alura

Conhecendo HTML, CSS e Javascript_

Construa suas primeiras aplicações em 10 dias.

SOBRE ESTE LIVRO

A ideia de criar este livro veio durante a produção da Imersão Dev da Alura que aconteceu entre 22 de março e 2 de abril de 2021. A Imersão foi pensada para pessoas sem nenhum contato anterior com programação, e sabemos que esta primeira experiência pode ser decisiva! Assim, ficamos felizes em acompanhar quem está começando agora da melhor forma possível, e sabemos que isso pode ser feito de várias maneiras: síncrona, assíncrona, por vídeo e também por texto.

O conteúdo a seguir está totalmente baseado nos projetos desenvolvidos durante a Imersão Dev, mas conta com algumas explicações extras e diferentes. Ele foi pensado para servir de apoio ao conteúdo da Imersão, mas não é apenas uma transcrição dos vídeos e sim material extra em um formato mais adequado para o texto. Há várias formas de estudar e vários tipos de suporte para isso: vídeo, texto, áudio e inclusive papel. Parte do desafio do estudo de programação é justamente encontrar o que funciona melhor para você. Esperamos que este livro possa lhe ajudar de alguma forma a entrar com o pé certo nesse novo mundo.

Bons estudos!

SOBRE QUEM ESCREVEU

A produção dos capítulos foi dividida entre nosso time de conteúdo de programação:

Guilherme Lima Um dos condutores da Imersão Dev junto

com a Rafaella, é desenvolvedor Python e instrutor back end na Alura.

Rafaella Ballerini Condutora da Imersão Dev com o Guilherme Lima, é instrutora front end da Alura.

Juliana Amoasei Desenvolvedora JavaScript e instrutora de programação back end na Alura.

AGRADECIMENTOS

A todas as 93.900 pessoas que se inscreveram para participar da Imersão Dev, motivadas pela mudança de carreira, reforço nos estudos ou, também, por que não, pela curiosidade de aprender algo novo. A Imersão foi feita para vocês, assim como todo este livro e os materiais extras.

A todos os times da Alura! Quem aparece são as devs e os devs, mas todas as imersões são produzidas por muitas mãos: time de devs que pensa e produz o conteúdo, didática que garante que o conteúdo faça sentido para quem está vendo de fora, edição e produção de vídeo, marketing, que faz com que tanta gente entre em contato com o evento e possa aprender programação com a gente, e o Scuba Team que dá suporte nas plataformas. Nossos agradecimentos!

COMO UTILIZAR ESTE LIVRO

Tudo que se refere a **trechos de código** neste livro utiliza uma das duas notações abaixo:

- trechos pequenos ou termos, como nomes de funções,

palavras-chaves ou expressões aparecerão no texto `desta forma`. Por exemplo, quando mencionamos uma função chamada `executarCodigo()` ou uma palavra-chave como `.length`.

- trechos maiores de código, como blocos de função, objetos ou similares aparecerão no texto em um parágrafo separado, como por exemplo:

```
function exibeTexto() {
   console.log("Olá Mundo!")
}
```

Todo o código será disponibilizado via link para consulta ou download (mais sobre isso abaixo), **mas é importante que você leia para entender o que acontece em cada parte!**

Sobre o GitHub

Utilizamos GitHub para disponibilizar o código finalizado de cada um dos projetos. O GitHub é uma plataforma de hospedagem de códigos que permite que qualquer pessoa cadastrada possa acessar e contribuir com projetos de código aberto.

O Github representa todo um universo ligado à programação, o chamado **controle de versão ou versionamento de código**. É uma das ferramentas principais no dia-a-dia de quem trabalha na área e você pode ir aprendendo como utilizá-la aos poucos.

Ao final de cada capítulo, você terá acesso ao código completo disponibilizado no GitHub, com a seguinte estrutura:

```
app.js
index.html
style.css
```

Sinta-se a vontade para executar os programas no Codepen ou em algum editor de código de sua preferência.

Redes sociais

Paulo Silveira

- Instagram
- Twitter
- Linkedin

Rafaella Ballerini

- Instagram
- Linkedin
- Youtube

Guilherme Lima

- Instagram
- Youtube
- Linkedin

Juliana Amoasei

- Linkedin

Sumário

Versão: 25.5.6

Capítulo 1

CONVERSOR DE MOEDAS

1.1 OBJETIVO DA AULA

Nesta primeira aula, o objetivo é criar um programa que peça para o usuário fornecer um valor em dólar, e a partir desse valor o programa vai exibir na tela o equivalente em real.

Neste programa focamos nas primeiras ferramentas principais de qualquer linguagem de programação: variáveis e operadores, além das funções `alert` e `prompt` para trocar nossas primeiras mensagens na tela com o usuário.

1.2 PASSO A PASSO

Antes de começarmos a escrever qualquer linha de código, temos que pensar no caminho que o programa deve percorrer e quais informações ele precisa para ser executado corretamente. No caso do nosso conversor de moedas:

1. O que o programa deve fazer? Converter dólar para real…

Ok, mas como isso é feito? Se fôssemos converter 10 dólares para real "na mão", faríamos a seguinte conta: `10 * 5.50 =`

`55.00`.

Onde `10` é o valor em reais que queremos converter e `5.50` é a taxa média do dólar no momento em que escrevemos este texto. Multiplicando um valor pelo outro, temos o resultado em reais: `55.00`.

2. Como o programa vai receber essas informações?

Podemos concluir, a partir do que vimos no ponto 1, que os dados que o programa precisa para funcionar são:

- o valor em dólar que queremos converter (será fornecido pelo usuário);
- a taxa do dólar que será utilizada na conversão (que nós vamos fornecer ao programa no momento em que escrevemos o código).

3. Qual o "resultado" do programa:

Após executar todo o código, o programa deve exibir ao usuário um número que representa o resultado da operação `valor em dólar * taxa de câmbio`.

Agora que já sabemos quais dados o programa precisa para funcionar, o que ele precisa fazer e qual o resultado final dele, podemos começar a escrever código!

1.3 ESCREVENDO O PROGRAMA

Já que precisamos receber do usuário o valor em dólar que ele quer converter, a primeira coisa a fazer é "pegar" esse valor de

alguma forma. Vamos utilizar uma ferramenta do JavaScript chamada `prompt`, que abre no navegador uma caixa de texto onde o usuário pode inserir informações. Utilizamos os parênteses `( )` para passar para o usuário uma mensagem, assim ele sabe qual informação estamos pedindo — **não esqueça das aspas!**.

```
var valorEmDolarTexto = prompt("Qual o valor em dolar que você qu
er converter?")
```

É muito importante sempre lembrar de "guardar" todos os dados que queremos usar em nossos programas e, para isso, usamos variáveis. Variáveis são como pequenos espaços de memória onde guardamos dados, como por exemplo um número ou um texto. Nesse caso, criamos com a palavra-chave `var` uma variável chamada `valorEmDolarTexto` que vai "guardar" a informação que o usuário vai passar para o programa. E que informação é essa? O valor em dólar para ser calculado.

> O nome da variável é muito importante! É através dele que o programa consegue identificar e localizar todos os dados que "guardamos" para serem utilizados pelo programa, e que podem ser muitos!

Agora que já temos o valor em dólar, vamos esbarrar em um probleminha muito comum e característico da programação, que é a distinção entre textos e números. Por padrão, quando usamos o `prompt` para pedir alguma informação ao usuário, o JavaScript **sempre** interpreta essa informação como texto, não importa se o usuário escreve `2`. Essa é uma questão relacionada a *tipos de dados em computação*, que neste momento não vamos detalhar

muito, mas você pode pesquisar para saber mais.

Já que é assim, se o usuário inserir o valor `10` para converter, o JavaScript pode ficar confuso, afinal de contas ele não sabe que `"10"` é um número, e como é que ele vai fazer essa multiplicação?

Felizmente, como já dissemos, essa questão *texto x número* é comum. E o próprio JavaScript tem uma palavra-chave pronta para fazer essa conversão. Então, antes de qualquer outra coisa, devemos pegar o valor que está guardado na variável `valorEmDolarTexto` e transformá-lo de texto para número:

```
var valorEmDolarTexto = prompt("Qual o valor em dolar que você qu
er converter?")

var valorEmDolarNumero = parseFloat(valorEmDolarTexto)
```

A palavra-chave `parseFloat()` é essa ferramenta pronta do JavaScript para converter o que está entre os parênteses de texto para número. No caso, o que vai ser convertido é o valor que o usuário vai inserir no programa através do `prompt` e que está guardado na variável `valorEmDolarTexto`.

Como sempre, precisamos guardar em uma variável esse valor convertido, para que o programa possa usá-lo. Já que agora o valor é um número, chamamos essa nova variável de `valorEmDolarNumero`.

Já podemos fazer a conta? E a taxa do dólar? Há algumas formas de passarmos para o programa a informação da taxa do dólar. Para esta primeira versão do nosso conversor vamos dizer ao JavaScript exatamente qual é a taxa e utilizá-la na conta de multiplicação:

```
var valorEmDolarTexto = prompt("Qual o valor em dolar que você qu
```

```
er converter?")

var valorEmDolarNumero = parseFloat(valorEmDolarTexto)

var valorEmReal = valorEmDolarNumero * 5.50
```

Sem esquecer de "guardar" o valor da multiplicação em uma variável, afinal de contas precisamos dele para exibir ao usuário! Neste caso, a taxa utilizada é `5.50`; se lembrarmos que a variável `valorEmDolarNumero` já está guardando um número (informado pelo usuário), o JavaScript consegue fazer a substituição e usar os valores para fazer a conta :)

Estamos guardando o resultado desta conta na variável `valorEmReal`.

Uma última coisa antes de exibir o resultado para o usuário! Quando fazemos contas que envolvem valores "quebrados" — ou seja, com um ou mais dígitos depois da vírgula — pode acontecer a tal da *dízima periódica*, ou o número após a vírgula ficar com muitas casas decimais. Então vamos usar uma outra palavra-chave que o JavaScript já deixou pronta para esses casos:

```
var valorEmDolarTexto = prompt("Qual o valor em dolar que você qu
er converter?")

var valorEmDolarNumero = parseFloat(valorEmDolarTexto)

var valorEmReal = valorEmDolarNumero * 5.50
var valorEmRealFixado = valorEmReal.toFixed(2)
```

Essa palavra-chave é `.toFixed()` e deve ser usada da forma que está no exemplo: nome da variável, ponto, `toFixed` e entre parênteses vai um número com as casas decimais que queremos arredondar.

Ou seja, `nomeDaVariavel.toFixed(2)` vai arredondar o

número que está guardado dentro da variável `nomeDaVariavel`. Se for o número `5.77777777` ele vai ser arredondado para `5.77`. Sem esquecer de guardar esse novo valor em uma variável! No caso, usamos a variável `valorEmRealFixado` para isso.

Agora já está tudo pronto para exibir o resultado para o usuário. Podemos usar a ferramenta `alert()` que vai exibir a informação em uma caixa de texto no navegador. Só precisamos passar essa informação dentro dos parênteses; se informarmos o nome da variável onde este dado está guardado, o JavaScript consegue fazer a substituição:

```
var valorEmDolarTexto = prompt("Qual o valor em dolar que você qu
er converter?")

var valorEmDolarNumero = parseFloat(valorEmDolarTexto)

var valorEmReal = valorEmDolarNumero * 5.50
var valorEmRealFixado = valorEmReal.toFixed(2)

alert(valorEmRealFixado)
```

Alguns detalhes que sempre temos que lembrar:

- Podemos dar o nome que quisermos para as variáveis, porém o JavaScript tem algumas regras: não comece com letra maiúscula nem com números, não use espaços, caracteres especiais e nem acentos. Escolha um nome que descreva o que a variável vai "guardar"! O padrão de nomes do JavaScript é `nomeDaVariavel`, usando maiúsculas para diferenciar as palavras (mas nunca no início da palavra).

- O JavaScript tem muitos trechos de código prontos para utilizarmos em nossos programas, como por exemplo `parseFloat()` para converter um texto em número e

`variavel.toFixed(2)` para arredondar. Estes códigos estão sempre escritos em inglês e devem ser usados exatamente como estão nos exemplos para funcionar — não coloque pontos nem espaços onde não tem, não troque maiúsculas por minúsculas e não esqueça dos parênteses nos lugares certos, pois esses pequenos detalhes fazem com que o JavaScript "entenda" o que está sendo escrito. Com prática e estudo você vai entender melhor para que servem e o que significam os pontos, parênteses e tudo mais.

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

1.4 RESUMO

Vamos repassar todos os passos do programa:

- Obter do usuário o valor que vai ser convertido
- Converter esse dado para número
- Fazer a operação matemática (multiplicação)
- Arredondar o resultado
- Exibir o resultado na tela para o usuário

Capítulo 2

CALCULADORA

2.1 OBJETIVO DA AULA

Criar um programa que pergunte e armazene dois valores e o usuário escolha qual operação deseja realizar com um número. Por exemplo: para somar a pessoa teria que digitar 1, para subtrair, 2, etc. **Neste programa, focamos em laços condicionais, operações booleanas, if e else**.

2.2 PASSO A PASSO

Vamos começar criando uma variável para armazenar o primeiro valor.

```
var primeiroValor
```

Agora vamos pedir para o usuário digitar o primeiro valor.

```
var primeiroValor = prompt("Digite o primeiro valor:"))
```

Como queremos realizar operações matemáticas com esses valores, vamos converter o primeiro valor em um número inteiro com o `parseInt` .

```
var primeiroValor =prompt("Digite o primeiro valor:")
primeiroValor = parseInt(primeiroValor)
```

Dica: essa conversão pode ser feita em apenas uma linha. Podemos usar as funções `prompt` e `parseInt` na mesma linha, e teremos o mesmo resultado.

```
var primeiroValor = parseInt(prompt("Digite o primeiro valor:"))
```

Agora, vamos criar uma segunda variável para armazenar o segundo valor.

```
var primeiroValor = parseInt(prompt("Digite o primeiro valor:"))
var segundoValor = parseInt(prompt("Digite o segundo valor:"))
```

Agora que temos dois valores, precisamos perguntar qual operação será realizada (se multiplicação, soma, divisão ou subtração). Para isso, vamos representar cada operação com um número, como ilustra a tabela abaixo.

número da operação	operação
1	Divisão
2	Multiplicação
3	Soma
4	Subtração

Sabendo disso, vamos criar uma variável para armazenar o número da operação escolhida através da função `prompt`.

```
var operacao = prompt("Digite 1 para fazer uma divisão, 2 para mu
ltiplicação, 3 para soma e 4 para subtração: ")
```

Agora precisamos saber **se** a variável `operacao` é igual a 1. No JavaScript e em muitas outras linguagens, essa condição **se** é escrita

no idioma inglês **if** (que significa “se” em português).

```
if
```

Para criar a condição que verifica **se** a variável `operacao` é igual a 1 vamos incluir o sinal de parênteses.

```
if ()
```

Dentro dos parênteses, vamos incluir nossa condição, e aqui tem um detalhe importante. Comparar **se** a variável `operacao` é **igual** a **1** não será feita com apenas um sinal de igual, e sim com dois (`==`)..

```
if (operacao == 1)
```

Isso acontece porque quando usamos apenas um sinal de igual, estamos atribuindo um valor a uma variável, e não é o que queremos fazer. Queremos comparar se o valor da variável operação é igual a um número, neste caso, 1.

> Dica: com um sinal de igual (`=`), estamos atribuindo um valor a uma variável (`var nome = "Maria"` , por exemplo). Com dois sinais de igual (`==`), estamos comparando dois valores.

Caso isso seja verdadeiro, isto é, caso a operação escolhida seja *igual igual* a 1, queremos fazer alguma coisa. Esse fazer alguma coisa será indicado pelos sinais de abrir e fechar chaves (`{ }`).

```
if (operacao == 1) {

}
```

Ao fazermos isso, todo o código escrito dentro dos "bigodes", quer dizer, chaves, será executado caso a operação seja igual a 1. Dentro das chaves, precisamos dividir o valor armazenado na variável `primeiroValor` pelo `segundoValor` e guardar esse resultado em algum lugar. Podemos criar uma nova variável chamada `resultado` .

```
if (operacao == 1) {
  var resultado
}
```

Na programação, as operações são representadas de formas diferentes. Veja isso no quadro abaixo.

símbolo na programação	operação
/	Divisão
*	Multiplicação
+	Soma
-	Subtração

Sabendo disso, podemos usar a barra para dividir o primeiro valor pelo segundo valor.

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
}
```

Vamos executar nosso programa, atribuir os valores `10` no primeiro valor e `2` no segundo, selecionar a opção 1, e ver o que acontece?

Nada :(Não aconteceu nada. O resultado da operação não apareceu na tela e isso é triste. Podemos incluir um alert exibindo o resultado.

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  alert(resultado)
}
```

Vamos testar agora?

Deu certo :)

Maaaaas podemos fazer algo melhor: no lugar de exibirmos um *alert*, que tal escrevermos na tela? Que tal exibirmos o resultado da divisão na tela?

Para isso, podemos usar um outro comando, chamado `document.write` , e passar o valor que queremos exibir entre os parênteses.

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write(resultado)
}
```

Agora sim! O resultado apareceu na tela e ficou bem legal. Podemos melhorar ainda mais.

Que tal incluirmos os valores das variáveis `primeiroValor` e `segundoValor` com o símbolo de divisão entre elas, e o sinal de igual informando o resultado? Para juntar os valores das variáveis com os símbolos do tipo texto, usaremos o sinal de soma (`+`), e as aspas para os símbolos, assim:

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write(primeiroValor + " / " + segundoValor + " = " + r
esultado)
}
```

Dica: dentro do `document.write` , podemos incluir as tags do HTML e estilizá-las no CSS, alterando o tamanho, posicionamento, cores e muitas outras coisas.

A equipe da Alura estilizou uma tag para exibir a operação bem bonita na tela. A tag usada foi o `<h2>` . Podemos incluí-lo no `document.write` e realizar um novo teste.

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write("<h2>" + primeiroValor + " / " + segundoValor +
" = " + resultado + "</h2>")
}
```

Ficou sensacional, mas temos um problema. Nossa calculadora realiza apenas a divisão quando a operação é 1. Mas e as outras operações?

Podemos verificar o caso de **se** a operação **não** for igual a `1` . Esse **se não** é chamado por meio do comando `else` .

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write("<h2>" + primeiroValor + " / " + segundoValor +
" = " + resultado + "</h2>")
} else
```

Agora, precisamos de uma nova condição: comparar **se** a variável `operacao` é **igual** a `2` , realizar a operação de multiplicação com o símbolo do asterisco (`*`) e alterar o texto do `document.write` :

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
```

```
  document.write("<h2>" + primeiroValor + " / " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 2) {
  var resultado = primeiroValor * segundoValor
  document.write("<h2>" + primeiroValor + " x " + segundoValor +
" = " + resultado + "</h2>")
}
```

Podemos realizar o mesmo para as operações soma e subtração:

```
if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write("<h2>" + primeiroValor + " / " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 2) {
  var resultado = primeiroValor * segundoValor
  document.write("<h2>" + primeiroValor + " x " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 3) {
  var resultado = primeiroValor + segundoValor
  document.write("<h2>" + primeiroValor + " + " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 4) {
  var resultado = primeiroValor - segundoValor
  document.write("<h2>" + primeiroValor + " - " + segundoValor +
" = " + resultado + "</h2>")
}
```

Para finalizar, pense no seguinte cenário: assim que o programa inicia, uma pessoa entra com `10` no primeiro valor, `2` no segundo valor e escolhe a opção **5**. O que será que vai acontecer?

Nosso calculadora realiza quatro operações, e não temos a opção **5**. Quando esta opção for escolhida, nada vai acontecer. Podemos incluir uma condição informando que a operação é inválida caso a opção dada seja diferente de 1, 2, 3 e 4.

Para isso, vamos usar apenas o comando `else` . Ou seja, se a

operação não for com 1, 2, 3 e nem com 4, nossa calculadora indicará "opção inválida", sem precisar saber qual opção foi atribuída.

```
else {
  document.write("<h2>Opção inválida</h2>")
}
```

Aqui está o código completo desta aula:

```
var primeiroValor = parseInt(prompt("Digite o primeiro valor:"))
var segundoValor = parseInt(prompt("Digite o segundo valor:"))

var operacao = prompt("Digite 1 para fazer uma divisão, 2 para mu
ltiplicação, 3 para soma e 4 para subtração: ")

if (operacao == 1) {
  var resultado = primeiroValor / segundoValor
  document.write("<h2>" + primeiroValor + " / " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 2) {
  var resultado = primeiroValor * segundoValor
  document.write("<h2>" + primeiroValor + " x " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 3) {
  var resultado = primeiroValor + segundoValor
  document.write("<h2>" + primeiroValor + " + " + segundoValor +
" = " + resultado + "</h2>")
} else if (operacao == 4) {
  var resultado = primeiroValor - segundoValor
  document.write("<h2>" + primeiroValor + " - " + segundoValor +
" = " + resultado + "</h2>")
} else {
  document.write("<h2>Opção inválida</h2>")
}

// escrevendo na tela - document.write()
// concatenação (juntar palavra com variáveis) - ("palavra" + var
iavel)
// == comparação é diferente do = que usamos para fazer atribuiçã
o

// if = se
```

```
// else = se não
// else if = se não se
```

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

2.3 RESUMO

Vamos repassar todos os passos do programa:

1. Interagir com usuário para receber os valores que serão calculados;
2. Criar a lógica para saber qual operação será executada;
3. Comparar o valor da operação escolhida para descobrir qual função será realizada;
4. Exibir uma mensagem de erro, caso a operação escolhida seja inválida;
5. Exibindo os valores escolhidos e o resultado da operação com document.write;

Capítulo 3

MENTALISTA

3.1 OBJETIVO DA AULA

Criar um programa que sorteia um número secreto entre 0 e 10, em que o usuário possui três tentativas para acertar esse número. Para auxiliar quem joga, quando o chute é incorreto dizemos se ele é maior ou menor que o número secreto. Quando o chute é correto, informamos o fim do jogo e o valor do número secreto. **Neste programa, focamos no uso do while, condicionais e comparação entre variáveis**.

3.2 PASSO A PASSO

Para iniciar esse programa, vamos criar uma variável chamada numeroSecreto e atribuir um número aleatório entre 0 e 10. Vamos utilizar uma função chamada `Math.random` :

```
> Math.random()
<- 0.20982489919137293

> Math.random()
<- 0.43822764751936005

> Math.random()
<- 0.2983069465358348

> Math.random()
```

```
<- 0.6899012983471291
```

Segundo a documentação, a função Math.random retorna um número pseudo-aleatório no intervalo entre 0 e 1, sendo que o valor 1 é exclusivo. Como queremos um número inteiro, vamos multiplicar a função Math.random por 10, movendo o ponto para esquerda:

```
> Math.random() * 10
<- 3.029855123323566

> Math.random() * 10
<- 4.808136849238727
```

Para finalizar, vamos utilizar a função `parseInt` que, basicamente, vai fazer a conversão para número inteiro, removendo os números depois do ponto.

```
var numeroSecreto = parseInt(Math.random() * 10)
```

Para ver o número secreto, você pode usar o código `console.log(numeroSecreto)` . Podemos usar o console.log para realizar diversas verificações. Lembrando que Você pode usar a função `alert(numeroSecreto)` ou `console.log(numeroSecreto)` para ver o número secreto.

Vamos criar uma nova variável para armazenar as 3 tentativas para acertar o número secreto.

```
var tentativas = 3
```

Enquanto tentativas for maior que 0, queremos continuar nosso jogo. Para criar este comportamento, vamos utilizar o comando `while` (que significa "enquanto" em português).

```
while (tentativas > 0) {
```

```
}
```

Para receber o chute da pessoa que está jogando, vamos criar uma nova variável e utilizar a função `prompt` com "Digite um número entre 0 e 10". Como queremos que o valor digitado seja um número inteiro, vamos utilizar a função `parseInt` , como ilustra o código abaixo.

```
while (tentativas > 0) {
  var chute = parseInt(prompt("Digite um número entre 0 e 10"))
}
```

Vamos utilizar a condição `if` e verificar se o número secreto é igual ao chute. Essa comparação é feita com dois símbolos de igual (==). Lembrando que quando usamos apenas um sinal de igual, estamos atribuindo um valor (=). Se o número secreto for igual ao chute, exibiremos um alerta informando que houve acerto e pausaremos o while com o comando `break` .

```
while (tentativas > 0) {
  var chute = parseInt(prompt("Digite um número entre 0 e 10"))

  if (numeroSecreto == chute) {
    alert("Acertou")
    break
  }
}
```

Vamos criar mais duas condições para verificar se o chute é maior ou igual ao número secreto usando o comando `else if` e a condição entre parênteses (não esqueça os bigodes, ou seja, de abrir e fechar as chaves). Além disso, vamos remover as tentativas nesses casos de erro.

```
  if (numeroSecreto == chute) {
    alert("Acertou")
    break
  } else if (chute > numeroSecreto) {
```

```
    alert("O número secreto é menor")
    tentativas = tentativas - 1
  } else if (chute < numeroSecreto) {
    alert("O numero secreto é maior")
    tentativas = tentativas - 1
  }
```

Assim que as tentativas acabam, precisamos informar a pessoa que está jogando que o jogo acabou e revelar qual era o número secreto. Para isso, fora fo while vamos verificar se o chute não for igual ao número secreto, exibimos a mensagem de Game over e revelamos o número secreto.

Para verificar se o chute não é igual ao número secreto, usamos o sinal `!=` , como ilustra o código abaixo:

```
if (chute != numeroSecreto){
  alert("Suas tentativas acabaram. O número secreto era " + numer
oSecreto)
}
```

O código completo fica assim:

```
var numeroSecreto = parseInt(Math.random() * 10)
//alert(numeroSecreto)
var tentativas = 3

while (tentativas > 0) {
  var chute = parseInt(prompt("Digite um número entre 0 e 10"))

  if (numeroSecreto == chute) {
    alert("Acertou")
    break
  } else if (chute > numeroSecreto) {
    alert("O número secreto é menor")
    tentativas = tentativas - 1
  } else if (chute < numeroSecreto) {
    alert("O numero secreto é maior")
    tentativas = tentativas - 1
  }
}
```

```
if (chute != numeroSecreto){
  alert("Suas tentativas acabaram. O número secreto era " + numer
oSecreto)
}
```

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

3.3 RESUMO

Vamos repassar todos os passos do programa:

- Fazer testes utilizando console.log();
- Criar a lógica por trás do "chute" com if, else if e else;
- Criar a lógica para controlar a quantidade de tentativas com while;
- Utilizar a função Math.random() do JavaScript para criar números aleatórios;
- Praticar o uso em conjunto de condicionais e loops, e refletir sobre em que momento cada trecho de código é executado.

Capítulo 4

ALURAFLIX

4.1 OBJETIVO DA AULA

Criar um programa que permite você listar os alguns filmes em formato de imagem com a plataforma Aluraflix! **Nesse projeto, focamos em arrays**.

4.2 PASSO A PASSO

Vamos começar criando três variáveis para armazenar os primeiros valores, que no caso serão nomes de um filme:

```
var filme1 = “Star Wars”
var filme2 = “Toy Story”
var filme3 = “Interestellar”
```

Agora, vamos imprimi-los no console da seguinte maneira:

```
console.log(filme1)
console.log(filme2)
console.log(filme3)
```

Mesmo que dê tudo certo, é possível perceber que quanto mais filmes adicionamos no nosso “catálogo” do Aluraflix, mais linhas de código temos que usar para armazenar e imprimir as variáveis. Sendo assim, podemos utilizar uma **variável especial, chamada array**. Ele nada mais é do que uma **variável que pode armazenar**

um conjunto de outras variáveis. Vamos ver na prática como isso funciona, declarando o array e imprimindo no console:

```
var filmes = [“Star Wars”, “Toy Story”, “Interestellar”]
console.log(filmes)
```

> Dica: existe outra forma de armazenar diferentes valores dentro de um array, pois nem sempre já temos todos eles definidos no momento em que os declaramos. Nestes casos, podemos declarar o array vazio e ir adicionando cada valor com a função `push`.

```
var filmes = []
filmes.push(“Star Wars”)
filmes.push(“Toy Story”)
filmes.push(“Interestellar”)

console.log(filmes)
```

Mas e se quisermos apenas imprimir um valor específico desse array? O que devemos fazer?

Para isso, podemos chamar esse valor pelo seu índice. **É importante destacar que o primeiro elemento sempre terá índice `0`**, o segundo índice terá `1`, o terceiro índice, `2`, e assim por diante. Eles servem como identificadores únicos de cada valor dentro do array, então você sempre saberá qual valor será impresso, da seguinte forma:

```
console.log(filmes[0]) // ”Star Wars”
console.log(filmes[1]) // ”Toy Story”
console.log(filmes[2]) // ”Interestellar”
```

Porém, ainda caímos no mesmo problema de antes. Se

decidirmos adicionar 200 filmes, precisaremos de 200 linhas de `console.log()` para imprimir cada um deles. **Sendo assim, existe um laço de repetição (*loop*) que podemos utilizar, diferente do `while` visto na aula anterior, o `for` .** A sintaxe dele é a seguinte:

```
for (var i = 0; i < 3; i++) {
    //aqui serão escritos os comandos a serem executados dentro d
o laço de repetição
}
```

Não se assuste, vamos entender exatamente o que está acontecendo em cada parte desse código agora.

É possível perceber que existem três divisões dentro do parênteses do `for` , separados por ponto e vírgula. A primeira delas, `var i = 0` , está declarando uma nova variável `i` com o valor 0. **Essa será a variável utilizada como contadora do laço de repetição.**

Já na segunda divisão, temos `i < 3` . Essa é a condição para que o programa entre no loop e execute os comandos dentro dele. Por fim, na última divisão temos `i++` , que é `i = i + 1` escrito de forma **mais compacta e simples**.

Então, agora podemos sintetizar melhor o que está acontecendo: o programa cria uma nova variável `i` , e atribui o valor `0` a ela. Ele verifica se a variável `i` é menor que 3. Caso seja, todos os comandos dentro das chaves serão executados. No final dessa execução, o valor `i` é aumentado por 1, de 0 passando para 1 e assim por diante, até ele deixar de ser menor que 3 e deixar de entrar no laço de repetição.

Agora que entendemos o `for` , podemos adicionar um

comando dentro dele para vermos na prática.

```
for (var i = 0; i < 3; i++) {
    console.log(i)
}
```

Estamos imprimindo o valor de `i` cada vez que o programa entra no loop. Sendo assim, podemos perceber que ele vai diminuindo essa quantidade de vezes com o passar do tempo, a cada vez que entra no laço.

Vamos testar utilizar o `for` para imprimir os valores do array sem precisarmos escrever várias linhas de `console.log()`, assim:

```
for (var i = 0; i < 3; i++) {
    console.log(filmes[i])
}
```

Cada vez que entramos no loop, imprimimos o valor do array com o índice igual ao valor de `i`, percorrendo todos os elementos dentro dele.

Porém, ainda podemos melhorar o nosso código, pois dessa forma teremos sempre que saber a quantidade de elementos a serem colocados dentro da condição para compararmos com a variável `i`.

Vamos utilizar o `.length`, que vai retornar o número de elementos que temos dentro de um array, da seguinte forma:

```
for (var i = 0; i < filmes.length; i++) {
    console.log(filmes[i])
}
```

Tudo certo até aqui? Agora podemos apagar isso tudo que vimos e começar a implementar o projeto dessa aula!

A ideia agora é, em vez de colocarmos os nomes dos filmes para aparecerem na tela, mostrarmos as imagens de cada um deles.

Para isso, podemos buscar as capas dos filmes que queremos na internet e copiarmos o endereço dessas imagens. Isso pode ser feito clicando com o botão direito do mouse e selecionando a opção "Copiar endereço da imagem".

Podemos colar este endereço copiado dentro do nosso novo array:

```
var filmes = [“https://m.media-amazon.com/images/M/MV5BZDgzNzdmNj
EtMDAwMC00M2FiLTlkMTEtMDE0MDIyNTEwYmJlXkEyXkFqcGdeQXVyMjY3MjUzNDk
@._V1_UY268_CR12,0,182,268_AL_.jpg”]
```

Parece um pouco assustador, mas o que você deve entender olhando para esse link, é que ele tem no final um `.jpg`, indicando que trata-se realmente de uma imagem.

Podemos agora adicionar as imagens dos outros filmes dentro do nosso array:

```
var listaFilmes = [“https://m.media-amazon.com/images/M/MV5BZDgzN
zdmNjEtMDAwMC00M2FiLTlkMTEtMDE0MDIyNTEwYmJlXkEyXkFqcGdeQXVyMjY3Mj
UzNDk@._V1_UY268_CR12,0,182,268_AL_.jpg”, “https://m.media-amazon
.com/images/M/MV5BMDU2ZWJlMjktMTRhMy00ZTA5LWEzNDgtYmNmZTEwZTViZWJ
kXkEyXkFqcGdeQXVyNDQ2OTk4MzI@._V1_UX182_CR0,0,182,268_AL_.jpg”, “
https://m.media-amazon.com/images/M/MV5BZjdkOTU3MDktN2IxOS00OGEyL
WFmMjktY2FiMmZkNWIyODZiXkEyXkFqcGdeQXVyMTMxODk2OTU@._V1_UX182_CR0
,0,182,268_AL_.jpg”]
```

Já aprendemos a imprimir valores de um array com o `for`, então vamos fazer isso:

```
for (var i = 0; i < listaFilmes.length; i++){
    //comando para imprimir os filmes
}
```

Dessa vez, não vamos imprimir os valores dentro do nosso console com `console.log()`, e sim usar um comando para que as imagens sejam exibidas na tela, com `document.write()`. O nosso programa ficará assim:

```
for (var i = 0; i < listaFilmes.length; i++){
    document.write(listaFilmes[i])
}
```

Ok, acho que não ficou da forma que queríamos, certo? Isso porque queremos que as imagens apareçam na tela, e não os links escritos em forma de texto.

Para resolvermos isso, precisaremos de um pouquinho de HTML. Criaremos elementos na nossa página que indiquem que aquilo é uma imagem, e não mais um texto. Colocaremos uma nova tag `<img>` dentro do nosso `document.write()`:

```
for (var i = 0; i < listaFilmes.length; i++){
    document.write("<img src=" + listaFilmes[i] + ">")
}
```

O `src` dentro da nossa tag `img` é a propriedade que recebe o endereço da imagem, que no caso é o nosso valor dentro do array.

> Legal né? O interessante agora é você colocar vários outros filmes, com a intenção de criar realmente um catálogo com os que mais gosta, ou os que indica para alguém assistir!

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

4.3 RESUMO

Vamos repassar todos os passos do programa:

- Criar uma primeira array de filmes usando a sintaxe [];
- Utilizar o método filmes.push("Nome Do Filme") para inserir um novo elemento na lista (ou seja, um novo filme na array);
- Descobrir a quantidade de elementos em uma array com o método array.length;
- Selecionar elementos de uma array utilizando a sintaxe array[número], lembrando sempre que o primeiro índice começa com zero, ou seja, array[0] para o primeiro elemento;
- Utilizar a instrução for para iterar, ou seja, percorrer todos os elementos de uma array;
- Criar uma array com imagens de pôsters de alguns filmes que gostamos;
- Montar a lógica do programa que vai iterar esta array de filmes e exibir cada um deles na tela, integrando o for do JavaScript com a tag `img` do HTML.

CAPÍTULO 5

ALURAFLIX, BOTÕES VALIDAÇÕES E FUNÇÕES

5.1 OBJETIVOS DA AULA

Criar um programa que armazene a imagem de um filme, caso a entrada seja uma imagem. Caso contrário, informar ao usuário que a imagem é inválida. Fazemos essa verificação pedindo ao JavaScript que localize no fim do link o trecho de texto `.jpg` (nesse caso, não serão aceitas imagens em .png). Criar um botão que execute uma função para validar o link da imagem do filme.

> Neste programa, focaremos em aprender e praticar FUNÇÕES e como vincular o JavaScript a um botão no HTML para executar uma função no momento do clique.

5.2 PASSO A PASSO

Primeiro vamos pensar no que o programa deve fazer: uma listagem de filmes composta por imagens (de pôsteres, personagens, etc). Quem monta a lista é o próprio usuário, então

precisamos prever **de que forma o usuário vai inserir os itens da lista no programa**. Para isso, é bem comum utilizarmos um campo no HTML e "unir" essa ponta (a parte que o usuário interage) com o JavaScript (onde é feito o processamento do programa).

A segunda parte é utilizar o JavaScript para modificar o HTML que exibe as informações da página. Mas — esse é um detalhe importante — essa etapa só deve acontecer depois que o usuário inserir uma informação no campo de input (no caso, a imagem de um filme). Se deixarmos esse código solto, ele não vai esperar o usuário inserir o dado necessário e vai executar com erro. É aí que entram as funções.

Função para adicionar filme na lista

Pensando sempre no fluxo do programa, não queremos que nada seja processado enquanto o usuário não clicar no botão para adicionar um novo filme - vamos considerar aqui que o usuário só vai clicar no botão depois de adicionar o endereço de uma imagem de filme no input.

Esse é um caso para uso de funções: existem muitas razões para organizarmos nosso código em funções, e um deles é justamente poder **controlar** quando um trecho de código é executado. No caso do nosso programa, se não colocarmos o código para adicionar um novo filme na lista dentro de uma função, o código seria executado logo no carregamento da página, e não daria nem tempo do usuário preencher qualquer dado! O código seria executado com erro.

Então, a primeira coisa a fazer é criar uma função para que um

certo trecho de código dentro dela só seja executado no clique do botão.

O código responsável por avisar o HTML que existe uma função para ser chamada no clique do botão é o atributo `onclick="adicionarFilme()"`:

```
<input type="text" id="filme" name="filme" placeholder="Insira en
dereço de imagem">
<button onClick="adicionarFilme()">Adicionar Filme</button>
```

Mas por enquanto essa função não existe. Vamos criá-la no arquivo `script.js`:

```
function adicionarFilme() {
 var campoFilmeFavorito = document.querySelector('#filme')
 var filmeFavorito = campoFilmeFavorito.value
 if (filmeFavorito.endsWith('.jpg')) {
     listarFilmesNaTela(filmeFavorito)
 } else {
     alert("Imagem inválida")
 }
 campoFilmeFavorito.value = ""
}
```

Esta função está executando os seguintes passos na lógica que imaginamos para o programa: 1 . Criando uma variável para encontrar o "local" no HTML (ou seja, no conteúdo exibido na tela) onde o usuário vai inserir o link para a imagem. Fazemos isso através do identificador `'#filme'` que está identificando o `input` no HTML (veja o código acima).

```
var campoFilmeFavorito = document.querySelector('#filme')
```

1. Usamos a palavra-chave `.value` para que o JavaScript consiga capturar o texto que foi inserido dentro do campo input.

```
var filmeFavorito = campoFilmeFavorito.value
```

1. Agora que já conseguimos capturar o link que o usuário inseriu no input, hora de validar se é um link de imagem válido, com final `.jpg`. Fazemos isso usando o condicional `if` e a ferramenta `.endsWith()`, que podemos traduzir literalmente para "termina com". Ou seja, se o texto (`string`) da variável que recebemos termina com o trecho `.jpg`.

```
 if (filmeFavorito.endsWith('.jpg')) {
     listarFilmesNaTela(filmeFavorito)
 } else {
     alert("Imagem inválida")
}
```

No código acima, passamos uma instrução para o caso de sucesso, ou seja, se o link de imagem for válido. Essa instrução é `listarFilmesNaTela(filmeFavorito)`. Se lermos o código novamente, vemos que `filmeFavorito` é a variável que "guarda" o link com a imagem do filme que o usuário inseriu na tela e pegamos lá do HTML. Mas o que essa variável está fazendo?

Aqui, vemos as funções novamente em ação. Não só podemos dizer exatamente quando um trecho de código vai ser executado, como podemos organizar melhor o programa, separando cada parte da lógica em uma função diferente, passando para cada uma delas as informações necessárias para funcionar. Essas informações são passadas entre parênteses, em forma de dados (números, strings, etc) ou variáveis — chamamos essas informações de **parâmetros** ou **argumentos**.

```
listarFilmesNaTela(filmeFavorito)
```

Ainda não escrevemos a função `listarFilmesNaTela`, vamos

fazer isso em seguida. Podemos executar funções dentro de outras funções!

1. Lembrando: temos que passar para o computador exatamente todos os passos do algoritmo. Então, para "limpar" o campo depois de clicar no botão, temos que avisar que o `.value` (que antes tinha o valor do link que o usuário inseriu) agora não tem mais valor de texto, ou o que chamamos também de "string vazia":

```
campoFilmeFavorito.value = ""
```

Função para exibir o filme na tela

De acordo com os passos do programa, depois de "pegarmos" o link da imagem, temos que inclui-la na tela sem apagar o que já está lá. Vamos escrever a função `listarFilmesNaTela` :

```
function listarFilmesNaTela(filme) {
 var listaFilmes = document.querySelector('#listaFilmes')
 var elementoFilme = "<img src=" + filme + ">"
 listaFilmes.innerHTML = listaFilmes.innerHTML + elementoFilme
}
```

1. De forma parecida com o que fizemos antes, vamos usar o identificador `id="listaFilmes"` para localizar a parte da tela (ou seja, do HTML) onde queremos adicionar a imagem do filme escolhido.

```
<div id="listaFilmes"></div>
```

No JavaScript, criamos uma variável para salvar as informações que existem na tela nesse momento:

```
var listaFilmes = document.querySelector('#listaFilmes')
```

1. Adicionando um filme novo à lista (ou incluindo o primeiro se a lista estiver vazia): aqui utilizamos o JavaScript para criar uma nova "tag" de imagem de HTML. Uma vez que o JavaScript inserir esta nova linha no HTML, este trecho de código será reconhecido e exibido como imagem.

```
var elementoFilme = "<img src=" + filme + ">"
```

Esta forma de escrever, misturando "strings" (ou seja, dados de texto) com variáveis, parece um pouco estranha. Porém é bastante comum utilizarmos formas similares a esta para atualizar e modificar elementos da tela (HTML) com JavaScript.

Se você se perguntou o que é a variável `filme` neste código, observe novamente a primeira linha da função `listarFilmesNaTela` : `filme` é o nome que damos ao **parâmetro** (ou **argumento**) da função. Ou seja, quando esta função for executada — ou "chamada", como costumamos dizer — a variável `filme` será substituída pela string que está sendo informada dentro do `if` da função anterior.

1. A última coisa a fazer é adicionar na tela a linha de HTML com a tag de imagem.

```
listaFilmes.innerHTML = listaFilmes.innerHTML + elementoFilme
```

Já havíamos criado antes a variável `listaFilmes` para salvar as tags de imagem que já existiam na tela. Agora estamos **reatribuindo** um valor a esta mesma variável: este novo valor é composto do `innerHTML` (mais sobre esta palavra-chave abaixo) somado à nova "tag" de imagem que acabamos de criar e salvar na variável `elementoFilme` . Quando trabalhamos com JavaScript e usamos o operador de soma `+` com dados de texto, os dois textos

vão ser unidos em um só.

Sobre o `innerHTML`

Quando utilizamos a palavra-chave `innerHTML`, normalmente estamos nos referindo a todo o conteúdo (*tags* e seus atributos) que é interno a determinado elemento HTML.

Ou, colocando de forma mais prática o uso que estamos fazendo no código do nosso programa, o `innerHTML` se refere a todo o conteúdo interno de determinado elemento. Por exemplo, o `innerHTML` do elemento identificado por `id="lista"`:

```
<div id="lista">

  <!-- inner HTML -->
  <p>item 1</p>
  <p>item 2</p>
  <p>item 3</p>
  <!-- fim do inner HTML -->

</div>
```

De acordo com o exemplo acima, vemos que `innerHTML` se refere a tudo que está entre entre as *tags* de abertura e fechamento do elemento `<div id="lista">`. Colocando de outra forma:

```
<div id="lista">innerHTML</div>
```

> Caso queira saber mais, você pode consultar a documentação sobre `innerHTML` no MDN.

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

5.3 RESUMO

Vamos repassar todos os passos do programa:

- Sintaxe e criação de funções no JavaScript;
- Integrando funções criadas no JavaScript com o HTML que está sendo exibido na tela;
- Condicionando a execução (ou "chamada") de uma função a um clique em um botão na tela;
- Usando o JavaScript para acessar o que está sendo exibido na tela e pegar valores digitados pelo usuário com querySelector() e .value;
- Passar informações que as funções precisam para funcionar, através dos parâmetros;
- Utilizar o .endsWith() para verificar se um texto termina com determinados caracteres;
- Ver mais um exemplo de reatribuição de variável para "limpar" o texto do campo com "".

Capítulo 6

TABELA DE CLASSIFICAÇÃO E OBJETOS NO JAVASCRIPT

6.1 OBJETIVOS DA AULA

Criar um programa que mostre uma tabela informando as vitórias, empates, derrotas e a quantidade de pontos. Cada vitória soma 3 pontos e cada empate 1 ponto na tabela. Criar 3 botões chamados vitórias, empates e derrotas. Quando a vitória for clicada, adicionar 1 na quantidade de vitórias e calcular a quantidade de pontos com base nas regras acima. O mesmo com empate e derrota.

Neste programa, vamos aprender o que são objetos no Javascript.

6.2 PASSO A PASSO

Quando criamos uma página com HTML e CSS, todo o

conteúdo desta página não muda, a menos que troquemos seus valores no HTML. Isso pode ser ruim, pois, no nosso caso, quando uma vitória ou empate for alterado, não podemos esquecer de alterar também o valor dos pontos.

Sabendo disso, vamos realizar toda a manipulação dos dados desta tabela usando o Javascript. Quando uma vitória for computada, queremos adicionar o valor 1 ao número de vitória e recalcular os pontos do jogador ou jogadora. Mas como armazenar no Javascript o nome do jogador, o número de vitórias, empates, derrotas e os pontos?

Objetos no Javascript

Objetos são tipos de dados compostos: eles agregam vários valores em uma única unidade e nos permitem armazenar e recuperar esses valores por nome. Outra maneira de explicar isso é dizer que um objeto é uma coleção não ordenada de propriedades, cada uma delas com um nome e um valor. Os valores nomeados mantidos por um objeto podem ser valores como números e textos.

Ao observar os dados da tabela, observe que cada jogador possui um nome, a quantidade de vitórias, empates, derrotas e os pontos.

Nome	Vitórias	Empates	Derrotas	Pontos
Paulo	2	3	5	9
Rafa	3	2	1	11

Certo mais, como posso criar um objeto no Javascript que represente cada jogador?

Criando cada jogador como objeto no Javascript

Para criar cada jogador, vamos definir uma nova variável e atribuir o sinal de chaves `{ }`

```
var paulo = {}
```

O que queremos fazer agora é armazenar no objeto `paulo` seu nome, o número de vitórias, empates, derrotas e pontos. Para isso, vamos criar cada propriedade com seu `nome` , o sinal de dois pontos `:` e atribuir o valor de cada atributo separado por vírgula entre eles, como mostra o código abaixo:

```
var paulo = {
    nome: "Paulo",
    vitorias: 2,
    empates: 5,
    derrotas: 1,
    pontos: 0
}
```

> Dica: posso criar o objeto usando apenas uma linha também.

```
var paulo = {nome: "Paulo", vitorias: 2, empates: 5, derrotas: 1,
 pontos: 0}
```

Podemos criar a jogadora `rafa` , da mesma forma:

```
var rafa = {
    nome: "Rafa",
    vitorias: 3,
    empates: 5,
    derrotas: 2,
    pontos: 0
}
```

Em ambos jogadores, deixamos seus pontos zerados e vamos criar uma função que realiza o cálculo dos pontos. Mas como pegamos os valores de um objeto?

Atributos de um objeto

Para recuperar os valores de um objeto, podemos utilizar o `.` (ponto), seguido do nome do atributo que queremos ver. Vamos realizar alguns testes:

Para visualizar o número de vitórias da Rafa:

```
console.log(rafa.vitorias)
```

Para visualizar o número de empates ou o nome do Paulo:

```
console.log(paulo.empates)
console.log(paulo.nome)
```

Sendo assim, usamos o ponto para acessar os campos de um objeto.

Função para calcular pontos

Para alterar o valor de um campo em um objeto, podemos usar o sinal de igual `=` e atribuir um novo valor. Observe o exemplo a seguir:

```
console.log(paulo.vitorias)
paulo.empates += 1
console.log(paulo.vitorias)
```

Dica: podemos também atribuir 1 com o código `paulo.empates++`

Observe que o valor da vitória foi alterado. O que podemos fazer é criar uma função que realiza o cálculo dos pontos e atribui o valor na variável pontos. Para função conseguir identificar de quem são os pontos, vamos passar entre os parênteses o objeto jogador.

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

Dentro da função, vamos criar uma variável chamada pontos e atribuir o número de vitórias do jogador multiplicado por 3 e somar com número de empates.

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

Toda a vez que essa função é executada, queremos que ela devolva os pontos já calculados. Para isso, vamos incluir a palavra `return` seguido do nome da variável `pontos`

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

Podemos calcular os pontos da Rafa e do Paulo, executando a função e atribuindo os pontos no campo `pontos` de cada objeto.

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

Maravilha! Isso ficou realmente incrível, mas não estamos vendo nada na tela. Isso que faremos a seguir.

Exibindo os jogadores na tela

Queremos exibir todos os jogadores na tela de uma só vez. Para isso vamos criar uma lista com todos nossos jogadores.

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

Além disso, vamos criar uma função que exibe todos os jogadores, passando nossa lista de jogadores.

```
function exibirJogadoresNaTela(jogadores) {}
```

Como queremos exibir os objetos que estão no Javascript na página HTML, podemos usar o `innerHTML` para isso. Vamos criar uma variável chamada `html` e juntar as informações como vitórias, empates, derrotas e pontos nela.

```
function exibirJogadoresNaTela(jogadores) {
    var html = ""
}
```

Podemos criar um `for` para exibir todos os jogadores de nossa lista.

```
function exibirJogadoresNaTela(jogadores) {
    var html = ""
    for (var i = 0; i < jogadores.length; i++) {}
}
```

Vamos varrer nossa lista icluíndo uma `tr` indicando uma nova linha e uma `td` para cada atributo do jogador.

```
function exibirJogadoresNaTela(jogadores) {
    var html = ""
    for (var i = 0; i < jogadores.length; i++) {
```

```
        html += "<tr><td>" + jogadores[i].nome + "</td>"
        html += "<td>" + jogadores[i].vitorias + "</td>"
        html += "<td>" + jogadores[i].empates + "</td>"
        html += "<td>" + jogadores[i].derrotas + "</td>"
        html += "<td>" + jogadores[i].pontos + "</td>"
    }
}
```

Para posicionar cada jogador de forma correta, criamos um elemtento no HTML com um ID chamado `tabelaJogador` , para exibir as informações de nossa tabela. Vamos recuperar ele elemento do HTML com `document.getElementById` e atribuir com `innerHTML` a variável `html` que criamos.

```
function exibirJogadoresNaTela(jogadores) {
    var html = ""
    for (var i = 0; i < jogadores.length; i++) {
        html += "<tr><td>" + jogadores[i].nome + "</td>"
        html += "<td>" + jogadores[i].vitorias + "</td>"
        html += "<td>" + jogadores[i].empates + "</td>"
        html += "<td>" + jogadores[i].derrotas + "</td>"
        html += "<td>" + jogadores[i].pontos + "</td>"
    }
    var tabelaJogadores = document.getElementById('tabelaJogadore
s')
    tabelaJogadores.innerHTML = html
}
```

Estamos vendo os elementos na tela, mas… Onde estão os botões?

Botões de cada objeto

Nossa função exibe os jogadores, mas não exibe os botões para adicionar vitórias, empates ou derrotas. Precisamos ajustar isso.

Vamos adicionar na função que exibe os jogadores na tela os botões e o concatenando o índice de cada jogador:

```
 html += "<td><button onClick='adicionarVitoria(" + i + ")'>Vitór
ia</button></td>"
 html += "<td><button onClick='adicionarEmpate(" + i + ")'>Empate
</button></td>"
 html += "<td><button onClick='adicionarDerrota(" + i + ")'>Derro
ta</button></td></tr>"
```

Veja como ficou nossa função completa:

```
function exibirJogadoresNaTela(jogadores) {
    var html = ""
    for (var i = 0; i < jogadores.length; i++) {
        html += "<tr><td>" + jogadores[i].nome + "</td>"
        html += "<td>" + jogadores[i].vitorias + "</td>"
        html += "<td>" + jogadores[i].empates + "</td>"
        html += "<td>" + jogadores[i].derrotas + "</td>"
        html += "<td>" + jogadores[i].pontos + "</td>"
        html += "<td><button onClick='adicionarVitoria(" + i + ")
'>Vitória</button></td>"
        html += "<td><button onClick='adicionarEmpate(" + i + ")'
>Empate</button></td>"
        html += "<td><button onClick='adicionarDerrota(" + i + ")
'>Derrota</button></td></tr>"
    }
    var tabelaJogadores = document.getElementById('tabelaJogadore
s')
    tabelaJogadores.innerHTML = html
}
```

Temos os botões aparecendo na linha de cada jogador. Porém, quando clicamos um erro informa que essas funções não foram definidas.

Função adicionar vitória

Vamos criar uma função chamada `adicionaVitoria` (mesmo nome do onclick do botão), variável chamada `jogador` e atriibuir da lista de jogadores no íindice `i` .

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

```
}
```

Agora que sabemos o jogador, vamos somar `1` na quantidade de vitória, adicionar o pontos no jogador e chamar a função para exibir os jogadores na tela.

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

Função adicionar empate

Para o botão de empate, vamos criar uma função chamada `adicionarEmpate` , somar 1 no empate do jogador com código `jogador.empate++` e assim como fizemos na função acima, atribuir os pontos calculados no jogador e chamar a função `exibirJogadoresNaTela` .

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

Função adicionar derrota

Está função será semelhante a outras, mas como não faremos nenhuma manipulação nos pontos do jogador, não precisamos atribuir nada nos pontos do jogador.

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

```
}
```

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

6.3 RESUMO

Vamos repassar todos os passos do programa:

- Remover o código estático do HTML;
- Criar um objeto no Javascript para cada jogador;
- Criar uma função que receba um objeto como parâmetro para calcular os pontos;
- Exibir o objeto na página HTML;
- Criar uma função para adicionar vitória;
- Criar uma função para adicionar empate e outra para adicionar derrota;
- Recalcular os pontos quando vitória ou empate for adicionado.

Capítulo 7

SUPER TRUNFO - LÓGICA DO JOGO

7.1 OBJETIVO DA AULA

Depois de toda essa prática, já podemos pensar em algum projeto um pouco mais extenso para aplicar tudo que vimos até agora e adicionar mais funcionalidades - as chamadas *features*. E esse projeto será um jogo clássico, o Super Trunfo!

Para criar este jogo, vamos focar em outra estrutura de dados muito utilizada, o `objeto` , além de mais funções e operadores.

7.2 PASSO A PASSO

Como das vezes anteriores, vamos fazer o exercício de colocar o processo em um *fluxo* com as seguintes informações:

- O que deve acontecer no programa em caso de sucesso;
- O que deve acontecer no programa em caso de falha;
- De que dados esse programa precisa;
- De onde virão esses dados.

Trabalhando com dados em objetos

Vimos anteriormente que é possível utilizar arrays `[]` para guardar conjuntos de dados. Isso funciona bem, por exemplo, para listas, quando os dados são do mesmo "tipo":

```
var arrayTelefones = [“11991231234”, “21992342342”, “31993453453”
];
```

Mas em alguns casos temos que passar grupos de dados um pouco mais complexos, por exemplo as informações de uma pessoa, por exemplo se for estudante de uma escola: estudantes têm nomes, datas de nascimento, e-mails de contato, em que classe estão, etc. Imaginando esses dados em uma tabela:

```
| Propriedade | Valor |
| --- | --- |
| nome | Nome Sobrenome |
| data de nascimento | 19/03/2008 |
| email | email@email.com |
|série | 5a B |
```

Não é o tipo de conjunto de dados que funcione muito bem em arrays, pois quando utilizamos arrays não é possível "classificar" cada um dos índices. Esse é um dos usos do *objeto*, que vamos ver aqui.

Para adaptarmos as informações de estudante acima para um *objeto*, é possível utilizar a seguinte sintaxe:

```
var estudante = {
 nome: "Nome Sobrenome",
 dataNascimento: "19/03/2008",
 email: "email@email.com",
 serie: "5a B"
}
```

Os objetos podem guardar dados de forma bastante complexa.

Por exemplo, se temos mais de uma informação de contato para cada estudante, podemos salvá-las juntas, usando objetos dentro de objetos:

```
var estudante = {
 nome: "Nome Sobrenome",
 dataNascimento: "19/03/2008",
 contato: {
   email: "email@email.com",
   telefone: "11923452345"
 },
 serie: "5a B"
}
```

Certo, mas como podemos "pegar" as informações de dentro de um objeto para usar no código?

Fluxo do programa

Primeiro vamos relembrar bem resumidamente os passos do jogo Super Trunfo, pois já vimos essa parte no vídeo da aula:

1. Cada oponente tira a primeira carta de seu monte;
2. A pessoa escolher qual atributo da carta quer usar para "apostar" na comparação;
3. As cartas são comparadas e vence o turno a pessoa que tiver a carta com o valor maior no atributo que tiver sido escolhido.

Para nosso primeiro código, simplificamos algumas partes do jogo, em comparação com a forma que jogamos ao vivo; nesta primeira versão, vamos jogar contra o computador.

Começar de forma simples, testando a lógica com poucas "funcionalidades", é a forma mais comum de trabalho durante o desenvolvimento de um projeto. Sempre incrementamos nosso código aos poucos!

Podemos traduzir os passos acima de uma forma mais parecida com o fluxo de código:

1. Iniciamos com uma lista de cartas pré-determinadas;
2. O programa sorteia 2 cartas a partir dessa lista, uma para quem está jogando e outra que vai ser a primeira carta da "pilha" do computador;
3. O programa exibe na tela a carta que foi sorteada, para quem está jogando possa escolher o atributo que vai "apostar";
4. O usuário que está jogando faz sua escolha;
5. O programa automaticamente compara com a outra carta sorteada (a carta do computador) e anuncia quem venceu a rodada.

Hora de colocar esses passos todos em formato de código!

7.3 ESCREVENDO O CÓDIGO

Os objetos para cada carta

Cada carta do Super Trunfo segue um padrão de nome, propriedade, etc, mas com seus próprios valores. Podemos pensar no exemplo abaixo:

```
var cartaSeiya = {
 nome: "Seiya de Pégaso",
 atributos: {
     ataque: 80,
     defesa: 60,
     magia: 90
 }
}

var cartaPokemon = {
 nome: "Bulbasauro",
 atributos: {
     ataque: 70,
     defesa: 65,
     magia: 85
 }
}

var cartaStarWars = {
 nome: "Lorde Darth Vader",
 atributos: {
     ataque: 88,
     defesa: 62,
     magia: 90
 }
}
```

Neste código, cada carta tem seu próprio objeto, salvo em sua própria variável. Já aprendemos anteriormente que uma array é um tipo de lista *ordenada* de elementos, então podemos usar estas duas estruturas de dados em conjunto, da seguinte forma:

```
// índice      0           1          2
var cartas = [ cartaSeiya, cartaPokemon, cartaStarWars ]
```

O programa vai ter que separar quais cartas serão sorteadas, então já vamos deixar as variáveis para a carta do computador e do usuário *declaradas, mas sem valor*:

```
var cartaMaquina
var cartaJogador
var cartas = [ cartaSeiya, cartaPokemon, cartaStarWars ]
```

Sorteando as cartas

Outra coisa que também já vimos foi como gerar números aleatoriamente. Então, agora que temos uma lista de cartas em uma array, é possível usar o JavaScript para determinar que cartas serão sorteadas!

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

Com o código acima, os resultados possíveis são os abaixo. Revise o conteúdo anterior sobre `Math.random()` caso precise relembrar!

```
cartas[0] // cartaSeiya
cartas[1] // cartaPokemon
cartas[2] // cartaStarWars
```

Antes de continuar, temos que garantir que os números sorteados não sejam os mesmos para jogador e computador! Ou seja, **enquanto** (*while*) as duas cartas sorteadas forem as mesmas - o que pode acontecer bastante, pois nosso baralho por enquanto só tem três cartas - o programa deve sortear uma carta nova:

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

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

No código acima, o programa sorteia primeiro a carta do computador, salva o número em `numeroCartaMaquina` e utiliza o recurso de array `array[numeroDoIndice]` para salvar o resultado em `cartaMaquina` .

Em seguida, o processo para sortear a carta do jogador deveria ser o mesmo, certo? Porém temos um passo a mais: uma *iteração* que verifica se o número sorteado (lembrando que, no caso do código acima, é um número entre 0 e 2) não é o mesmo, para que computador e jogador não acabem com o mesmo número de carta. Essa iteração está sendo feita com e ferramenta `while` ; ou seja, *enquanto* o número sorteado para computador e jogador forem iguais, peça ao programa para sortear novamente *apenas para o jogador*.

Lembrando que, uma vez dentro do *loop*, o JavaScript não sai até que esteja resolvido. No caso, até que os números sorteados sejam diferentes. Só depois disso o programa realmente define qual é a carta sorteada para o jogador.

Finalizando a função do sorteio

Agora que já temos a lógica principal da primeira parte do programa (sortear as cartas), podemos finalizar esta função, que será chamada/executada quando o usuário clicar no botão `<button onclick="sortearCarta()" id="btnSortear">Sortear carta</button>` .

No trecho de HTML acima, o botão está executando a função `sortearCarta()` , então vamos escrevê-la com o código que já temos:

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

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

```
    }
    cartaJogador = cartas[numeroCartaJogador]
    console.log(cartaJogador)

    document.getElementById('btnSortear').disabled = true
    document.getElementById('btnJogar').disabled = false
    // exibirOpcoes()
}
```

O código dessa função é em grande parte o que já vimos acima. Além de organizado e "cercado" por uma função, veja algumas linhas que acrescentamos no final:

```
    document.getElementById('btnSortear').disabled = true
    document.getElementById('btnJogar').disabled = false
    // exibirOpcoes()
```

Para ajudar o usuário a fazer decisões na tela, é comum habilitarmos e desabilitarmos as botões, campos de texto e etc. A propriedade `.disabled` (ou "desabilitado" em inglês) pode fazer isso para nós; Depois de sorteadas as cartas, pedimos ao JavaScript que desabilite o botão de sorteio para não ser clicado novamente enquanto o jogo não se resolve - e habilitar o botão de jogo. Se deixarmos esses botões livres para serem clicados pelo usuário **fora do momento certo** todo o programa pode ficar *bugado*.

> Você consegue imaginar algumas razões para esse controle que fazemos de quando e onde os usuários podem clicar e interagir com um programa na tela? Esta é uma boa hora para refletir sobre a importância dessas ferramentas!

Agora que a parte de sortear as cartas está definida, podemos passar para as próximas etapas do programa:

- Jogador escolhe um atributo;
- O programa faz a comparação com a carta sorteada para o computador.

Já podemos deixar a próxima função - estamos chamando aqui de `exibirOpcoes()` - chamada na última linha, porém "comentada" para não interferir no código por enquanto.

Percorrendo as opções/atributos

Quando programamos, sempre temos que lembrar que a quantidade de dados que temos que lidar pode ser grande, desde uma lista com 120 Pokémons até cadastros de clientes de um negócio. Nosso programa tem apenas três objetos, mas as pergunta são as mesmas para três cartas ou para 120 Pokémons:

1. Como "extrair" as informações que queremos de dentro de cada um dos objetos da lista, de forma automatizada?
2. Como exibir estas informações na tela?

Assim como aprendemos anteriormente o método `for` para *iterar* uma array, o JavaScript tem outros métodos tanto para iterar arrays quanto objetos. Um deles é o chamado `for...in` , que serve explicitamente para iteração de *objetos*:

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

/* neste ponto, você já pode tirar o comentário da última linha d
a função anterior, para que exibirOpcoes() possa ser chamada/exec
utada.
```

A sintaxe é parecida com a do `for` clássico, porém a variável que normalmente chamamos apenas de `i` (lembrando que é somente uma convenção, você poderia dar o nome que quiser) agora chama `atributo`. E o que é exatamente `(var atributo in cartaJogador.atributos)` ou, traduzindo para o português, "variável `atributo` em `cartaJogador.atributos"?

Primeira coisa é entender de onde estão saindo os dados. No caso acima, `cartaJogador` é uma variável que está definida no início do código, em teoria salvando um objeto com os dados de uma carta do jogo… Você pode sempre recorrer ao `console.log(cartaJogador)` para confirmar!

E quando falamos de `cartaJogador.atributos`? Aqui começamos a lidar um pouco mais com os objetos e como acessar os dados dentro deles. Vamos rever como estão estruturados os objetos de cada carta, com um exemplo:

```
{
   nome: "Seiya de Pégaso",
   atributos: {
       ataque: 80,
       defesa: 60,
       magia: 90
   }
}
```

O objeto acima é uma das opções que podem ser sorteadas para a variável `cartaJogador`, de acordo com a lógica que vimos anteriormente. O JavaScript utiliza o que chamamos de *notação de ponto* para dizer ao código como acessar propriedades dentro de um objeto. Teste as seguintes opções em seu código:

```
console.log(cartaJogador.nome) //texto (string) com o nome na car
ta
console.log(cartaJogador.atributos) // objeto com os atributos da
```

```
carta
```

Juntando tudo: O método `for...in` serve para iterar em objetos; no nosso código, o objeto em questão é o que está dentro de `cartaJogador.atributos` , ou seja, a lista de "poderes" de cada personagem. Pensando em **iteração**, ou seja, em pedir ao JavaScript para percorrer uma lista de informações, podemos concluir que essa iteração vai acontecer nesta parte do código:

```
   atributos: {
       ataque: 80,
       defesa: 60,
       magia: 90
   }
```

> Você vai perceber cada vez mais, durante seus estudos, que as estruturas de arrays e objetos muitas vezes são utilizadas em conjunto. O segredo é PRATICAR SEMPRE!

Exibindo as opções/atributos

Já descobrimos como o JavaScript pode acessar um objeto e "extrair" dados de dentro dele, então podemos exibir essas informações na tela para que o jogador escolha em qual "poder" da carta quer apostar.

```
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
}
```

A função `exibirOpcoes()` completa está acima. As linhas que adicionamos estão relacionadas aos métodos do JavaScript usados para acessar os elementos corretos no HTML e adicionar mais elementos sem sobrescrever o que já está na tela:

```
var opcoes = document.getElementById('opcoes')
var opcoesTexto = ""
```

A variável `opcoes` acessa uma tag específica do HTML que esteja identificada por `id="opcoes"` . Em seguida, criamos a variável `opcoesTexto` com um valor de que chamamos de *string vazia* - um valor de texto, porém sem conteúdo.

Agora, dentro do iterador `for...in` podemos substituir o `console.log()` anterior, adicionando com o operador `+=` mais um valor de string na variável `opcoesTexto` . Como fizemos anteriormente, utilizamos strings de texto com a sintaxe do HTML *concatenadas* (ou seja, `"valor da string" + variavel`) com variáveis para que o JavaScript crie o HTML de forma automática. A tag de HTML utilizada aqui é a `<input type='radio' >` , vamos falar mais sobre isso um pouco mais abaixo.

A última linha adicionada é `opcoes.innerHTML = opcoesTexto` ; depois da iteração, o JavaScript já pode acessar `innerHTML` do elemento que salvamos na variável `opcoes` e inserir o novo conteúdo. Quando isso acontece, o HTML que está na tela vai ser atualizado.

As opções já estão na tela e o usuário pode selecionar em uma das "bolinhas" que acompanham as opções… Mas por enquanto nada vai acontecer, pois ainda não avisamos o JavaScript que ele tem que fazer qualquer coisa com essa informação… Ou seja, salvar a opção escolhida e utilizá-la no programa.

O usuário escolhe um atributo

O HTML reconhece vários tipos de interação que o usuário pode fazer na tela: preencher um texto, clicar em um botão, selecionar o que chamamos de *checkboxes*. E existe um tipo de interação específico para o caso deste jogo, o `radio`. Escolhemos esta tag pois limita o usuário a escolher somente uma opção de uma lista que já está pronta, então quem escolher não consegue nem adicionar opções nem clicar em mais de uma.

Agora que:

1. As carta já foram sorteadas e
2. O jogador já pode escolher um "poder" para comparar

É hora de jogar mesmo! O botão já deve estar liberado graças ao código que fizemos anteriormente (com o `.disabled = false`). Falta escrever a função que vai ser executada com o clique nesse botão, que chamamos de função `jogar()`. Não esqueça de conferir no HTML se ela está sendo executada em `onclick="jogar()"`:

```
<form id="form">
<h2>Escolha o seu atributo</h2>
<div class="opcoes" id="opcoes"></div>
<button type="button" id="btnJogar" onclick="jogar()" disabled="f
alse">Jogar</button>
</form>
```

A função `jogar()` vai ser responsável pela lógica de fazer as comparações entre as cartas e definir quem ganhou ou perdeu a rodada. Mas temos que voltar uma etapa, pois esta função depende de uma informação que não temos ainda: qual foi o poder escolhido pelo jogador na tela.

Vamos então já criar a função `jogar()`, já criando também a função que vai buscar as informações no HTML:

```
function jogar() {
   var atributoSelecionado = obtemAtributoSelecionado()

}
```

A linha acima executa a função `obtemAtributoSelecionado()` (que ainda vamos escrever!) e salva as *informações de retorno* dela na variável `atributoSelecionado`. Veja mais sobre retornos de função abaixo!

Hora de criar a função `obtemAtributoSelecionado()`:

```
function obtemAtributoSelecionado() {
   var radioAtributo = document.getElementsByName('atributo')
   for (var i = 0; i < radioAtributo.length; i++) {
       if (radioAtributo[i].checked) {
           return radioAtributo[i].value
       }
   }
}
```

Neste ponto do aprendizado começamos a rever conceitos que já aprendemos antes e como reaproveitá-los em diversas situações.

A variável `radioAtributo` procura no HTML todos os elementos com `name='atributo'` e salva em uma *lista*. Falando em listas, já pensamos que esta lista deve ser *iterável*, utilizando `for` por exemplo.

> Você consegue localizar no HTML e no código que já escrevemos onde `name='atributo'` está sendo utilizado?

O `for` nós já vimos em ação e você pode incluir alguns `console.log()` para relembrar o que está acontecendo em cada parte da iteração. Agora veja as linhas:

```
if (radioAtributo[i].checked) {
    return radioAtributo[i].value
}
```

O `for` está percorrendo a lista de ítens de "poderes" e verificando qual deles tem a propriedade `checked`, uma propriedade automática que o HTML adiciona toda vez que o usuário clica no botão de rádio correspondente na tela. Caso encontre, o JavaScript vai **retornar `.value` deste elemento**. Mas o que isso significa?

A palavra-chave `return` é muito importante e tem diversos usos quando trabalhamos com funções. Neste primeiro momento, o importante é entender que é através dela que *o valor guardado em `radioAtributo[i].value` consegue ser acessado por outras funções, salvo em variáveis, etc*. Sem o `return`, o JavaScript pode fazer todas as operações dentro de uma função, mas outras partes do código que estão fora dela - ou seja, fora do bloco de código

`{}` da função - não conseguem acessar a informação.

Por último, o que é `value` nessa parte específica do código? É onde está o valor (em texto mesmo, ou *string*) de cada um dos "poderes" da carta. Dê uma olhada no código que criamos para a função `exibirOpcoes()` e veja como o atributo HTML `value=` está sendo gerado com o nome de cada um dos "poderes".

Finalizando a lógica do jogo

Por enquanto a função `jogar()` somente executa a função `obtemAtributoSelecionado()` e, por causa do `return`, consegue salvar o `value` do "poder" selecionado pelo usuário na tela em uma variável.

```
function jogar() {
   var atributoSelecionado = obtemAtributoSelecionado()

}
```

Os passos finais desta primeira versão de jogo são:

1. Comparar os "poderes" nas cartas sorteadas;
2. Informar o resultado para o jogador

Os valores dos poderes são numéricos, então podemos concluir que uma comparação entre dois números inteiros pode ter três resultados: *valor A é menor que valor B, valor A é maior que valor B ou empate (ambos os valores são iguais)*.

Já praticamos anteriormente com as ferramentas para comparar valores e também como passar **condições** ao JavaScript: *se tal condição se cumprir, vá por tal caminho; caso contrário, siga por outro caminho*. Vamos reaproveitar tudo aqui:

```
function jogar() {
   var atributoSelecionado = obtemAtributoSelecionado()

   if (cartaJogador.atributos[atributoSelecionado] > cartaMaquina
.atributos[atributoSelecionado]) {
       alert('Venceu. A carta do computador é menor')
   } else if (cartaJogador.atributos[atributoSelecionado] < carta
Maquina.atributos[atributoSelecionado]) {
       alert('Perdeu. A carta do computador é maior')
   } else {
       alert('Empatou!')
   }
   console.log(cartaMaquina)
}
```

E como exatamente o JavaScript consegue obter o **valor** de cada atributo para comparar? Através da sintaxe `objeto[“nomeDaPropriedade”]` quando sabemos exatamente o nome da propriedade ou `objeto[variavel]` (sem aspas) quando não temos como passar o nome da propriedade ou será feita uma iteração.

Faça os seguintes testes no seu código, logo abaixo da variável `cartaSeiya`:

```
//usamos colchete e aspas quando sabemos exatamente o nome da pro
priedade e queremos acessar o valor correspondente
console.log(cartaPaulo.atributos["ataque"])

//usamos colchete e uma variável quando não sabemos exatamente ou
 não temos como fixar o nome da propriedade, por exemplo em caso
de iteração, para acessar o valor correspondente a cada proprieda
de
for (atributo in cartaPaulo.atributos) {
   console.log(cartaPaulo.atributos[atributo])
}
```

Dica: agora é possível acessar os VALORES de cada propriedade (ou seja, cada "poder" das cartas) e aí sim fazer a comparação entre o valor do "poder" na carta sorteada para o jogador e para o computador. O fluxo do código que deve ser executado para cada resultado da comparação pode ser resolvido com `if/else if/else`.

Observe um detalhe na execução deste código: separamos o código em funções para que só sejam executados no momento certo e podemos ver um exemplo disso na função `obtemAtributoSelecionado()`; esta função está sendo chamada/executada a partir da execução da função `jogar()`. Como a própria função `jogar()` não é executada antes do jogador clicar no botão, não há o risco da função `obtemAtributoSelecionado()` ser executada sem os dados necessários, ou em um momento errado.

Esta primeira versão do Super Trunfo já tem bastante código, então vamos parar por aqui e respirar um pouco.

7.4 PARA SABER MAIS

Os temas que estamos vendo nestas aulas vão te acompanhar durante toda sua trajetória no desenvolvimento web. Abaixo seguem alguns temas interessantes para você já ir se aprofundando:

- escopo

- a palavra-chave `return`
- manipulação de arrays (iteração, filtros, etc)
- manipulação de objetos (iteração, acesso a propriedades, acesso a valores)

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

7.5 RESUMO

Vamos repassar todos os passos do programa:

- Criar as cartas do jogo e definir seus atributos;
- Desenvolver uma função para sortear uma carta para o jogador e outra para a máquina;
- Exibindo os atributos das cartas na tela para o jogador;
- Obter o atribudo escolhido pelo jogador e comparar com a carta da máquina;
- Comparar o atributo de ambas as cartas e definir um vencedor.

Capítulo 8

SUPER TRUNFO: MONTAGEM DAS CARTAS

8.1 OBJETIVO DA AULA

Além de resolver problemas de forma lógica, trabalhar com programação também envolve a **integração de ferramentas**. Para esta Imersão, utilizamos o JavaScript em conjunto com o HTML e CSS, que são a base para o que chamamos de **desenvolvimento web front end**.

Nesta aula, vamos evoluir o que já criamos para o Super Trunfo, focando justamente na integração da lógica com a tela.

8.2 PASSO A PASSO

Como das vezes anteriores, vamos começar pelo fluxo do programa. O jogo já está fazendo o **mínimo necessário** para que o fluxo lógico do Super Trunfo funcione, então podemos pensar em incrementar a experiência de jogo, adicionando alguns passos:

- após o sorteio das cartas, extrair de cada objeto a imagem da carta e o valor de cada "poder";
- exibir na tela estas informações, fazendo com que

apareçam no local correto (ou seja, dentro da tag esperada).

Adicionando imagens para as cartas

Vamos pegar estas imagens da internet, como fizemos com os pôsteres do AluraFlix:

```
var cartaSeiya = {
   nome: "Seiya de Pégaso",
   imagem: "https://i.pinimg.com/originals/c2/1a/ac/c21aacd5d092b
f17cfff269091f04606.jpg",
   atributos: {
       ataque: 80,
       defesa: 60,
       magia: 90
   }
}

var cartaPokemon = {
   nome: "Bulbasauro",
   imagem: "http://4.bp.blogspot.com/-ZoCqleSAYNc/UQgfMdobjUI/AAA
AAAAACP0/s_iiWjmw2Ys/s1600/001Bulbasaur_Dream.png",
   atributos: {
       ataque: 70,
       defesa: 65,
       magia: 85
   }
}

var cartaStarWars = {
   nome: "Lorde Darth Vader",
   imagem: "https://images-na.ssl-images-amazon.com/images/I/51VJ
BqMZVAL._SX328_BO1,204,203,200_.jpg",
   atributos: {
       ataque: 88,
       defesa: 62,
       magia: 90
   }
}
```

Note que acrescentamos em cada objeto de carta um **atributo** a mais, o atributo `imagem`, cada um com um valor de `string`

correspondente ao link da imagem.

> Você pode escolher outras imagens se quiser, mas não esqueça de conferir se o link termina com `.jpg` ou `.png`.

Exibindo as informações adicionais na tela

Por enquanto, o programa está exibindo na tela somente uma lista dos "poderes", que pegamos a partir da carta sorteada para o jogador — estávamos mostrando só a lista, sem valores! Hora de exibir também os valores, para que o jogador possa escolher melhor em qual poder quer apostar.

Para isso, precisamos:

- localizar um elemento HTML específico na tela, usando o `getElementById()` ou o `getElementsByName()`, de acordo com o caso;
- adicionar novas informações formatadas como tags de HTML, utilizando JavaScript;
- descobrir como incluir estilos CSS a estas novas informações que estão sendo adicionadas.

Vamos começar criando a função `exibeCartaJogador()` e usando o `document.getElementById("carta-jogador")` para localizar o elemento com o identificador `id="carta-jogador"` no HTML e salvar este "endereço" em uma variável.

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
```

```
}
```

Para testar esse código, substitua a chamada da função `exibirOpcoes()` na úlfima linha da função `sortearCarta()` por esta que estamos começando a criar agora: `exibeCartaJogador()`.

Agora, quando o jogador clica no botão `sortear carta`, além da lógica do sorteio que já vimos anteriormente, o JavaScript também tem que "injetar" pedaços de código em HTML e em CSS para que os dados apareçam nos locais corretos da tela e com a aparência que esperamos.

Tanto o HTML quanto o CSS têm suas próprias sintaxes, que são diferentes do JavaScript. Porém, o JavaScript tem ferramentas que utilizam strings (caracteres entre aspas `" "`) combinados com variáveis para criar novos trechos de código em HTML e CSS, fazendo com que o navegador reconheça as instruções. Já fizemos um pouco disso nos projetos anteriores, toda vez que utilizamos a palavra-chave `innerHTML` para inserir tags HTML no formato de string.

O HTML inicial deste projeto já conta com imagem pronta para ser a base de todas as cartas, e o jogo já inicia com esta imagem aplicada na `<div>` de carta do jogador e de carta do computador:

```
<div>
   <div id="carta-jogador">
       <img src="https://www.alura.com.br/assets/img/imersoes/dev
-2021/card-super-trunfo-transparent-ajustado.png"
           style=" width: inherit; height: inherit; position: abs
olute;">
       <h3></h3>
   </div>
```

```
</div>
<div>
   <div id="carta-maquina" class="carta"><img
    src="https://www.alura.com.br/assets/img/imersoes/dev-2021/ca
rd-super-trunfo-transparent-ajustado.png"
           style=" width: inherit; height: inherit; position: abs
olute;"></div>
</div>
```

Já temos a base, mas como exibir as informações nela? Lembrando que, nessa hora, só devemos mostrar a carta sorteada para o jogador!

As informações são:

- nome da carta;
- imagem;
- poderes **e seus valores**.

Quando queremos adicionar estilo ao HTML, utilizamos as chamadas **propriedades de CSS**; por exemplo, trocar as cores dos textos, alinhar elementos, entre muitas outras coisas.

É possível atribuir estas propriedades a tags HTML de algumas formas. Vamos ver um exemplo baseado no código do primeiro projeto da Imersão:

No arquivo HTML atribuímos `class` a um elemento:

```
<h1 class="page-title">
 Conversor de moedas
</h1>
```

E no CSS utilizamos o nome de `class` para mudar propriedades; no exemplo abaixo, a cor do texto:

```
.page-title {
   color: #ffffff;
```

```
   margin: 0 0 5px;
 }
```

O exemplo acima é uma forma bastante utilizada. Mas também é possível passar estilos direto na tag HTML do elemento, dispensando o arquivo CSS. O exemplo acima ficaria, então, da seguinte forma:

```
<h1 class="page-title" style="color: #ffffff; margin: 0 0 5px;">
 Conversor de moedas
</h1>
```

O JavaScript consegue manipular tags HTML e adicionar `style` a uma tag, e é esta ferramenta que vamos utilizar para adicionar a imagem correspondente:

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
}
```

No código acima, utilizamos `divCartaJogador.style.backgroundImage = url(${cartaJogador.imagem}) ` para adicionar um atributo style ao elemento que já estava salvo na variável divCartaJogador . Além de style adicionamos também qual é a propriedade de estilo que queremos adicionar e seu valor, uma url (um caminho de um link) que estamos obtendo do objeto cartaJogador.imagem`.

> Se precisar, relembre o processo de acessar dados de um objeto no material da aula anterior.

Quando o JavaScript processar esse trecho de código, o HTML gerado vai ser semelhante ao abaixo:

```
<div id="carta-jogador" style="background-image: url('https://[en
dereço da sua imagem].jpg');">
</div>
```

Ainda temos que exibir o nome da carta e os poderes com valores. Ou seja, **também temos que utilizar o JavaScript para gerar código HTML destes elementos**.

Podemos pegar o nome no mesmo objeto `cartaJogador` e criar uma tag `<p>` (tag genérica de texto) com esta informação:

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>
`
}
```

O nome ainda não está aparecendo na tela, pois ainda não atualizamos o valor de `divCartaJogador` com mais este trecho de código HTML.

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>
`

   divCartaJogador.innerHTML += nome
}
```

Para atualizar uma variável com um novo valor sem substituir totalmente o que já está salvo nela, podemos usar o operador `+=`, como já vimos nos exercícios com iteradores.

Imagem e nome da carta estão na tela, hora de lidar com a lista de poderes e os valores. Também temos que usar o JavaScript para criar tags HTML e inserir tudo no código; a diferença é que, nesse caso, **temos que fazer isso para cada um dos poderes que está dentro de um objeto**.

No estágio inicial deste projeto, vimos como percorrer objetos utilizando `for… in` . Podemos reutilizar este método para criar tags HTML de forma automatizada com cada poder e valor.

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>
`
   var opcoesTexto = ""

   for (var atributo in cartaJogador.atributos) {
       opcoesTexto += "<input type='radio' name='atributo' value=
'" + atributo + "'>" + atributo + " " + cartaJogador.atributos[at
ributo] + "<br>"
   }

   divCartaJogador.innerHTML += nome + opcoesTexto
}
```

Para cada um dos poderes, temos que criar uma `string` com a sintaxe do HTML e as informações que vamos acessar do objeto JavaScript referente à carta do jogador. Então, a primeira coisa a fazer é criar uma variável para salvar todas essas tags, começando com um valor de "vazia":

```
var opcoesTexto = ""
```

A estrutura do `for… in` é a mesma que fizemos para a fase anterior deste projeto! Você pode consultar o material caso precise relembrar o que foi feito.

```
for (var atributo in cartaJogador.atributos) {
 // código aqui
}
```

O que será feito em cada **iteração** é que muda. Além disso, já vimos o `input type="radio"` na fase anterior do projeto, e também como acessar os valores em um objeto, e não apenas as propriedades:

```
for (var atributo in cartaJogador.atributos) {
   opcoesTexto += "<input type='radio' name='atributo' value='" +
 atributo + "'>" + atributo + " " + cartaJogador.atributos[atribu
to] + "<br>"
}
```

Agora o JavaScript vai unir cada `atributo` (ou seja, cada item dentro do objeto `cartaJogador.atributos`) com strings que representam código HTML, salvando o resultado de cada iteração na variável `opcoesTexto` — repare que estamos usando o operador `+=` novamente para atualizar a variável ao invés de substituir seu valor.

Temos que atualizar também o valor que será enviado para a variável `divCartaJogador` : isso está sendo feito com a linha `divCartaJogador.innerHTML += nome + opcoesTexto` … Mas se tentarmos atualizar a tela neste momento, perceberemos que os poderes não estão aparecendo na parte da tela onde deveriam.

Aqui vamos entrar um pouco mais em como HTML e CSS trabalham juntos para que todas as partes da tela estejam onde deveriam estar. Para este projeto, nosso time de front end já deixou algumas propriedades de CSS criadas no arquivo `style.css`, prontas para serem acessadas pelo HTML… Só precisamos criar o código para isso!

```
   divCartaJogador.innerHTML += "<div id='opcoes' class='carta-st
atus'>" + nome + opcoesTexto + "</div>"
```

A classe `class='carta-status'` vai cuidar do alinhamento, o que precisamos fazer é inserir uma tag para agrupar tudo. No caso, vamos criar uma tag de elemento do tipo `<div>`, adicionar as variáveis com o código HTML que acabamos de criar, sem esquecer de fechar a tag no final da linha, com `</div>`.

O resultado final desta função fica da seguinte forma:

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>
`
   var opcoesTexto = ""

   for (var atributo in cartaJogador.atributos) {
       opcoesTexto += "<input type='radio' name='atributo' value=
'" + atributo + "'>" + atributo + " " + cartaJogador.atributos[at
ributo] + "<br>"
   }

   divCartaJogador.innerHTML += "<div id='opcoes' class='carta-st
atus'>" + nome + opcoesTexto + "</div>"
}
```

HTML e CSS são competências à parte da lógica de programação em si! Durante a Imersão decidimos não focar muito em conceitos destas linguagens, pois seria muita coisa para ver ao mesmo tempo! Mas se você gostar, pode se aprofundar também nestes assuntos e revisitar este código mais tarde para ver como estas partes estão trabalhando juntas.

Exibir a carta do computador

Uma boa parte do código que fizemos na versão anterior pode ser aproveitada neste trecho.

A função `obtemAtributoSelecionado()` não vai ser modificada, pois a lógica de obter a lista de elementos `<input type='radio' name='atributo'>` para verificar qual está selecionado (`.checked`) é a mesma.

Na função `jogar()` — a função que usamos, entre outras coisas, para chamar/executar `obtemAtributoSelecionado()` —, vamos manter a lógica de verificação usada no `if` (valor maior/ valor menor/ empate), porém agora substituindo o `alert` por uma mensagem na tela:

```
function jogar() {
   var atributoSelecionado = obtemAtributoSelecionado()
   var htmlResultado = ""

   if (cartaJogador.atributos[atributoSelecionado] > cartaMaquina
.atributos[atributoSelecionado]) {
       htmlResultado = '<p class="resultado-final">Venceu</p>'
   } else if (cartaJogador.atributos[atributoSelecionado] < carta
```

```
Maquina.atributos[atributoSelecionado]) {
        htmlResultado = '<p class="resultado-final">Perdeu</p>'
    } else {
        htmlResultado = '<p class="resultado-final">Empatou</p>'
    }

    var divResultado = document.getElementById("resultado")
    divResultado.innerHTML = htmlResultado

    exibeCartaMaquina()
}
```

Começamos com a variável `htmlResultado` vazia, e para cada uma das **condições** do `if`, salvamos uma `string` com a tag HTML e a informação de acordo com a comparação (jogador perdeu, jogador ganhou, etc).

Os passos seguintes, como nas outras funções que criamos, é localizar o elemento `"resultado"` no HTML e inserir no `.innerHTML` deste elemento a string que o JavaScript salvou na variável `htmlResultado` de acordo com o caminho seguido pelo `if`.

Agora que já temos o resultado dessa rodada do jogo, podemos passar para o último passo, que é revelar a carta do computador. Já vamos deixar esta função sendo chamada na última linha, `exibeCartaMaquina()`.

Exibindo a carta do computador

Já temos a função `exibeCartaJogador()`, e se formos pensar na lógica do que precisamos fazer para exibir a carta do computador, podemos identificar coisas semelhantes:

- acessar objeto da carta sorteada;
- exibir nome e imagem a partir deste objeto;

- percorrer (iterar) o objeto `cartaMaquina.atributos` para acessar a lista de poderes;
- exibir estes atributos da forma correta na tela (nesse caso, não precisamos dos botões `type="radio"` para selecionarmos uma opção).

Assim, a função final fica da seguinte forma:

```
function exibeCartaMaquina() {
   var divCartaMaquina = document.getElementById("carta-maquina")
   divCartaMaquina.style.backgroundImage = `url(${cartaMaquina.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaMaquina.nome}</p>
`
   var opcoesTexto = ""

   for (var atributo in cartaMaquina.atributos) {
       opcoesTexto += "<p type='text' name='atributo' value='" +
atributo + "'>" + atributo + " " + cartaMaquina.atributos[atribut
o] + "<br>"
   }

   divCartaMaquina.innerHTML += "<div id='opcoes' class='carta-st
atus --spacing'>" + nome + opcoesTexto + '</div>'
}
```

As **diferenças** entre as funções `exibeCartaJogador()` e `exibeCartaMaquina()` são:

- o momento em que são chamadas: exibimos na tela a carta do jogador quando o usuário clica no botão para sortear sua carta. Já a carta do computador só pode ser exibida **após** o usuário escolher um poder e clicar no botão `jogar`. Afinal de contas, não podemos deixar o usuário ver os poderes da carta do computador antes de fazer a escolha e confirmar clicando no botão!
- o HTML gerado no `for... in`: na carta do jogador, temos

que criar uma tag do tipo `<input type="radio">` para o usuário selecionar o poder que quer comparar. Para a carta do computador, podemos criar uma tag de texto normal do HTML (`<p>`) já que o jogador não vai fazer nenhuma escolha nesse caso, só acompanhar os resultados.

Você já ouviu falar em "reaproveitamento de código"? Quando utilizamos partes de código similares, como no caso das funções `exibeCartaJogador()` e `exibeCartaMaquina()` é normal tentarmos escrever de uma forma que tenha o melhor aproveitamento possível. Ou seja, tentamos escrever somente um trecho de código que sirva para várias situações; afinal de contas, o código para exibir cartas na tela deveria funcionar independente de ser a carta do jogador, do computador, ou qualquer outra situação. Você vai se deparar muito com esta questão durante seus estudos, e vai ter a oportunidade de estudar situações em que isso pode inclusive não ser vantajoso… **No código que acabamos de fazer, você consegue pensar em como reaproveitar a função de exibir cartas para o mesmo código funcionar nos dois casos, sem precisar ser reescrito?**

8.3 PARA SABER MAIS

Lembre-se que quase sempre existe mais de uma forma de fazer qualquer tarefa de programação. À medida em que você for aprendendo novos métodos, pode voltar neste projeto e praticar nele!

Abaixo, algumas questões que você pode usar para direcionar seus estudos:

- por que os trechos de código que estamos adicionando ao HTML direto pelo JavaScript já aparecem nos lugares certos?
- quais são as formas de se trabalhar com código CSS e quando cada caso é melhor, ou mais utilizado?

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

8.4 RESUMO

- Adicionando o campo imagem nos objetos com o caminho da imagem;
- Criar uma função que exibe a carta do jogador após o sorteio das cartas;
- Adicionar a moldura da carta;
- Escrever o resultado na tela após o duelo das cartas informando se o jogador venceu ou perdeu;
- Criar uma função que exibe a carta da máquina;
- Exibir os atributos e pontos da carta da máquina.

CAPÍTULO 9

SUPER TRUNFO: JOGANDO COM MAIS CARTAS

9.1 OBJETIVO DA AULA

Continuando com o desenvolvimento do Super Trunfo, as próximas funcionalidades (também chamadas de *features*) do programa vão ser:

- aumentar a quantidade de cartas;
- retirar as cartas que já foram jogadas para não repetir;
- contar os pontos a cada rodada;
- estabelecer em que momento o jogo termina.

> Listar por extenso cada passo separado que conseguimos pensar para uma tarefa facilita bastante na hora de escrever o código. Às vezes algo que parece simples tem muito mais partes do que imaginamos no começo.

Nesta aula, o foco vai ser desenvolver a lógica da contagem

de pontos e controle das cartas, assim como representar na tela o que está acontecendo.

9.2 PASSO A PASSO

Aumentando a quantidade de cartas

A estrutura dos objetos permanece a mesma; vamos deixar o código pronto, mas você pode criar as suas estruturas, assim como modificar para que todas tenham o mesmo tema como o Super Trunfo clássico. Você pode criar cartas para um jogo com 8, 16 ou 32 cartas. O código abaixo também está no repositório, no fim do capítulo.

```
var cartaSeiya = {
   nome: "Seiya de Pegaso",
   imagem: "https://i.pinimg.com/originals/c2/1a/ac/c21aacd5d092b
f17cfff269091f04606.jpg",
   atributos: {
       ataque: 80,
       defesa: 60,
       magia: 90
   }
}

var cartaPokemon = {
   nome: "Bulbasauro",
   imagem: "http://4.bp.blogspot.com/-ZoCqleSAYNc/UQgfMdobjUI/AAA
AAAAACP0/s_iiWjmw2Ys/s1600/001Bulbasaur_Dream.png",
   atributos: {
       ataque: 70,
       defesa: 65,
       magia: 85
   }
}

var cartaStarWars = {
   nome: "Lorde Darth Vader",
```

```
   imagem: "https://images-na.ssl-images-amazon.com/images/I/51VJ
BqMZVAL._SX328_BO1,204,203,200_.jpg",
   atributos: {
       ataque: 88,
       defesa: 62,
       magia: 90
   }
}

var cartaLol = {
   nome: "Caitlyn",
   imagem: "http://1.bp.blogspot.com/-K7CbqWc1-p0/VLc98v85s0I/AAA
AAAAABqk/-ZB684VVHbg/s1600/Caitlyn_OriginalSkin.jpg",
   atributos: {
       ataque: 95,
       defesa: 40,
       magia: 10
   }
}

var cartaNaruto = {
   nome: "Naruto",
   imagem: "https://conteudo.imguol.com.br/c/entretenimento/16/20
17/06/27/naruto-1498593686428_v2_450x337.png",
   atributos: {
       ataque: 80,
       defesa: 60,
       magia: 100
   }
}

var cartaHarry = {
   nome: "Harry Potter",
   imagem: "https://sm.ign.com/ign_br/screenshot/default/89ff10dd
-aa41-4d17-ae8f-835281ebd3fd_49hp.jpg",
   atributos: {
       ataque: 70,
       defesa: 50,
       magia: 95
   }
}

var cartaBatman = {
   nome: "Batman",
   imagem: "https://assets.b9.com.br/wp-content/uploads/2020/09/B
```

```
atman-issue86-heder-1280x677.jpg",
   atributos: {
       ataque: 95,
       defesa: 70,
       magia: 0
   }
}

var cartaMarvel = {
   nome: "Capitã Marvel",
   imagem: "https://cinepop.com.br/wp-content/uploads/2018/09/cap
itamarvel21.jpg",
   atributos: {
       ataque: 90,
       defesa: 80,
       magia: 0
   }
}
```

Como nas versões anteriores, ainda precisamos criar variáveis para salvar as cartas sorteadas para o jogador e para o computador, assim como criar a *array* que vai ser usada para o sorteio:

```
var cartaMaquina
var cartaJogador
var cartas = [cartaSeiya, cartaPokemon, cartaStarWars, cartaLol,
cartaNaruto, cartaHarry, cartaBatman, cartaMarvel]
```

Esta parte do código não está dentro de nenhum bloco de função, para poder ser acessada a partir de qualquer parte do código.

> Para entender melhor por que algumas variáveis podem ou não ser acessadas pelo restante do código — o que chamamos de VARIÁVEL COM ESCOPO GLOBAL — o tema para estudo é **escopos em JavaScript**.

Agora, temos que salvar a pontuação de cada jogador, que também vai ser controlada com variáveis, começando com `0`:

```
var pontosJogador = 0
var pontosMaquina = 0
```

Já temos como controlar o placar! Vamos escrever a lógica de quando e de que forma ele é atualizado.

Atualizando na tela o placar e as cartas disponíveis

Vamos criar uma função para mostrar na tela a quantidade de cartas em jogo, assim podemos chamar esse trecho de código para ser executado no momento certo:

```
function atualizaQuantidadeDeCartas() {
   var divQuantidadeCartas = document.getElementById('quantidade-
cartas')
   var html = "Quantidade de cartas no jogo: " + cartas.length

   divQuantidadeCartas.innerHTML = html
}
```

Esta função localiza no HTML o elemento com identificador `id="quantidade-cartas"`, e insere no elemento um texto mais a informação `cartas.length`, ou seja, o comprimento da array `cartas` que definimos um pouco antes.

Veja que esta função não está responsável pela lógica que vai **modificar** a quantidade de cartas na array, apenas faz a atualização da informação na tela! Esta distinção é importante, e também é bom separarmos bem o código para ele ficar organizado e fácil de atualizar.

A função para atualizar o placar na tela vai ter uma lógica parecida:

```
function atualizaPlacar() {
   var divPlacar = document.getElementById('placar')
   var html = "Jogador " + pontosJogador + "/" + pontosMaquina +
" Máquina"

   divPlacar.innerHTML = html
}
```

Nesta função, vamos utilizar o valor das variáveis `pontosJogador` e `pontosMaquina` que criamos anteriormente e que, **neste momento**, têm o valor de `0`.

Sortear cartas e exibir carta do jogador

As funções para sortear as cartas e exibir a carta sorteada para o jogador são similares às que fizemos na aula passada, com algumas alterações.

O que precisamos fazer neste momento do código é **retirar as cartas sorteadas do baralho (a array `cartas`)**. Também retiramos a moldura da carta do HTML inicial para que ela só apareça após o início do jogo, então temos que passar isso para o JavaScript também.

```
function sortearCarta() {
   var numeroCartaJogador = parseInt(Math.random() * cartas.lengt
h)
   cartaJogador = cartas[numeroCartaJogador]
   cartas.splice(numeroCartaJogador, 1)

   var numeroCartaMaquina = parseInt(Math.random() * cartas.lengt
h)
   cartaMaquina = cartas[numeroCartaMaquina]
   cartas.splice(numeroCartaMaquina, 1)

   document.getElementById('btnSortear').disabled = true
   document.getElementById('btnJogar').disabled = false

   exibeCartaJogador()
```

```
}
```

No código anterior, era necessário um iterador `while` para garantir que jogador e computador não recebessem a mesma carta. Mas agora, após ser sorteada, a carta deve ser retirada do baralho (ou seja, da array). Então, não precisamos mais dessa verificação via `while`, e sim de uma forma de **retirar um elemento de um array**.

Há algumas formas de se fazer isso com JavaScript. Aqui, vamos usar o método `cartas.splice(numeroCartaJogador, 1)` que vai automaticamente acessar a array `cartas` no índice indicado pelo número sorteado em `numeroCartaJogador` e retirar este elemento da lista. A quantidade de elementos retirados (somente um) está sendo indicada pelo número `1`.

Seguimos a mesma lógica para sortear a carta do computador e retirar o elemento correspondente da array `cartas`.

A lógica para exibir somente a carta do jogador também se mantém, a diferença é que agora não vamos mostrar as molduras vazias das cartas — isso vai melhorar o visual do jogo.

```
function exibeCartaJogador() {
   var divCartaJogador = document.getElementById("carta-jogador")
   var moldura = '<img src="https://www.alura.com.br/assets/img/i
mersoes/dev-2021/card-super-trunfo-transparent.png" style=" width
: inherit; height: inherit; position: absolute;">';
   divCartaJogador.style.backgroundImage = `url(${cartaJogador.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaJogador.nome}</p>
`
   var opcoesTexto = ""

   for (var atributo in cartaJogador.atributos) {
       opcoesTexto += "<input type='radio' name='atributo' value=
'" + atributo + "'>" + atributo + " " + cartaJogador.atributos[at
```

```
ributo] + "<br>"
   }

   var html = "<div id='opcoes' class='carta-status'>"

   divCartaJogador.innerHTML = moldura + nome + html + opcoesText
o + '</div>'
}
```

No código acima, acrescentamos a variável `moldura` com o valor da tag HTML `<img>` que está exibindo as molduras das cartas na tela, ao invés de deixar esta tag desde o início no meio do código HTML.

No fim da função, atualizamos o `innerHTML` com esta moldura, além dos outros elementos que já tínhamos acessado anteriormente.

A função para exibir a carta do computador segue com alterações semelhantes:

```
function exibeCartaMaquina() {
   var divCartaMaquina = document.getElementById("carta-maquina")
   var moldura = '<img src="https://www.alura.com.br/assets/img/i
mersoes/dev-2021/card-super-trunfo-transparent.png" style=" width
: inherit; height: inherit; position: absolute;">';
   divCartaMaquina.style.backgroundImage = `url(${cartaMaquina.im
agem})`
   var nome = `<p class="carta-subtitle">${cartaMaquina.nome}</p>
`
   var opcoesTexto = ""

   for (var atributo in cartaMaquina.atributos) {
       console.log(atributo)
       opcoesTexto += "<p type='text' name='atributo' value='" +
atributo + "'>" + atributo + " " + cartaMaquina.atributos[atribut
o] + "<br>"
   }

   var html = "<div id='opcoes' class='carta-status --spacing'>"
```

```
   divCartaMaquina.innerHTML = moldura + nome + html + opcoesTexto + '</div>'
}
```

Atualizando a lógica do jogo

Agora que adicionamos funcionalidades, temos mais coisas que têm que acontecer quando o jogador clica no botão `jogar`: além de exibir a carta do computador, agora também precisa atualizar a pontuação e verificar se o jogo termina ou continua.

Ainda estamos usando o `<input type='radio'>` para salvar qual poder o jogador escolheu, então não precisamos alterar esta função:

```
function obtemAtributoSelecionado() {
   var radioAtributo = document.getElementsByName('atributo')
   for (var i = 0; i < radioAtributo.length; i++) {
       if (radioAtributo[i].checked) {
           return radioAtributo[i].value
       }
   }
}
```

Na função `jogar()`, a lógica para verificar se o jogador perdeu ou ganhou a rodada permanece **quase** a mesma, e temos que pensar no que acontece **depois** de termos o resultado desta lógica.

- Se o jogador vence a rodada — ou seja, se o código entra no primeiro `if` —, além da mensagem "venceu" na tela, o jogo também precisa aumentar o placar do jogador com +1 ponto.
- Se quem vence a rodada é o computador — o código entrou no `else if` —, deve aparecer a mensagem "perdeu" na tela, e se computar +1 no placar do

computador.

- Caso o número de cartas no monte (array `cartas`) tenha acabado, o programa deve iniciar a lógica de comparar os placares e ver quem venceu.

```
 function jogar() {
   var divResultado = document.getElementById("resultado")
   var atributoSelecionado = obtemAtributoSelecionado()

   if (cartaJogador.atributos[atributoSelecionado] > cartaMaquina
.atributos[atributoSelecionado]) {
       htmlResultado = '<p class="resultado-final">Venceu</p>'
       pontosJogador++
   } else if (cartaJogador.atributos[atributoSelecionado] < carta
Maquina.atributos[atributoSelecionado]) {
       htmlResultado = '<p class="resultado-final">Perdeu</p>'
       pontosMaquina++
   } else {
       htmlResultado = '<p class="resultado-final">Empatou</p>'
   }

   if (cartas.length == 0) {
       alert("Fim de jogo")
       if (pontosJogador > pontosMaquina) {
           htmlResultado = '<p class="resultado-final">Venceu</p>
'
       } else if (pontosMaquina > pontosJogador) {
           htmlResultado = '<p class="resultado-final">Perdeu</p>
'
       } else {
           htmlResultado = '<p class="resultado-final">Empatou</p
>'
       }
   } else {
       document.getElementById('btnProximaRodada').disabled = fal
se
   }

   divResultado.innerHTML = htmlResultado
   document.getElementById('btnJogar').disabled = true

   atualizaPlacar()
   exibeCartaMaquina()
```

```
    atualizaQuantidadeDeCartas()
}
```

No primeiro bloco de `if/else`, temos uma instrução a mais em cada uma das condições: `pontosJogador++` ou `pontosMaquina++`. Estas duas variáveis foram criadas no começo da aula e devem estar no topo do código; o operador `++` significa que o valor de cada uma (lembrando que começam com `0`) será aumentado sempre de 1 em 1. Ou seja, se o valor salvo na variável `pontosJogador` atualmente é 2, o operador `++` vai fazer com que seja atualizado para 3, se o valor atual é 3, será atualizado para 4, e daí em diante.

Então podemos concluir que, se o código entrar no primeiro `if` (o valor do atributo é maior na carta do jogador), além da mensagem na tela, a variável `pontosJogador` vai ter seu valor aumentado em 1. A mesma coisa se a carta do computador for a vencedora da rodada.

O próximo passo é verificar se o jogo continua, ou seja, se ainda há elementos dentro da array `cartas`; caso isso seja verdade, isto é, a comparação `cartas.length == 0` é `true`, é hora de comparar os placares e ver quem ganhou. Caso ainda haja cartas na array para serem jogadas (ou seja, o valor de `cartas.length` é **diferente** de `0`) o JavaScript tem que pular a verificação do placar, permitir que o jogador clique no botão `próxima rodada`, e seguir com a execução do restante do código.

```
    if (cartas.length == 0) {
        alert("Fim de jogo")
        if (pontosJogador > pontosMaquina) {
            htmlResultado = '<p class="resultado-final">Venceu</p>
'
        } else if (pontosMaquina > pontosJogador) {
            htmlResultado = '<p class="resultado-final">Perdeu</p>
```

```
'
        } else {
            htmlResultado = '<p class="resultado-final">Empatou</p
>'
        }
    } else {
        document.getElementById('btnProximaRodada').disabled = fal
se
    }
```

Depois de fazer todas as verificações que têm que acontecer **em todas as rodadas do jogo**, podemos chamar três funções:

```
atualizaPlacar()
exibeCartaMaquina()
atualizaQuantidadeDeCartas()
```

Lembrando do que falamos nas aulas anteriores, **blocos de código dentro de funções só são executados quando a função é chamada**, que é o que está acontecendo aqui. Só neste momento, depois que o JavaScript já comparou valores, modificou valores de variáveis e fez verificações é que devemos executar a função `atualizaPlacar()` (que exibe na tela o novo valor das variáveis `pontosJogador` e `pontosMaquina`), `exibeCartaMaquina()` (para exibir a carta do computador na tela e o jogador poder comparar) e, por último, `atualizaQuantidadeDeCartas()`, que exibe na tela quantas cartas ainda estão disponíveis no jogo.

Continuando com as rodadas do jogo

Agora que o jogo tem várias cartas e placar, não podemos mais atualizar a tela depois de cada rodada. Se você fizer isso, o jogo vai reiniciar completamente. Então o JavaScript tem que atualizar a tela para o jogador:

```
function proximaRodada() {
    var divCartas = document.getElementById('cartas')
```

```
   divCartas.innerHTML = `<div id="carta-jogador" class="carta"><
/div> <div id="carta-maquina" class="carta"></div>`

   document.getElementById('btnSortear').disabled = false
   document.getElementById('btnJogar').disabled = true
   document.getElementById('btnProximaRodada').disabled = true

   var divResultado = document.getElementById('resultado')
   divResultado.innerHTML = ""
}
```

O código dessa função existe basicamente para "voltar" o HTML que está sendo exibido na tela ao estado em que estava antes, sem nenhuma carta aparecendo:

```
   divCartas.innerHTML = `<div id="carta-jogador" class="carta"><
/div> <div id="carta-maquina" class="carta"></div>`
```

E também habilitar ou desabilitar os botões conforme o caso, para ajudar o jogador:

```
   document.getElementById('btnSortear').disabled = false
   document.getElementById('btnJogar').disabled = true
   document.getElementById('btnProximaRodada').disabled = true
```

E, por último, limpar da tela a mensagem "perdeu" ou "venceu" da rodada anterior:

```
   var divResultado = document.getElementById('resultado')
   divResultado.innerHTML = ""
```

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

9.3 RESUMO

- Nesta aula vimos um pouco mais sobre variáveis que estão

no chamado *escopo global*. Entender como os escopos funcionam e para que servem vão te ajudar muito no futuro.

- Criamos uma regra diferente da regra clássica, onde quem vence a rodada pega a carta de quem perdeu, mas você pode praticar para implementar a solução clássica! Dica: estude bastante manipulação de **arrays**.
- Ainda sobre manipulação de arrays, o JavaScript tem vários métodos para isso. Nesta aula usamos o `splice`, mas existem muitos outros para vários casos de uso.

Capítulo 10

CERTIFICADO

10.1 OBJETIVOS DA AULA

Durante toda imersão, focamos na lógica de programação com Javascript e aprendemos muita coisa. Nessa última aula, vamos focar exclusivamente no HTML e no CSS para construir um certificado com uma foto pessoal e uma lista cada projeto desenvolvido durante a imersão.

> Neste programa, vamos aprender a estruturar um site com HTML e como deixá-lo bem lindão com CSS.

10.2 PASSO A PASSO

Escrever um texto como este que está lendo e desenvolver um site não são coisas tão diferentes assim. Nesse texto, você está lendo um parágrafo e no começo da página existe um título bem grande escrito **Certificado**. No HTML, precisamos especificar o que é um título, o que é um parágrafo, uma imagem, entre muitas outras coisas.

Tags no HTML

O HTML é composto de diversas tags, cada uma com sua função e significado. Desde 2013, com a atualização da linguagem para o HTML 5, muitas novas tags foram adicionadas. Nesse momento, vamos focar em tags que representam títulos, parágrafo e ênfase.

Títulos

Quando queremos indicar que um texto é um título em nossa página, utilizamos as tags de heading em sua marcação. Escreva seu nome entre a tag `h1` .

```
<h1>Guilherme Lima</h1>
<h2>Insígnias da imersao.dev</h2>
```

As tags de heading são para exibir conteúdo de texto e contém 6 níveis, ou seja de `h1` à `h6` , seguindo uma ordem de importância, sendo `h1` o título principal, o mais importante, e `h6` o título de menor importância.

Utilizamos, por exemplo, a tag `h1` para o nome, título principal da página, e a tag `h2` como subtítulo ou como título de seções dentro do documento.

> Dica: recomendamos que a tag `h1` seja utilizada uma vez em cada página porque não pode existir mais de um conteúdo mais importante da página.

A ordem de importância tem impacto nas ferramentas que

processam HTML. As ferramentas de indexação de conteúdo para buscas, como o Google, Bing ou Yahoo! levam em consideração essa ordem e relevância. Os navegadores especiais para acessibilidade também interpretam o conteúdo dessas tags de maneira a diferenciar seu conteúdo e facilitar a navegação do usuário pelo documento.

Parágrafos

Quando exibimos qualquer texto em nossa página, é recomendado que ele seja sempre conteúdo de alguma tag filha da tag . A marcação mais indicada para textos comuns é a tag de parágrafo:

```
<h1>Guilherme Lima</h1>
<h2>Insígnias da imersao.dev</h2>
<p>Isso é um parágrafo. Legal né?</p>
```

Estruturando o certificado

Agora que aprendemos algumas tags HTML, vamos criar a estrutura principal da página, sem a preocupação inicial na estilização. Vamos criar uma lista ordenada para listar cada projeto que fizemos. Para isso, vamos usar a tag `ol` e criar cada linha com a tag `li` . Não esqueça de fechar as tags, como ilustra o código abaixo:

```
<h1>Mario Souto (@omariosouto/devsoutinho)</h1>

  <div>
    <h2>Insígnias da imersao.dev</h2>
    <ol>
      <li> Conversor de Moedas</li>
      <li> Calculadora</li>
      <li> Mentalista</li>
      <li> Aluraflix</li>
```

```
      <li> Tabela de classificação</li>
      <li>★ Supertrunfo</li>
      <li> Certificard</li>
    </ol>
  </div>
```

CSS

Quando escrevemos o HTML, marcamos o conteúdo da página com tags que melhor representam o significado daquele conteúdo. Quando abrimos a página no navegador é possível perceber que ele mostra as informações com estilos diferentes.

Um `h1`, por exemplo, por padrão é apresentado em negrito numa fonte maior. Parágrafos de texto são espaçados entre si, e assim por diante. Isso quer dizer que o navegador tem um estilo padrão para as tags que usamos. Porém para fazer sites bonitos, ou com o visual próximo de uma dada identidade visual (design), vamos precisar personalizar a apresentação padrão dos elementos da página.

```
h1 {
  font-size: 20px;
}
```

> Revisando então a estrutura de uso do CSS.

```
seletor {
    propriedade: valor;
}
```

Podemos estilizar todo conteúdo que estamos vendo através do seletor `body`, alterando a cor de fundo da página e a fonte.

```
body {
  background-color: #ff5722;
  font-family: 'Open Sans';
}
```

Alterando a fonte

Para alterar o tipo de fonte, podemos buscar Google Fonts e incluir no início do HTML:

```
<link rel="preconnect" href="https://fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css2?family=Open+Sans:it
al,wght@0,300;0,400;0,600;0,700;0,800;1,300;1,400;1,600;1,700;1,8
00&display=swap" rel="stylesheet">
```

Dividindo o HTML em partes

Sem aprofundar muito, alguns elementos do HTML 5 tem função semântica e podemos aplicar um estilo a mais de uma tag. Observe o código abaixo:

```
<link rel="preconnect" href="https://fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css2?family=Open+Sans:it
al,wght@0,300;0,400;0,600;0,700;0,800;1,300;1,400;1,600;1,700;1,8
00&display=swap" rel="stylesheet">

<section>
  <header>
    <img src="https://unavatar.now.sh/github/omariosouto" />
  </header>
  <h1>Mario Souto (@omariosouto/devsoutinho)</h1>

  <div>
    <h2>Insígnias da imersao.dev</h2>
    <ol>
      <li> Conversor de Moedas</li>
      <li> Calculadora</li>
      <li> Mentalista</li>
      <li> Aluraflix</li>
      <li> Tabela de classificação</li>
      <li>★ Supertrunfo</li>
```

```
      <li> Certificard</li>
    </ol>
  </div>
</section>
```

Observe que temos uma tag `img` dentro da tag `header` e todo conteúdo está dentro de uma `section` que envolve toda a aplicação. Além disso, a lista das insígnias e o `h2` está incluso em uma `div` .

Certo, mas como podemos estilizar o visual da aplicação com base nas divisões que fizemos?

Box-Shadow

Para começar, vamos criar um card que envolva todo nosso código. Podemos usar sites como box-shadow para criar esse card e incluir no seletor `section` do CSS.

```
section {
  -webkit-box-shadow: 5px 5px 50px 8px rgba(0,0,0,0.59);
  box-shadow: 5px 5px 50px 8px rgba(0,0,0,0.59);
  display: block;
  margin-left: auto;
  margin-right: auto;
  max-width: 500px;
  margin-top: 100px;
  background-color: white;
  padding: 15px;
  border: 1px solid black;
  border-radius: 5px;
}
```

Podemos também estilizar o `header` que criamos.

```
header {
  background-image: url('https://cdn.cinepop.com.br/2021/02/pokem
on-list.jpg');
  background-size: cover;
  height: 80px;
```

```
  border-radius: 5px;
}
```

Para finalizar, vamos estilizar também a `div`.

```
div {
  padding: 15px;
  border-radius: 5px;
  border: 1px solid black;
  background-color: #bac3d6;
}
```

Estilizando a imagem

A imagem também pode ser estilizada, alterando seu tamanho, altura, incluir bordas e posicionamento, como ilustra o código abaixo.

```
img {
  width: 150px;
  height: 150px;
  border-radius: 100%;
  margin-left: auto;
  margin-right: auto;
  display: block;
  transform: translateY(-50%);
  border: 8px solid white;
}
```

> Dica: como a tag `h1` está incluido na `section`, podemos alterar o CSS desta forma:

```
section h1 {
  font-size: 20px;
}
```

Tudo certo? Então é hora de praticar!

Clique neste link para acessar o código completo no GitHub.

10.3 RESUMO

Vamos repassar todos os passos do programa:

- Criamos um certificado do zero;
- Adicionamos o HTML para estruturar nossa página;
- Estilizamos a página com CSS.