CONCATENAÇÃO

Concatenar é uma termo de programação que significa "juntar" ou "colocar junto".

Concatenação

ARRAYS

Anteriormente aprendemos que para juntar um ou mais arrays a um já existente, usamos a função *concat*

```
var array1 = ['ola'];
var array2 = ['de novo'];

array1.concat(array2);
// [ 'ola', 'de novo' ]
```

ARRAYS e OBJECTS

É possível concatenar um objecto a um array, e não ao contrário, visto que a função *concat* é relativa a arrays.

```
var array1 = ['ola'];
var object1 = { nome: 'isto é um nome!' };

array1.concat(object1);
// [ 'ola', { nome: 'isto é um nome!' } ]
```

STRINGS

Para colocar strings juntas em JavaScript, usamos o operador (+)

```
var variavel = 'ola' + ' ' + 'de novo';
// 'olá de novo'

var texto = "de novo";
var variavel = 'ola' + ' ' + texto;
// 'olá de novo'
```

STRINGS e NÚMEROS

Para colocar strings e números juntos em JavaScript, usamos o operador (+).

```
var variavel = 'ola' + 5;
// 'ola5'

var variavel = 5 + 5;
// '55'
```

Exercício

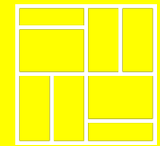
CONCATENAÇÃO

- Juntem as strings: ‘Isto é um teste’ e ‘tu vais passar!’;
- Formem uma string com strings e números para obter o resultado:
  ‘Isto é 1 string com 2 números’

PROPRIEDADES GLOBAIS

- Propriedades globais retornam um valor simples;

- Não têm propriedades ou métodos.



Propriedades globais

UNDEFINED()

O valor global undefined representa um valor indefinido.

```
var variavel = undefined;
var outraVariavel;
```

NAN

A propriedade global NaN é um valor especial que significa Not-A-Number (não é um número).

```
var variavel = 3 * 'a';
// NaN
```

NULL()

O valor null representa um valor nulo ou "vazio".

```
var variavel = null;
```

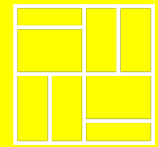
INFINITY

A propriedade global Infinity é um valor numérico que representa o infinito.

```
var variavel = 2**1024
// Infinity
```

FUNÇÕES GLOBAIS

Funções globais são funções que são chamadas globalmente ao invés de num objeto.



FUNÇÕES GLOBAIS

PARSEINT(argumento)

Analisa um *argumento* string (convertendo-o para uma string primeiro caso necessário) e retorna um inteiro.

```
parseInt("3");
// 3
```

PARSEFLOAT(argumento)

Analisa um *argumento* (convertendo-o para uma string primeiro caso necessário) e retorna um número decimal.

```
parseFloat("3.3");
// 3.3
```

ISNAN()

Determina se o valor é NaN ou não. Se necessário, o parâmetro é primeiro convertido a um número.

```
isNaN("ahahahaha");      // true
isNaN("233333");         // false
```

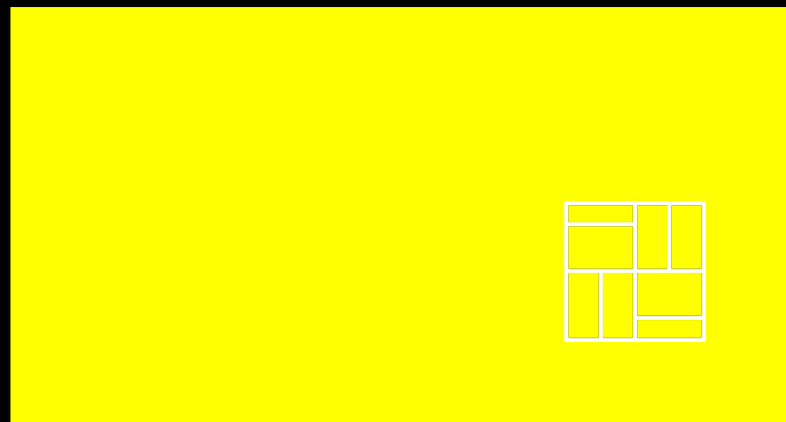
ISFINITE()

Determina se o valor transmitido é um número finito. Se necessário, o parâmetro é primeiro convertido a um número.

```
isFinite(2**10);         // true
isFinite(2**1024);       // false
```

DATE

Cria uma instância JavaScript de Date que representa um único momento no tempo.



Criar um Date

NEW DATE()

Cria a data do momento exacto do sistema.

```
new Date();
// Sun Oct 16 2022 10:46:52 GMT+0100
```

NEW DATE(valor)

Cria a data a partir do *valor* inteiro representando o número de milissegundos desde 1 de Janeiro de 1970 00:00:00 UTC

```
new Date(1000);
// Thu Jan 01 1970 01:00:01 GMT+0100
```

NEW DATE(dataString)

Um valor do tipo String que representa uma data.
A string deverá estar uma formato reconhecido pelo método Date.parse()

```
new Date('2014-12-23');
// Tue Dec 23 2014 00:00:00 GMT+0000

new Date('2014-25-23');
// Invalid Date

new Date('10 06 2023');
// Fri Oct 06 2023 00:00:00 GMT+0100
```

NEW DATE(ano, mês, dia, hora, minuto, segundo, milissegundo)

Cria a data a partir de todos os valores inteiros.
Nota: Os meses vão de 0 até 11.

```
new Date(1990, 1, 10);
// Sat Feb 10 1990 00:00:00 GMT+0000
```

Funções de leitura

GETDATE()

Retorna o dia do mês da data especificada.

```
new Date('2014-12-23').getDate();
// 23
```

GETDAY()

Retorna o dia da semana da data especificada.
Os valores da semana começam em 0 para Domingo.

```
new Date('2014-12-23').getDay();      // terça-feira
// 2
```

GETFULLYEAR()

Retorna o ano da data especificada com 4 dígitos.

```
new Date('2014-12-23').getFullYear();
// 2014
```

GETMONTH()

Retorna o mês da data especificada.

```
new Date('2014-12-23').getDate();
// 11
```

GETHOURS()

Retorna as horas da data especificada.

```
new Date('December 25, 1995 23:15:30').getHours();
// 23
```

GETMINUTES()

Retorna os minutos da data especificada.

```
new Date('December 25, 1995 23:15:30').getMinutes();
// 15
```

Funções de escrita

SETDATE()

Altera o dia do mês da data especificada.

```
var data = new Date('2014-12-23');
data.setDate(25);
data.getDate();
// 25
```

SETMONTH()

Altera o mês da data especificada.

```
var data = new Date('2014-12-23');
data.setMonth(10);
data.getMonth();
// 10
```

SETFULLYEAR()

Altera o ano da data especificada com 4 dígitos.

```
var data = new Date('2014-12-23');
data.setFullYear(1990);
data.getDate();
// 1990
```

SETHOURS()

Altera as horas da data especificada.

```
var data = new Date('December 25, 1995 23:15:30');
data.setHours(10);
data.getHours();
// 10
```

Exercício

DATE

- Criem um Date com a data 6 de Fevereiro de 2020.
- Com a data criada obtenham o dia da semana e o ano;
- Alterem a data para 6 de Março de 2021.

FUNÇÕES ANÓNIMAS

- Uma função anónima é uma função sem nome.
- Geralmente não está acessível após a sua criação.

Instanciar uma função

Uma função anónima não tem nome entre a palavra-chave function e os parênteses ():

```
var funcaoAnonima = function () {
        console.log('isto é uma função anónima!');
}
funcaoAnonima();
```

Costumamos usar funções anónimas como argumentos de outras funções. Por exemplo:

```
setTimeout(function() {
        console.log('isto é uma função dentro de um setTimeout!');
}, 1000);
```

Neste exemplo, passamos por argumento uma função anónima para a função *setTimeout()*. A função passa a ser então um parâmetro da função setTimeout. A função *setTimeout()* executa essa função anónima um segundo depois.

Exercício

FUNÇÕES ANÓNIMAS

- Criem uma função anónima que faça *console.log* do vosso nome completo.
- Criem uma função anónima que aceite 2 números e faça a sua soma.

IIFE

- Função anónima que é executada imediatamente após a sua declaração;

- IIFE = Immediately Invoked Function Expression;

- As variáveis definidas dentro da expressão não podem ser acedidas fora do seu contexto (scope), mesmo que seja um *var*.

Instanciar uma função

Uma função anónima não tem nome entre a palavra-chave function e os parênteses ():

```
(function (argumentos) {
        console.log('Isto é uma IIFE');
}) ('isto é um argumento');
```

Uma IIFE divide-se em duas secções:

FUNÇÃO

Nesta secção definimos a função anónima, que deverá estar dentro de parêntesis:

```
(function () {
        console.log('Isto é uma IIFE');
})
```

CHAMAR A FUNÇÃO

Após a definição da função adicionamos parêntesis de chamada de função, que podem ou não ter argumentos:

```
(argumentos);
```

Exercício

IIFE

- Criem uma IIFE que faça *console.log* do vosso nome completo.
- Criem uma IIFE que aceite 2 números e faça a sua soma.

CLOSURES

- Funções definidas dentro de um "contexto léxico" (i.e. o corpo de uma função) acessam variáveis definidas nesse contexto;

- Uma closure dá acesso ao scope de uma função externa a partir de uma função interna;

- Closures são usadas para manter as variáveis privadas, não podendo ser acedidas externamente.

Definição

Consideremos o seguinte código:

```
function funcaoTeste() {
    var mensagem = 'isto é uma mensagem!';

    (function () {
        console.log(mensagem);
    })();
}
console.log( funcaoTeste() );          // 'isto é uma mensagem!'
```

A função *funcaoTeste()* é chamada e internamente executa a IIFE que está dentro do seu scope. Daí o resultado do console.log ser *'isto é uma mensagem!'*

Definição

Consideremos agora o seguinte código:

```
var primeiroValor = 5;

function funcaoTeste() {
    return function () {
        var segundoValor = 10;
        console.log(primeiroValor + segundoValor);
    };
}
var funcao = funcaoTeste();

console.log( funcao() );            // 15
console.log( primeiroValor );       // 10
console.log( segundoValor );        // ReferenceError: segundoValor is not defined
```

- A função *funcaoTeste()* ao ser chamada devolve uma função, e para ser executada essa função interna, deverá ser chamada novamente.
- A variável *segundoValor* não consegue ser acedida externamente.

Definição

Compliquemos agora um pouco:

```
function multiplicacao(primeiroValor) {
        return function (segundoValor) {
                return primeiroValor * segundoValor;
        };
}
var funcao = multiplicacao(5);

console.log(multiplicacao(5));          // (segundoValor) { return primeiroValor * segundoValor; }
console.log(funcao(2));                 // 10
console.log(multiplicacao(5)(2));       // 10
```

- O código *multiplicacao(5)* devolve a função interna, que é retornada pela função *multiplicacao*;
- O código *funcao(2)* devolve o valor correcto, porque *funcao* é o mesmo que *multiplicacao(5)*;
- O código *multiplicacao(5)(2)* devolve o valor correcto, porque ambas as funções são chamadas.

Exercício

CLOSURES

- Criem uma closure que divida por 2 um número que forneçam como parâmetro. Exemplo:
  funcao(5); // 2.5

- Criem uma closure que faça uma divisão: o primeiro número é argumento da primeira função; o segundo argumento da divisão é passado para a função interna.