que estamos fazendo isso de inicializar ela e depois dar o valor? É aquela história do escopo, quando criamos a variável fora de cada uma das funções, ela vai poder ser acessível em várias outras funções, então é justamente por isso que estamos criando ela aí fora em vez de colocar dentro da função.

Guilherme: Excelente comentário. Vou criar a nossa função com o nome de `sortearCarta()`.

```
function sortearCarta()
```

Rafaella: Nós temos, né, Gui, o botão de sortear? Temos lá o método, né?

Guilherme: Tem! Nossa, melhor ainda. Olha só, no nosso projeto tem um botão de "Sortear carta", vou salvar o projeto e rodar mais uma vez e clicar nesse botão. Vou tirar

a linha da função `sortearCarta()` e salvar o projeto. E lá no nosso HTML temos o botão `onClick()` chamado `sortearCarta()`. Quando eu clico no botão "Sortear carat" ele aponta que algo do JavaScript não está certo e no console ele fala que o `sortearCarta is not defined`, não está definido. Ou seja, queremos de fato sortear as cartas do nosso jogo quando nós clicarmos neste botão. Então, vou dar o nome da nossa função que por coincidência está com o mesmo nome.

```
function sortearCarta() {}
```

Guilherme: Vou salvar e executar mais uma vez e quando eu clicar no botão é uma função, ele entende que é uma função e não dará mais erro. Mas não acontece nada porque a nossa função de `sortearCarta()` não faz nada! E não é esse comportamento que queremos. Então vamos adicionar o comportamento.

Quero sortear uma carta para a máquina e uma carta para o jogador nessa nossa função. A primeira coisa que temos que ter em mente são as possibilidades, as formas que podemos pensar para resolver esse problema. Bom, temos uma lista de cartas e temos aqui três cartas dentro dessa nossa lista. Então se eu coloco aqui `console.log()` vamos supor que a carta da máquina vai ser a carta do índice `0` e quando eu apertar nesse "Sortear carta" ele vai mostrar a carta da máquina no índice `0`.

```
// [object Object]
{
  "nome": "Bulbasaur",
  "atributos": {
    "ataque": 7,
    "defesa": 8,
    "magia": 6
  }
}
```

Guilherme: Deu certo. Se eu colocar aqui o valor `1` e valor `2` e atribuir isso para a carta da máquina eu posso falar assim que a carta da máquina vai ser igual a essa carta `0`.

```
function sortearCarta() {  
    cartaMaquina = ccarta[0];  
}
```

Guilherme: Dessa forma, nosso jogo não teria muita graça, porque todas as vezes que fossemos jogar a carta da máquina sortearia o Bulbassauo e isso deixaria o jogo totalmente óbvio. Eu quero que essa escolha da carta da máquina aqui o índice seja escolhido de forma aleatória.

O que eu quero fazer? Quero escolher um número aleatório de 0 a 2. Já sabemos que existe uma função chamada `Math.random()` que vai fazer isso para nós.

```
Math.random * 3
```

Guilherme: Vamos executar essa função multiplicada por três no console para lembrarmos e não ficarmos com nenhuma dúvida. Quando dou esse `Math.random * 3` ele dá um `NaN`, porque isso é uma função, eu tenho que chamar o `()`.

```
Math.random() * 3  
  
2.586780573130609
```

Guilherme: Dá aquele número muito doido. Mas eu não preciso dessa parte decimal, o que eu posso fazer? Eu tenho duas opções aqui, o que vocês sugerem? Eu tenho uma em mente.

Rafaella: Eu tenho aquela opção para transformarmos em inteiro.

Guilherme: Boa. Vamos fazer isso então. Vamos transformar em um `parseInt()` e englobar ele todo.

```
parseInt(Math.random() * 3)
```

Guilherme: Agora ao dar enter temos `0, 2, 2, 2, 0, 1` e assim por diante.

Rafaella: Exatamente os números do array, né?

Paulo: Porque não queremos o 3, né? Porque seria a quarta carta.

Rafaella: Exato.

Guilherme: Então vou copiar toda essa estrutura e falar que isso será o índice da `cartaMaquina`. Para ficar mais fácil vou colocar a variável `numeroCartaMaquina` e falar que vai ser igual a toda essa nossa estrutura. E o índice da `cartaMaquina` vai ser `numeroCartaMaquina`.

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3)  
  
    cartaMaquina = cartas[numeroCartaMaquina];  
}
```

Guilherme: Vamos colocar um `console.log()` exibindo a `cartaMaquina`. Salvaremos, limparemos todo o Console e o código de `parseInt()` que estávamos executando, e rodaremos.

```
//código anterior omitido  
  
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);  
}
```

Guilherme: Quando sortearmos a carta clicando no botão a primeira vez, o Console irá mostrar uma carta, que neste caso foi a "Shiryu de dragão" do Paulo.

Se clicarmos no botão "Sortear carta" novamente, ele está habilitado. Como a Rafa comentou no início, não conseguiremos clicar no botão de "Jogar". Mas conseguimos clicar no botão de sortear várias vezes e exibir várias cartas sorteadas diferentes no Console.

Agora temos um outro problema; estamos sorteando a carta da máquina, mas não estamos conseguindo sortear a carta do jogador, e temos que fazer isso.

Se copiarmos toda a estrutura da variável `numeroCartaMaquina`, colarmos em seguida - que é um método bem comum mas não recomendamos pois devemos ter sempre muito cuidado -, e alterarmos o nome para `numeroCartaJogador`, falaremos que também será um `parseInt()` vezes três, terá a variável `cartaJogador` que criamos acima ao invés de `cartaMaquina`, e falaremos que esta carta terá o mesmo comportamento.

Ainda, no lugar de chamarmos o `numeroCartaMaquina` em `cartas[]`, será o `numeroCartaJogador`, e o `console.log()` exibirá a `cartaJogador`. Depois vamos salvar e rodar o projeto.

```
//código anterior omitido
```

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);  
  
    var numeroCartaJogador = parseInt(Math.random() * 3);  
    cartaJogador = cartas[numeroCartaJogador];  
    console.log(cartaJogador);  
}
```

Guilherme: Temos o `console.log()` das duas cartas, da máquina e do jogador, e clicando sobre o botão de sorteio, teremos um cenário interessante em que exibimos as informações das duas cartas sorteadas.

Podemos clicar várias vezes para vermos os resultados. Acontece que um número de zero a dois do índice foi sorteado, e quando a mesma carta é mostrada para a máquina e o jogador, teremos um empate.

Mas isso não é um comportamento que queremos, pois queremos que escolha um número para a máquina e outro para o jogador. Porém, caso este índice seja igual em ambas as cartas, queremos que faça outro sorteio até que o número seja diferente, ao invés de empatar. Assim garantiremos que a carta da máquina seja uma e a do jogador seja outra.

Rafaella: Quando falamos de condição, ou seja, "se isso acontecer...", lembramos sempre do `if ()`. Então poderíamos colocar, por exemplo, "se o `numeroCartaJogador` for igual ao `numeroCartaMaquina`, executaremos a função `sortearCarta()` novamente.

Guilherme: Faz sentido, mas o `if ()` tem um ponto importante, ele verifica uma vez só.

Então criaremos o `if ()` quando sortearmos a carta do jogador, dizendo que, se o `numeroCartaMaquina` entre os parênteses que indica a condição executada for igual usando `==` ao `numeroCartaJogador`, queremos refazer o comportamento da estrutura da variável `numeroCartaJogador` para sortear novamente.

//código anterior omitido

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);  
  
    var numeroCartaJogador = parseInt(Math.random() * 3);  
    if (numeroCartaMaquina == numeroCartaJogador) {  
        numeroCartaJogador = parseInt(Math.random() * 3);  
    }  
    cartaJogador = cartas[numeroCartaJogador];  
    console.log(cartaJogador);  
}
```

Guilherme: Porém, isso não garante que as cartas sejam diferentes se o sorteio acontecer de novo e for o mesmo número para as duas.

Rafaella: Poderíamos chamar a função novamente, mas de fato teríamos que refazer muita coisa só para isso.

Guilherme: Certo. Então temos um problema na nossa estrutura, pois o `if ()` faz o sorteio de novo mais uma vez se o índice for igual para as duas cartas, mas ainda não garantimos que este número sorteado seja diferente dessa próxima vez, e pode passar pela condição mesmo sendo igual.

Existe uma estrutura na programação que garantirá a execução do sorteio enquanto o número não for diferente, chamada `while ()`, que é a tradução em inglês de "enquanto".

Então diremos que, enquanto os índices das cartas da máquina e do jogador forem iguais, iremos sortear quantas vezes forem necessárias até que os números seja

diferentes, ou seja, somente quando a repetição `numeroCartaMaquina == numeroCartaJogador` for falsa, sairemos do `while ()` e este não será mais executado.

//código anterior omitido

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);  
  
    var numeroCartaJogador = parseInt(Math.random() * 3);  
    while (numeroCartaMaquina == numeroCartaJogador) {  
        numeroCartaJogador = parseInt(Math.random() * 3);  
    }  
    cartaJogador = cartas[numeroCartaJogador];  
    console.log(cartaJogador);  
}
```

Guilherme: Salvando, executando o projeto, limpando o console e sorteando novamente, teremos duas cartas diferentes. Testaremos clicando no botão mais uma vez para vermos se ainda aparecem cartas iguais.

Logo, conseguimos garantir que sempre teremos um sorteio de cartas diferentes para a máquina e para o jogador ou jogadora.

Paulo: Observando o código da função `sortearCarta()`, sorteamos o número da carta da máquina, e na hora que sorteamos a do jogador, vimos que, se forem iguais, continuamos a usar a estrutura de laço `while ()`.

Lembra o `for ()` de alguma forma, mas somente com seu miolo com a parte que faz o `if ()` para decidir. As sintaxes são idênticas, mas o `if ()` apenas executa uma vez de novo, e o `while ()` só executa enquanto a condição de empate dentro dos parênteses não for satisfeita pelo sorteio.

Há várias formas de resolvermos este problema, e esta é uma sugestão. Mas quando temos três cartas e o computador sorteia uma, poderíamos remover a carta sorteada e depois escolheríamos entre as que sobraram para o jogador ou jogadora.

Então uma outra possibilidade para evitarmos a necessidade do `while()` é remover esta carta de `cartas[]` antes de sortearmos a próxima. Na documentação da `array` com a sequência de objetos que criamos, veremos que existe um `.push()` e `.length()`, mas há outro elemento que nos permite fazer o sorteio novamente depois desta primeira remoção.

Porém, se fazemos o sorteio vezes três com `3`, agora faremos o vezes três menos um, ou seja, apenas vezes dois pois só sobraram duas cartas. Com isso, teremos certeza de que não iremos sortear a mesma carta para a máquina e para o jogador.

Existem várias ideias possíveis para resolver essa questão, como o uso do método que embaralha uma array chamado `shuffle()`. Aí entra a ideia do algoritmo, mas claro que possui algo mais rebuscado para resolver casos mais complexos do que o que temos agora.

O desafio de gerar o mesmo resultado escrevendo de maneiras diferentes é muito importante para buscar soluções eficientes.

Guilherme: Isso mesmo! Então temos várias formas de solucionar o mesmo problema, e isso é programação. Então vale muito a pena ver o que outras pessoas pensaram.

Aqui utilizamos o `while ()` e funcionou, mas a proposta de retirar a primeira carta sorteada antes de escolher a próxima para que não sejam iguais também é ótima.

Rafaella: Então agora já conseguimos sortear as cartas e exibir no Console. Porém, temos que ajustar os nossos botões, pois além de sortearmos o número e vermos nossos atributos, queremos jogar de fato, mas o botão "Jogar" ainda está desabilitado.

Quando fizermos o sorteio, iremos desabilitar o botão de "Sortear carta" que clicamos e habilitar o de "Jogar" ao mesmo tempo. O botão estar ou não habilitado é uma propriedade da própria *tag* dentro do HTML.

Abriremos a aba "HTML" para vermos o `<button>`. O `SortearCarta()` está habilitado, mas veremos mais adiante o de `Jogar()` que está desabilitado desde o começo para entendermos esta propriedade.

Dentro do `<form>`, encontraremos o `<button>` do botão "Jogar" com `type`, `id`, `onClick` e o `disabled` igual a `"true"`. Esta última propriedade é a tradução em português da palavra inglesa "desabilitar".

Se estamos passando o valor `"true"`, é porque realmente queremos que esteja desabilitado de fato. Agora precisaremos trocar por `"true"`.

De volta ao código JavaScript, pegaremos o `document.getElementById()` ao final da nossa função `sortearCarta()` quando as cartas acabarem de serem sorteadas.

Em seguida, pegaremos o `id` igual a `"btnSortear"` do botão "Sortear carta" antes de mais nada, e o selecionaremos em `document.getElementById()`. Depois, selecionaremos a propriedade `disabled` lembrando que, quando queremos pegar apenas o valor, usamos a propriedade `disabled`, e neste caso queremos pegar a que desabilita o botão, usaremos `.disabled`.

Passaremos o valor `true` para o `.disabled` do nosso botão "Sortear", e assim o desabilitaremos depois que tudo dentro da nossa função for executado. Vamos testar rodando o código e sorteando a carta novamente.

```
//código anterior omitido
```

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);  
  
    var numeroCartaJogador = parseInt(Math.random() * 3);  
    while (numeroCartaMaquina == numeroCartaJogador) {  
        numeroCartaJogador = parseInt(Math.random() * 3);  
    }  
    cartaJogador = cartas[numeroCartaJogador];  
    console.log(cartaJogador);  
  
    document.getElementById("btnSortear").disabled = true;  
}
```

Rafaella: Ao clicarmos em "Sortear carta" iremos sortear as cartas que serão exibidas no Console, e também veremos o botão ser desabilitado imediatamente impedindo que o acionemos de novo.

Agora habilitaremos o botão de "Jogar". Para isso, iremos ao HTML para encontrarmos seu `id`, o copiaremos e depois faremos outro `.getElementById()` recebendo o `"btnJogar"` desta vez.

Colocaremos o `.disabled` igual a `false` ao invés de `true` que tínhamos no HTML para habilitarmos o botão e jogarmos de fato. Salvaremos, limparemos o Console e rodaremos novamente para clicarmos no "Sortear carta" e no "Jogar" em seguida.

```
//código anterior omitido
```

```
function sortearCarta() {  
    var numeroCartaMaquina = parseInt(Math.random() * 3);  
    cartaMaquina = cartas[numeroCartaMaquina];  
    console.log(cartaMaquina);
```

```

    var numeroCartaJogador = parseInt(Math.random() * 3);
    while (numeroCartaMaquina == numeroCartaJogador) {
        numeroCartaJogador = parseInt(Math.random() * 3);
    }
    cartaJogador = cartas[numeroCartaJogador];
    console.log(cartaJogador);

    document.getElementById("btnSortear").disabled = true;
    document.getElementById("btnJogar").disabled = false;
}

```

Guilherme: Vamos sortear e ver as cartas que aparecem no Console. Depois, veremos que o botão "Jogar" está habilitado para clicarmos.

Rafaella: Perfeito! Então agora já poderemos jogar de fato. Mas antes disso, precisamos escolher um atributo de acordo com a orientação "Escolha o seu atributo" do nosso programa.

Abaixo deste enunciado, precisaremos exibir quais atributos escolheremos, se será a "magia", "ataque" ou "defesa". Quando clicarmos em "Jogar", saberemos realmente quem ganhou o jogo.

Portanto, criaremos uma função para exibir essas opções de atributos. Após a `function sortearCarta()`, pularemos uma linha e criaremos a `exibirOpcoes()`. Abriremos as chaves e, dentro delas, colocaremos o que queremos executar quando chamarmos a função.

Primeiramente, já temos uma tag no `<form>` do nosso HTML com o `class="opcoes"` cujo `id` é igual a `"opcoes"`, que será exatamente onde iremos colocar nossas opções para serem exibidas na tela.

Então, na função `exibirOpcoes()` aplicaremos novamente o `getElementById()` recebendo `"opcoes"` dentro de uma nova variável chamada `opcoes`.

```

//código anterior omitido

function sortearCarta() {
    var numeroCartaMaquina = parseInt(Math.random() * 3);
    cartaMaquina = cartas[numeroCartaMaquina];
    console.log(cartaMaquina);

    var numeroCartaJogador = parseInt(Math.random() * 3);
    while (numeroCartaMaquina == numeroCartaJogador) {

```

```

        numeroCartaJogador = parseInt(Math.random() * 3);
    }
    cartaJogador = cartas[numeroCartaJogador];
    console.log(cartaJogador);

    document.getElementById("btnSortear").disabled = true;
    document.getElementById("btnJogar").disabled = false;
}

function exibirOpcoes() {
    var opcoes = document.getElementById("opcoes")
}

```

Rafaella: Em seguida, percorreremos todos os atributos e os imprimiremos na tela. Iremos imprimir no Console primeiro para entendermos bem como percorremos esses atributos.

Relembrando, estávamos agrupando dentro de um objeto que é o valor da chave do objeto das cartas. No início do nosso arquivo JS, temos as variáveis `carta1`, `carta2` e `carta3`.

O `atributos` é uma chave cujo valor é um objeto com `ataque`, `defesa` e `magia`. Queremos percorrer cada um desses atributos e imprimir cada um desses valores.

Podemos utilizar várias outras formas, mas uma bem comum quando usamos uma lista de itens e queremos percorrer cada um deles é o `for`. Ele também tem um nome `forEach()` que podemos encontrar em outras linguagens.

Mas o `for ()` funciona executando algo para cada elemento dentro de uma lista, como uma tradução "para cada". Criaremos uma nova variável chamada `atributo` para chamar cada um desses itens dentro dos parênteses.

Em seguida, usaremos `in` para percorrer cada atributo da carta, ou seja, `in cartaJogador` chamando `.atributos` para imprimirmos novamente, como vimos no início da aula quando usamos o `console.log()` com `cartaJogador.atributos`.

Então esse é um outro formato de estrutura de repetição ou *loop* muito utilizado para listas, em que criamos uma nova variável para chamar cada um dos elementos, e usamos o `in` como tradução de "dentro de" ou "em" para chamar a lista.

Depois, abriremos as chaves para colocar o bloco de comandos que executaremos para cada um dos atributos, mas nesse caso apenas imprimiremos com `console.log()` recebendo o `atributo`.

É como se esta nova variável fosse o próprio índice. Antes imprimíamos o `console.log()` com o `i`, usávamos o `array` e os colchetes do índice, mas aqui poderemos utilizar o próprio elemento dentro desta lista para ser o que percorre.

Guilherme: Antes de salvarmos e rodarmos, vamos tirar a impressão da `cartaMaquina` e deixaremos somente a exibição da `cartaJogador` no Console para podermos escolher um atributo sem saber a carta da máquina.

//código anterior omitido

```
function sortearCarta() {
    var numeroCartaMaquina = parseInt(Math.random() * 3);
    cartaMaquina = cartas[numeroCartaMaquina];

    var numeroCartaJogador = parseInt(Math.random() * 3);
    while (numeroCartaMaquina == numeroCartaJogador) {
        numeroCartaJogador = parseInt(Math.random() * 3);
    }
    cartaJogador = cartas[numeroCartaJogador];
    console.log(cartaJogador);

    document.getElementById("btnSortear").disabled = true;
    document.getElementById("btnJogar").disabled = false;
}

function exibirOpcoes() {
    var opcoes = document.getElementById("opcoes");

    for (var atributo in cartaJogador.atributos) {
        console.log(atributo);
    }
}
```

Guilherme: Notaremos que o programa ainda não mostrou as opções de atributo para escolhermos em nenhum momento, nem no Console antes de apresentarmos na tela.

Rafaella: Não exibimos as opções porque apenas criamos o que queremos que seja executado quando a função é chamada, mas ainda não a chamamos de fato.

Chamaremos a função ao final no momento em que quisermos realmente exibir as opções logo depois de terminarmos o sorteio das cartas.

Ou seja, dentro da função de `sortearCarta()`, quando tudo já tiver terminado, iremos exibir as opções chamando a `exibirOpcoes()` para de fato apresentarmos os atributos.

Com isso, vamos salvar e rodar para vermos os atributos no Console.

```
//código anterior omitido
```

```
function sortearCarta() {
    var numeroCartaMaquina = parseInt(Math.random() * 3);
    cartaMaquina = cartas[numeroCartaMaquina];

    var numeroCartaJogador = parseInt(Math.random() * 3);
    while (numeroCartaMaquina == numeroCartaJogador) {
        numeroCartaJogador = parseInt(Math.random() * 3);
    }
    cartaJogador = cartas[numeroCartaJogador];
    console.log(cartaJogador);

    document.getElementById("btnSortear").disabled = true;
    document.getElementById("btnJogar").disabled = false;

    exibirOpcoes()
}

function exibirOpcoes() {
    var opcoes = document.getElementById("opcoes");

    for (var atributo in cartaJogador.atributos) {
        console.log(atributo);
    }
}
```

Rafaella: Conseguimos imprimir a carta do jogador no Console, e embaixo teremos os três atributos.

Também queremos imprimir o valor que a carta tem, pois agora estamos colocando o `.atributos` e precisamos acessar o valor de cada um.

Então aproveitaremos para imprimir na própria tela da página. Estávamos entendendo melhor o funcionamento do *loop*, e já vimos como usar o `console.log()` da mesma maneira que sempre fizemos até agora, usando o `.innerHTML` para imprimir na página mesmo.

Agora criaremos um texto para imprimirmos dentro da tag do `innerHTML` em uma nova variável `opcoesTexto` na função `exibirOpcoes()`.

Guilherme: A ideia é montarmos alguma forma ou tag para, depois que clicarmos no botão do sorteio, exibirmos as opções de atributos para podermos escolher.

Rafaella: Exatamente, vamos criar uma tag, ou seja, escreveremos um HTML pelo JavaScript.

Paulo: Queremos que os atributos de "ataque", "defesa" e "magia" apareçam na página e não somente no Console, e queremos também conseguir clicar para selecionar a opção.

Rafaella: Sim, veremos um tipo diferente de tag. Inclusive é uma que já abordamos, mas com um tipo diferente para que possamos clicar de fato.

Já inicializamos a `var opcoesTexto` ainda vazia, e dentro do nosso `var opcoesTexto`, escreveremos o texto passando o valor para a variável. Desta vez, criaremos a nossa tag `<input>`, lembrando de colocar entre aspas pois é o formato *string*.

O `type` deste `input` será `radio`, que é o tal do `radioButton` de quando queremos selecionar uma opção e há várias, seja na vertical ou horizontal.

Neste caso deixaremos na horizontal mesmo, pois o CSS já deixa assim pronto. Assim, receberemos várias opções para escolhermos.

Também passaremos a propriedade `name` igual a `name`, lembrando que não poderemos escrever com aspas duplas, pois como as estamos utilizando para passar o valor da *string*, se fecharmos com as aspas duplas o sistema entenderá que é uma frase, e irá quebrar.

Então precisamos colocar aspas dentro das aspas, e para isso utilizamos as aspas simples `'` para não quebrar a nossa *string* mesmo. Desta forma, continuará como string até fecharmos o `<input>` de fato com as aspas duplas.

Além do `name`, também passaremos o `value` que é realmente o valor recebido pelo atributo. Por exemplo, no `input` usávamos o formato de texto e o valor é o que escrevíamos dentro, mas neste caso, cada um dos valores será o próprio atributo.

Por exemplo, o atributo `radioButton` de ataque terá o valor `"ataque"`, já o de defesa terá o `value` como `defesa`. Então o `value` no caso será a variável `atributo` que já criamos e está dentro do nosso *loop* funcionando como índice que imprimimos, como é possível ver no Console.

Então o `value` não precisa mais da *string*, basta fecharmos as aspas duplas. Aplicaremos o "mais", como já estávamos concatenando a variável JavaScript com string, colocando o `atributo`.

Com isso, somaremos e fecharemos a tag. Colocaremos novamente as aspas duplas para podermos escrever o texto e fechar de vez com.

Para imprimirmos na tela, usaremos o `.innerHTML` com a variável `opcoes` que criamos anteriormente, a qual já "puxa" o `id "opcoes"`, que é aquele espaço de `<div>` dentro do HTML, que é onde já sabemos que queremos imprimir os elementos.

Colocaremos `opcoes` com `.innerHTML` da mesma forma que já estamos acostumados a fazer, e passaremos a frase `opcoesTexto`.

Então vamos salvar e rodar para testarmos, mas sabemos que teremos que alterar algumas coisas depois. Podemos até limpar e fechar o Console para vermos apenas a página.

```
//código anterior omitido

function exibirOpcoes() {
    var opcoes = document.getElementById("opcoes");
    var opcoesTexto = "";

    for (var atributo in cartaJogador.atributos) {
        opcoesTexto =
            "<input type='radio' name='atributo' value='" + atributo + "'>";
    }
    opcoes.innerHTML = opcoesTexto;
}
```

Guilherme: Sortearemos a carta clicando no botão "Sortear carta", e apareceu somente um pequeno botão redondo e branco abaixo do enunciado "Escolha o seu atributo".

Rafaella: O que acontece é que, primeiramente, precisaremos voltar ao nosso JS para imprimir o `opcoesTexto` somados uns aos outros. Da mesma forma como estávamos concatenando *strings* com a variável, precisaremos concatenar todos os textos e botões que queremos exibir na tag HTML.

Então o `opcoesTexto` terá o sinal de "mais" antes do sinal de igualdade, que na verdade é o que já tínhamos visto antes também, que é o valor igual ao valor mais alguma coisa.

Portanto será o `opcoesTexto +=` com o texto, somando os *inputs*. Porém, é uma forma mais simples de escrever que já aprendemos, apenas escrevendo o sinal de mais antes do igual, que é como se "puxássemos" este `opcoesTexto` do lado direito e o mais para o `opcoesTexto`.

É mais para entendermos bem que precisamos de fato imprimir os três atributos, do contrário só imprimiríamos um deles.

Outra coisa importante é que não estamos escrevendo quais são os atributos, pois estamos passando apenas a tag. No HTML, veremos que normalmente tudo o que está escrito em texto está na cor branca como podemos ver no *syntax highlight*.

Tudo o que for escrito direto na página, como nos enunciados ou botões, estará em branco. Colocamos dentro da tag, a qual abre e fecha com o texto dentro.

Neste caso, teremos que passar o que queremos escrever em nosso JavaScript também, que é cada um desses atributos. Com isso, já será fora da nossa tag.

Então colocaremos o "mais" e utilizaremos a própria variável `atributo` que já tem o nome de defesa, ataque e magia fora da tag para podermos escrever. É como se estivéssemos escrevendo na parte branca do HTML.

```
//código anterior omitido
```

```
function exibirOpcoes() {  
    var opcoes = document.getElementById("opcoes");  
    var opcoesTexto = "";  
  
    for (var atributo in cartaJogador.atributos) {  
        opcoesTexto =  
            "<input type='radio' name='atributo' value='" +  
            atributo +  
            "'>" +  
            atributo;  
    }  
    opcoes.innerHTML = opcoesTexto;  
}
```

Guilherme: Vamos testar? Vou salvar, rodar e clicar em "Sortear".

Rafaella: Olha só, já apareceram os três botões, que nós somamos as três tags de input, e agora temos "Ataque", "Defesa" e "Magia" descritos. Já estamos conseguindo exibir na tela todos esses atributos para escolhermos.

O `radio button` são essas "bolinhas" que conseguimos selecionar como opções. Inclusive existem algumas propriedades que ele pode receber, como "já selecionado" ou "pode selecionar mais de um". Você pode consultar essas propriedades na documentação. No caso do Super Trunfo, só poderemos escolher um atributo, já que esse é o formato do nosso jogo.

Guilherme: Isso ficou muito legal. E a nossa carta está aparecendo no console, certo?

Rafaella: Certo, está aparecendo no console.

Paulo: Do jeito que vocês estão fazendo, o `exibirOpcoes()` faz esse `for` nesse formato novo para pegar as chaves dos nossos atributos. Poderíamos ter feito aquilo que chamamos de "hard coded", deixando os valores sempre fixos no campo HTML.

Como a Rafa e o Guilherme decidiram fazer assim, dinamicamente, se adicionarmos outro atributo em nossas cartas, por exemplo a "Mana", já vai aparecer sem que seja necessário alterar em vários lugares do sistema.

Deixar dinâmico ou fixo sempre vai ser uma dúvida e a decisão vai depender do software que você está criando, assim como o método de iteração - se você usa o `for`, o `while` ou até mesmo o `do while`.

Nesse caso temos um `for` onde não é necessário incrementar com o `i++`, já que nossa chave é um texto. Tem muitos lugares onde podemos escolher como resolver o problema, uma decisão que pode trazer vantagens ou desvantagens no futuro. São coisas que vamos aprendendo com o tempo.

Rafaella: Já selecionamos nosso atributo, e o próximo passo é escolhermos qual jogar e pegarmos essa informação. Estamos com a carta no console, por exemplo o "Darth Vader", que tem um "ataque" de valor 9. Nesse caso, faz sentido selecionarmos o "ataque" e jogarmos com ele.

Entretanto, ainda não temos a função de "Jogar", que é a parte de comparação, e também não estamos pegando essa informação de qual atributo foi escolhido. É isso que vamos fazer agora.

Guilherme: Uma coisa é mostrarmos na tela as três opções. Agora precisamos pegar a informação de que o usuário ou usuária selecionou o ataque, a defesa ou a magia. Para isso, vou criar uma nova função que obtém qual foi o atributo selecionado, chamada `obtemAtributoSelecionado()`.

Essa função será invocada quando clicarmos em "Jogar". Sendo assim, vou criar também uma função `jogar()`, que terá uma variável `atributoSelecionado()` recebendo o retorno da invocação de `obtemAtributoSelecionado()`.

Paulo: É exatamente o que a Rafa estava falando. Essa função `obtemAtributoSelecionado()` ainda está vazia, mas já estamos organizando a lógica. Quando clicarmos em "Jogar", nós vamos obter o atributo selecionado, pegar a carta que foi escolhida e a carta da máquina, comparar os atributos das duas e dar o resultado.

Não tem problema ainda não termos o código de `obtemAtributoSelecionado()`, estamos fazendo o raciocínio geral e depois escreveremos as peças que estão faltando. Em alguns momentos é mais interessante seguir essa abordagem.

Guilherme: Nossos atributos estão dinâmicos e, se criarmos outros, como "velocidade", "força" e assim por diante, eles vão aparecer. Mas como conseguiremos o atributo selecionado para comparar com a máquina? Vamos focar nisso.

Na função `obtemAtributoSelecionado()`, criaremos uma variável `atributoSelecionado`. Ele precisa acessar o nosso HTML, então sabemos que usaremos o `document`. No HTML a Rafa incluiu uma propriedade `name`, o que nos permite utilizar a função `getElementsByName()`, passando nosso elemento "atributo" como parâmetro.

```
function obterAtributoSelecionado() {  
    var atributoSelecionado = document.getElementsByName("atributo");  
  
}
```

Porém, dessa forma nós traremos todo o atributo, ou seja, "ataque", "defesa" e "magia". É necessário descobrirmos qual o atributo selecionado. Vou até renomear essa variável para `radioAtributos`, já que ele contém todas essas informações.

```
function obterAtributoSelecionado() {  
    var radioAtributos = document.getElementsByName("atributo");  
  
}
```

Prosseguindo, usarei a instrução `for` para percorrer todo o nosso `radioAtributos`. Faremos `var i = 0` e, enquanto o `i` for menor que `radioAtributos.length` (a quantidade de elementos no input), incrementaremos com `i++`.

```
function obterAtributoSelecionado() {  
    var radioAtributos = document.getElementsByName("atributo");  
  
    for (var i = 0; i < radioAtributos.length; i++) {  
  
        }  
}
```

Dentro do `for` colocaremos nossa verificação. Se o `radioAtributos` no índice `[i]` estiver marcado (`checked == true`), saberemos que esse é o atributo selecionado.

Rafaella: Esse `checked` é igual a qualquer outra propriedade que estávamos usando, você consegue passá-la no HTML mas também utilizá-la no Javascript. Quando nós selecionamos um atributo na página, o próprio programa automaticamente entende que esse `checked` está com o valor `true` - do contrário, está como `false`.

Guilherme: Se o `checked` for verdadeiro, o comportamento esperado é devolver para a função `jogar()` o atributo que de fato foi escolhido. Sendo assim, vamos retornar (`return`) o valor no índice selecionado com `radioAtributos[i].value`.

```
function obterAtributoSelecionado() {
  var radioAtributos = document.getElementsByName("atributo");

  for (var i = 0; i < radioAtributos.length; i++) {
    if (radioAtributos[i].checked == true) {
      return radioAtributos[i].value;
    }
  }
}
```

Na função `jogar()`, farei um `console.log()` do `atributoSelecionado` para verificarmos se o comportamento está acontecendo como esperado.

```
function jogar() {
  var atributoSelecionado = obterAtributoSelecionado();
  console.log(atributoSelecionado)
}
```

Após salvar, vou clicar em "Sortear carta" e receber o "Darth Vader" no console. Seleccionarei o ataque, que é o atributo mais forte da carta, e clicarei em "Jogar". No console, ele mostrará o texto "ataque" - ou seja, conseguimos recuperar o atributo selecionado.

O desafio agora é compararmos as informações do atributo selecionado da minha carta com a carta da máquina e mostrar qual foi a carta vencedora.

Rafaella: Exato. Vamos lembrar o que fizemos no início da aula, que foi acessar o valor do atributo que está dentro da chave de atributos que está dentro do nosso objeto. Ainda na função `jogar()`, vamos fazer um `console.log()` de `cartaJogador.atributos`.

Paulo: Se eu fizer `cartaJogador.atributos.atributoSelecionado`, não vai acontecer o que eu gostaria, certo? O Javascript vai tentar pegar dentro da `cartaJogador` um atributo chamado `atributoSelecionado`, que não existe. Sendo assim, precisamos de outra forma de acessar a chave desse atributo passando essa variável. Tem sintaxe para isso?

Rafaella: Tem um formato que é abrindo os nossos colchetes e, dentro deles, passar de fato o `atributoSelecioneado`.

```
function jogar() {  
    var atributoSelecioneado = obterAtributoSelecioneado();  
    console.log(atributoSelecioneado)  
    console.log(cartaJogador.atributos[atributoSelecioneado])  
}
```

Paulo: A mesma sintaxe que tínhamos nas listas, mas isso não é uma lista, é um objeto. Ao invés de acessar a enésima posição desse objeto, nós passamos uma chave para ele.

Rafaella: Exatamente, estamos passando a própria chave. Da mesma forma que chamamos o `atributos` ou o `nome` para pegarmos seus valores, estamos chamando o próprio `ataque` para pegarmos esse valor.

Claro que, para isso, precisamos chamar dentro do próprio `atributos`, por isso não fizemos `cartaJogador[atributoSelecioneado]`. Vamos executar para verificar se realmente conseguimos pegar esse valor.

Guilherme: Vou sortear a carta, saiu o "Shiryu de dragão", que tem o valor 10 em magia. Vou selecionar esse atributo e clicar em "Jogar". No console, teremos "magia", que é o atributo selecionado, e o valor 10.

Rafaella: Agora que conseguimos acessar esse valor, queremos comparar qual carta tem o atributo de maior valor, se é a do jogador ou a da máquina. Além disso, vamos imprimir na tela "você venceu", "você empatou" ou "você perdeu".

Em nosso HTML já temos uma tag "resultado" que será responsável por essa impressão. Já podemos remover os `console.log()` que usamos para esse teste.

Paulo: Por enquanto só temos o valor do atributo da carta do jogador, precisamos pegar da carta da máquina e comparar.

Rafaella: Exato. Vamos criar uma variável `elementoResultado` recebendo o `document.getElementById("resultado")`.

```
function jogar() {
  var atributoSelecioneado = obterAtributoSelecioneado();
  var elementoResultado = document.getElementById("resultado");

  console.log(cartaJogador.atributos[atributoSelecioneado])
}
```

Guilherme: Rafa, eu tenho uma dúvida. A estrutura

`cartaJogador.atributos[atributoSelecioneado]` está bem grande. Nós vimos que é possível pegar o valor do atributo selecionado na carta do jogador, será que eu posso criar uma nova variável para armazenar esse valor e outra para a carta da máquina para então fazermos essa comparação? Assim não precisaremos usar toda essa estrutura dentro do `if`.

Rafaella: Pode, vamos fazer isso. Criaremos as variáveis `valorCartaJogador` recebendo esse conteúdo do `console.log()`, e `valorCartaMáquina`, fazendo a mesma coisa mas para a carta da máquina.

```
function jogar() {
  var atributoSelecioneado = obterAtributoSelecioneado();
  var elementoResultado = document.getElementById("resultado");
  var valorCartaJogador =
  cartaJogador.atributos[atributoSelecioneado];
  var valorCartaMáquina =
  cartaMáquina.atributos[atributoSelecioneado];
}
```

Rafaella: Agora vamos comparar os valores desses atributos selecionados. Usaremos a instrução `if` com os seguintes cenários: se `valorCartaJogador` for maior que `valorCartaMáquina`, vamos vencer; se for menor, vamos perder; e se for igual, vamos empatar.

Para imprimirmos na tela, acessaremos a variável `elementoResultado.innerHTML` e escreveremos os respectivos textos "Você venceu", "Você perdeu, a carta da máquina é

maior" ou "Empatou". Lembrando que, após a primeira verificação, podemos utilizar o `else if` para as condições posteriores.

Paulo: Inclusive, futuramente podemos colocar os valores para a pessoa acreditar no resultado.

Rafaella: Sim, podemos fazer um `console.log()` da `cartaMaquina` para conferirmos se essa lógica realmente está certa, se os valores batem com o resultado.

```
function jogar() {
  var atributoSelecioneado = obterAtributoSelecioneado();
  var elementoResultado = document.getElementById("resultado");
  var valorCartaJogador =
  cartaJogador.atributos[atributoSelecioneado];
  var valorCartaMaquina =
  cartaMaquina.atributos[atributoSelecioneado];

  if (valorCartaJogador > valorCartaMaquina) {
    elementoResultado.innerHTML = "Você venceu";
  } else if (valorCartaMaquina > valorCartaJogador) {
    elementoResultado.innerHTML = "Você perdeu, a carta da máquina é
maior";
  } else {
    elementoResultado.innerHTML = "Empatou";
  }
  console.log(cartaMaquina);
}
```

Guilherme: Vou limpar o console e executar novamente. Clico em "Sortear carta", saiu o "Darth Vader", que tem o ataque bem alto. Vou selecionar o ataque e clicar em "Jogar". Apareceu embaixo o texto "Você venceu", a carta da máquina saiu no console, o "Shiryu de dragão", com ataque mais fraco. Deu certo.

Rafaella: Perfeito. O nosso texto "Você venceu" está em preto ainda, seria legal mudar, mas isso é coisa do CSS e quando vocês pegarem o código já vai estar alterado. Paulo, o que você quer deixar de desafio para o pessoal?

Paulo: Guilherme, só sorteia a carta, não joga. Primeiro eu quero que você, que está desenvolvendo, tente jogar sem selecionar nenhum dos atributos para verificar o que vai acontecer. E como resolvemos esse tipo de problema?

Tem várias formas de fazer isso, por exemplo mostrando uma mensagem para a pessoa selecionar, deixar algum atributo selecionado por padrão e assim por diante ou sortear um atributo aleatório. Dá para fazer isso com `if`, com `random`, com HTML, vários mecanismos possíveis.

Fica para você resolver. Repare que nessa aula começamos a misturar várias coisas, trabalhamos com input do usuário, condições de execução, entre outras.

Outro desafio é você popular essas cartas com vários personagens que você gosta. Amanhã vamos inclusive colocar imagens, dando uma roupagem bem legal. Compartilhe suas cartas e seu código lá no Discord para criarmos uma rede, um deck, bem interessante e completo para você realmente impressionar as pessoas.

Rafaella: Muito legal. Eu tenho um desafio também, acho que seria interessante logo que você sorteia aparecer uma imagem do personagem da carta sorteada. Você pode escolher onde colocar essa imagem. Já vimos com o Aluraflx que é possível pegar o endereço de uma imagem e incluir na tela usando a tag `` do HTML, e acho que ficaria bem bonito.

Vamos ver isso nas próximas aulas, mas é legal você começar a brincar com a ideia de adicionar mais uma chave e valor no objeto e com essa tag de imagem. Você tem algum desafio, Gui?

Guilherme: Acho que não, vocês já pegaram pesado demais. Por misericórdia aos nossos alunos e alunas, não vou passar nenhum desafio, ter chegado até aqui já é ótimo.

Paulo: Estamos nos aproximando do final da Imersão Dev. Essa aula tem alguns detalhes, é um pouco mais pesada, pois envolve HTML, CSS e vários conceitos que você pode errar. Tente resolver e, se não conseguir, traga para a gente no Discord!

É muito importante você saber até quando insistir sozinho tentando descobrir o erro e quando pedir ajuda, existe um meio termo aí. Claro que você deve ter coragem de mexer no código, mas também não faz sentido perder horas em um mesmo ponto sem

sair do lugar. É necessário conhecer esse momento de conversar, pedir ajuda, discutir ou mesmo copiar uma solução.

Vamos dar continuidade a esse projeto amanhã, e depois faremos uma finalização incrível, envolvendo ainda mais coisas. A ideia é você ter no seu portfólio e mostrar com muito orgulho para as pessoas o que você fez na Imersão Dev e sua capacidade de construir em cima daquilo que a gente trouxe para você. Queremos que você participe desse Aluraverso e seja mais uma dessas pessoas que falam da gente, da Imersão, e que participam da comunidade que temos construído nesses vários anos.

A gente se vê amanhã com mais código para mergulharmos nessa Imersão!