Estrutura de Dados: Arrays

Em um dado momento de qualquer projeto que estará trabalhando você precisará armazenar dados mais complexos que os habituais números e textos.

Nesse cenário o array é uma forma muito especial de armazenar dados estruturados em um programa. Já que, com apenas uma variável, um programa pode guardar *n* valores.

Antes de começarmos a explicar em detalhes do funcionamento do array, primeiro vamos entender o problema...

É uma fria manhã de inverno, você pega uma xícara de café, liga seu computador e começa a ler seus e-mails como de costume.

Bate o olho no Inbox e não pode deixar de ler o assunto desse e-mail: "Preciso de um programador urgente". Click, você abre depressa!

"Preciso de um programador para criar um pequeno aplicativo de controle de lista de convidados...". Dizia o e-mail.

Você troca uma ideia com o cliente e então você começa a jornada...

Você decide criar este "programinha" usando suas habilidades *JavaScriptcas* atuais. Só que logo percebe que existe um problema. Veja:

```
var nome1 = prompt("Digite o nome do convidado 1");
var nome2 = prompt("Digite o nome do convidado 2");
var nome3 = prompt("Digite o nome do convidado 3");
var nome4 = prompt("Digite o nome do convidado 4");
var nome5 = prompt("Digite o nome do convidado 5");
var nome6 = prompt("Digite o nome do convidado 6");
var nome7 = prompt("Digite o nome do convidado 7");
// ...
var nome112 = prompt("Digite o nome do convidado 112");
```

Você percebe no centésimo décimo segundo convidado que:

1. Seus dedos doem de tanto digitar a variável `nomeXXX`.
2. Existe uma limitação na quantidade de dados de entrada.

Claro que o cliente que receberá este programa não irá aceitar, então você tem duas escolhas:

- Fazer musculação nos dedos e digitar até a variável `nome12371235234`, ou.
- Pensar em uma forma melhor de guardar estes dados.

Com sorte (e estudo) você descobre que existe uma estrutura de dados que uma vez declarada aceita "infinitos" valores.

Este tipo de dado trabalha como um "Arquivo Indexado" (aqueles que estávamos acostumados a ver em escolas onde a gaveta 1 continham as fichas A, a gaveta 2 as fichas B, etc).

Você entende imediatamente e cria isso:

```
var nomes = [];
var quantConvidados = Number(prompt("Digite o número de convidados"));

for(i = 0;i < quantConvidados;i++) {
    nomes[i] = prompt("Digite o nome do convidado " + i);
}

// ...continua...
```

Parabéns! Você é mais um programador que acaba de aprender sobre Array e colocar mais algum no bolso!

Como um array é indexado

O array é uma variável que pode receber *n* valores. Fato!

A beleza do array é que esses valores podem ser acessados através de um índice numérico (como as gavetas do arquivo).

Essas "gavetas" possuem um índice numérico, que **inicia com 0** e termina com a quantidade de valores que você inseriu. Veja:

```
var nomes = ["Felipe", "Douglas", "João"];

console.log(nomes[0]);
console.log(nomes[1]);
console.log(nomes[2]);
// → "Felipe"
// → "Douglas"
// → "João"
```

Se você ir um pouco mais além, vai perceber que dá para usar perfeitamente o `for` para acessar os índices. Sendo assim você pode ser mais esperto e deixar um "robô" trabalhar por você:

```
var nomes = ["Felipe", "Douglas", "João"];

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

// → "Felipe"
// → "Douglas"
// → "João"
```

Adicionando e removendo valores no array

O array é uma estrutura de dados dinâmica, sendo assim você poderá inserir ou excluir valores ao seu bel prazer ou necessidade.

Por sorte um programador já deixou estes métodos prontos, chamou o comando de `push` para inserir e `pop` para excluir.

Veja os comandos do abençoado trabalhando:

```
var nomes = []; // avisa o JavaScript que a variável é um array

nomes.push("Felipe");  // Insere o nome no final do array
nomes.push("Douglas");
nomes.push("João");

console.log(nomes);
// → ["Felipe", "Douglas", "João"]
```

Ou então:

```
var nomes = []; // avisa o JavaScript que a variável é um array

nomes.push("Felipe");  // Insere o nome no final do array
nomes.push("Douglas");
nomes.push("João");
nomes.pop();           // Exclui o último valor do array
nomes.push("Mariana");

console.log(nomes);
// → ["Felipe", "Douglas", "Mariana"]
```

Há ainda outros métodos de manipulação e pesquisa de array, veja:

- `length` - Retorna a quantidade de elementos de um array.
- `forEach` - Itera sob um array.
- `unshift` - Adiciona no início do array.
- `shift` - Exclui um elemento no **início** do array.
- `splice` - Altera (insere/excluí) um elemento em uma **posição** específica do array.
- `indexOf` - Procura por um valor no array retornando seu índice.

Estes são os métodos principais para trabalhar com array, ainda existem muitos outros que você pode consultar na documentação oficial da linguagem: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

O método `length`

Bom, o `length` não é propriamente um método de manipulação e pesquisa de array. Ele na verdade é um método que retorna a quantidade de valores/elementos.

O `length` auxilia muito na em métodos de iteração, que é exatamente o caso do `for`. Veja:

```
var nomes = ["Felipe", "Douglas", "João"];

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

// → "Felipe"
// → "Douglas"
// → "João"
```

Como você pode ver, quando iteramos com o `for` não precisamos saber exatamente a quantidade de elementos para definir o critério de parada.

Basta apenas utilizarmos o método `length` para isso. Uau!

O método `forEach`

Se o `for` facilita o processo o `forEach` facilita mais ainda! Com esse método você pode iterar no array diretamente sem precisar consultar seu índice. Veja:

```
var nomes = ["Felipe", "Douglas", "João"];

nomes.forEach(function(nome) {
    console.log(nome);
});
// → "Felipe"
// → "Douglas"
// → "João"
```

Ou ainda acessar receber o seu índice:

```
var nomes = ["Felipe", "Douglas", "João"];

nomes.forEach(function(nome, indice) {
    console.log(indice + " - " + nome);
});
// → "0 - Felipe"
// → "1 - Douglas"
// → "2 - João"
```

Para ser bem sincero eu mais utilizo o método `forEach` para iterar com um array do que propriamente o `for`, já que esse método faz o trabalho duro pra gente.

Uma coisa legal é que se você reparar o método `forEach` utiliza uma função como parâmetro (como falamos nos capítulos sobre `functions` como parâmetros).

Isso chama-se **callback**.

Você poderia facilmente deixar o comando um pouco mais "verboso" fazendo desta forma:

```
var nomes = ["Felipe", "Douglas", "João"];

var callback = function(nome) {
    console.log(nome);
}

nomes.forEach(callback);
// → "Felipe"
// → "Douglas"
// → "João"
```

Não é comum usarmos desta forma, já que podemos passar uma `function` diretamente para a outra. Mas é legal de compartilhar isso contigo já que aprender coisas novas é sempre legal!

O método `unshift`

O método `push` insere valores no final do array, enquanto o método `unshift` faz o inverso, adiciona no início.

Vejamos o `unshift` em seu habit natural:

```
var nomes = ["Felipe", "Douglas"];

nomes.unshift("Bruno");

console.log(nomes);
// → ["Bruno", "Felipe", "Douglas"]
```

O método `shift`

Na pegada dos métodos de manipulação de cabeça de lista, o método `shift`, ao contrário do `pop`, exclui elementos no início do array. Veja:

```
var nomes = ["Bruno", "Felipe", "Douglas"];

nomes.shift();

console.log(nomes);
// → ["Felipe", "Douglas"]
```

Assim como o método `pop` (que exclui no final do array), este método retorna o valor do elemento excluído. Veja:

```
var nomes = ["Bruno", "Felipe", "Douglas"];

var excluido = nomes.shift();

console.log(nomes);
console.log(excluido);
// → ["Felipe", "Douglas"]
// → "Bruno"
```

O método `splice`

Vamos supor que você queira alterar elementos em uma posição específica do array. Para este trabalho nós utilizamos o método `splice`.

Este método é um pouco mais elaborado que seus primos, veja os parâmetros de acordo com a doc oficial:

```
array.splice(indice[, deleteCount[, elemento1[, ...[, elementoN]]]])
```

- `indice`: Índice o qual deve iniciar a alterar a lista. Se maior que o tamanho total da mesma, nenhum elemento será alterado. Se negativo, irá iniciar a partir daquele número de elementos a partir do fim.
- `deleteCount`: Um número de antigos elementos que devem ser removidos. Se o parâmetro `deleteCount` não é especificado, ou se é maior que o número de elementos restantes na lista iniciando pelo índice, então todos os elementos até o fim da lista serão deletados. Se `deleteCount` é 0, nenhum elemento é removido. Neste caso você deve especificar pelo menos um novo elemento.
- `elemento1, ..., elementoN`: Os elementos a adicionar na lista. Se você não especificar nenhum elemento, splice simplesmente removerá elementos da mesma.

Com todos esses parâmetros para se trabalhar o céu é o limite, vejamos o caso simples primeiro:

```
var nomes = ["Bruno", "Felipe", "Douglas"];

nomes.splice(1); // parâmetro deleteCount não foi específicado - todos os
elementos a partir do 1 serão apagados.

console.log(nomes);
// → ["Bruno"]
```

Ou:

```
var nomes = ["Bruno", "Felipe", "Douglas"];

nomes.splice(1, 1); // parâmetro deleteCount foi específicado - 1 elemento
após a posição 1 será apagado.
```

```
console.log(nomes);
// → ["Bruno", "Douglas"]
```

Ou então:

```
var nomes = ["Bruno", "Felipe", "Douglas"];

nomes.splice(1, 1, "Zezinho", "Fulano"); // parâmetro deleteCount foi
especificado - 1 elemento após a posição 1 será apagado. E também o
elemento "Zedinho" e "Fulano" serão na posição especificada.

console.log(nomes);
// → ["Bruno", "Zezinho", "Fulano", "Douglas"]
```

Maneiro!

O método `indexOf`

Vamos supor que você quer encontrar a posição de um elemento específico, por exemplo, o "Fulano" na lista de convidados.

Se você não for um programador do século XXI você provavelmente vai usar isso:

```
var nomes = ["Bruno", "Zezinho", "Fulano", "Douglas"];
var busca = "Fulano";
var indice = -1;

for(i = 0;i < nomes.length;i++) {
    if (nomes[i] == busca) {
        indice = i;
        break;
    }
}

console.log("Encontrei o " + busca + " em: " + indice);
// → Encontrei o Fulano em: 2
```

Caso o "Fulano" seja o penetra na festa:

```
var nomes = ["Bruno", "Zezinho", "Douglas"];
var busca = "Fulano";
var indice = -1;

for(i = 0;i < nomes.length;i++) {
    if (nomes[i] == busca) {
        indice = i;
```

```
            break;
        }
    }

    console.log("Encontrei o " + busca + " em: " + indice);
    // → Encontrei o Fulano em: -1
```

Trampo!

Agora usando a mesma função usando o `indexOf` (que aquele programador macanudo deixou pronto pra gente):

```
var nomes = ["Bruno", "Zezinho", "Fulano", "Douglas"];
var busca = "Fulano";
var indice = nomes.indexOf(busca);

console.log("Encontrei o " + busca + " em: " + indice);
// → Encontrei o Fulano em: 2
```

Ou então, sem o pilantrinha:

```
var nomes = ["Bruno", "Zezinho", "Douglas"];
var busca = "Fulano";
var indice = nomes.indexOf(busca);

console.log("Encontrei o " + busca + " em: " + indice);
// → Encontrei o Fulano em: -1
```

O retorno do método `indexOf` retorna a posição do valor que encontrou ou então `-1` caso não tenha encontrado nada.

É interessante observar que o `indexOf` faz apenas buscas exatas, caso sua pesquisa esteja faltando um pedaço da palavra, faltando acento ou até com letras caixa diferente (minúsculas ou maiúsculas)... Não vai funcionar.

Veja:

```
var nomes = ["Bruno", "Zezinho", "Fulano", "Douglas"];
var busca = "fulano"; // busca com o F minúsculo
var indice = nomes.indexOf(busca);

console.log("Encontrei o " + busca + " em: " + indice);
// → Encontrei o Fulano em: -1
```

Sendo assim o `indexOf` é bem útil quando estamos em busca de um array conhecido (que foi produzido pelo seu próprio programa). Se for algo que um usuário digitou, por experiência, não use isso!

String também é Array

Eu sei que isso vai te soar estranho mas uma string também é um array.

Strings no fundo são uma cadeia de caracteres. E como array elas se comporta como tal:

```
var mensagem = "Olá";

console.log(mensagem[0]);
// → O
```

Ou então:

```
var mensagem = "Olá";

for(i = 0;i < mensagem.length;i++) {
    console.log(mensagem[i]);
}
// → O
// → l
// → á
```

Devido as propriedades *arraylisticas* de uma string os mesmos métodos de array podem ser aplicados tranquilamente!

Desafio

Com o conceito de array em mente, implemente na sua calculadora o cálculo de média aritmética.

Resposta

```
var numeros = [];

while(true) {
    var numero = prompt("Digite um número ou S para sair");

    if (numero == 'S') {
        break;
    } else {
        numeros.push(Number(numero));
    }
}

var soma = 0;

for (i = 0;i < numeros.length;i++) {
    soma += numeros[i];
}

var resultado = soma / numeros.length;

console.log(resultado);
```