MÓDULO

JAVASCRIPT/AJAX

UNIDADE

OBJETOS JAVASCRIPT

Instituto Tecnológico

ÍNDICE

OBJETIVOS **3**
INTRODUÇÃO **4**
1. PROGRAMAÇÃO ORIENTADA A OBJETOS **5**
2. OBJETOS, ATRIBUTOS E MÉTODOS **6**
2.1. PROPRIEDADES 6
2.2. CRIAÇÃO DE OBJETOS 7
2.3. MÉTODOS 11
3. OBJETOS PREDEFINIDOS EM JAVASCRIPT **15**
3.1. STRING 15
3.2. NUMBER 19
3.3. MATH 20
3.4. DATE 22
3.5. ARRAY 25
CONCLUSÃO **27**
AUTOAVALIAÇÃO **29**
SOLUÇÕES **35**
PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO **36**
BIBLIOGRAFIA **37**

OBJETIVOS

Com esta unidade didática, pretende-se que desenvolva os seguintes objetivos de aprendizagem:

- Saber o que é programação orientada a objetos.
- Aprender o método de implementação de um sistema orientado a objetos em JavaScript.
- Aprender a criar objetos com os seus atributos e relacionamentos.
- Conhecer um pouco melhor as classes predefinidas do JavaScript.

INTRODUÇÃO

A programação orientada a objetos é uma forma de programação cada vez mais difundida. Graças a ela, pode criar código mais complexo de uma forma mais simples. É amplamente utilizada na programação de jogos e programas complexos.

O JavaScript possui uma implementação em objetos que permite aprimorar as páginas web (e não só).

1. PROGRAMAÇÃO ORIENTADA A OBJETOS

A POO (programação orientada a objetos), como o próprio nome sugere, concentra-se no objeto como unidade principal do sistema.

Um objeto é uma estrutura definida que contém propriedades (que são variáveis) e métodos (ou funções) que manipulam essas mesmas propriedades.

O sistema de POO separa os objetos em dois tipos: classes e instâncias.

As classes são os conjuntos abstratos de vários objetos, enquanto as instâncias são os diferentes membros das classes. As instâncias têm as mesmas propriedades da classe na qual estão incluídas e mais uma série de propriedades denominadas de herança ou polimorfismo.

O JavaScript usa um sistema orientado a objetos simplificado (apenas existem objetos, e não classes abstratas).

2. OBJETOS, ATRIBUTOS E MÉTODOS

2.1. PROPRIEDADES

Os objetos em JavaScript têm propriedades associadas e podem ser utilizados de uma maneira simples:

```
objeto.propriedade
```

Ao definir objetos e as suas propriedades, o JavaScript faz distinção entre maiúsculas e minúsculas, portanto, deverá ter cuidado com o modo como os nomes são definidos.

Os valores destas propriedades são declarados através do sistema de atribuição, o sinal de igual (=). E cada objeto pode ter quantas propriedades forem necessárias.

Por exemplo, para uma classe em que há um programa para definir os funcionários de uma empresa, poderia definir um objeto "funcionario" (que mais tarde aprenderá a criar) com as seguintes propriedades:

```
funcionario.antiguidade = '1980';
funcionario.precoHora = 23;
```

```
funcionario.horas = 6;
```

Os objetos em JavaScript podem ser tratados como arrays. Para aceder aos dados de um array, basta chamar o array com o seu índice e, no caso dos objetos, o nome da propriedade atua como um índice. Portanto, para aceder às propriedades do exemplo anterior, basta escrever este código:

```
document.write(funcionario[“antiguidade”]);
```

Este código mostrará na página web o valor da propriedade “antiguidade” do objeto “funcionario”.

2.2. CRIAÇÃO DE OBJETOS

Em JavaScript, existem duas maneiras diferentes de criar um objeto:

- **Com o objeto inicial.**

  Esta forma de criar objetos não é a forma padrão usada noutras linguagens de programação. Para pequenos objetos esta é a sintaxe mais simples e rápida de se utilizar:

```
objeto = {propriedade1:valor1,propriedade2:valor2, …}
```

  O ”objeto” será o nome do objeto e cada propriedade e valor serão os identificadores e os seus devidos valores para as propriedades do objeto.

  Com este sistema foram definidas diretamente as instâncias dos objetos, ou seja, não teria um objeto “funcionario” geral a partir do qual criaria as instâncias, mas antes geraria diretamente as instâncias.

  Também é possível criar um objeto que contém outro objeto, com este sistema. Para fazer isso, no objeto iniciador, basta substituir o valor da propriedade pela definição de outro objeto.

Por exemplo:

```
funcionario1 = {precoHora:23,horas:6,secretaria:{nome:
‘mario’,numero:2}};

funcionario2 = {precoHora:20,horas:8,secretaria:{nome:
‘nuno’,numero:12}};
```

Neste exemplo, o objeto “funcionario1” (instância do objeto) possui três propriedades: as duas primeiras que são valores simples, e a terceira, que possui um outro objeto como valor, que por sua vez possui mais duas propriedades.

- **Com uma função construtora.**

  Neste caso irá criar um objeto genérico com uma função construtora que invocará sempre que quiser criar uma instância desse objeto.

  Nesta função construtora definirá as propriedades e os métodos que cada objeto irá ter. Segue-se um exemplo de como criar uma função construtora e as instâncias do objeto:

```
function funcionario(antiguidade, precoHora, horas){

this.antiguidade = antiguidade;

this.precoHora = precoHora;

this.horas = horas;

}
```

O código acima é a função construtora do objeto “funcionario” e é a função que recebe os valores das propriedades de cada objeto a criar.

```
funcionario1 = new funcionario(‘1980’,23,6);

funcionario2 = new funcionario(‘1980’,20,8);
```

Este código cria duas instâncias do objeto “funcionario” chamado “funcionario1” e “funcionario2”.

Tal como no caso anterior, um objeto pode pertencer a outro objeto. Para isso, basta criar previamente o objeto que quer utilizar como propriedade, criar as suas instâncias e passá-las como mais um parâmetro para a função construtora (que aceita a passagem de objetos como variáveis para propriedades). Segue-se um exemplo:

```
function secretaria(nome, numero, piso){
this.nome = nome;
this.piso = piso;
this.numero = numero;
}
nome_mario = secretaria('mario',2,13);
nome_nuno = secretaria('nuno',12,3);
```

Com este código criou o objeto "secretaria" e as suas duas instâncias dos detalhes dos dois funcionários que criar. Agora, basta modificar a função anterior, da criação de funcionários, para aceitar a nova propriedade:

```
function funcionario(antiguidade, precoHora, horas, secretaria){
this.antiguidade = antiguidade;
this.precoHora = precoHora;
this.horas = horas;
this.secretaria = secretaria;
}
funcionario1 = new funcionario('1980',23,6, nome_mario);
funcionario2 = new funcionario('1980',20,8, nome_nuno);
```

Como podemos ver na chamada às instâncias, basta passar os objetos que criam o endereço como parâmetro da função.

Para aceder aos valores das propriedades simples do objeto "funcionario1" procede-se da seguinte forma:

```
document.write(funcionario1.horas);
```

Para as propriedades que, por sua vez, são objetos terá de lhes aceder da seguinte forma:

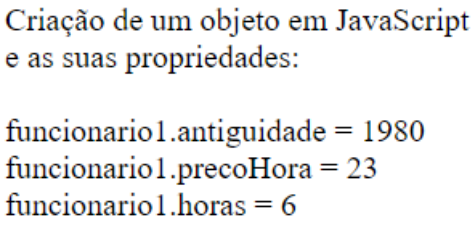
```
document.write(funcionario1.secretaria.nome);
```

Exemplo 01

Neste exemplo, será criada uma função construtora do objeto "funcionario", seguida da criação de uma instância do objeto e demonstração de todos os valores das suas propriedades.

Exemplo:

```
<html>
<head>
<script type="text/JavaScript">
function funcionario(antiguidade, precoHora, horas){
 this.antiguidade = antiguidade;
 this.precoHora = precoHora;
 this.horas = horas;
}
</script>
</head>
<body>
Criação de um objeto em JavaScript <br>e as suas
propiedades:<br><br>
<script type="text/JavaScript">
funcionario1 = new funcionario('1980',23,6);
for (var i in funcionario1){
 document.write("funcionario1." + i + " = " + funcionario1[i] +
"<br>");
}
</script>
</body>
</html>
```



Resultado do exemplo.

Um objeto criado não é um elemento estático; pode sempre criar novas propriedades para os objetos, utilizando a função prototype, que permite adicionar propriedades a todas as instâncias do objeto. Por exemplo, na classe "funcionario", se quiser adicionar a propriedade "posto", proceda da seguinte forma:

```
funcionario.prototype.posto = null;
```

O valor padrão null é atribuído para que todas as instâncias de "funcionario" tenham o valor null. Depois de a nova propriedade ser criada, pode atribuir-se valores às instâncias:

```
funcionario1.posto = "segurança";
funcionario2.posto = "chefe";
```

2.3. MÉTODOS

Um método é uma função apropriada de um objeto. Para definir um método, utiliza-se a mesma sintaxe da criação de qualquer função:

```
function saldoMensal()
{
…
código
…
}
```

Depois de criada a função, é necessário atribuí-la ao objeto (isto é feito dentro do construtor do objeto, para que todas as instâncias desta função tenham aquele método associado). Basta incluir dentro do construtor:

```
this.saldoMensal = saldoMensal;
```

Um método tem as mesmas características que qualquer outra função, portanto, se necessário, poderá incluir parâmetros; mas é importante referir que as propriedades do objeto são acessíveis sem ter de as passar como parâmetros para a função; ou seja, a função de calcular o salário seria a seguinte:

```
function saldoMensal()
{
document.write(this.precoHora * this.horas * 30)
}
```

E o construtor da classe utilizada ficaria:

```
function funcionario(antiguidade, precoHora, horas){
this.antiguidade = antiguidade;
this.precoHora = precoHora;
this.horas = horas;
this.saldoMensal = saldoMensal;
}
```

Para executar um método de um objeto, basta invocar o método com um agregado do objeto da seguinte maneira:

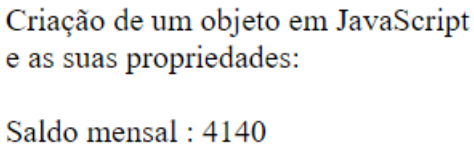
```
funcionario1.saldoMensal();
```

Exemplo 02

Este exemplo mostra como criar um objeto com propriedades e um método que interage com os valores dessas mesmas propriedades para fornecer um valor específico.

Exemplo:

```
<html>
<head>
<script type="text/JavaScript">
function saldoMensal(){
document.write('Saldo mensal : ' + this.precoHora * this.horas *
30)
}
function funcionario(antiguidade, precoHora, horas){
this.antiguidade = antiguidade;
this.precoHora = precoHora;
this.horas = horas;
this.saldoMensal = saldoMensal;
}
</script>
</head>
<body>
Criação de um objeto em JavaScript <br>e as suas
propiedades:<br><br>
<script type="text/JavaScript">
funcionario1 = new funcionario('1980',23,6);
funcionario1.saldoMensal();
</script>
</body>
</html>
```



Resultado do exemplo.

Como viu anteriormente, com a função prototype pode incluir novas propriedades num objeto. No entanto, também existe a possibilidade de apagar as propriedades; para esse fim, utiliza-se a função (e palavra reservada) delete, seguida da instância do objeto e da sua propriedade:

```
delete funcionario1.antiguidade;
```

É uma função muito raramente utilizada: normalmente, não é desejável remover propriedades dos objetos, dado que existe a possibilidade de as modificar para o valor null. Caso o faça e mais tarde tente aceder ao valor da propriedade, ela devolverá um valor undefined.

3. OBJETOS PREDEFINIDOS EM JAVASCRIPT

3.1. STRING

O objeto string é o objeto de texto, ou seja, as propriedades e métodos deste elemento podem ser utilizados na variável na qual insere a string ou diretamente na string:

```
texto = “curso de JavaScript”
```

Este objeto tem apenas uma propriedade chamada length, que indica o número de caracteres que a string possui:

```
ex = text.length
// irá devolver 19,que é o número de caracteres
ex = “curso de JavaScript”.length
// devolve o mesmo que o anterior
```

Este objeto tem vários métodos, alguns alteram a string e outros simplesmente lhe dão forma. Segue-se uma lista com os métodos mais usados deste objeto:

- **grande, negrito, itálico, pequeno, rasurado, sub, sup**: estes métodos devolvem a string com um formato diferente:
  - texto.big devolve a maior string.
  - my_string.bold devolve a string em negrito.
  - my_string.italics devolve a string em itálico.
  - my_string.small devolve a string com o menor tamanho.
  - my_string.strike devolve a string rasurada.
  - my_string.sub devolve a string como um subscrito.
  - my_string.sup devolve a string como sobrescrito.
- **charAt**: devolve o carácter especificado com o índice pedido pelo parâmetro.

```
“curso de JavaScript”.charAt(3) // devolve ‘r’
```

- **charCodeAt:** devolve o valor único do carácter na posição especificada pelo parâmetro.

```
“curso de JavaScript”.charCodeAt(11)
// devolve o código unicode 65 da letra ‘a’
```

- **indexOf**: devolve a posição dentro da string (objeto) da primeira correspondência com o carácter ou caracteres indicados pelo parâmetro. Um segundo parâmetro também pode ser indicado com a posição inicial da pesquisa; se não for indicado, começará do início.

```
“curso de JavaScript”.indexOf(“de”) //devolve 8
```

- **lastIndexOf**: devolve a posição da última correspondência dentro da string (objeto) de um parâmetro passado. Como no caso anterior, um segundo parâmetro pode ser passado para indicar de onde a pesquisa começará.

```
“curso de JavaScript”.lastIndexOf(“a”) //devolve 13
```

- **link**: cria um link HTML sobre a string (objeto) para a qual o método é chamado. O link apontará para o link indicado pelo parâmetro do método.

```
“curso de JavaScript”.link(“http://www.masterd.pt”)
// gera o mesmo que o <a href ></a> do HTML
```

- **concat**: devolve uma string à qual são adicionadas tantas strings quanto desejar (colocando-as nos parâmetros).

```
“curso de JavaScript”.concat(“da masterd” , “centro de estudos”)
//devolve “curso de JavaScript da masterd centro de estudos”
```

- **fromCharCode**: devolve uma string criada a partir da união de vários códigos Unicode.

```
String.fromCharCode(65,66,67,68,69) //devolve ABCDE
```

- **split**: divide um objeto string num array de strings a partir de um separador indicado pelo parâmetro. Opcionalmente, pode indicar um segundo parâmetro como número máximo de divisões.

```
“curso de JavaScript”.split(“ “) //divide em 3 sub-strings
```

- **slice**: corta a string (objeto) numa nova string, que será a indicada entre os parâmetros enviados, o primeiro indica onde começa a nova string e o segundo a posição até à qual é extraída.

```
“curso de JavaScript”.slice(2,6) //devolve “rso”
```

- **substring:** devolve um subconjunto da string (objeto). Recebe dois parâmetros, extraindo do primeiro parâmetro para o segundo sem o incluir. Tendo em consideração que, se ambas forem iguais, a string devolvida estará vazia, e se a primeira for maior que a segunda, elas serão trocadas.

```
“curso de JavaScript”.substring(3,7) //devolve “so d”
```

- **substr:** tal como o anterior devolve uma string que começa no primeiro carácter especificado, mas, neste caso, o segundo valor é o número de caracteres que leva e não a posição onde termina.

```
“curso de JavaScript”.substr(3,7) //devolve “so de j”
```

- **replace**: devolve uma nova string na qual o conteúdo do primeiro parâmetro indicado para a função é substituído pelo valor do segundo parâmetro.

```
“curso de JavaScript”.replace(“JavaScript”,”js”)
//devolve “curso de js”
```

- **search**: devolve a posição da string em que for encontrada uma correspondência com o valor indicado no parâmetro. Caso não seja encontrado, devolve -1.

```
“curso de JavaScript”.search(“vas”) //devolve 11
```

- **toLowerCase**: devolve a mesma string, mas em minúsculas.

```
“Curso De JavaScript”.toLowerCase()
// devolve “curso de javascript”
```

- **toUpperCase:** devolve a mesma string, mas em maiúsculas.

```
“curso de JavaScript”.toUpperCase();
//devolve “CURSO DE JAVASCRIPT”
```

3.2. NUMBER

Este objeto possui propriedades para as constantes numéricas. Visto que nas constantes não se pode modificar os seus valores, é apenas permitido aceder aos valores para os conhecer.

Por exemplo:

```
Number.MAX_VALUE //devolve 1.7976931348612157e+308
```

As constantes são:

- **MAX_VALUE**: valor máximo do tipo numérico.
- **MIN_VALUE:** valor mínimo do tipo numérico.
- **NaN**: indicador de que não é numérico.
- **NEGATIVE_INFINITY**: define o valor a partir do qual o limite negativo ocorrerá.
- **POSITIVE_INFINITY**: define o valor a partir do qual um limite positivo ocorrerá.

Este objeto possui vários métodos aos quais é possível aceder:

- **toExponential**: devolve a string representada com notação exponencial.

```
a = new Number(3.2);
alert(a.toExponential()) // devolve 3.2+e0
```

- **toFixed**: devolve a string representada como um número inteiro ignorando os números decimais.

```
a = new Number(3.2);
alert(a.toFixed()) // devolve 3
```

- **toPrecision**: devolve uma string com o número com casas decimais indicado pelo parâmetro.

```
a = new Number(3.2234);
alert(a.toPrecision(2)) // devolve 3.22
```

- **toString**: devolve o valor do objeto número em string.
- **valueOf:** devolve o valor primitivo do objeto especificado.

```
a = new Number(3.2234);
alert(a.valueOf ()) // devolve 3.2234
```

3.3. MATH

O objeto math contém várias propriedades e métodos para usar funções matemáticas ou constantes.

Todos os métodos do objeto math recebem os argumentos (se forem indicados) em radianos (unidade de ângulo).

Seguem-se as constantes pertencentes ao objeto math e às quais se tem acesso a partir do JavaScript:

- **Math.PI**: devolve o valor do PI.
- **Math.E**: devolve o valor da constante de Euler.
- **Math.LN2**: devolve o logaritmo natural de 2.
- **Math.LOG2E**: devolve o valor do logaritmo de base 2 de E.
- **Math.LN10**: devolve o logaritmo natural de 10.
- **Math.LOG10E**: devolve o logaritmo de base 10 de E.
- **Math.SQRT2:** devolve a raiz quadrada de 2.
- **Math.SQRT12:** devolve a raiz quadrada da metade.

Métodos da classe math:

- **Math.abs(número)**: devolve o valor absoluto do número.
- **Math.min(número1, número2)**: devolve o menor dos dois números indicados no parâmetro.
- **Math.max(número1, número2)**: devolve o maior dos dois números indicados no parâmetro.
- **Math.sin(número)**: devolve o seno do ângulo indicado pelo parâmetro.
- **Math.cos(número)**: devolve o cosseno do ângulo indicado pelo parâmetro.
- **Math.tan(número)**: devolve a tangente do ângulo indicado como parâmetro.
- **Math.asin(número)**: devolve o arco seno do ângulo indicado pelo parâmetro.
- **Math.acos(número)**: devolve o arco cosseno do ângulo indicado pelo parâmetro.
- **Math.atan(número)**: devolve o arco tangente do ângulo indicado pelo parâmetro.
- **Math.atan2(num1, num2)**: devolve o arco tangente cujas projeções nos eixos Y e X são os números num1 e num2.
- **Math.sqrt(número)**: devolve a raiz quadrada do número indicado pelo parâmetro.
- **Math.ceil(número)**: devolve o número indicado pelo parâmetro para o seu maior inteiro.
- **Math.floor(número)**: devolve o número indicado pelo parâmetro para o seu menor inteiro.
- **Math.round(número)**: devolve o inteiro mais próximo do número indicado pelo parâmetro.
- **Math.exp(número)**: devolve o resultado de elevar o número E ao expoente indicado pelo parâmetro.
- **Math.log(número)**: devolve o logaritmo natural do número indicado pelo parâmetro.

- **Math.pow(base, expoente)**: devolve o resultado da potência da base para o expoente indicado pelos parâmetros.
- **Math.random():** devolve um número pseudoaleatório entre 0 e 1, inclusive.

3.4. DATE

Este objeto permite trabalhar com datas. É importante saber que o JavaScript trata todas as datas em milissegundos, assim como acontece com os arrays e índices JavaScript, janeiro será o mês 0 e dezembro o 11, enquanto para os dias da semana, por exemplo, domingo é o dia 0 e sábado é o 6. O formato da data é HH: MM: SS (horas, minutos, segundos).

O objeto date pode ou não receber parâmetros, e pode criar um objeto de data vazio que será carregado, ou criar a data indicando os parâmetros que desejar:

```
data = new Date(ano,mês)
data = new Date(ano, mês, dia)
data = new Date(ano, mês, dia, hora)
data = new Date(ano, mês, dia, hora, minutos)
data = new Date(ano, mês, dia, hora, minutos, segundos)
```

Este objeto depende de uma série de métodos para obter a data ou trabalhar com ela. Segue-se uma lista de métodos:

- **getDate():** devolve o dia do mês atual entre 1 e 31.
- **getDay():** devolve o dia atual da semana entre 0 (domingo) e 6 (sábado).
- **getHours():** devolve a hora atual do dia entre 0 e 23.
- **getMinutes():** devolve os minutos da hora atual entre 0 e 59.
- **getSeconds():** devolve os segundos do minuto atual entre 0 e 59.
- **getMonth():** devolve o mês do ano atual entre 0 e 11.
- **getTime():** devolve o tempo em milissegundos desde 1 de janeiro de 1970.

- **getYear():** devolve o ano atual.
- **setDate(dia):** define o dia indicado pelo parâmetro como o dia do objeto de data.
- **setDay(dia da semana):** define como dia do objeto data o dia indicado pelo parâmetro (de 0 a 6).
- **setHours(hora):** define a hora do objeto data para aquela indicada pelo parâmetro.
- **setMinutos (minutos):** define os minutos do objeto data para aqueles indicados pelo parâmetro.
- **setMonth(mês):** define o mês indicado pelo parâmetro como o mês do objeto data.
- **setSeconds(segundos):** define os segundos do objeto data para aqueles que forem indicados no parâmetro.
- **setTime(milissegundos):** define a data, a partir dos milissegundos que forem indicados desde 1de janeiro de 1970, no objeto data utilizado.
- **setYear(ano):** define o ano indicado pelo parâmetro como o ano do objeto data.
- **toGMTString():** devolve a string convertida para o fuso horário GMT.

Exemplo 03

Neste exemplo, é criado um relógio simples na página.

```
<html>
<head>
<title>Relógio com Javascript</title>
<script type="text/javascript">
function moverRelogio(){
    momentoActual = new Date()
    hora = momentoActual.getHours()
    minuto = momentoActual.getMinutes()
    segundo = momentoActual.getSeconds()
```

```
    imprimirHora = hora + " : " + minuto + " : " + segundo


    document.getElementById("relogio").value = imprimirHora
}
</script>
</head>


<body onload="moverRelogio()">


Aqui está o relógio a funcionar...


<form name="formRelogio">
<input type="text" name="relogio" id="relogio" size="10">
<input type="button" name="rf" value="Atualizar"
onclick="moverRelogio()"/>
</form>
</body>
</html>
```

Resultado da visualização do exemplo.

3.5. ARRAY

O Java Script usa o objeto Array predefinido e os seus métodos para trabalhar com arrays. Um array é um conjunto ordenado de valores com um nome e um índice.

Para criar um array basta chamar o objeto Array com uma série de valores:

```
vector = new Array(elemento1, elemento2, elemento3, …);
```

Seguem-se alguns métodos que o objeto Array permite fazer:

- **concat**: junta os dois arrays num novo.

```
vector = new Array(‘flor’ , ‘casa’, ‘cão’);
vector2 = vector.concat(‘gato’ , ‘guitarra’);
// devolve ‘flor’, ‘casa’, ‘cão’, ‘gato’, ‘guitarra’
```

- **join**: junta os elementos de um array de uma string separada pelo delimitador indicado por parâmetro.

```
vector = new Array(‘gato’, ‘cão’, ‘tartaruga’);
lista = vector.join(‘ – ‘);
// devolve ‘gato – cão - tartaruga’
```

- **reverse**: inverte a ordem dos elementos do array.

```
vector = new Array(‘1’, ‘2’,’3’);
vector.reverse();
// devolve ‘3’, ‘2’,’1’
```

- **slice**: recebe dois parâmetros: o primeiro indica de onde, e o segundo a que distância deve ser extraído o slice do array.

```
vector = new Array(‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’);
vector = vector.slice(2,3);
// devolve ‘3’, ‘4’, ‘5’
```

- **splice**: inserido num array a partir da posição indicada no primeiro parâmetro, o segundo parâmetro indica se elimina algum dos valores existentes e finalmente adiciona-se tantos parâmetros quantos os valores que se pretende incluir.

```
vector = new Array(‘a’, ‘b’, ‘c’, ‘d’, ‘e’);
vector.splice(1,2, ‘1’, ‘2’);
// devolve ‘a’, ‘1’, ‘2’, ‘d’, ‘e’
vector = new Array(‘a’,‘b’,‘c’,‘d’,‘e’);
vector.splice(1,0, ‘1’, ‘2’);
// devolve ‘a’, ‘1’,‘2’,‘b’,‘c’,‘d’,‘e’
```

- **sort**: organiza os elementos do array.

```
vector = new Array(‘gato’,‘cão’,‘águia’);
vector.sort();
// devolve ‘águia’, ‘cão’, ‘gato’
```

CONCLUSÃO

A programação orientada a objetos torna muito mais fácil programar páginas web mais complexas. Também é bastante importante estudar bem os objetos predefinidos para obter o máximo da programação JavaScript.

Executar POO permite criar um código mais limpo e personalizado para uma página web.

AUTOAVALIAÇÃO

1. **O que é um objeto?**

   a) Uma estrutura física criada pelo JavaScript.

   b) Uma estrutura definida pelo que contém: propriedades e métodos.

   c) Um elemento simples que possui propriedades, mas sem métodos.

   d) Um elemento complementar para as funções.

2. **Quais são as propriedades dos objetos?**

   a) Funções definidas para o objeto.

   b) Os dados do nome do objeto, tamanho, entre outros.

   c) Os valores das variáveis que compõem o objeto.

   d) Objetos não possuem propriedades.

3. **Como é que os objetos podem ser tratados em JavaScript?**

   a) Como funções.

   b) Como arrays.

   c) Como variáveis locais.

   d) Como variáveis globais.

4. O que é um objeto indicador?

a) Um objeto JavaScript que indica as variáveis.

b) Um objeto JavaScript para definir funções matemáticas.

c) Uma forma de criar objetos em JavaScript.

d) Um método de controle dos objetos definidos numa página.

5. É possível criar um objeto que possua um array como propriedade com um objeto indicador?

a) Sim, definindo o objeto dentro da definição.

b) Não, não é possível.

c) Apenas para objetos sem métodos.

d) Apenas para objetos sem outras propriedades.

6. O que são os métodos dos objetos?

a) Os valores das variáveis que o objeto contém.

b) Os nomes das variáveis que o objeto contém.

c) As funções set e get já definidas pelo JavaScript.

d) As funções JavaScript que estão associadas ao objeto.

7. Como é que se pode aceder a uma propriedade de um método?

a) Não é possível aceder.

b) Passando-a como parâmetro para o método.

c) Criando-a como variável global e atribuir-lhe o valor antes do método.

d) As propriedades são acessíveis a partir dos métodos do objeto.

8. Que instrução usaremos para criar uma nova propriedade para um objeto existente?

a) newProperty.

b) addProperty.

c) Não pode ser adicionado.

d) Prototype.

9. O que é que delete vector.property permite fazer?

a) Excluir uma propriedade de uma instância de um objeto.

b) Excluir um objeto se a propriedade indicada for nula.

c) Esta declaração não existe.

d) Excluir todas as instâncias do array indicado.

10. Para que serve o método charAt do objeto string predefinido?

a) Devolve o valor Unicode do parâmetro passado.

b) Devolve a posição do carácter passado por parâmetro.

c) Devolve a última posição do carácter indicado.

d) Devolve o carácter indicado no índice passado por parâmetro.

11. Para que serve o método split do objeto string predefinido?

a) Devolver uma string à qual se adicionam outras strings.

b) Dividir um objeto string num array de outras strings a partir de um determinado separador.

c) Criar um link como se fosse um <a href> de HTML.

d) Dividir a string num array de outras strings com cada palavra.

12. Qual é o método que permite extrair uma string indicando dois parâmetros: o primeiro com o início da string e o segundo com o número de caracteres a serem usados?

a) substring.

b) search.

c) substr.

d) getString.

13. Qual é o método do objeto number que devolve a string em formato exponencial?

a) toExponential.

b) toExponent.

c) toFixed.

d) valueEXponent.

14. Qual é o método do objeto number que devolve a string no tipo de texto?

a) toString.

b) toStr.

c) makeSTR.

d) Não existe esse método.

15. O que é que devolve o método abs do objeto math?

a) O valor em formato de texto.

b) O valor absoluto do número anterior.

c) O número de casas decimais de um número abstrato.

d) O valor do PI.

16. Qual é o método que, após indicar um número, devolve o seu inteiro superior?

a) sqrt.

b) atan.

c) ceil.

d) floor.

17. O que é que o método random() devolve?

a) Um número inteiro menor que 30.

b) O valor do logaritmo natural de 10.

c) Um valor aleatório de 1 a 100.

d) Um valor pseudoaleatório de 0 a 1.

18. O que é que o método getDate do objeto Date devolve?

a) O dia, mês e ano atuais.

b) O dia atual.

c) O dia e o mês atuais.

d) As horas.

19. Qual é o método do objeto Array que permite remover o último elemento de um array?

a) pop.

b) push.

c) delete.

d) reverse.

20. Qual é o método do objeto Array que devolve os dados vetoriais numa string, separados pelo carácter indicado pelo parâmetro?

a) concat.

b) push.

c) pop.

d) join.

SOLUÇÕES

1.	b	**2.**	c	**3.**	b	**4.**	c	**5.**	a
6.	d	**7.**	d	**8.**	d	**9.**	a	**10.**	d
11.	b	**12.**	c	**13.**	a	**14.**	a	**15.**	b
16.	c	**17.**	d	**18.**	b	**19.**	a	**20.**	d

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para saber mais sobre a programação orientada a objetos em geral, visite os seguintes sites:

- https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_objetos
- https://www.w3schools.com/js/js_objects.asp

BIBLIOGRAFIA

- MDN Web Docs (2021). "Trabalhando com objetos". Disponível em:

  https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objetos. Consultado a 28 de janeiro de 2021.

- VV. AA. (2010). *JavaScript*. Madrid: Anaya Multimedia.