Prof. Esp. Thalles Canela

- **Graduado:** Sistemas de Informação - Wyden Facimp
- **Pós-graduado:** Segurança em redes de computadores - Wyden Facimp

- **Consultor de Tecnologia - [aXR6] Cyber Security e NtecSoftware**
- **Professor no Senac (contratado)**
- **Professor na Wyden Facimp (contratado)**
  - **Pós-graduação:** Segurança em redes de computadores - Wyden Facimp
- **Professor na Wyden Facimp (Efetivado)**
  - **Graduação:** Análise e desenvolvimento de sistemas - Wyden Facimp

Redes sociais:

- **Linkedin:** https://www.linkedin.com/in/thalles-canela/
- **YouTube:** https://www.youtube.com/aXR6CyberSecurity
- **Facebook:** https://www.facebook.com/axr6PenTest
- **Instagram:** https://www.instagram.com/thalles_canela
- **Github:** https://github.com/ThallesCanela
- **Github:** https://github.com/aXR6
- **Twitter:** https://twitter.com/Axr6S

Linguagem compilada vs. Linguagem script

- Compilação: **o código-fonte é lido pelo compilador que gera então um arquivo de saída com uma tradução daquele código-fonte para linguagem de máquina** (o código executável); esse arquivo em linguagem de máquina pode ser então executado no computador e não pode ser facilmente editado, pois não é compreensível por nós seres humanos. Assim, se desejarmos alterar alguma parte desse programa, precisaremos alterar seu código-fonte e compilá-lo novamente para que o executável novo seja gerado.

Linguagem compilada vs. Linguagem script

- para executar um programa em uma linguagem interpretada (script) precisamos apenas digitar o código-fonte e o interpretador irá ler esse código e executar as instruções, comando por comando, a partir do próprio texto do código-fonte, cada vez que o script for rodado; assim, não é necessário a criação de um arquivo estático, para alterar o programa basta alterar o código e ele já estará pronto para rodar novamente.

- **JavaScript é uma linguagem de script.**

Tipagem Dinâmica

- **Nesse modo de tipagem, as variáveis podem assumir qualquer tipo ou objeto definido pela linguagem**, por exemplo, se uma variável X receber um valor inteiro ela irá se comportar como uma variável inteira, e se mais tarde X receber uma string passará a se comportar como uma string daquele ponto em diante.
- **Assim, não é preciso definir o tipo da variável no momento da sua declaração.** O tipo da variável é definido implicitamente pelo seu valor.

- **JavaScript é uma linguagem de programação de tipagem dinâmica.**

Funções de ordem superior

- São funções que recebem uma ou mais funções como argumentos ou que têm uma função como saída. Com isso, é possível criar o que são chamadas function factories que são funções que a partir de outras funções simples são capazes de realizar ações mais complexas.

- **JavaScript é uma linguagem de programação que possibilita a definição de funções de ordem superior.**

Programação Client-side vs. Server-side

- **Client-side:** Quando o programa é criado com esta característica ele é enviado para o computador cliente ainda na forma de código-fonte, que só então é interpretado e executado, dependendo assim unicamente da capacidade de processamento do cliente.
- **Server-side:** é executado no computador Servidor e somente é enviado para o cliente o resultado da execução, sejam dados puros ou uma página HTML.

- **Neste estudo, tratamos JavaScript apenas como uma linguagem de programação Client-side.**

Segurança

- Por ser uma linguagem que é executada no computador do cliente, o JavaScript precisa ter severas restrições para evitar que se façam códigos maliciosos que possam causar danos ao usuário.
- As principais limitações do JavaScript para garantia de segurança são a proibição de:

1. Abrir e ler arquivos diretamente da máquina do usuário;
2. Criar arquivos no computador do usuário (exceto cookies);
3. Ler configurações do sistema do usuário;
4. Acessar o hardware do cliente;
5. Iniciar outros programas;
6. Modificar o valor de um campo de formulário do tipo <input>;

Tipos de dados - Tipos numéricos

- **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 = 12,3; *//declarado na forma comum*

var x = 4,238e2; *//declarado como potência de 10 que equivale a 423,8*

Tipos de dados - Tipos numéricos

- **Booleano**

```
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.
```

Tipos de dados - Tipos numéricos

- **Indefinido**

```
var marvin;
window.alert(marvin);
// Quando tentamos imprimir a variável marvin 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 caractéres nula e nada será impresso.
```

Tipos de dados - Tipos numéricos

- **null**

```
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.
```

Tipos de dados - Tipos numéricos

- **Strings**

```
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.
```

Tipos de dados - Tipos numéricos

- **Arrays**

```
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.
```

Acessando as varáveis dentro de um array:

```
arr[0] = "Até mais e obrigado pelos peixes";
arr[1] = 42;
document.write(arr[1]);
//imprime o conteúdo de arr[1]
```

Tipos de dados - Operadores

- **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

Tipos de dados - Operadores

- **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)

Tipos de dados - Operadores

- **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)

Tipos de dados - Operadores

- **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

Tipos de dados - Operadores

- **Lógicos**

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

Estruturas de controle

- **if ... else**

- A estrutura if é usada quando se deseja verificar se determinada expressão é verdadeira ou não, e executar comandos específicos para cada caso.

Estruturas de controle

- **if ... else**

```
var a = 12;
var b = 5;
if (a == b) {
    window.alert("12 é igual a 5?!?!");
} else {
    window.alert("a é diferente de b");
}
// No caso acima a frase escrita seria "a é diferente de b"
```

Estruturas de controle

- **if ... else**

```
var a = 10;
if (a < 6) {
    window.alert("a  menor que 6");
} else  if (a > 6) {
    window.alert("a  maior que 6");
} else {
    window.alert("se  a não é maior nem menor que 6, a é 6!");
}
```

Estruturas de controle

- **switch ... case**

- As estruturas do tipo switch são usadas quando queremos selecionar uma opção dentre várias disponíveis.

Estruturas de controle

- **switch ... case**

```
var marvin = "robot";
switch (marvin) {
	case "human":
		document.write("hello carbon unit!");
		break;
	case "alien":
		document.write("brrr I hate aliens!");
		break;
	case "robot":
		document.write("emergency, to the rescue!");
		break;
	default:
		document.write("what are you?");
		break;
}
```

Estruturas de controle

- **while**

- Os laços do tipo while são usados quando se deseja que uma sequência de ações seja executada apenas no caso da expressão de condição ser válida. Assim, primeiro a expressão é testada, para depois o conteúdo do laço ser executado ou não.

Estruturas de controle

- **while**

```
var cont = [5,2];
while ((cont[0]+cont[1]) < 15) {
        cont[0]+=1;
        cont[1]+=2;
        document.write('cont0 = '+cont[0]+'cont1 = '+cont[1]);
}
```

// Com o uso de while, no primeiro teste, cont[0]+cont[1] vale 7;

Estruturas de controle

- **do ... while**

- Diferentemente do while, o do ... while primeiro executa o conteúdo do laço uma vez e, depois disso, realiza o teste da expressão pare decidir se continuará executando o laço ou irá seguir o resto do programa.

Estruturas de controle

- **do ... while**

```
var cont = [5,2];
do{
        cont[0]+=1;
        cont[1]+=2;
        document.write('cont0 = '+cont[0]+'cont1 = '+cont[1]);
} while ((cont[0]+cont[1]) < 15)
```

// Com o uso de do...while, no primeiro teste, cont[0]+cont[1]
// já valerá 10, e os contadores já terao sido impressos uma vez
// pois o laço já foi executado a primeira vez antes do teste!

Estruturas de controle

- **for**

- Na maioria das vezes, quando usamos um laço do tipo while também construímos uma estrutura com um contador que é incrementado a cada passo para controle do laço e manipulação interna de objetos, arrays como nos exemplos anteriores. Os laços for oferecem a vantagem de já possuírem em sua estrutura essa variável de contador e incrementá-la de maneira implícita.

Estruturas de controle

- **for**

```
var cont = [5,2,3];
for(var i=0 ; i < 3 ; i++) {
        cont[i]++;
}
```

// Ao final do laço cada elemento do vetor cont foi incrementado em 1

Estruturas de controle

- **for ... in**

- Existe uma segunda forma de se utilizar os laços for para percorrer propriedades de um objeto.

Estruturas de controle

- **for ... in**

```
var doc = document;
for(var prop in doc) {
    document.write(prop+"<br />");
}
```

// Esse laço automaticamente itera pelas propriedades do objeto,
// No caso ele listara todas as propriedades do objeto Document
// responsavel pelo controle do documento exibido na tela.
// Se olhar com cuidado encontrará nessa lista o proprio método
// Write que usamos para imprimir no documento com document.write.