universidade de aveiro

deti departamento de electrónica, telecomunicações e informática

Introdução às Tecnologias Web

JAVASCRIPT

Gonçalo Matos | MEC 92972

Licenciatura em Engenharia Informática

1.º Semestre | Ano letivo 2018/2019

Índice

Linguagens de programação

O que são?

Uma "linguagem de programação" é uma linguagem que possui uma sintaxe (formato) e uma semântica (significado), e é usada para expressar uma sequência de ações computacionais que formam um "programa".

Existem milhares de linguagens de programação e novas linguagens surgem frequentemente, trazendo novos paradigmas e estabelecendo novos padrões para programadores.

Por isso, e importante conhecer as diferenças principais entre as linguagens e quando o uso de cada uma delas e mais adequado

Conceitos básicos

Compilador verifica a sintaxe do código escrito para garantir que está de acordo com a semântica adequada e, caso tudo esteja correto, gera um código executável a partir do código fonte escrito pelo programador.

O código executável não possui o conteúdo do código fonte, portanto programas de linguagens compiladas são melhores de distribuir quando o programador não quer que o seu código seja publico.

Interpretador executa o código fonte diretamente, lendo trechos do código fonte em tempo de execução, converte em um formato que o computador consegue ler (compilação em tempo de execução) e realiza a sua execução.

Linguagem compilada vs. Script

Chamamos linguagem compilada à que precisa de ser compilada antes de ser executada, caso contrário, é chamada de **script**.

Linguagem Script

Este tipo de linguagem costuma ter uma **performance inferior** às compiladas, pois exigem mais passos, para que seja possível disponibilizar o programa em tempo de execução.

No entanto, são **mais produtivas**, uma vez que cada vez que são alteradas dispensam uma nova compilação.

Algumas linguagens combinam as duas, criando uma representação intermédia, **bytecode**, que depois é interpretada/compilada em tempo de execução.

Linguagens "tipadas" (typed)

Nestas linguagens as operações são realizadas sobre estruturas de dados bem definidas e cada operação define os tipos de dados que deve receber.

Nas linguagens pouco tipadas, as operações são aplicadas para qualquer estrutura de dados, porém, estas podem falhar em tempo de execução caso a estrutura não suporte a operação, aumentando assim a possibilidade de erros, sendo por isso recomendada a realização de testes exaustivos antes de colocar o sistema em reprodução.

//Exemplo

//JAVASCRIPT

```
float soma(float a, int b) {

return a+b;

}
```

//PYTHON

```
def soma(a,b):

    soma=a+b

    return soma
```

Em JAVASCRIPT os tipos de dados da operação soma estão bem definidos: a é do tipo float, b é integer e o tipo de dado que a função devolve também é float.

Em PYTHON, por outro lado, o resultado pode receber vários tipos de dados, dependendo o resultado destes, por exemplo:

- Se a e b forem do tipo integer, vai concatenar os dois e devolver um resultado do tipo int também;
- Se a e b forem inteiros, vai realizar a soma e devolver um resultado do tipo int;
- Se a for int e b float, vai realizar a soma e devolver um resultado do tipo float;

A linguagem Javascript

Introdução

É uma linguagem de **script**, (**interpretada**, portanto), orientada a objetos e que funciona em múltiplas plataformas – tanto do lado do **cliente** como do **servidor**.

Pode ser executada no **browser** ou no **sistema operativo**, podendo estar ligada a objetos do ambiente, fornecendo controlo pragmático (prático) sobre os mesmos (ex.: window.location, window.document, …).

Possui uma biblioteca padrao de objetos, tais como Array, Date, Math, e um conjunto fundamental de elementos da linguagem tais como operadores, estruturas de controle, e *statements*.

O Javascript no desenvolvimento de páginas web

O Javascript é uma das três linguagens que todos os desenvolvedores de páginas *web* devem dominar, sendo estas:

1. HTML para definir o conteúdo das páginas;
2. CSS para especificar o layout das páginas;
3. JAVASCRIPT para programas o comportamento das páginas!

Para que serve?

Originalmente esta linguagem foi implementada nos *web browsers* para que estes pudessem executar programas (scripts) do lado do cliente e interagissem com o utilizador sem a necessidade de recorrer ao servidor.

Um script javascript permite:

- Controlar o *web browser*;
- Realizar comunicações assíncronas (que não se realizam em simultâneo);
- Alterar o conteúdo, de modo dinâmico, do documento exibido.

Os elementos básicos do JavaScript podem ser estendidos com objetos adicionais para uma variedade de propósitos, por exemplo:

Um programa em JavaScript a correr no **cliente** fornece objetos para controlar o browser e o seu Document Object Model (DOM).

Por exemplo, extensões de cliente permitem a uma aplicação adicionar elementos num formulário HTML e responder a eventos do utilizador tais como cliques, input adicionado, e navegação na página.

Um programa em JavaScript a correr no **servidor** fornece objetos relevantes para aceder a funcionalidades diversas.

Por exemplo, extensões do lado do servidor permitem que uma aplicação comunique com uma base de dados, garante continuidade de informação entre invocações da aplicação, ou executa manipulações de ficheiros no servidor.

Vantagens e desvantagens

Como é uma linguagem interpretada, é processada aos blocos e compilada à medida que é necessário converter as diversas estruturas para uma representação capaz de ser executada.

Vantagens aparentemente basta executar diretamente o código escrito pelo programador, evita o *reload* da página HTML cada vez que há uma comunicação com o servidor.

Desvantagens muitos erros só são detetados quando o fluxo de execução atinge a linha onde o erro está presente, que pode provocar paragens na execução.

Inclusão de javascript no documento HTML

À semelhança do CSS, o programa (script) pode ser definido no documento (**inclusão direta**), através dos marcadores **`<script></script>`** no cabeçalho da página ou até mesmo no final do body (por vezes necessário, quando script só pode ser executado no final do carregamento da página), de modo a não interferir com a normal apresentação do documento.

Quando pretendemos atuar por Javascript sobre elementos html representados no DOM (op1 e op2) e como a página É construída de modo incremental e à medida que o documento vai sendo lido pelo browser, e imprescindível que os elementos HTML já estejam representados na DOM quando o código JavaScript que os referencia for executado.

Em alternativa e também como o CSS, este pode ser obtido de uma fonte externa, de ficheiros com a extensão **.js**, obtidos através do marcador **`<link>`** no head do documento, como por exemplo, acontece na utilização do *Bootstrap*.

Versatilidade vs segurança

Alta versatilidade uma vez que para além de ser bastante poderosa pode ser executada em qualquer browser de qualquer sistema operativo, o que permite desenvolver aplicações que podem ser distribuídas de uma forma muito eficaz;

Pouca segurança uma vez que o código Javascript é sempre enviado ao cliente na sua forma textual, podendo por isso ser rapidamente copiado.

Para dificultar a leitura do código, protegendo a autoria do mesmo, e para poupar no espaço ocupado pelo ficheiro, de modo a nao prejudicar o carregamento e posterior apresentação da pagina, este código e muitas vezes "minimizado" (tradução livre de minified) ou "ofuscado" (ofuscated) (ex.: bootstrap.min.js).

A linguagem javascript

É inspirada na linguagem C e semelhante à Java, herdando assim algumas características de ambas.

Esta é baseada em **instruções**, devendo cada uma destas terminar com o caracter "**;**" (ponto e virgula).

Outra característica é que javascript é uma linguagem **case-sensitive**, devendo por isso haver um cuidado redobrado na sua escrita.

```
//INSTRUÇÃO JAVASCRIPT
instrucao;

//CASE SENSITIVE
Sim ≠ sim ≠ sIm ≠ SIM …
```

Sintaxe da linguagem

Declaração e atribuição de variáveis

A declaração de variáveis e feita através da utilização da palavra reservada var seguida pelo nome_da_variável.

A atribuição de valores a uma variável faz-se de modo convencional: <nome_da_variável> <operação> <valor>.

```
//DECLARAÇÃO
var x;

//ATRIBUIÇÃO
x = 3;
```

Funções

De forma a melhor organizar o código, e evitar a replicação desnecessária, e possível organizar um programa em funções.

Tal como a declaração das variáveis e indicada pela palavra reservada var, a declaração de funções faz uso da palavra reservada **function**, sendo estes elementos constituídos por um **nome**, uma **lista de argumentos** e um **corpo**.

```
//FUNÇÃO
function (agr1, agr2) {
    corpo da função
}
```

Declaração de variáveis na consola do navegador e no documento

```
//SCRIPT
var x;
x = 3;
console.log(x);
document.write(x);
alert(x);
```

```
//CONSOLA DO NAVEGADOR
3

//PÁGINA WEB
3
```


```
//NOTA
Existe a chamada a uma função console.log com o argumento x.

O método alert() exibe uma caixa de alerta com mensagem especificada e um botão [OK].
```

Quebras no documento

```
//SCRIPT document.write("<br/>")
```

Operações

Podem ser aplicados operadores aritméticos às variáveis, como a soma (+) ou a subtração (-), dependendo o seu significado do tipo de variáveis da operação.

Por exemplo a soma de int resulta na sua concatenação.

```
//SCRIPT
var x;
var y;
x = 3;
y = "3";
console.log(x+1);
console.log(y+1);
```

```
//CONSOLA DO NAVEGADOR
4
31
```

Operações com erros

Por exemplo, subtrair um int a uma str (não existe operação inversa à concatenação).

Operações condicionais

```
if (compracao){
  instrução caso positivo
} else {
  Instrução caso negativo
}
```

If…else

Utilizada quando queremos verificar uma única condição.

```
switch (a) {
case (val1):
    instrucao;
    break;
case (val2):
    instrucao;
    break;
default:
    intrucao;
}
```

Switch…case

Utilizada quando queremos testar mais do que uma condição (alternativa à estrutura if encadeada).

Cada instrução case deve ser separada por uma instrução **break**, caso contrário o programa continuará a fazer as comparações seguintes e irá executar as instruções por defeito (default).

A instrução defaut (opcional) é executada caso nenhuma das anteriores se verifique.

Ciclos

Do…while

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

While

```
while (condição) {
  instrução;
}
```

For

```
for (inicio,compracao,incremento) {
  instrução;
}
```

Diferenças entre os ciclos

No ciclo **do…while** as instruções são executadas pelo menos uma vez, porque a condição de comparação só é executada no fim do ciclo enquanto que no **while** são executadas 0 ou mais vezes, pois o ciclo só se realiza se a condição se verificar à partida. De outra forma, no ciclo **for**, as instruções são executadas um número fixo de vezes (desde o início até à compração, com um incremento gradual).

Interação com o DOC

Document Object Model (DOC)

O grande potencial da linguagem Javascript quando e executado no web browser advém da possibilidade de aceder a qualquer elemento de uma página HTML.

Isso permite manipular, em tempo real, o conteúdo da página, os estilos e as marcas apos a pagina ter sido carregada sem necessidade de a recarregar novamente.

A caraterística que possibilita esta interação e chamada de Document Object Model (DOM).

Tal como o nome indica, o "modelo de objetos do documento (HTML)" permite que estes sejam utilizados / acedidos / manipulados através de Javascript .

Como aceder aos objetos do DOM?

No HTML DOM tudo são nós:

- O documento em si é um DOM do tipo **document**;
- Todos os elementos HTML são nós do tipo **element**;
- Todos os atributos HTML são nós do tipo **attribute**,
- Texto dentro de elementos HTML são nós do tipo **text**;
- Comentários são nós do tipo **comment**;

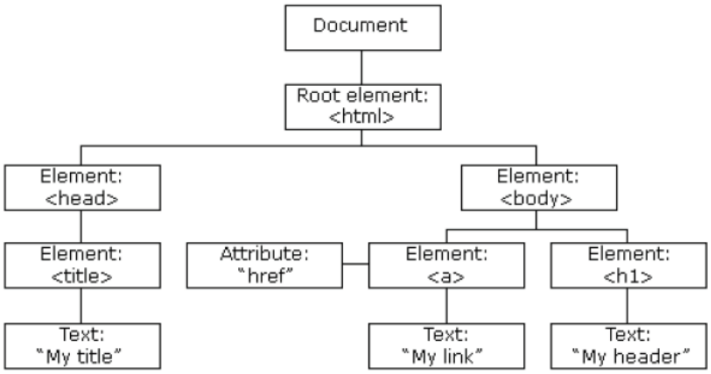
O objeto document

Quando um documento HTML é carregado num browser, ele passa a ser um objeto do tipo document.

Assim, o objeto document é o nó raiz do documento HTML e o "proprietário" de todos os outros nós: (elements, text, atribute, comment).

O objeto document fornece as propriedades e os métodos que permitem aceder a todos os nós, através do JavaScript.

Estrutura DOM de uma página HTML



Interação com o DOM

Noção de objeto, propriedades, métodos e eventos

Podemos considerar que cada nó de um documento HTML é, ele próprio, um **objeto**, possuindo cada um um conjunto de **propriedades, métodos e eventos**.

```
//ACESSO PROGRAMÁTICO A UM OBJETO | MÉTODO

/SCRIPT

var x = document.getElementById("id1")


/HTML

a id="id1" href="url">Texto</a>
```

Elementos inexistentes

Caso se procure um elemento com ID inexistente, o valor devolvido pelo método getElementById será **null**.

Métodos de interação

getElementById(id**)** procura um elemento no DOM que tenha o atributo ID especificado no parâmetro

parseFloat(var**)** converte uma string num valor real (float)

```
//EXEMPLO

/HTML

<body>

<input id="op1" value="2" />

<input id="op2" value="2" />

<input id="res" value="2" />

<script type="text/javascript"
src="dom.js"></script>

</body>
```

```
/SCRIPT

var x = document.getElementById(op1);

var y = document.getElementById(op2);

console.log(parseFloat(x.value));

console.log(parseFloat(y.value));
```

Propriedades dos objetos

var.**value**

Acede à propriedade value de cada um dos objetos devolvidos.

Esta propriedade é de escrita e de leitura, o que significa que se pode alterar facilmente o texto apresentado num dado campo <input>, apenas modificando a propriedade value.