Array

Definição: Array ou Vetores são estruturas utilizadas para armazenar uma lista de dados.

Como identificar: utilizamos [ ]

Como declaramos: em uma variável, em vez de armazenar um valor único, podemos armazenar vários valores separados por vírgula, dentro de colchetes [ ].

Implementação de Referência: lista de compras.

Antes de iniciar a implementação, pergunte se os alunos já aprenderam let ou ainda estão usando var. Siga o que eles já aprenderam.

```
let listaDeCompras = ["pão", "leite", "farinha", "ovos"];
```

Como acessar o valor:

```
nomeDaVariavel[posicaoIniciandoNoZero]
```

Ex.: Se quisermos trazer o 'pão':

```
listaDeCompras[0] // este código retorna 'pão'
```

Ex.: Se quisermos trazer o 'leite':

```
listaDeCompras[1] // este código retorna 'leite'
```

E assim por diante.

JSON

Definição: JSON ou Java Script Object Notation são estruturas utilizadas para armazenar mais do que um valor em uma mesma variável, representando por exemplo objetos da vida real

Como identificar: utilizamos { }

Como declaramos: em uma variável, em vez de armazenar um valor único, podemos armazenar vários valores, dentro de chaves { } e cada um com seu próprio "identificador", em uma estrutura comum de "chave-valor".

Implementação de Referência: livro.

Antes de iniciar a implementação, pergunte se os alunos já aprenderam let ou ainda estão usando var. Siga o que eles já aprenderam.

Em um livro, temos vários atributos, por exemplo o autor, o título, o ano... cada um destes atributos é uma "chave" e cada chave tem um "valor"

```
let meuLivro = {"autor": "J.K. Rowling","titulo": "Harry Potter e
a Pedra Filosofal", "ano": 1998};
```

Como acessar o valor:

```
nomeDaVariavel.chave
```

Ex.: Se quisermos trazer o 'ano':

```
meuLivro.ano // este código retorna 1998
```

Ex.: Se quisermos trazer o 'autor':

```
meuLivro.autor // este código retorna 'J.K. Rowling'
```

E assim por diante.

setTimeout / setInterval

Introdução: Na vida real, fazemos tudo de maneira "assíncrona", ou seja, podemos iniciar mais de uma tarefa e executá-las simultaneamente, sem que uma interfira na outra. Exemplo: quando fazemos macarrão, podemos colocar a água para ferver e não precisamos aguardar ferver para fazer o molho; podemos fazer o molho enquanto a água aquece e só então voltar à panela e colocar o macarrão. Quando programamos, também podemos programar para que as funções aconteçam em simultâneo ou sem que elas travem umas às outras.

Quando invocamos uma função, ela é executada quase que instantaneamente. Podemos fazer com que uma função seja invocada uma vez daqui um intervalo de tempo ou regularmente a cada X segundos, com setTimeout e setInterval.

Como identificar: usamos as palavras "setTimeout" se quisermos que uma função seja invocada daqui um determinado tempo e "setInterval" se quisermos que uma função seja invocada regularmente com um determinado intervalo de tempo.

Como declaramos:

```
setTimeout(funcaoDesejada, tempoEmMilissegundos)

setInterval(funcaoDesejada, intervaloDeTempoEmMilissegundos)
```

Implementação de Referência:

Quando estiver implementando, informe que `() =>` é utilizada em casos específicos e é uma maneira de declarar função sem dar um nome, por exemplo.

setTimeout

Digite a função abaixo e informe que ela irá ser executada apenas uma vez e instantaneamente.

```
console.log('oi, agora mesmo')
```

Digite a função abaixo e informe que ela irá ser executada apenas uma vez, daqui 5 segundos

```
setTimeout(() => console.log('oi, mas daqui a pouco'), 5000)
```

setInterval

Digite as duas funções abaixo e informe que elas ocorrerão regularmente e sem que se interrompam.

```
var dizerOla = setInterval(() => console.log('OLÁ!'), 2000);
var fazerPsiu = setInterval(() => console.log('psiu'), 5000);
clearInterval(dizerOla) // este código interrompe as repetições
clearInterval(fazerPsiu) // este código interrompe as repetições
```

Funções com parâmetros e com retorno

Introdução: Quando executamos as funções, é interessante que reutilizemos a mesma estrutura para algumas variações.

Implementação de Referência:

```
function somarDoisMaisDois() {
      return 2+2 // return utilizamos para "devolver para quem pediu". Se o backend pediu, retorne para ele. Se o front, idem.
}

function somarDoisMaisTres() {
      return 2+3
}
```

Assim é insustentável, precisaria ter milhares de funções fazendo praticamente a mesma coisa. Faremos então com que a função fique mais reutilizável.

```
function somar(a, b) {
      return a+b
}
```

Assim, podemos somar 2 e 2, 2 e 3, 10 e 1000, 50 e 20... Execute alguns exemplos:

```
somar(2,2) // este código retorna 4

somar(2,3) // este código retorna 5

somar(10, 50) // este código retorna 60
```

sessionStorage

Introdução: Quando estamos navegando nas páginas da internet, é normal que precisemos usar, na próxima página, uma informação que tínhamos na página anterior. Há várias maneiras de fazer isso e usando algumas variáveis que o navegador nos permite manipular é uma delas.

Passo a passo da explicação:

1. Execute o projeto Acquatec, efetue o login e mostre aos alunos que a informação de nome de usuário, que não foi inserida no momento do login, aparece em tela.
2. Abra as ferramentas de desenvolvedor no navegador, abra a aba Application e mostre os valores salvos em SessionStorage
    a. Se tiver explicado o JSON no mesmo dia, mostre que as informações nesta aba seguem a mesma estrutura de "chave-valor".
    b. Execute o código: `sessionStorage` no console e mostre a estrutura JSON explicada anteriormente.
    c. Execute o código `sessionStorage.NOVO_COOKIE = 'teste'` no console e volte à aba de Application. Informe que adicionamos ao objeto de SessionStorage um novo atributo, um novo par chave-valor.
3. Abra o código da API e mostre que no arquivo login.html o código armazena os valores de email, nome e id do usuário logado no sessionStorage.
4. Abra o código da API e mostre que no arquivo funções.js o código insere em innerHTML de componente em tela o valor que está em sessionStorage.

Definição: sessionStorage é um cookie que existe apenas na sessão da aba (mostre que a outra aba não contém o dado anterior no Chrome)

Analogia: uma variável global que é armazenada no seu navegador / uma gavetinha do seu navegador

Por que usar no projeto: armazenar as informações do usuário assim que efetuar o login para que não precise ficar buscando no banco (por exemplo, música favorita).