Tutorial JavaScript

MAC0425/5739 - Inteligência Artificial

Departamento de Ciência da Computação
Instituto de Matemática e Estatística - IME
Universidade de São Paulo - USP

20 de agosto de 2015

Agenda

- Introdução
- Sintaxe
- Orientação a objetos
- Estruturas de dados
- Depuração

Introdução (1/2)

- JavaScript NÃO é Java;
- Linguagem de script para adicionar interatividade em páginas web criada por Brendan Eich (Netscape) em 1995;
- Arcabouço para desenvolvimento de aplicações web (*front-end*): jquery.js, backbone.js, angular.js, ember.js, ...;
- Arcabouço para desenvolvimento de aplicações do lado do servidor (*back-end*): node.js;
- **Linguagem de propósito geral com suporte para paradigmas procedural, orientado a objetos e funcional.**

Para o EP nosso objetivo será usar JavaScript como linguagem de propósito geral! Além de aproveitar a flexibilidade e interface de navegador Web! Não é necessário utilizar nenhum framework Web.

Introdução (2/2)

- Influciada por Java (sintaxe), Perl (expressões regulares), Self (herança por protótipos), Scheme (lambda);
- Linguagem interpretada e dinamicamente tipada;
- Alocação de memória gerenciada pelo interpretador (*garbage collector*);
- Orientação a objetos sem classes (protótipos);
- Funções são valores (*first-class citizens*);
- *global namespace* e *mono-thread* no navegador.

Tipos, valores e variáveis (1/3)

```
typeof 42 === "number";
typeof 0.1 === "number";
typeof "hello" === "string";
typeof [] === "object";
typeof true === "boolean;"
typeof undefined === "undefined";
typeof null === "object";
typeof Infinity === "number";
typeof NaN === "number";
```

Tipos, valores e variáveis (2/3)

```
(12 + 9) * 2;  // = 42
0.1 + 0.2;     // = 0.30000000000000004
5 / 2;         // = 2.5
10 % 3;         // = 1

false;         // 0, "", '', undefined, null
true;          // "0", "abc", [], 1, -1, 10, ...

// strings com aspas duplas " ou simples '
'abc'; "hello, world!";

undefined !== null;
Infinity;      // = 1/0
NaN;           // = 0/0
```

Tipos, valores e variáveis (3/3)

```
var foo = 12345;    // não é necessário declaração de tipo
var FOO;            // case-sensitive

var ABC_123, _snake_case, camelCase;   // OK!
var 123_ABC, my-foo, hungry?;          // NOK!

c = 10;  // sem ‘‘var’’ a variável fica no escopo global
d;       // ReferenceError: d is not defined

"use strict"; // verifica erros de uso de variáveis
x = 10  // ReferenceError: assignment to undeclared variable x
```

Controle de fluxo (1/4)

```
// Imprime mensagem no console do navegador
// caso tenha encontrado solução
if (solution !== null) {
  console.log("Solution found!");
}

// comparações são feitas com os operadores relacionais:
// atenção com os símbolos === e !== para evitar problemas!
42 === 21 * 2;
"hello" !== "HELLO";
-3.14 < 0;
Math.abs(-10) >= 1;
```

Controle de fluxo (2/4)

```
// Versão alternativa com cláusula else
if (solution) {
  console.log("Solution found!");
}
else {
  console.log("Oooops... no solution was found!");
}

// Atenção: não esqueça de colocar as
// chaves {} para evitar problemas com o navegador!
```

Controle de fluxo (3/4)

```
// Imprime os 10 primeiros números da sequência
// de Fibonacci armazenados em um Array
var fib = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34];

for (var i = 0; i < 10; i++) {
  console.log("fib[" + i + "] = " + fib[i]);
}

// operador + pode concatenar strings e números
var age = 31;
var msg = "Your age is " + age;
```

Controle de fluxo (4/4)

```
// Versão alternativa usando Array.length
var fib = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34];

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

// Array.length devolve o número de elementos
fib.length === 10;

// Arrays são indexados como em C a partir de 0
var first = fib[0];
var last = fib[fib.length-1];
```

Funções (1/2)

```
// Devolve enésimo número de fibonacci
function fibonacci(n) {

  if (n <= 1) { return n; }

  var fib; var fib0 = 0; var fib1 = 1;
  for (var i = 2; i <= n; i++) {
    fib = fib1 + fib0;
    fib0 = fib1;
    fib1 = fib;
  }

  return fib;
}
```

Funções (2/2)

```
// Versão alternativa: funções são valores
var fibonacci = function (n) {

  if (n <= 1) { return n; }

  var fib; var fib0 = 0; var fib1 = 1;
  for (var i = 2; i <= n; i++) {
    fib = fib1 + fib0;
    fib0 = fib1;
    fib1 = fib;
  }

  return fib;
}; // note que se trata de uma atribuição de variável

// a variável fibonacci recebe uma função anônima
```

Orientação a objetos (1/5)

```
// Um objeto em JavaScript é simplesmente um hashmap ...
// (também conhecido como dicionário)

var initialPosition = {
  xpos: 1,
  ypos: 5
};

typeof initialPosition === "object";

// initialPosition é um literal objeto
// podemos acessar seus atributos através da notação . ou []
console.log(initialPosition.xpos);
console.log(initialPosition["ypos"]);
```

Orientação a objetos (2/5)

```
// Estrutura de dados de um nó na árvore de busca
// pode ser modelado como um objeto

function Node(action, parent, state, depth, g, h) {
  this.state = state;      // representação de estado do nó
  this.parent = parent;    // nó pai na árvore de busca
  this.action = action;    // ação que gerou o nó
  this.depth = depth;      // profundidade do nó na árvore
  this.g = g;              // custo de caminho até o nó
  this.h = h;              // heurística de custo até meta
};

// Em JavaScript não há classes!
// Uma função pode ser usada como construtor de objetos,
// em geral nomeada com primeira letra em maiúscula...
```

Orientação a objetos (3/5)

```
// Podemos criar um novo objeto com base em um construtor
// usando o operador ‘‘new’’

var s0 = problem.initialState;
var initialNode = new Node(null, null, s0, 0, 0, null);

// initialNode é um objeto que representa o nó inicial
// de uma busca não informada:
// Note que o nó inicial não tem pai nem utiliza heurística!
```

Orientação a objetos (4/5)

```
// Podemos criar métodos para os objetos Node
// adicionando funções ao protótipo do objeto...

// Devolve caminho da raiz até o nó
Node.prototype.getPath() = function () {
  var path = [];
  var node = this;
  while (node.parent !== null) {
    var action = node.action;
    path.unshift(action); // adiciona ação no início do array
    node = node.parent;   // sobe um nível na árvore de busca
  }
  return path;
};

// Acessamos o método através da notação .
var solution = goalNode.getPath();
```

Orientação a objetos (5/5)

```
// Um objeto em JavaScript é simplesmente um hashmap!
var Point = function (x, y) {
  this.x = x;
  this.y = y;
};

// É comum definir um método de representação em string
// do objeto para fins de visualização e/ou identificação
Point.prototype.toString = function () {
  return "(" + this.x + ";" + this.y + ")";
};

var origin = new Point(0, 0);
console.log(origin.toString()); // imprime (0;0)
```

Estruturas de dados (1/4)

```
// ATENÇÃO: JavaScript tem suporte nativo para operações de
// manipulação de pilhas e filas..

// Para esse EP a fim de facilitar o desenvolvimento,
// disponibilizamos uma interface para acesso a
// essas estruturas de dados em src/js/utils.js
// juntamente com uma implementação de fila de prioridades

var myStack = new Stack();  // pilhas (LIFO)
var myQueue = new Queue();  // filas (FIFO)
var myPriorityQueue = new PQueue(scoreFunction);
```

Estruturas de dados (2/4)

```
// Definição de pilhas em src/js/utils.js no EP
// Stack.push(), Stack.pop(), Stack.empty()

var myStack = new Stack();
var numbers = [1, 2, 3, 4, 5];
for (var i = 0; i < numbers.length; i++) {
  myStack.push(numbers[i]);
}

// Imprime no console do navegador numbers em ordem inversa
while (!myStack.empty()) {
  console.log(myStack.pop());
}
```

Estruturas de dados (3/4)

```
// Definição de filas em src/js/utils.js no EP
// Queue.put(), Queue.get(), Queue.empty()

var myQueue = new Queue();
var numbers = [1, 2, 3, 4, 5];
for (var i = 0; i < numbers.length; i++) {
  myQueue.put(numbers[i]);
}

// Imprime no console do navegador numbers em ordem de chegada
while (!myQueue.empty()) {
  console.log(myQueue.get());
}
```

Estruturas de dados (4/4)

```
// Definição de fila de prioridade em src/js/utils.js no EP
// PQueue.put(), PQueue.get(), PQueue.empty()
var scoreFn = function (node) { return node.g + node.h; };
var myPriorityQueue = new PQueue(scoreFn);

myPriorityQueue.put(node1);
myPriorityQueue.put(node2);
myPriorityQueue.put(node3);

// Imprime os estados em ordem crescente da função de avaliação
while (!myPriorityQueue.empty()) {
  var node = myPriorityQueue.get();
  console.log(node.state.toString());
}
```

Console

`https://developer.mozilla.org/en/docs/Tools/Web_Console`

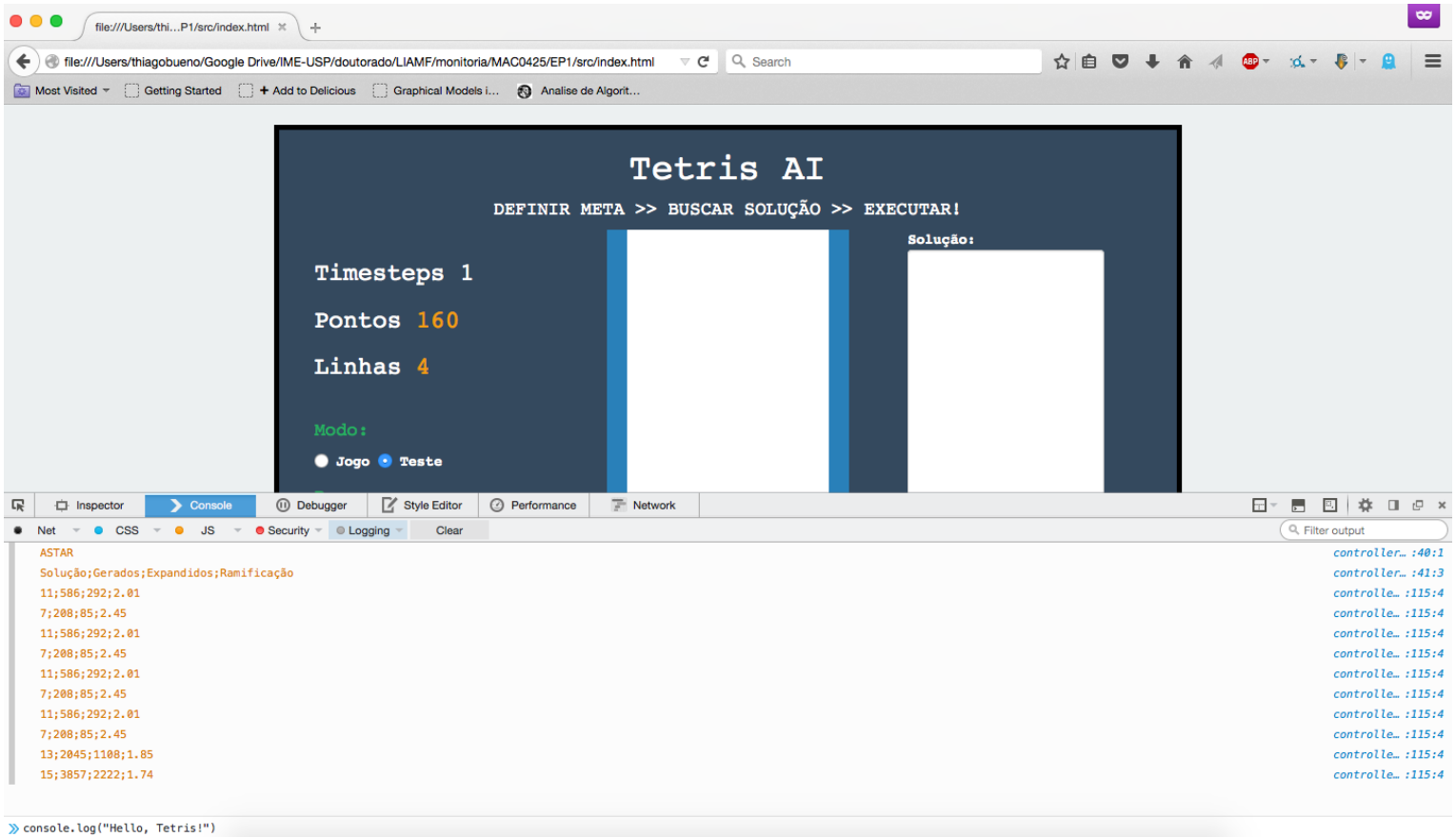

Figura 1: O console JavaScript do navegador pode ser acessado no menu "Ferramentas >> Desenvolvimento Web": você pode executar código JavaScript na linha de comando na parte inferior

Depurador

`https://developer.mozilla.org/en-US/docs/Tools/Debugger`

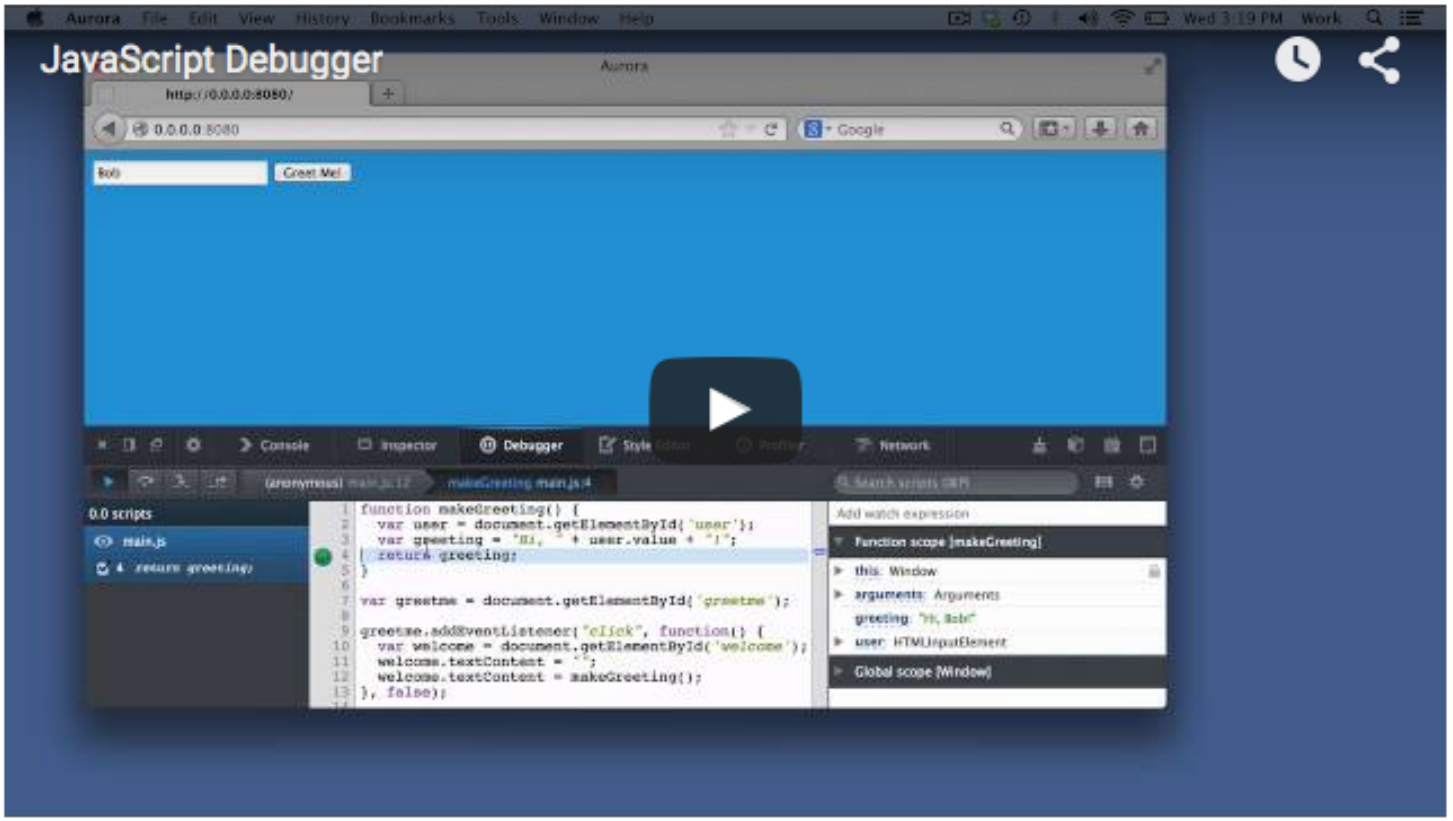

Figura 2: Usando o depurador podemos executar o código passo-a-passo inspecionando e modificando dinâmicamente as variáveis.

Bibliografia I

[1] Learn X in Y minutes Where X=javascript

`http://learnxinyminutes.com/docs/javascript/`

[2] A re-introduction to JavaScript (JS tutorial)

`https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript`

[3] Mozilla Developer Network JavaScript (MDN)

`https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[4] JavaScript: The Good Parts - Douglas Crockford

`https://www.youtube.com/watch?v=hQVTIJBZook`

GoogleTechTalks.