

PROGRAMAÇÃO I

JavaScript:
Fundamentos

APRESENTAÇÃO

AS ORIGENS DO JAVASCRIPT

- Desenvolvida Brendan Eich da **Netscape**:
 - Primeiro com o nome **Mocha**;
 - Com o Netscape 2 passou a chamar-se **LiveScript**;
- 1995: surge a designação **JavaScript** (trabalho conjunto entre a Netscape e a Sun Microsystems);
- 1997: primeiro standard para JavaScript designado **ECMA-262** (European Computer Manufactures Association)
- 1998: A ISO (Internation Standards Organization) adotou o **ECMAScript** como um standard
- Versão atual: **ECMAScript 2015** (ECMA-262 6ª ed.)
- Nome oficial: **ECMAScript**

COMPONENTES DO JAVASCRIPT

■ CORE

- Inclui os elementos base da linguagem (operadores, expressões, instruções, subprogramas)

■ CLIENTE SIDE

- Inclui uma coleção de objetos que permitem o controlo do browser e as interações com os utilizadores

■ SERVER SIDE

- Inclui um conjunto de objetos úteis no lado dos servidores web, como por exemplo: acesso a base de dados, operação com ficheiros e comunicações

LINGUAGEM DE *SCRIPTING*

- Linguagem de alto nível;
- Scripts são programas escritos para ambientes particulares, cujo código é interpretados em tempo de execução;
- Servem para automatizar a execução de tarefas;
- Ambientes que podem ser automatizados por meio de scripts:
 - Aplicações;
 - Páginas *web* dentro de um browser;
 - Shell dos sistemas operativos;
- Script de aplicação são chamados extensão de uma linguagem.

UTILIZAÇÃO DO JAVASCRIPT

- **Objetivo original:** providenciar tecnologia de programação para estabelecer a comunicação Web entre servidores e clientes
- **JavaScript Cliente-side:**
 - Código embutido em HTML
 - Interpretado pelos browsers
 - Permite a interação com o utilizador (*forms* e caixas de diálogo)
 - Permite criar/modificar conteúdo de forma dinâmica no browser (acesso ao conteúdo HTML e a propriedades CSS), através da API *DOM*
- **JavaScript Server-side:**
 - Operações com ficheiros
 - Acesso a base de dados
 - Comunicações

A ESTRUTURA BASE DE UMA PÁGINA HTML

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Título da página</title>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <!-- conteúdo HTML da página -->  
  </body>  
</html>
```

SCRIPTS INSERIDOS EM HTML (EXPLÍCITA)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título da página</title>
    <meta charset="UTF-8">
    <script>
      alert("Olá!...");
    </script>
  </head>
  <body>
    <!-- conteúdo HTML da página -->
    <script>
      alert("Olá, mais uma vez!...");
    </script>
  </body>
</html>
```


SCRIPTS INSERIDOS EM HTML (IMPLÍCITA)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título da página</title>
    <meta charset="UTF-8">
    <script src="exemplo1.js"></script>
  </head>
  <body>
    <!-- conteúdo HTML da página -->
    <script src="exemplo2.js"></script>
  </body>
</html>
```

PALAVRAS RESERVADAS

abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

COMENTÁRIOS

■ Comentar uma linha

// Comentário de uma linha

■ Comentar várias linhas

/*

Este é um comentário
efetuado em várias linhas

*/

TIPOS DE DADOS E VARIÁVEIS

JavaScript

CONCEITO DE VARIÁVEL

■ Variáveis:

- São **espaços de memórias** requisitados pelos programas ao sistema operativo;
- Permitem **guardar valores** de determinado tipo;
- São **declaradas no código** das linguagens de programação.
- Em tempo de desenvolvimento esses **espaços** são devidamente identificados.

TIPOS PRIMITIVOS DO JAVASCRIPT

- **Number** - aceita dados inteiros e reais de precisão dupla (64 bits)
- **String** - (sequências de 0 ou mais caracteres alfanuméricos delimitados por “” (aspas) ou ‘’ (apóstrofes))
- **Boolean** - (aceita os valores booleanos *true* e *false*)
- **Null** - (aceita apenas um valor *null* e indica variável sem valor)
- **Undefined** - (aceita apenas o valor *undefined* e indica que a variável foi definida mas não lhe foi atribuída um valor)
- **Object** - (variável que guarda a referência a um objecto criado com o operador *new* ou inicializada com o valor *null*)

LITERAIS

■ Literais numéricos:

- 72 7.2 .72 72. 7E2 7e2 .7e2 7.e2
7.2E-2 Infinity -Infinity
- 0x7A (valor inteiro hexadecimal) 05 (valor inteiro octal)

■ Literais String

- 'Ana' "Alfa Romeo \'159\'" "" (string vazia)
- Sequências de escape
 - Ex: \n (new line); \t (tab); \' (apóstrofe dentro de strings); \" (aspas dentro de strings), etc.

■ Outros literais

- null undefined true false

DECLARAÇÃO DE VARIÁVEIS

- Em JavaScript as variáveis são *loosely typed* (a variável pode armazenar qualquer tipo de dados).
- A declaração de variáveis: recorre a palavra-chave **var**.

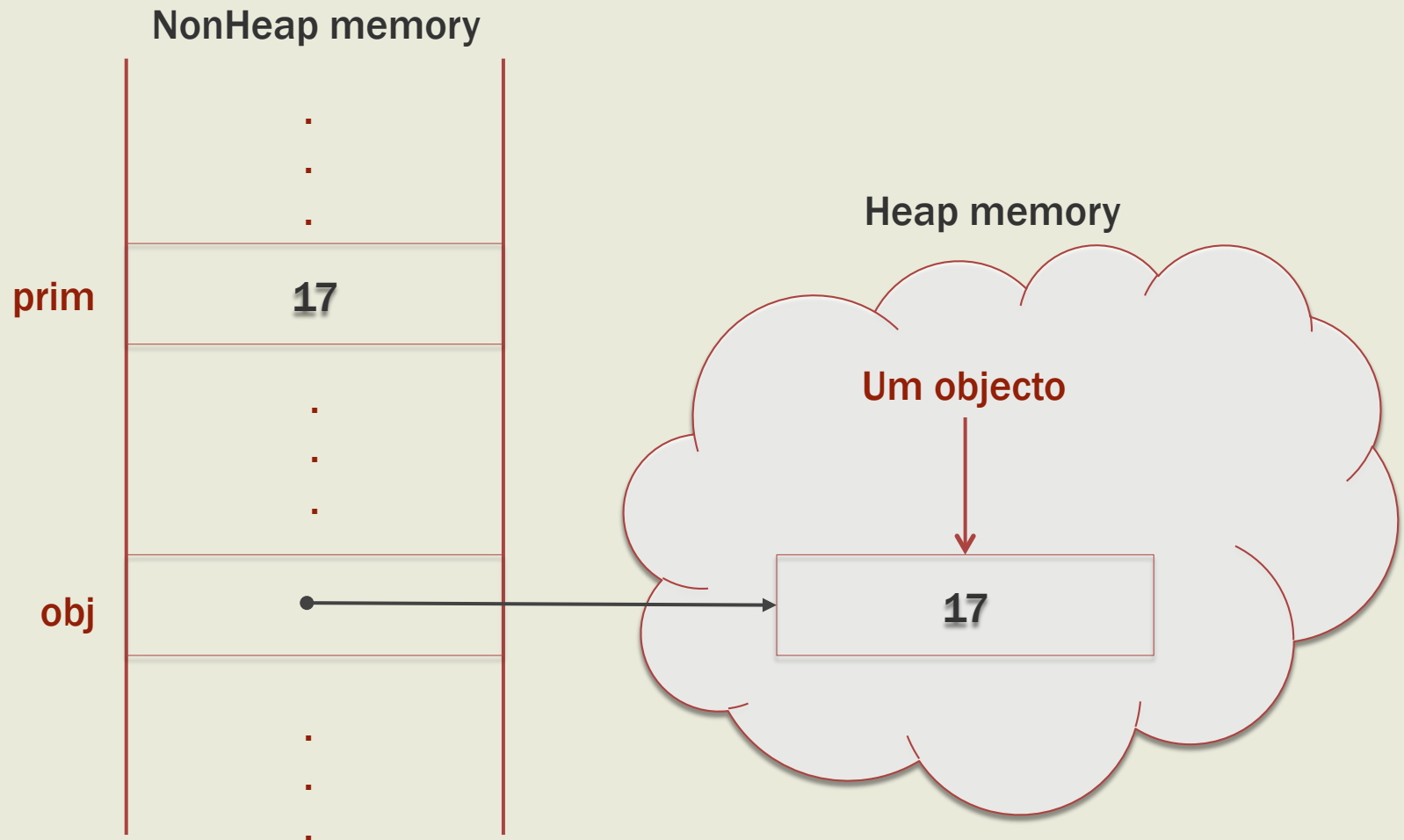
- Exemplos:

- `nome = "Ana";` // declaração não recomendada
- `var mensagem;` // variável do tipo *undefined*
- `var mensagem = "Olá";` // variável do tipo *string*
- `var valor = 7.3;` // variável do tipo *number*
- `var casado = true;` // variável do tipo *boolean*
- `var idade = 34;` // variável do tipo *number*
- `var carro = null;` // variável do tipo *Object*
- `var my_objeto = new Object();` // variável tipo *Object*
- `var counter, index, pi=3.14159265, jogador="Ronaldo", stop = true;` // declaração de várias variáveis

NOMES DE VARIÁVEIS (REGRAS)

- **Começar** por uma **letra**, **_** (underscore) ou **\$** (dolar)
- Os **restantes** caracteres podem ser: **letras**, **_**, **\$** ou **dígitos**
- Não há limitação do tamanho do nome da variável
- O nomes não podem ser iguais a palavras reservadas da linguagem
- A linguagem *JavaScript* é **casesensitive** (distingue minúsculas de maiúsculas: nome ≠ Nome ≠ NOME ≠ noMe)
- Não pode incorporar caracteres especiais da linguagem (operadores aritméticos, relacionais, lógicos, espaço, vírgula, ponto, etc.)
- Por convenção as variáveis são escritas em minúsculas

TIPO PRIMITIVO VS OBJECTO



OPERADORES

JavaScript

OPERADORES

- **Tipos de operadores**
 - Operadores aritméticos
 - Operadores relacionais
 - Operadores lógicos
 - Operadores de atribuição
 - Operadores de bit (bitwise)

OPERADORES ARITMÉTICOS

Operador	Operação	Exemplo
+	Soma	$a + b$
-	Subtração	$a - b$
*	Produto	$a * b$
/	Divisão	a / b
%	Resto da divisão inteira	$a \% b$
++	Incremento	$a++$; $++a$;
--	Decremento	$a--$; $--a$;

OPERADORES RELACIONAIS

Operador	Operação	Exemplo
>	Maior que	$a > b$
<	Menor que	$a < b$
>=	Maior ou igual a	$a \geq b$
<=	Menor ou igual a	$a \leq b$
==	Igual a (igualdade efetua conversão)	$2 == \text{"2"} \text{ (true)}$
===	Igual a (igualdade não efetua conversão)	$2 === \text{"2"} \text{ (false)}$
!=	Diferente de (efetua conversão)	$2 != \text{"2"} \text{ (false)}$
!==	Diferente de (não efetua conversão)	$2 !== \text{"2"} \text{ (true)}$

OPERADORES LÓGICOS

Operador	Significado	Exemplo
&&	Operador lógico AND	<code>a > 5 && a < 10</code>
	Operador lógico OR	<code>a < 5 a > 10</code>
!	Operador lógico NOT	<code>! a > 5</code>

Tabelas de verdade

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

A	B	A B
false	false	false
false	true	true
true	false	true
true	true	true

A	!A
false	true
true	false

OPERADORES DE ATRIBUIÇÃO

Operador	Operação	Exemplo
=	Atribuição simples	<code>a = 5;</code>
+=	Atribuição com soma	<code>a += 5; // a = a + 5;</code>
- =	Atribuição com subtração	<code>a -= 5; // a = a - 5;</code>
*=	Atribuição com produto	<code>a *= 5; // a = a * 5;</code>
/*	Atribuição com divisão	<code>a /= 5; // a = a / 5;</code>
%=	Atribuição com resto divisão inteira	<code>a %= 5; // a = a % 5;</code>

OPERADORES BIT A BIT (BITWISE)

Operador	Operação	Exemplo
&	Operação AND bit a bit	25 & 3; //=1
	Operação OR bit a bit	25 3; //=27
~	Operação NOT bit	~25; //= -26
^	Operação XOR bit a bit	25 ^ 3; //=26
<<	<i>Left Shift</i> (deslocação de bits à esquerda)	2 << 5; //=64
>>	<i>Right Shift</i> (deslocação de bits à direita)	64 >> 5; //=2
>>>	<i>Unsigned Right Shift</i> (deslocação de bits à direita sem sinal)	64 >>> 5; ; //=2 -64 >>> 5; //=134217726

CAIXAS DE MENSAGEM

JavaScript

CAIXAS DE MENSAGEM

■ Tipos de caixas de mensagem ou caixas de diálogo:

■ Alert:

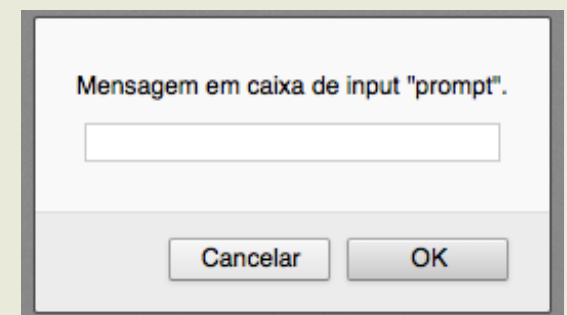
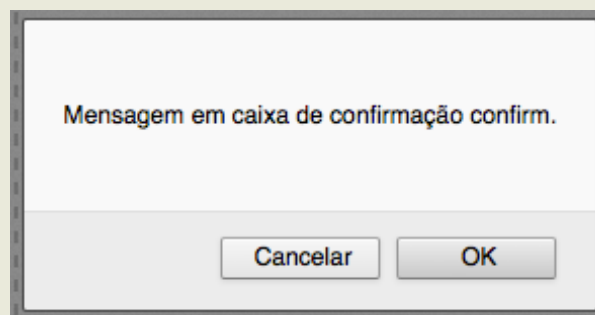
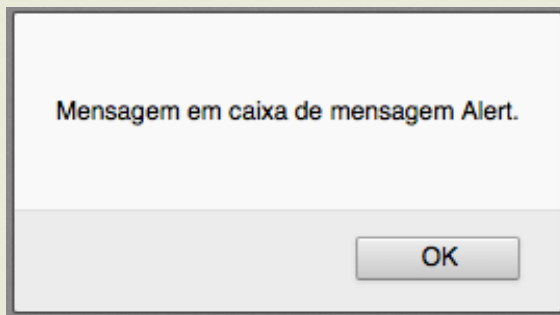
- `window.alert("mensagem");`

■ Confirm:

- `var conf = window.confirm("mensagem");`

■ Prompt:

- `var resp = window.prompt(mensagem, valor);`



ESTRUTURAS DE CONTROLO

INSTRUÇÕES DE CONTROLO

■ Decisão

- if ... else
- if ... else if ... else

■ Seleção

- switch

■ Repetição / Iteração

- while
- do ... while
- for
- for-in

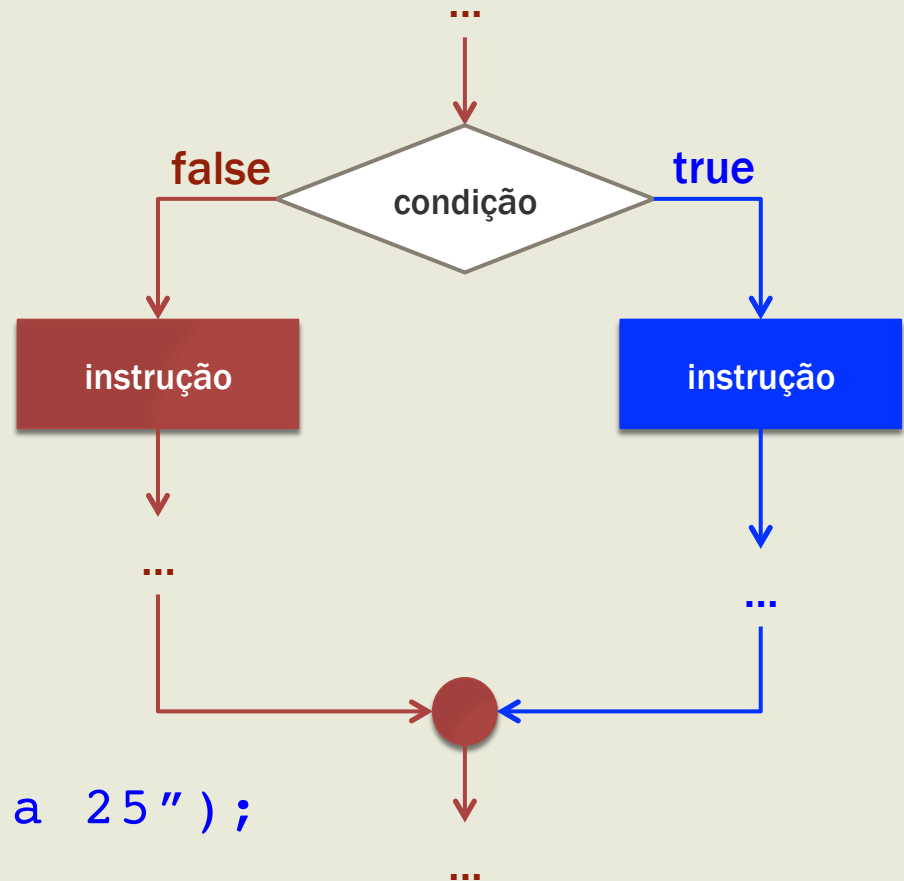
IF .. ELSE

■ Sintaxe:

```
if (condição) [ {  
    instruções;  
} ] [ else [ {  
    instruções;  
} ] ]
```

■ Exemplo:

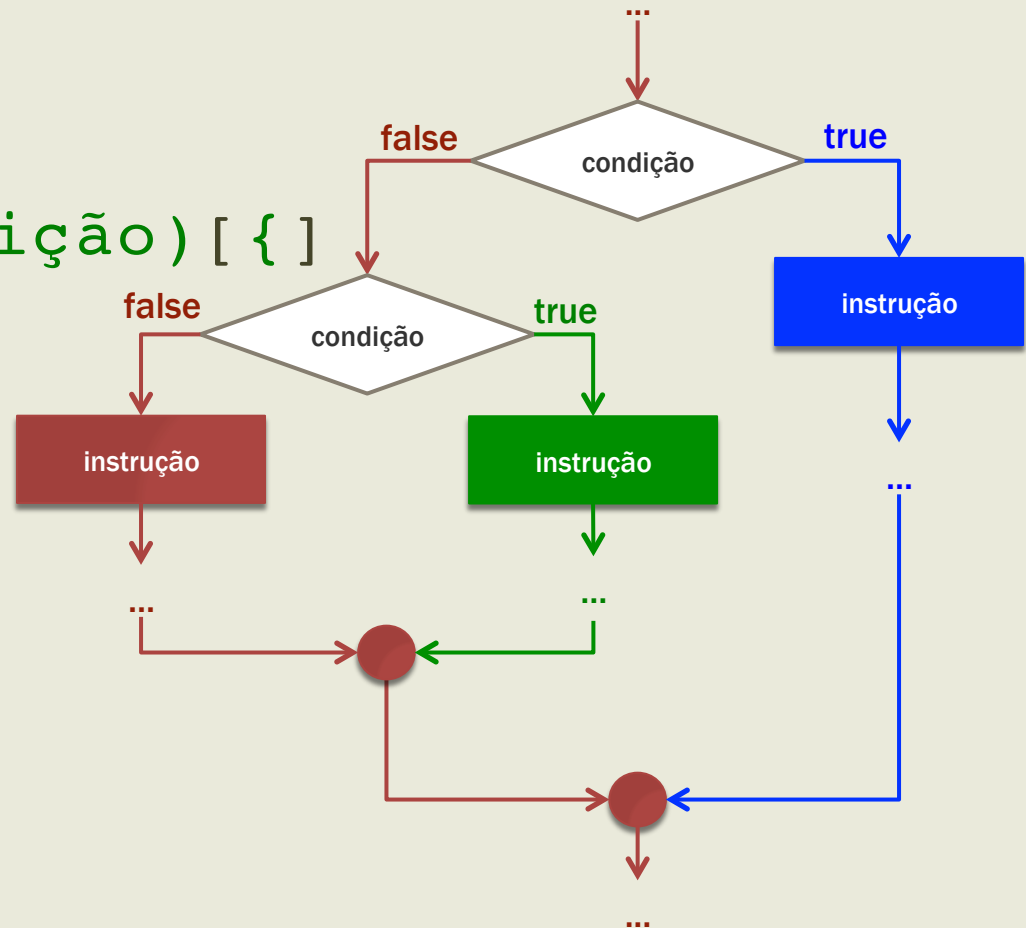
```
if (i > 25)  
    alert("maior que 25");  
else {  
    alert("Menor ou igual a 25");  
}
```



IF ... ELSE IF ... ELSE

■ Sintaxe:

```
if (condição) [ {  
    instruções;  
} ] else if (condição) [ {  
    instruções;  
} ] [ else [ {  
    instruções;  
} ] ]
```



IF ... ELSE IF ... ELSE

■ Exemplo:

```
if (i > 25) {  
    alert("Maior que 25.");  
} else if (i < 0) {  
    alert("Menor que 0.");  
} else {  
    alert("Entre 0 e 25, inclusivé.");  
}
```


SWITCH

■ Sintaxe:

```
switch(expressão | variável) {  
    case valor:  
        [instruções;]  
        [break;]  
    case valor:  
        [instruções;]  
        [break;]  
    ...  
    [default:  
        instruções;]  
}
```

SWITCH

■ Sintaxe:

```
switch(i) {  
    case 25:  
        alert("25");  
        break;  
    case 35:  
        alert("35");  
        break;  
    case 45:  
        alert("45");  
        break;  
  
    default:  
        alert("Outro valor");  
}
```

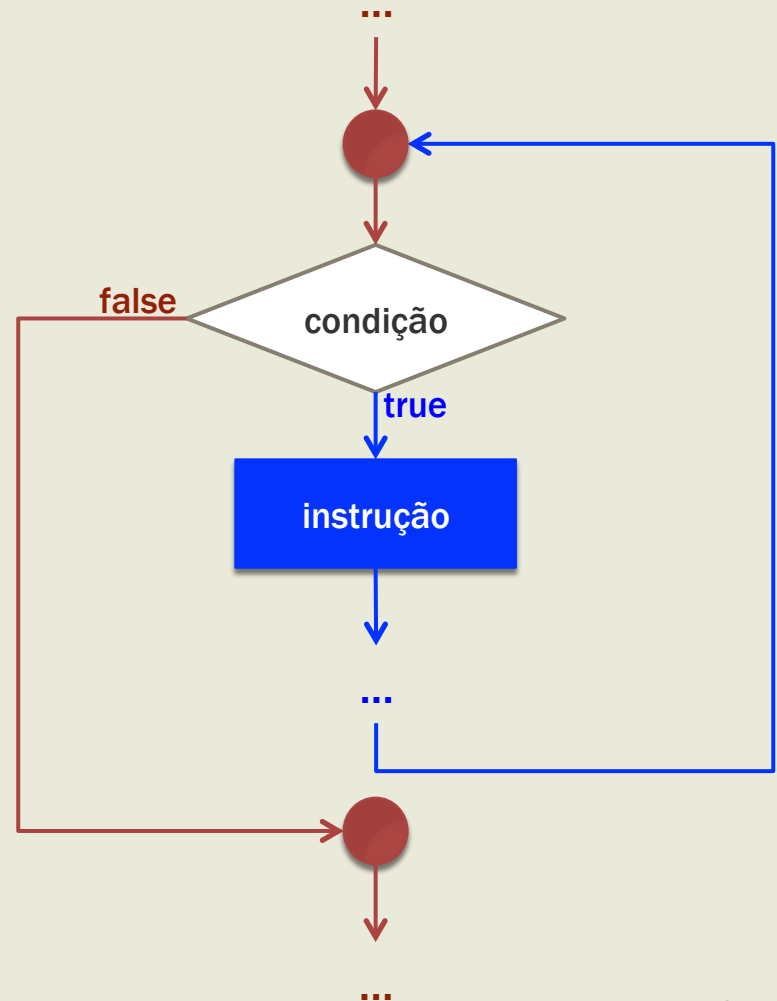
WHILE

■ Sintaxe:

```
while (condição) [ {  
    instruções;  
} ]
```

■ Exemplo:

```
var i = 0;  
while (i < 10) {  
    i += 2;  
}
```



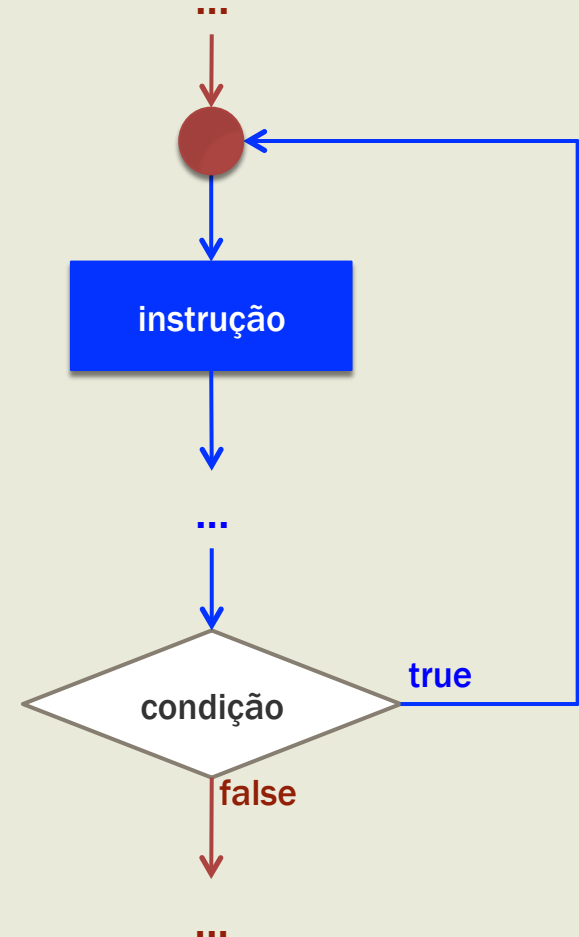
DO ... WHILE

■ Sintaxe:

```
do{  
    instruções;  
} while (condição);
```

■ Exemplo:

```
var i = 0;  
do{  
    i += 2;  
} while (i < 10);
```



FOR

■ Sintaxe:

```
for([inicialização]; [condição]; [incremento])[{]  
    instruções;  
[}]
```

■ Sintaxe:

```
var contador = 10;  
for(i = 0; i < contador; i++){  
    alert(i);  
}  
alert(i);           // mostra valor 10
```

FOR-IN

■ Sintaxe:

```
for(propriedade in objecto) [ { ]  
    instruções;  
[ } ]
```

■ Exemplo:

```
for(var nomeprop in window) {  
    document.write(nomeprop);  
}
```

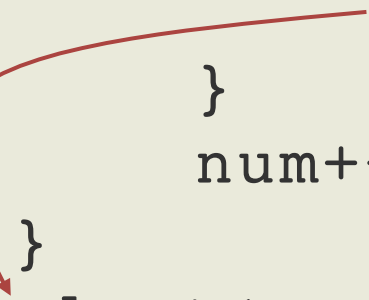
AS INSTRUÇÕES *break* E *continue*

- Permitem o controlo mais exato sobre a execução de um ciclo:
- *break* – força a saída do ciclo continuando a execução a seguir ao ciclo
- *continue* – força a verificação da condição do ciclo, ignorando as seguintes a sua utilização.

AS INSTRUÇÕES break E continue

■ Exemplo:

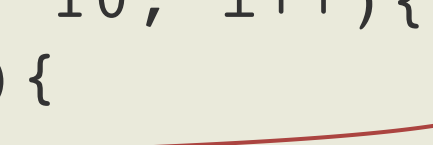
```
var num = 0;  
for(var i = 1; i < 10; i++){  
    if(i % 5 == 0){  
        break;  
    }  
    num++;  
}  
alert(num);    // 4
```



AS INSTRUÇÕES break E continue

■ Exemplo:

```
var num = 0;  
for(var i = 1; i < 10; i++){  
    if(i % 5 == 0){  
        continue;  
    }  
    num++;  
}  
alert(num);    // 8
```



EXERCÍCIOS

EXERCÍCIOS

1. Receber três número, através da caixa de input *prompt* e determinar qual o maior número introduzido.
2. Apresentar, através da instrução *alert*, uma tabela de números de 5 a 15 e os respectivos quadrados e cubos.
3. Apresentar os 20 primeiros números da série de Fibonacci, apresentados em sequência
1, 1, 2, 3, ...
onde cada número, a partir do segundo, é determinado pela soma dos dois números anteriores. Utilizar a instrução *document.write* para produzir o output .
4. Modificar o exercício 3, de forma a pedir o número de números da sequência da série de Fibonacci a apresentar.
5. Escrever um programa que apresente a tabela de multiplicação (sugestão: utilizar um ciclo dentro de outro).

EXERCÍCIOS

1. Apresentar o somatório dos 100 primeiros números positivos.
2. Apresentar o somatório dos números pares entre 0 e 100.
3. Calcular o fatorial de um número.
4. Implementar um jogo simples, considerando os seguintes requisitos: o computador deverá gerar um número entre 0 e 100, que o jogador tentará adivinhar. Sempre que o utilizador responda com um determinado valor, o programa responderá “Muito grande!” ou “Muito pequeno!” conforme o caso. O jogo termina quando o número for adivinhado.
(NB: utilizar a instrução `Math.round(100*(Math.random()))` para gerar o número aleatório entre 0 e 100);

EXERCÍCIOS

- Elaborar um programa que apresente os valores de conversão de graus *Celsius* em *Fahrenheit*, de 10 em 10, iniciando a contagem em 10 graus *Celsius* e finalizando em 100 graus *Celsius*. O programa deve apresentar os valores das duas temperaturas.
 - $^{\circ}\text{C} \times 9/5 + 32 = ^{\circ}\text{F}$
 - $(^{\circ}\text{F} - 32) \times 5/9 = ^{\circ}\text{C}$
- Elaborar um programa que efetue o cálculo e no final apresente o somatório do número de grãos de trigo que se pode obter num tabuleiro de xadrez, obedecendo a seguinte regra: colocar um grão de trigo no primeiro quadro e nos quadros seguintes o dobro do quadro anterior. Ou seja, no primeiro quadro coloca-se 1 grão, no segundo quadro colocam-se 2 grãos (neste momento tem-se 3 grãos), no terceiro quadro colocam-se 4 grãos (tendo neste momento 7 grãos), no quarto quadro colocam-se 8 grãos (tendo-se então 15 grãos) até atingir o sexagésimo quarto.

EXERCÍCIOS

- Elaborar um programa permita determinar o número de dias de um determinado mês. O mês e o ano são indicados pelo utilizador. É importante referir que existem anos bissextos que se caracterizam pelo facto de o mês de fevereiro ter 29 dias. Um ano é bissexto se:
 - For divisível por 400;
 - Ou for divisível por 4 mas não por 100.

FUNÇÕES

FUNÇÕES

- Funções permitem **encapsular instruções, às quais se pode dar um nome**. Através desse nome podem ser executadas onde e quando quisermos

- Declaração:

- É realizada através da palavra-chave **function**;

- Sintaxe:

```
function nomeDaFunção(arg0, arg1, ..., argN) {  
    instruções  
}
```


PARTES DE UMA FUNÇÃO

■ Exemplo

```
function soma(a, b){  
    var c = a + b;  
    return c;  
}
```

■ Parte constituintes do código:

- A instrução: *function*
- O nome da função: *soma*
- Os parâmetros da função: *a* e *b*
- O corpo da função: bloco de instruções entre chavetas

```
{  
    var c = a + b;  
    return c;  
}
```
- A instrução *return* (se não for indicada a função devolve *undefined*)

CHAMAR FUNÇÕES SEM *return*

■ Exemplo

```
function saudacao(nome, mensagem){  
    alert("Olá " + nome + ", " + mensagem);  
}
```

■ Exemplos de chamada da função

```
saudacao("Ana", "como estás hoje?");  
saudacao("Carlos", "desejo-te um bom dia!");
```

CHAMAR FUNÇÕES COM *return*

■ Exemplo

```
function soma(num1, num2){  
    return num1 + num2;  
}
```

■ Exemplos de chamada da função

```
var resultado = soma(5, 10);  
var valor = soma(100.50, 23.12);  
alert(soma(2, 14));
```

A INSTRUÇÃO COM *return*

■ Exemplos

```
function soma(num1, num2){  
    return num1 + num2;  
    alert("Olá mundo!"); // não é executada ()  
}  
  
function saudacao(nome, mensagem){  
    return;  
    alert("Olá " + nome + ", " + mensagem);  
    // esta última instrução não é executada  
}
```

FUNÇÕES COM *return*

- Exemplo de função com várias instruções *return*

```
function diferenca(num1, num2) {  
    if(num1 < num2) {  
        return num2 - num1;  
    } else {  
        return num1 - num2;  
    }  
}
```

PARÂMETROS VS ARGUMENTOS DA FUNÇÃO

Considerando:

```
function soma(a, b){  
    var c = a + b;  
    return c;  
}
```

```
var r = soma(1, 2);
```

- **a** e **b** são designados **parâmetros** da função e **1** e **2** são designados **argumentos** da função;

ARGUMENTOS DA FUNÇÃO

- O JavaScript não é rigoroso na definição do número de argumentos;

- Ou seja, é possível passar mais ou até menos argumentos do que aqueles que a função está à espera:

```
> sum(1);           // devolve NaN  
> sum(1, 2);        // devolve 3  
> sum(1, 2, 3, ,4, 5); //devolve 3
```

- Isto é possível graças a um valor especial *arguments*, criado automaticamente dentro de cada função.

ARGUMENTOS DA FUNÇÃO

■ Exemplo:

```
function args(){  
  return arguments;  
}
```

```
> args();    // devolve [] (array vazio)  
[]
```

```
//devolve array com argumentos passados à  
função
```

```
> args(1, 2, 3, 4, true, "Rui");  
[1,2,3,4,true,"Rui"]
```


ARGUMENTOS DA FUNÇÕES

- Exemplo: reescrevendo o função soma para contemplar um número de argumentos variável

```
function somatorio(){  
  var i, res = 0;  
  var nr_param = arguments.length;  
  for(i = 0; i < nr_param; i++){  
    res += arguments[i];  
  }  
  return res;  
}
```

```
> somatorio(1,1,1);    // devolve 3  
> somatorio(1,2,3,4);  // devolve 10  
> somatorio(1,1,3,4,4,3,2,1) // devolve 20  
> somatorio(5)         // devolve 5  
> somatorio()          // devolve 0
```

VARIÁVEIS LOCAIS E GLOBAIS

- As variáveis declaradas **fora de funções** são **globais** e podem ser acedidas em qualquer parte do código;
- As variáveis declaradas **dentro das funções** são **locais** e são apenas conhecidas dentro das respectivas funções;

VARIÁVEIS LOCAIS E GLOBAIS

■ Exemplo:

```
function abc(){  
    var a = 5;    // variável local  
    document.write(a);    // 5  
}  
  
abc( );  
  
document.write(a);    // ERRO: variável a não existe
```

VARIÁVEIS LOCAIS E GLOBAIS

■ Exemplo:

```
function abc(){  
    var a = 5;    // variável local  
    document.write(a);    // 5  
}  
  
var a = 10;    // variável global  
  
abc();  
  
document.write("<br/>");  
  
document.write(a);    // 10
```

VARIÁVEIS LOCAIS E GLOBAIS

- Exemplo de criação de variáveis criada dentro de função com âmbito global:

```
function abc(){  
    a = 5;    // variável global  
    document.write(a);    // 5  
}  
  
abc();  
  
document.write(a);    // 5
```

VARIÁVEIS LOCAIS E GLOBAIS

- As variáveis locais sobrepõem-se às variáveis globais com o mesmo nome:

```
var a = 123;
```

```
function f(){  
    alert(a);    // apresenta undefined  
    var a = 1;  
    alert(a);    // apresenta o valor 1  
}
```

FUNÇÕES SÃO DADOS

- Funções em JavaScript são dados especiais com duas características:
 - Contêm código;
 - São executáveis (podem ser invocadas)

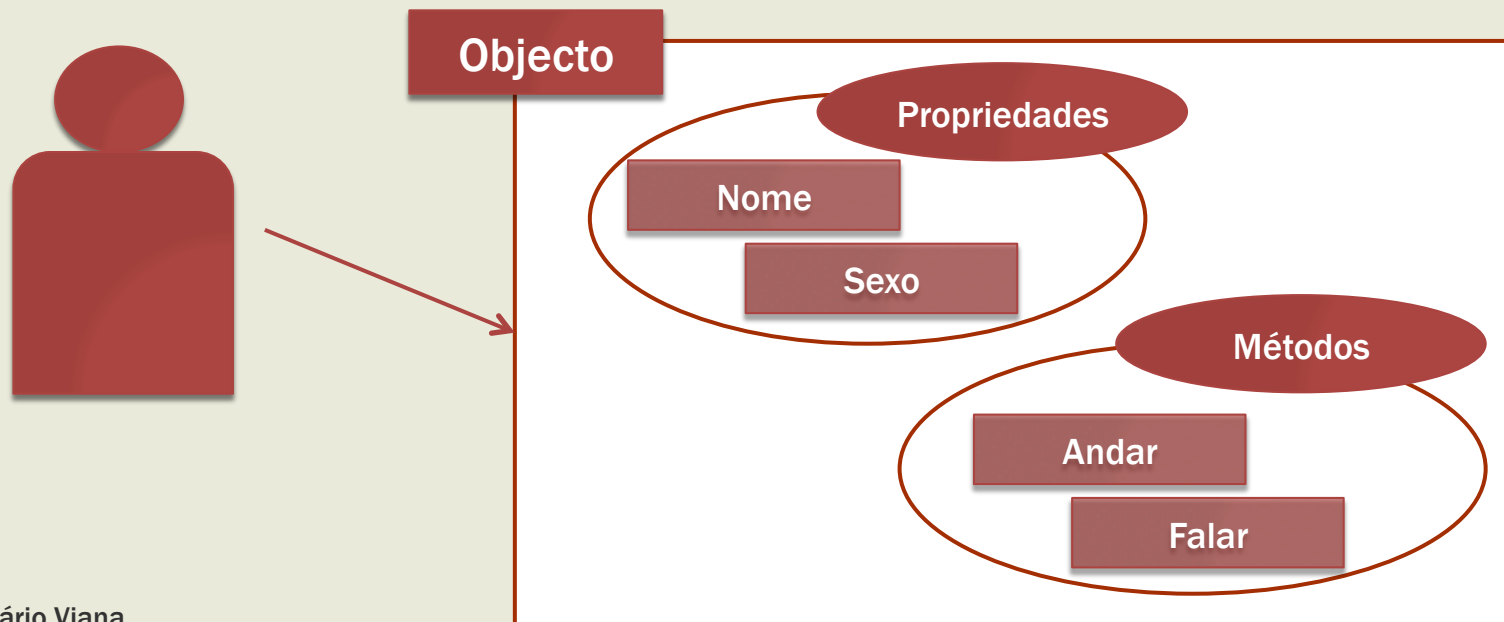
```
function define(){  
    return 1;  
}  
var express = function(){  
    return 1;  
};  
> typeof define; // "function"  
> typeof express; // "function"
```

CONCEITO DE OBJETO

JavaScript

FUNDAMENTOS DE OBJETOS

- Um objeto é uma entidade de software constituída por dados (propriedades) e funções (métodos);
- Permite, na programação, a modelização de situações do mundo real, através da manipulação das suas características (propriedades) e funcionalidades (métodos).



FUNDAMENTOS DE OBJETOS

- Objetos são uma coleção de propriedades (variáveis) e métodos (funções);
- Todos os objetos são acedidos indiretamente através de variáveis;
- O objeto ancestral de todos os objetos é **Object** e constitui o protótipo de todos os objetos. **Object** possui apenas métodos;
- Em JavaScript todos os objetos são uma lista de pares **propriedade: valor**
 - Propriedades: são nomes
 - Valor: são valores de dados ou funções

FUNDAMENTOS DE OBJETOS

- Nos objetos as funções são objetos e são referenciadas através de variáveis;
- Em JavaScript a coleção de propriedades dos objetos é **dinâmica**: as propriedades podem ser **adicionadas** ou **apagadas** a qualquer momento;
- Na criação de objetos utiliza-se o operador **new**;
`var my_object = new Object();`
- O acesso às propriedades do objeto é feita através do operador **.** (**ponto**) que permite especificar a propriedade do objecto:
`nomeobjecto.nomepropriedade`

FUNDAMENTOS DE OBJETOS

■ Exemplos

- Criar objecto e adicionar propriedades:

```
var carro = new Object();  
carro.marca = "Ford";  
carro.modelo = "Fiesta";
```

- Forma abreviada de criar objetos e respectivas propriedades:

```
var carro = {  
    marca: "Ford",  
    modelo: "Fiesta"  
};
```

FUNDAMENTOS DE OBJETOS

■ Exemplos

- Criar objetos dentro de objetos:

```
carro.motor = new Object();  
carro.motor.tipo = "V6";  
carro.motor.potencia = 200;
```

- ou:

```
var carro = {  
  marca: "Ford",  
  modelo: "Fiesta",  
  motor: {  
    tipo: "V6",  
    potencia: 200  
  }  
};
```

FUNDAMENTOS DE OBJETOS

■ Exemplos

- Aceder às propriedades do objeto:

```
var prop1 = carro.marca;  
var prop2 = carro["modelo"];  
var prop3 = carro.motor.tipo;  
var prop4 = carro["motor"]["potencia"];  
var prop5 = carro["motor"].tipo;  
var prop6 = carro.motor["potencia"];
```

- Apagar uma propriedade de um objeto:

```
delete carro.modelo;
```

FUNDAMENTOS DE OBJETOS

■ Exemplos

- Loop para aceder às propriedades de um objeto:

```
for(var prop in carro){  
    document.write("Propriedade: ", prop,  
        "; Valor: ", carro[prop], "<br/>");  
}
```

ARRAYS

ARRAYS

- Um *array* é uma estrutura de dados;
- Os elementos de um *array* são indexados;
- Os índices começam em 0 (zero) e são incrementados em 1 para cada elemento do *array*;
- Para aceder a um elemento, coloca-se o valor do índice do elemento entre parênteses retos ([]);
- Um *array* pode conter qualquer tipo de dados, incluindo outros *arrays*;
- Em *JavaScript* os *arrays* são dinâmicos.

CRIAÇÃO DO OBJETO ARRAY

■ Duas formas:

■ Criar um objeto array com o operador new.

//1. Criar um array sem elementos

```
var arrvazio = new Array();
```

//2. Criar um array com 4 elementos

```
var lista = new Array(1, 2, "três", "quatro");
```

//3. Criar um array de tamanho 100, sem elementos

```
var outra_lista = new Array(100);
```

■ Criar um objeto array com literais:

//1. Criar um array sem elementos

```
var arrvazio = [];
```

//1. Criar um array com 4 elementos através de literais

```
var lista = [1, 2, "três", "quatro"];
```

CARACTERÍSTICAS DOS OBJETOS ARRAY

- O primeiro índice é 0 (zero);
- O acesso aos elementos do *array* faz-se indicando o índice entre parênteses retos [];
- O tamanho do *array* é determinado pela maior posição onde foi colocado um elemento
`lista[47] = 2222;`
Neste caso o tamanho passa de 4 para 48
- É possível saber o tamanho do *array* através da propriedade `length`;
`//apresenta o tamanho do array lista`
`alert(lista.length);`

CARACTERÍSTICAS DOS OBJETOS ARRAY

- É possível alterar o tamanho de um *array* alterando o valor da propriedade *length*:
`lista.length = 1002;`
O tamanho do *array* passou para 1002.
- O tamanho de um *array* pode aumentar ou diminuir, alterando o valor da propriedade *length*;
- Apenas as posições com valores ocupam espaço → o tamanho do *array* não é necessariamente o espaço ocupado;
- O espaço é alocado dinamicamente através de uma lista ordenada de dados (heap)

ARRAYS BIDIMENSIONAIS

- Um *array* bidimensional é implementado como um *array de arrays*.
- Exemplos:

```
var a = [1,"dois",false,null,undefined];  
a[5] = [1,2,3];  
//a: [1,"dois",false,null,undefined,[1,2,3]]  
var bidim = [[2,4,6],[1,3,5],[10,20,30]];  
var subarray = bidim[0];  
//subarray: [2,4,6]  
var item = bidim[1][2];  
//item: 5
```

MÉTODOS DO OBJETO ARRAY

- **join()** : devolve o conteúdo de um *array* em *String*, com os elementos separados por , (vírgula) ou outro caractere indicado como argumento:

- Exemplos:

```
var nomes = ["Maria", "Ana", "Rui", "Zé"];
```

```
// devolve "Maria,Ana,Rui,Zé"
```

```
var snomes = nomes.join();
```

```
// devolve "Maria:Ana:Rui:Zé"
```

```
var snomes2 = nomes.join(":");
```

MÉTODOS DO OBJETO ARRAY

- **reverse()** : inverte a ordem dos elementos do array:

- Exemplos

```
// nomes fica ["Zé", "Rui", "Ana", "Maria"]  
nomes.reverse();
```

```
// inverte nomes e devolve uma cópia de nomes  
var rnomes = nomes.reverse();
```

MÉTODOS DO OBJETO ARRAY

- **concat()** : acrescenta novos elementos ao final do *array*, devolvendo o novo *array* resultado desse acréscimo. O *array* original fica inalterado:

- Exemplos

```
var nnomes = nomes.concat("Joel", "André");  
// nomes: ["Ana", "Maria", "Rui", "Zé"]  
// nnomes: ["Ana", "Maria", "Rui", "Zé", "Joel", "André"]  
  
nomes = nomes.concat("Joel", "André");  
// nomes ["Ana", "Maria", "Rui", "Zé", "Joel", "André"]
```


MÉTODOS DO OBJETO ARRAY

■ **slice()** : devolve parte do *array*:

■ **Exemplos**

```
var nrs = [2,4,6,8,10];
```

```
var sublista = nrs.slice(1, 3);  
// sublista: [4,6]
```

```
var sublista2 = nrs.slice(2);  
// sublista2: [6,8,10]
```

```
var sublista3 = nrs.slice(-3, -1);  
// sublista3: [8,10]  
// Equivalente a nrs.slice(3, 4); //nrs.slice(5-2, 5-1);
```

```
var sublista4 = nrs.slice(3, 1);  
// sublista4: [] (devolve array vazio)
```

MÉTODOS DO OBJETO ARRAY

- **toString()** : devolve uma string com todos os elementos, convertidos em *String*, separados por , (vírgula):

- Exemplos

```
var nrs = [2,4,6,8,10];
```

```
var sublista = nrs.toString();  
// sublista: "2,4,6,8,10"
```

MÉTODOS DO OBJETO ARRAY

- Métodos que permitem inserir/remover elementos no final ou no início do *array*:
 - **pop()** : retira e devolve o último elemento do array;
 - **push()** : adiciona elementos ao final do array;
 - **shift()** : retira e devolve o primeiro elemento do array;
 - **unshift()** : adiciona elementos no início do array;
- Este método combinados permite implementar pilhas (LIFO) e filas (FIFO).

MÉTODOS DO OBJETO ARRAY

■ Exemplos

```
var lista = ["Ana", "Beto"];  
lista.push("Rui");  
// lista: ["Ana", "Beto", "Rui"]  
var nome= lista.pop();  
// lista: ["Ana", "Beto"]  
// nome: "Rui"  
  
lista.unshift("Maria");  
// lista: ["Maria", "Ana", "Beto"]  
var primeiro= lista.shift();  
// lista: ["Ana", "Beto"]  
// primeiro: "Maria"
```

MÉTODOS DO OBJETO ARRAY

- **splice()** : permite:
 - **Eliminar** elementos se forem indicados dois argumentos (primeiro elemento a apagar, número de elementos a apagar)
 - **Inserir** elementos numa posição específica se forem indicados três argumentos (posição inicial, 0 e item a inserir)
 - **Alterar** elementos a partir de uma posição específica se forem indicados pelo menos três argumentos (posição inicial, nº de elementos a apagar e itens a inserir)

MÉTODOS DO OBJETO ARRAY

■ Exemplos

```
var cores = ["vermelho", "verde", "azul"];
```

```
var apagada= cores.splice(0,1);
```

```
// cores: ["verde", "azul"]
```

```
// apagada: ["vermelho"]
```

```
apagada = cores.splice(1,0,"amarelo","laranja");
```

```
// cores: ["verde", "amarelo", "laranja", "azul"]
```

```
// apagada: []
```

```
apagada= cores.splice(1,1,"vermelho","purpura");
```

```
// cores: ["verde", "vermelho", "purpura", "laranja", "azul"]
```

```
// apagada: [amarelo]
```

MÉTODOS DO OBJETO ARRAY

- **sort()** : ordena os elementos do *array*, convertendo previamente cada elemento em *String*:

- Exemplos

```
nomes.sort();  
// nomes fica ["Ana","Maria","Rui","Zé"]
```

```
var nomes2 = ["Maria","ana", "rui", "Zé"]  
nomes2.sort();  
// nomes2 fica ["Maria","Zé","ana","rui"]
```

```
var nrs = [5, 10, 2, 7, 3, 22];  
nrs.sort();  
// nrs fica [10, 2, 22, 3, 5, 7]
```

MÉTODOS DO OBJETO ARRAY

- **sort()** : como ordenar corretamente *arrays* com valores não alfanumericos?
- Neste caso, é necessário passar, ao método sort(), uma função específica que defina corretamente a ordenação. Esta função deve devolver um valor:
 - < 0 → se os dois elementos a comparar estiverem na ordem correta
 - $= 0$ → se os dois elementos são iguais
 - > 0 → se os dois elementos devem ser trocados

MÉTODOS DO OBJETO ARRAY

■ Exemplo 1:

```
var nrs = [5,10,2,7,3,22];

function num_ordem(a, b){
    if(a < b)
        return -1;
    else if (a == b)
        return 0;
    else
        return 1;
}

nrs.sort(num_ordem);
// nrs fica [2,3,5,7,10,22]
```

■ Exemplo 2:

```
// função mais otimizada
var nrs = [5,10,2,7,3,22];

function num_ordem(a, b){
    return a - b;
}

nrs.sort(num_ordem);
// nrs fica [2,3,5,7,10,22]
```

EXERCÍCIOS

- Elabore um programa que calcule o somatório e a média dos elementos de um vetor com N números reais.
- Elabore um programa que conte o número de vezes que cada elemento ocorre num vetor de inteiros compreendidos na faixa de 0 a 9. Por exemplo, no vetor A:
 - $A = \{4, 2, 5, 4, 3, 5, 2, 2, 4\}$
 - - 4 ocorre três vezes;
 - - 2 ocorre três vezes;
 - - 5 ocorre duas vezes;
 - - 3 ocorre uma vez.

EXERCÍCIOS

- Elabore um programa que apresente o maior e o menor elementos de um vetor de N números. Pressupõe que o vetor não tem elementos repetidos.
- Elabore um programa que apresente o maior e o segundo maior elementos de um vetor de N números. Pressupõe que o vetor não tem elementos repetidos.
- Elabore um programa que adicione e multiplique, elemento a elemento, dois vetores. Pressupõe que os dois vetores tem o mesmo número de elementos.