Aula 1 - Introdução a Javascript

Variáveis e Tipos

Tipo	Descrição
number	Valor numérico
string	texto
boolean	verdadeiro ou falso
function	Bloco de código com uma finalidade especifica
object	conjunto de dados sobre uma propriedade
undefined	valor indefinido
null	valor nulo

Como declarar uma variável

existem três maneiras: *var*, *const, let*

```
var name = 'icaro joel';

const id = 25346;

let age = 19;
```

Todas as 3 formas declaram uma variável, porem temos que entender as diferenças. O que difere elas é o *escopo*.

```
// as chaves {} representam o inicio e o fim do escopo da condicional
if (true) {
  // inicio

  var name = 'icaro';
```

```
  console.log(name); // icaro

  // fim
};

console.log(name); // icaro
```

Se você executar esse script, vai perceber que no terminal icaro vai aparecer duas vezes. A razão disso ocorrer é o var, ele não se limita ao escopo da função, trazendo muitos problemas num desenvolvimento de uma aplicação maior, pois pode-se redeclarar a variável perdendo os dados anteriores.

⚠️ Por isso usar o var não é uma boa prática ⚠️.

Palavra-chave	var	let	const
Escopo global	Sim	Não	Não
Escopo da função	Não	Sim	Sim
Escopo de bloco	Não	Sim	Sim
Pode ser reatribuído	Sim	Sim	Não

como o const se diferencia do let? Simples o const é uma constante, **o valor não pode ser alterado**, já no let os valores podem ser reatribuídos a vontade, Ex:

```
const favoriteFood = "Churrasco";
favoriteFood = "Lasanha";

console.log(favoriteFood); // TypeError: Assignment to constant variable.
```

```
let favoriteFood = "Churrasco";
favoriteFood = "Lasanha";

console.log(favoriteFood); // Lasanha
```

Tipos de Variáveis

o operador typeof é utilizado para saber o tipo da variável, Ex:

```
const name = "icaro joel";
const age = 19;
const isSlave = true;

console.log(typeof name); // string
console.log(typeof age); // number
console.log(typeof isSlave); // boolean
console.log(typeof id); // undefined
```

Exercícios

1. crie uma const chamada name e atribua a ela o seu nome.
2. crie uma const chamada age e atribua a ela a sua idade.
3. crie uma const chamada city e atribua a ela a cidade onde mora.
4. crie uma const chamada isSlave e atribua a ela se é bolsista (true ou false).
5. use o console.log() para exibir na tela as variáveis que você criou.

Boas práticas em nomeação de variáveis

- **Usar nomes descritivos:** o nome da variável deve ser descritivo o suficiente para que o objetivo da variável possa ser entendido sem a necessidade de comentários adicionais;
- **Usar camelCase:** em JavaScript, o estilo mais comum é o camelCase, em que a primeira palavra começa com letra minúscula e as subsequentes, com letra maiúscula, sem espaços;

- **Evitar abreviações:** a menos que as abreviações sejam amplamente conhecidas e aceitas, evite usá-las em nomes de variáveis. É preferível usar nomes completos e descritivos;
- **Evitar nomes muito longos demais:** nomes muito longos podem ser difíceis de digitar e tornar o código mais difícil de ler (nem muito curto, nem muito longo);
- **Usar convenções de nomenclatura padrão:** siga as convenções de nomenclatura padrão para o idioma e o *framework* que você está usando. Por exemplo, o padrão de nomenclatura no *Node.js* é usar *camelCase* para funções;
- **Evitar nomes reservados**: evite "var", "let", "const", "function", "if", "else" etc.

Exemplo:

```
// nomeando variaveis para strings
const firstName = 'Icaro Joel';
const birthCity = 'Petrolina';

// nomeando variaveis para numbers
const age = 19;
const height = 1.85;
```

Operações aritméticas

Precisamos de operadores para realizar operações em qualquer linguagem de programação, o js não é exceção, além dos aritméticos tem os lógicos, de atribuição, de comparação, etc. Os operadores são:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Obs: podemos incrementar com `++` e realizar um decremento com `--`, porém é recomendado trabalhar com o `+=` ou `-=` que podem utilizar valores personalizados, ex: `contador += 2` , pois

%	Modulo(resto da divisão)
+=	Incremento
-=	Decremento

`++` e `--` só trabalham com o incremento ou decremento de 1.

Exemplo:

```
// Soma
console.log(10 + 12); // 22

// Subtração
console.log(100 - 40); // 60

// Multiplicação
console.log(6 * 3); // 18

// Divisão
console.log(20 / 2); // 10
```

Exercícios

1. Crie uma constante chamada `base` e atribua a ela o valor 5;
2. Crie uma constante chamada `height` e atribua a ela o valor 8;
3. Crie uma constante chamada `area` e atribua a ela o resultado da multiplicação da `base` pela `height`;
4. Crie uma constante chamada `perimeter` e atribua a ela o resultado da soma de todos os lados de um retângulo.

Operadores de comparação

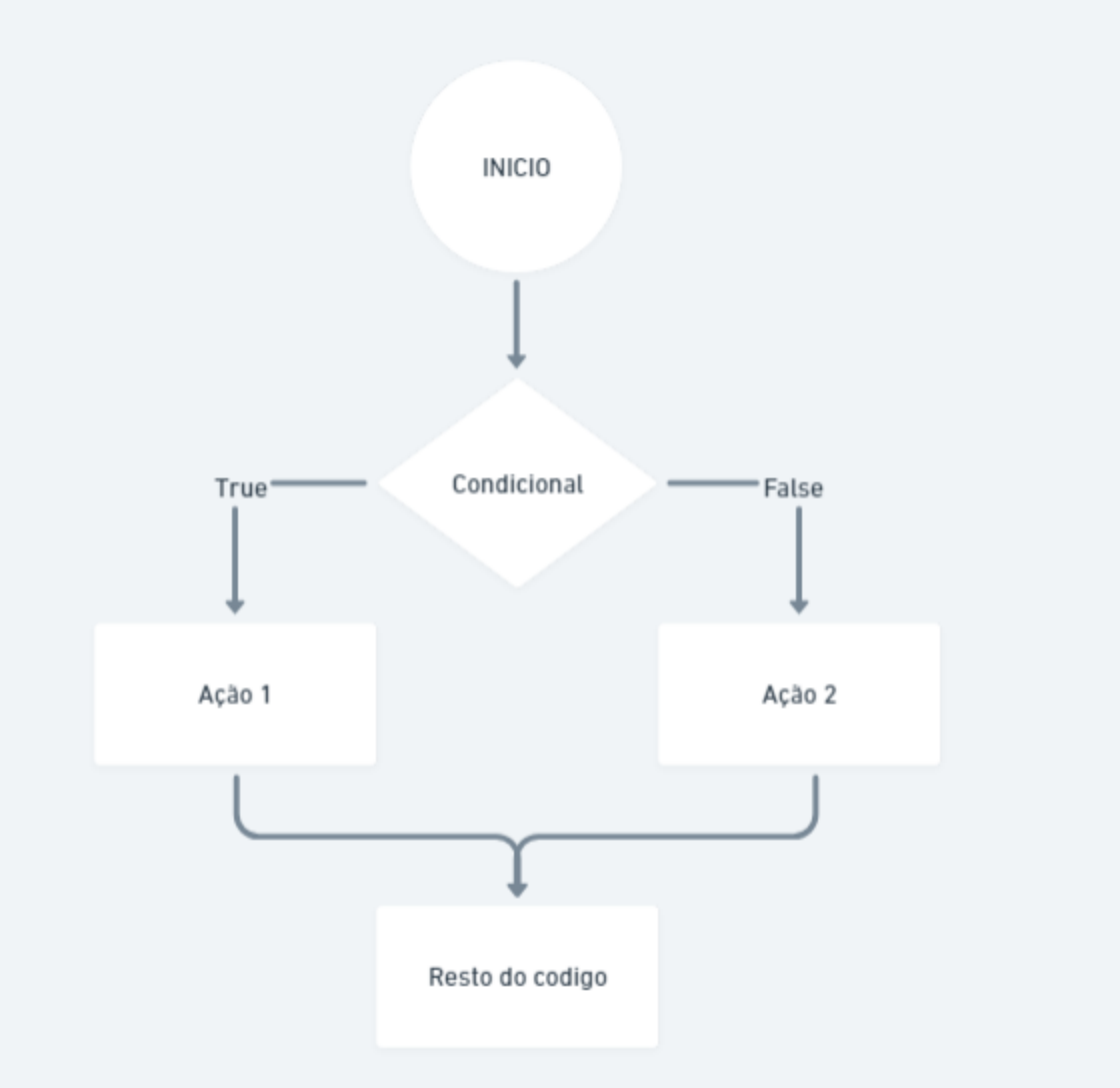
utilizados para realizar comparações para tornar as soluções mais simples e legiveis

Operador	Descrição
==	Igual, tem o mesmo valor
===	Idêntico, tem o mesmo valor e tipo
≠	diferente de, tem um valor diferente de
>	Maior que
≥	Maior igual que
<	Menor que
≤	Menor igual que

Condicionais

if … else

Fluxograma funcionamento de uma condicional



Exemplo:

```
const money = 10000;

if (money < 5000) {
  console.log('você recebeu um pix!');
} else {
  console.log('você não recebeu um pix!');
}
```

Neste exemplo, a mensagem exibida é você não recebeu um pix!, porque money é maior que 5000.

Ternário

Você acabou de ver como construir condições com `if...else`. No entanto, há uma sintaxe mais simplificada para a construção de lógicas condicionais: o ternário

Considere o seguinte problema: você precisa guardar dentro de uma constante a mensagem que traz a informação se uma pessoa pode entrar na festa. Com `if...else`, o código seria o seguinte:

```
const personAge = 17;
let canEntry;

if (personAge >= 18) {
  canEntry = 'Pode entrar na festa!';
} else {
  canEntry = 'Não pode entrar na festa!';
}

console.log(canEntry); // Não pode entrar na festa!
```

já com o ternário seria:

```
const personAge = 17;

const canEntry = personAge >= 18 ? 'Pode entrar na festa!' : 'Não pode entrar na festa!';

console.log(canEntry); // Pode votar!
```

O operador ternário é mais fácil de ser utilizado e entendido, sem falar que é muito mais sucinto do que escrever um bloco com `if...else` – o que gera um código mais limpo e simples.

Por outro lado, é bom saber que o operador ternário ***não substitui*** as expressões condicionais tradicionais. Se você precisa executar um conjunto de código a partir de uma condição, o operador ternário não vai te dar essa possibilidade. Além disso, em qualquer situação em que exista mais de uma condição a ser avaliada, o mais simples será utilizar as opções já estudadas.

Exercícios

1. Crie uma variável chamada `note` que receba a nota de uma pessoa candidata em um desafio técnico e atribua um valor entre 1 e 100;
2. Implemente uma lógica que verifique se a pessoa candidata foi `aprovada`, `reprovada` ou se está na `lista de espera`. Para isso, considere estas informações.
    - Se `grade` for maior ou igual a 80, armazene na variável `message`: “Parabéns, você faz parte do grupo de pessoas aprovadas!”;
    - Se `grade` for menor que 80 e maior ou igual a 60, armazene na variável `message`: “Você está na nossa lista de espera.”;
    - Se `grade` for menor que 60, armazene na variável `message`: “Infelizmente, você reprovou.”.

Operadores lógicos

A próxima ferramenta que vamos aprender são os operadores lógicos. Na linguagem JavaScript, há três principais operadores lógicos: `&&`, `||` e `!`. Podemos nos referir a eles como “*AND*“, “*OR*“ e “*NOT*“, respectivamente.

Operador	Descrição

&&	AND, e
||	OR, ou
!	NOT, negação

AND

O operador "*AND*" (`&&`) é binário. Isso significa que ele precisa de dois elementos para funcionar corretamente.

Para abstrair seu funcionamento, pense na seguinte situação: Você está em uma padaria para tomar café da manhã e faz o pedido: "Eu gostaria de um café `E` um pão na chapa, por favor".

Seria frustrante se você recebesse apenas o café ou apenas o pão, não é? Ou se recebesse um pão na chapa acompanhado de um caldo de cana, por exemplo. Isso acontece porque a expectativa é de que as duas condições sejam atendidas corretamente. Esse é exatamente o papel do `&&`. Ele só vai retornar `true` se as duas operações que estão em volta dele forem consideradas verdadeiras.

```
const food = 'pão';
const drink = 'café';

if (drink === 'café' && food === 'pão') {
  console.log('Obrigado!');
} else {
  console.log('Erraram meu pedido.');
}
```

Valor 1	Valor 2	Resultado
V	V	V
V	F	F
F	V	F
F	F	F

OR

Com o operador `OR` (ou `||`, no JavaScript), é necessário que apenas uma das condições seja verdadeira: *isso* `OU` *aquilo*.

Imagine novamente que você está em uma padaria e pede um café. Dessa vez, se não tiver café, você indica para a pessoa atendente que pode ser um suco de laranja. Como escreveríamos isso em JavaScript?

```
const principalDrink = 'café';
const alternativeDrink = 'suco de laranja';

if (principalDrink === 'café' || alternativeDrink === 'suco de laranja') {
  console.log("Agradeço por me atender :D");
} else {
  console.log("Ei, não foi isso que eu pedi!");
}
```

O símbolo `||` é a representação em caracteres desse operador. Novamente, são necessários dois elementos em torno dele para que funcione corretamente. Para que seu retorno seja verdadeiro, um de seus valores deve ser `true` **ou** ser considerado *truthy*.

> *um valor truthy é qualquer valor que seja avaliado como verdadeiro em uma condicional, caso contrário são chamados de falsy. Os valores falsy incluem:* `false`, `0`, `''`, `null`, `undefined`, e `NaN`. *Por exemplo, uma string que não esteja vazia ou um número diferente de zero são considerados valores truthy.*

Valor 1	Valor 2	Resultado
V	V	V
V	F	V
F	V	V
F	F	F

NOT

Vamos ver um exemplo do que o operador *NOT* (ou `!`, no JavaScript) faz.

```
console.log((2 + 2) === 4); // true
```

Esse código deve retornar `true`, não é? Afinal, 2 + 2 resultar em 4 é uma declaração verdadeira. E se adicionarmos o operador `NOT` antes dessa declaração?

```
console.log(!(2 + 2) === 4); // false
```

Estamos diante de um operador muito poderoso. Ele pode **inverter** o valor *booleano* de um elemento. Isso mesmo! Se tivermos uma variável ou um valor considerado `true`, podemos gerar o resultado oposto simplesmente fazendo `!variável`, ou seja, `false`.

Então, sabendo que o resultado original da operação é `true`, quando é inserido o operador `NOT` antes dela, obtemos o valor contrário à resposta final, que é `false`.

O conceito de `truthy` e `falsy` também se aplica nesse caso, por isso não estamos limitados apenas aos tipos primitivos *booleanos*.

Exercícios

1. Faça um script que pede duas notas de um aluno. Em seguida ele deve calcular a média do aluno e dar o seguinte resultado:

- A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
- A mensagem "Reprovado", se a média for menor do que sete;
- A mensagem "Aprovado com Distinção", se a média for igual a dez.

2. Faça um script que leia três números inteiros e mostre o maior deles.
3. Faça um script que pergunte em que turno você estuda. Imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.
4. As Inova resolveu dar um aumento de salário aos seus colaboradores e lhe contrataram para desenvolver um script que calculará os reajustes. Faça um script que recebe o salário de um colaborador e o reajuste segundo o seguinte critério, baseado no salário atual:

   - salários até R$ 280,00 (incluindo) : aumento de 20%
   - salários entre R$ 280,00 e R$ 700,00 : aumento de 15%
   - salários entre R$ 700,00 e R$ 1500,00 : aumento de 10%
   - salários de R$ 1500,00 em diante : aumento de 5%.

   Após o aumento ser realizado, informe na tela:

   - salário antes do reajuste;
   - percentual de aumento aplicado;
   - valor do aumento;
   - novo salário, após o aumento.

Exercícios

1. Elabore alguns códigos e imprima o resultado no console usando o `console.log()`, um para cada operação aritmética básica. É necessário que seu código tenha duas variáveis, `num` e `num2`, definidas no começo com os valores que serão operados. Escreva códigos para:

   - Adição `num1 + num2`
   - Subtração `num1 - num2`

- Multiplicação `num1 * num2`
- Divisão `num1 / num2`
- Módulo `num1 % num2`