Exercícios JS Intermediário

Dica: crie uma nova pasta para os exercícios dessa aula e um arquivo diferente para cada exercício, exemplo:

```
/js-intemrediario exercicio1.js exercicio2.js exercicio3.jsPara executar cada um dos exercícios, faça o seguinte:
```

Para executar cada um dos exercícios, faça o seguinte:

- Importe o arquivo de cada exercício do HTML, usando a técnica de utilizar o JS externamente
- Abra o arquivo HTML no seu navegador e inspecione a página (ctrl + shift + i), navegue até a aba “console”

---

1. **Declarações `if` são usadas para tomar decisões no código.**

   A palavra-chave `if` diz ao JavaScript para executar o código entre chaves sob certas condições, definidas entre parênteses. Essas condições são conhecidas como `Boolean` condições e podem ser apenas `true` ou `false`.

   Quando a condição é avaliada como `true`, o programa executa a instrução dentro das chaves. Quando a condição booleana for avaliada como `false`, a instrução dentro das chaves não será executada.

   ```
   //pseudocodigo if (*condição é verdadeira*) { *instrução é executada*
   }
   ```

   Exemplo

   ```
   var minhaVariavel = true; if (minhaVariavel) { console.log("é
   verdade") } else { console.log("é falso) }
   ```

   Crie uma variável chamada `hojeVaiChover` e uma condicional `if` que imprima na tela o seguinte:

   - "Leve seu guarda-chuva, hoje vai chover" se o valor for verdadeiro
   - "Fique tranquilo, hoje será um dia de sol" se o valor for falso

2. **Crie variáveis para cada uma das comparações e imprima o resultado delas**
    a. 5 é maior que 1
    b. 2000000 é menor que 3000000000
    c. "Banana" é igual a "Banana"
    d. "50" é estritamente igual a 50
    e. "Carro" é diferente de "bicicleta"
    f. 50 é maior que 27
    g. 975 é menor ou igual a 1200

```
// exemplos de operadores de comparação == - valor igual === valor igual e tipo igual != valor diferente !== valor e/ou tipo diferente > maior que < menor que >= maior ou igual <= menor ou igual
```

3. Os objetos são úteis para armazenar dados de forma estruturada e podem representar objetos do mundo real, como um gato.

```
const gato = { "nome": "Whiskers", "patas": 4, "caudas": 1, "principalInimigo": "água" };
```

Faça um objeto que representa um cachorro chamado `meuCachorro` que contém as propriedades `nome` (uma string) `patas`, `caudas` e `principalAmigo`.

Você pode definir essas propriedades de objeto para quaisquer valores que desejar, desde que `nome` e `principalAmigo` sejam string `patas` e `caudas` sejam números;

Após construir o objeto, imprima-o na tela.

4. Existem duas maneiras de acessar as propriedades de um objeto: notação de ponto (`.`) e notação de colchetes (`[]`). A notação de pontos é o que você usa quando sabe o nome da propriedade que está tentando acessar com antecedência. Aqui está um exemplo de como usar a notação de ponto (`.`) para ler a propriedade de um objeto:

```
const meuObj = { prop1: "val1", prop2: "val2" }; const primeiraProp =
meuObj.prop1; const segundaProp = meuObj.prop2;
```

`primeiraProp` teria um valor da string `val1` e `segundaProp` teria um valor da string `val2`.

Leia os valores de propriedade de `roupas` usando a notação de ponto. Defina a variável `valorChapeu` igual à propriedade do objeto `chapeu` e defina a variável `valorCamiseta` igual à propriedade do objeto `camiseta`.

```
// Setup const roupas = { "chapeu": "bone", "camiseta": "camiseta do
batman", "sapatos": "tenis da nike" }; // Apenas modifique dessa linha
para baixo const valorChapeu = roupas; const valorCamiseta = roupas;
console.log(valorChapeu); console.log(valorCamiseta);
```

5. A segunda maneira de acessar as propriedades de um objeto é a notação de colchetes (`[]`). Se a propriedade do objeto que você está tentando acessar tiver um espaço em seu nome, você precisará usar a notação de colchetes.

   No entanto, você ainda pode usar a notação de colchetes nas propriedades do objeto sem espaços.

   Aqui está um exemplo de como usar a notação de colchetes para ler a propriedade de um objeto:

   ```
   const meuObjeto = { "Nome com espaco": "Beyonce", "Mais espaco": "Jay-z", "SemEspaco": "Blue" }; meuObjeto["Nome com espaco"]; meuObjeto['Mais espaco']; meuObjeto["SemEspaco"];
   ```

   `meuObjeto["Nome com espaco"]` seria a string `Beyonce` , `meuObjeto['Mais espaco']` seria a string `Jay-z` e `meuObjeto["SemEspaco"]` seria a string `Blue` .

   Observe que os nomes das propriedades com espaços devem estar entre aspas (simples ou duplas).

   Leia os valores das propriedades `a entrada` e `o drink` do objeto `pedido` uso da notação de colchetes e atribua-os a `valorEntrada` e `valorDrink` respectivamente, por fim, imprima-os.

   ```
   // Setup const pedido = { "a entrada": "hamburger", "o adicional": "vegetais", "o drink": "coca-cola" }; // Apenas modifique dessa linha para baixo const valorEntrada = pedido; const valorDrink = pedido;
   ```

6. Você pode adicionar novas propriedades a objetos JavaScript existentes da mesma maneira que os modificaria ou acessaria. Veja como adicionaríamos a propriedade `brinquedoFavorito` no nosso objeto `gato` .

   ```
   gato.brinquedoFavorito = "arranhador"; //ou gato["brinquedoFavorito"] = "arranhador";
   ```

   Adicione uma propriedade `brinquedoFavorito` ao objeto `meuCachorro` criado anteriormente e defina-a, como "bolinha". Você pode usar a notação de ponto ou colchete. Imprima-o.

7. Uma outra coisa muito útil que conseguimos fazer em relação a objetos, é deletar propriedades deles.

```
const gato = { "nome": "Whiskers", "patas": 4, "caudas": 1,
"principalInimigo": "água", "cor": "caramelo" }; delete gato.cor;
```

Depois dessa última linha, teríamos o seguinte resultado

```
const gato = { "nome": "Whiskers", "patas": 4, "caudas": 1,
"principalInimigo": "água", };
```

Delete a propriedade `patas` do objeto `meuCachorro` e imprima-o em seguida. Você pode usar a notação de ponto ou colchete.

8. Depois de criar um objeto JavaScript, você pode atualizar suas propriedades a qualquer momento, da mesma forma que atualizaria qualquer outra variável. Você pode usar a notação de ponto ou colchete para atualizar.

Por exemplo, vejamos `gato`:

```
const gato = { "nome": "Whiskers", "patas": 4, "caudas": 1,
"principalInimigo": "água", }; gato.nome = "Bichano";
```

Depois dessa última linha, teríamos o seguinte resultado

```
const gato = { "nome": "Bichano", "patas": 4, "caudas": 1,
"principalInimigo": "água", };
```

Altere a propriedade `nome` do objeto `meuCachorro` definindo-a como "Pluto". Imprima o objeto em seguida.

8. Com as variáveis `array` JavaScript , podemos armazenar vários dados em um só lugar. Você inicia uma declaração de matriz com um colchete de

abertura, termina com um colchete de fechamento e coloca uma vírgula

```
const meusCarros = ["fusca", "corsa", "corolla"];
```

Modifique a nova matriz `meuArray` para que contenha uma string e um número (nessa ordem). Imprima-a.

```
const meuArray = [];
```

9. Podemos acessar os dados dentro de arrays usando *índices* .Os índices de array são escritos na mesma notação de colchetes que as strings e objetos usam, exceto que em vez de especificar um caractere ou valor, eles estão especificando uma entrada no array. Como strings, arrays usam indexação *baseada em zero* , então o primeiro elemento em um array tem um índice de `0` .

   ```
   const meuArray = [50, 60, 70]; meuArray[0];
   console.log(meuArray[0]); const data = meuArray[1];
   console.log(data);
   ```

   `array[0]` é agora `50` e `data` tem o valor `60` .

   Crie uma variável chamada `meuValor` e defina-a para ser igual ao primeiro valor `meuArray` usando a notação de colchetes, e imprima seu valor.

   ```
   const meuArray = [50, 60, 70];
   ```

10. Ao contrário das strings, as entradas dos arrays são *mutáveis* e podem ser alteradas livremente, mesmo que o array tenha sido declarado com `const`.

```
const nossoArray = [50, 40, 30]; nossoArray[0] = 15;
```

`nossoArray` agora tem o valor `[15, 40, 30]`.

Nota: Não deve haver nenhum espaço entre o nome do array e os colchetes, como `array [0]`. Embora o JavaScript seja capaz de processar isso corretamente, isso pode confundir outros programadores que estejam lendo seu código.

Modifique os dados armazenados no índice `0` de `meuArray` para um valor de `45` e `imprima-o.`

```
const meuArray = [23, 44, 98];
```

11. Aprendemos que os loops em Javascript podem ser usados em situações em que precisamos executar uma mesma ação diversas vezes porém usando um valor diferente em cada uma dessas execuções.

Por exemplo, suponhamos que temos uma lista (array) com várias tarefas (to-do list) para serem feitas atribuídas a uma variável `toDoList`, para saber o que precisamos fazer temos que imprimir essas tarefas no terminal. Uma das formas que aprendemos foi acessando valores de lista pelo índice, então poderia ser algo como:

```
const listaTarefas = ["Varrer a sala", "Lavar Roupa", "Comprar tomates",
"Enviar email"] console.log(listaTarefas[0]) //Varrer a sala
console.log(listaTarefas[1]) //Lavar Roupa console.log(listaTarefas[2])
//Comprar tomates console.log(listaTarefas[3]) //Enviar email
```

Porém, as instruções `console.log` são exatamente iguais, a não ser pelo item que queremos exibir, certo? Uma das formas de executar uma mesma ação várias vezes é usando o `for`, que tem a seguinte sintaxe:

Agora rode esse trecho de código e veja como ele acontece:

```
const listaTarefas = ["Varrer a sala", "Lavar Roupa", "Comprar tomates",
"Enviar email"] for(let indice = 0; indice < listaTarefas.length;
indice++) { console.log(listaTarefas[indice]); }
```

Testando os conhecimentos! Crie uma função chamada `criaPares` que receba como parâmetro o array `valoresNum`, nessa função deve ser executado um `for` que imprime na tela apenas os valores pares. Lembre-se que a sintaxe de um for é a seguinte:

```
for ([inicialização]; [condição]; [expressão final]) declaração
```