Valderi Leithardt, Dr.

IPW (Class notes - 1 week)

Summary

- Review of previous class *
- Objects
- Functions
- Arrays
- Exercises

Data Types

- **Tipos numéricos**

→ JS os números são representados pelo padrão IEEE 754. Todos os valores numéricos são "declarados" pela simples atribuição dos valores a uma variável.

Inteiros

```
var x = 35;  //atribuição na forma comum
var x = 0543; //notação octal que equivale a 357
var x = 0xBF; //notação hexadecimal que equivale a 191
```

Ponto Flutuante

```
var x = 35;  //atribuição na forma comum
var x = 0543; // notação octal que equivale a 357
var x = 0xBF; //notação hexadecimal que equivale a 191
```

Data Types (cont…)

- **Boolean**

  → JS converte automaticamente true para 1 e false para 0 quando isso for necessário.

Example:

```
var a = 14;
var b = 42;
var tr = (a == 14);
var fl = (a == b);
// Neste caso tr irá conter o valor true e fl o valor false.
var int1 = tr+1;
var int2 = fl+1;
// A variável int1 irá conter o valor 2 (true + 1), pois true é
// automaticamente convertido para 1 e int2 irá conter o valor 1
// (false + 1), pois false é convertido para 0.
```

Data Types (cont…)

- **Undefined**

→ A variável é indefinida quando ela foi declarada de alguma forma mas não possui nenhum valor concreto armazenado. Quando tentamos acessar uma variável que não teve nenhum valor associado a ela teremos como retorno "undefined“ (indefinido).

Example:

```
var isel;
window.alert(isel);
// Quando tentamos imprimir a variável isel na janela de alerta
// será impresso "undefined" pois não há nenhum valor associado a ela.
var text = "";
// O mesmo não ocorre com o caso acima, pois essa variável contém uma
// sequência de caracteres nula e nada será impresso.
```

Data Types (cont...)

- **Null**

→ É a ausência de valor; quando atribuímos **null** a um objeto ou variável significa que essa variável ou objeto não possui valor válido. Para efeito de comparação, se usarmos o operador de igualdade "==", JS irá considerar iguais os valores **null** e undefined. E isso não afeta o uso da comparação (var.metodo == null) quando pretendemos descobrir se um objeto possui determinado método. No entanto, se for necessário diferenciar os dois valores é recomendável o uso do operador "===" de identidade. Assim, para efeito de comparação, undefined e **null** são iguais, mas não idênticos.

Example:

```
var vazio = null;
var ind;
var res = (vazio == ind);
var res1 = (vazio === ind);
// Quando executado a variável res terá o valor true
// e res1 terá o valor false. E se tentarmos imprimir
// a variável vazio, teremos null impresso.
```

Data Types (cont...)

- **Strings**

→ São sequências de caracteres. **Em JS a string** pode ser tanto um tipo primitivo de dado como um objeto; no entanto, ao manipulá-la temos a impressão de que sejam objetos pois as strings em JavaScript possuem métodos que podemos invocar para realizar determinadas operações sobre elas.

Example:

```
var str = "Eu sou uma string!";
var str2 = ’Eu também sou uma string’;
// Declaração de strings primitivas
var str3 = new String("Outra string");
// Acima um objeto string declarado de forma explícita
// não há diferença nenhuma entre esses dois tipos no que se refere
// a seu uso.
```

Data Types (cont...)

- **Arrays**

→ São pares do tipo inteiro-valor para se mapear valores a partir de um índice numérico. Em JavaScript os Arrays são objetos com métodos próprios. Um objeto do tipo Array serve para se guardar uma coleção de itens em uma única variável.

Example:

```
var arr = new Array();
// Por ser um objeto podemos usar o "new" em sua criação
var arr = new Array(elem1,elem2, ... ,elemN);
// Dessa forma criamos um array já iniciado com elementos.
var arr = [1,2,3,4];
// outra forma é iniciar um array com elementos sem usar o "new".
var arr = new Array(4);
// Dessa forma criamos um array vazio de 4 posições.
```

Data Types (cont...)

- **Arrays**

  → Para acessar as variáveis de um array basta usar o nome do array e o índice da variável que se deseja acessar.

Example:

```
arr[0] = "Da-lhe Grêmio!";
arr[1] = 42;
document.write(arr[1]);
//imprime o conteúdo de arr[1]
```

→ Da mesma forma, que pode-se fazer atribuições ou simplesmente ler o conteúdo da posição

→ Em JS, arrays podem conter valores de tipos diferentes sem nenhum problema

→ Podemos colocar em um mesmo array inteiros, strings, booleanos e qualquer objeto que se desejar.

Operators

→ **Aritméticos**

Operador	Operação	Exemplo
+	Adição	x+y
-	Subtração	x-y
*	Multiplicação	x*y
/	Divisão	x/y
%	Módulo (resto da divisão inteira)	x%y
-	Inversão de sinal	-x
++	Incremento	x++ ou ++x
--	Decremento	x-- ou --x

Operators

→ **Comparação**

Operador	Função	Exemplo
==	Igual a	(x == y)
!=	Diferente de	(x != y)
===	Idêntico a (igual e do mesmo tipo)	(x === y)
!==	Não Idêntico a	(x !== y)
>	Maior que	(x > y)
>=	Maior ou igual a	(x >= y)
<	Menor que	(x < y)
<=	Menor ou igual a	(x <= y)

Operators

→ **Bit a bit**

Operador	Operação	Exemplo
&	E (AND)	(x **&** y)
|	OU (OR)	(x **|** y)
^	Ou Exclusivo (XOR)	(x **^** y)
~	Negação (NOT)	**~**x
>>	Deslocamento à direita (com propagação de sinal)	(x **>>** 2)
<<	Deslocamento à esquerda (preenchimento com zero)	(x **<<** 1)
>>>	Deslocamento à direita (preenchimento com zero)	(x **>>>** 3)

Operators

→ **Atribuição**

Operador	Exemplo	Equivalente
=	x = 2	Não possui
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
&=	x &= y	x = x & y
|=	x |= y	x = x | y
^=	x ^= y	x = x ^ y
>>=	x >>= y	x = x >>= y
<<=	x <<= y	x = x <<= y
>>>=	x >>>= y	x = x >>>= y

Operators

→ **Lógicos**

Operador	Função	Exemplo
&&	E Lógico	(x **&&** y)
||	OU Lógico	(x **||** y)
!	Negação Lógica	**!**x

→ **Estruturas de controle**

Operators

→ **Estruturas de controle**

if ... else
….
…
switch ... case
…
…..
while
….
..
do ... while
….
..
for
..
…
for ... in

Exercises

- …

References

- https://eloquentjavascript.net/01_values.html
- https://github.com/braziljs/eloquente-javascript/blob/master/chapters/01-valores-tipos-operadores.md

Valderi Leithardt, Dr.

Professor IPW

valderi.leithardt@isel.pt