5. Objects

Em JavaScript, a classe Object (ou objeto) funciona como uma coleção de propriedades, as propriedades de um objeto podem ser acessadas através de um nome (chave), chamamos isso de conjunto chave-valor. Os valores podem ser de qualquer tipo, como `string`, `number`, `boolean`, `function` ou até mesmo outro objeto. Dessa maneira, temos uma forma mais flexível de organizar e gerenciar nossos dados.

Os objetos são muito semelhantes aos arrays, primeiramente por serem tipos referenciados, ou seja, as variáveis que armazenam objetos não recebem diretamente o valor do objeto, mas sim uma referência na memória para aquele objeto, e assim como os Arrays, existem duas formas de declarar um Object:

```
// 1. Através do construtor Object
const a = new Object();
// {}

// 2. Instanciando um objeto vazio
const b = {};
// {}
```

Os objetos são especialmente úteis quando desejamos armazenar diversos valores relacionados a um mesmo elemento, por exemplo:

```
let pessoa = {
        nome: "Anderson",
        idade: 34,
        profissao: "Engenheiro",
        cumprimentar: function() {
                return "Olá, meu nome é " + this.nome + " e tenho " +
this.idade + " anos.";
        }
};
```

Para acessar uma propriedade definida, podemos utilizar alguns métodos, no primeiro deles utilizamos a chave fixa no código:

```
console.log(pessoa.nome);
// Anderson
```

A outra forma, serve para buscarmos uma chave dinâmica, ou seja, através do uso de outra variável contendo o nome da chave:

```
const chave = "idade";

console.log(pessoa[chave]);
// 34
```

Por fim, temos as sintaxes de desestruturação e "espalhamento", que servem para extrair valores específicos, ou todos os valores, respectivamente:

```
const { nome } = pessoa;

const clonePessoa = {  ... pessoa };
```

Conforme dito anteriormente, por ser tratar de um tipo referenciado, atribuir um objeto a duas variáveis faz com que as alterações afetem ambas as variáveis, pois a alteração ocorreu na referência. Porém, ao utilizarmos o operador de espalhamento para declarar a variável `clonePessoa`, todos os valores da variável `pessoa` são clonados em uma nova referência, fazendo com que as variáveis `pessoa` e `clonePessoa` sejam independentes.

Por fim, caso o valor selecionado seja uma função, podemos buscar a função através de sua chave, e executá-la normalmente:

```
console.log(pessoa.cumprimentar());
// Olá, meu nome é Anderson e tenho 34 anos.
```

this

Como outras linguagens, JavaScript possui a palavra reservara `this`, que representa uma referência dinâmica para o contexto atual do código, quando falamos de objetos, ao utilizar a palavra `this` (como no exemplo acima), o valor referenciado será o próprio objeto. Dessa forma, podemos acessar um atributo de um objeto dentro de uma função do mesmo objeto acessando uma chave de `this`:

```
const pessoa = {
        // outros atributos
        printaNome: function() {
                console.log(this.nome);
```

```
        }
}
```

Desafios

Fáceis

1. Crie um objeto "carro" com os seguintes com os seguintes atributos:
    1. Marca
    2. Modelo
    3. Ano
    4. Cor
    5. Ligado
2. Crie uma função que retorna a marca do carro
3. Crie uma função para `ligar` o carro, e outra para `desligar`

Médios

1. Crie três objetos "livro", com os seguintes atributos:
    1. Título
    2. Autor
    3. Ano
2. Crie uma função que recebe dois livros, e verifica se eles são do mesmo autor
3. Crie uma função chamada `descrever`, que retorna as informações do livro no seguinte formato:
   "O livro [titulo] foi escrito por [autor] no ano de [ano]"

Difíceis

1. Crie um objeto turma com os seguintes atributos:
    1. Nome
    2. Turno
    3. Alunos
2. Insira 3 alunos na turma, os alunos devem ter os seguintes atributos:
    1. Nome
    2. Idade
3. Crie uma função que retorna os nomes dos alunos da turma

4. Crie uma função que retorna a média das idades dos alunos da turma