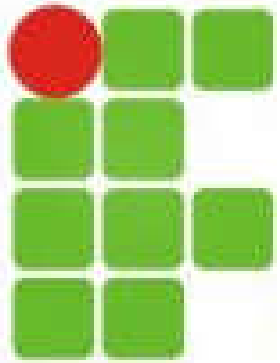
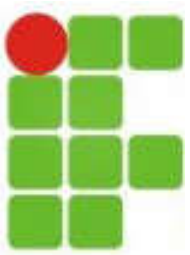
# Fundamentos de Web Design 2

Professor Eng. Dr. Will Roger Pereira



# Conteúdo

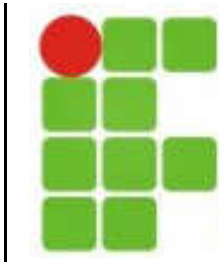


- Operadores;
- Instruções condicionais e repetitivas;
- Funções.

# Operadores

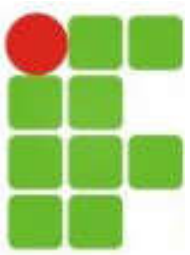
- Atribuição;
- Aritméticos;
- Relacionais;
- Lógicos;
- Unários;
- Ternário;
- Igualdade;
- Identidadade;
- Desigualdade.

# Atribuição



- Operador “=”;
- Envolve uma variável (ou constante) e um valor;
- O valor, à direita, será armazenado na variável, à esquerda;
- Sintaxe:
  - variável = valor;
- Valor pode ser;
  - Um valor literal;
  - Resultado de uma operação;
  - Retorno de uma função.

# Atribuição



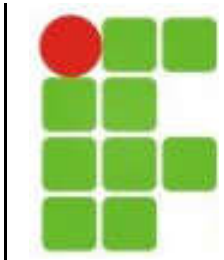
- Suponha variáveis já criadas:

- Atribuição de um literal:
  - var01 = 67;

- Atribuição de uma operação:
  - var02 = valor + 2;

- Atribuição do retorno de uma função:
  - var03 = func01();

# Operadores Aritméticos



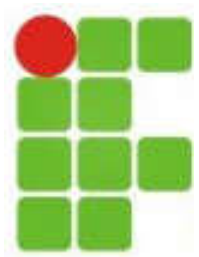
| Operador | Significado |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Resto da divisão |

O operador + pode ser utilizado também como operador de concatenação. Portanto, bastante cuidado ao utilizá-lo!

**Curiosidade**: É possível executar a expressão abaixo? Em caso afirmativo, que resultado será gerado?

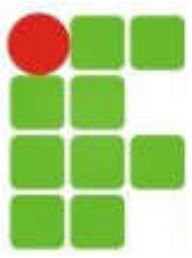
```
var valor = 10.3 % 3;
```

# Operações com String e Number



| Instrução(ões) | Resultado | Observação |
|---|---|---|
| var valor = 10;<br>alert(valor); | Exibe uma mensagem | alert espera uma string. Conversão implícita. |
| var valor = 14;<br>var msg = “Nro =” + valor; | “Nro = 14” | Conversão implícita. O operador + (concatenação neste caso) informa ao mecanismo de scripting para converter o conteúdo da variável valor em string. |
| var valor = “4” + 3 + 1 | “431” | Como o primeiro valor é uma string os demais serão tratados como string. Consequência: concatenação de strings |
| var valor = 4 + 3 + “1” | “71” | Soma os dois primeiros valores e por último concatena o resultado da soma a string “1” |
| var valor = “4” + 3 - 1 | 42 | Assim como no anterior, as operações foram feitas da esquerda para a direta. |
| var valor = “30” * 2 | 60 | A variável valor armazenará um literal do tipo numérico. |

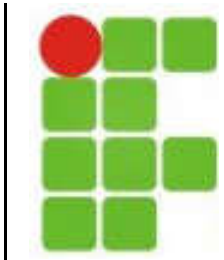
# Operadores Unários



- Podem ser pré ou pós-fixados:

| Operador | Significado | Posição |
|---|---|---|
| ++ | Incrementa 1 ao operando | Pós |
| -- | Diminui 1 do operando | Pós |
| - | Representa um valor negativo | Pré |

# Atribuição com Operação

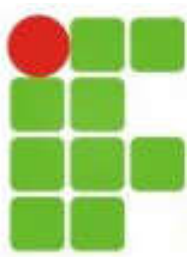


- É possível juntar a operação aritmética com a atribuição em uma única instrução simples se a mesma variável aparecer em ambos os lados do operador.

- Exemplos:
  - valor += 3.0;  // valor = valor + 3;
  - valor *= 3.0;  // valor = valor * 3;

- Todos os operadores aritméticos binários podem ser utilizados nesta técnica.

# Precedência de Operadores

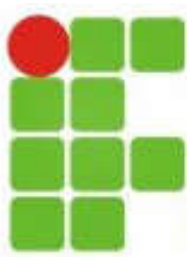


- Avaliação da expressão:

| Operadores | Prioridade (ordem de execução) |
|---|---|
| () | 1º |
| *, /, % | 2º |
| +, - | 3º |
| Atribuição | 4º |

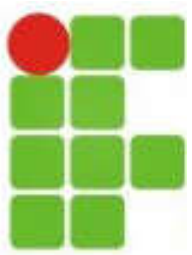
- Operadores com mesma prioridade são avaliados da esquerda para direita:

# Igualdade e Identidade



| Operador | Nome | Observação |
|---|---|---|
| == | Igualdade | Comparar o conteúdo de uma variável a outra ou a um literal |
| === | Identidade | Retorna true somente se ambos os operandos sejam do mesmo valor e tenham o mesmo tipo de dados |
| != | Desigualdade | Verifica a desigualdade entre variáveis, literais, etc |
| !== | Desigualdade escrita | Verifica a desigualdade entre variáveis, literais, além de verificar o tipo |

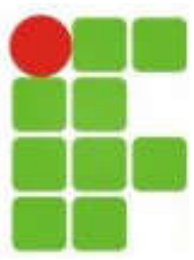
# Operadores Relacionais



- Servem para comparar string alfabeticamente ou números:

| Operador | Nome | Observação |
|---|---|---|
| > | Maior | Retorna true se o operando da esquerda for maior que o da direita |
| >= | Maior ou igual | Retorna true se o operando da esquerda for maior ou igual ao da direita. |
| < | Menor | Retorna true se o operando da esquerda for menor que o da direita |
| <= | Menor ou igual | Retorna true se o operando da esquerda for menor ou igual ao da direita. |

# Operadores Lógicos



- Normalmente utilizados para comparar resultados de duas operações comparativas (igualdade ou relacional):

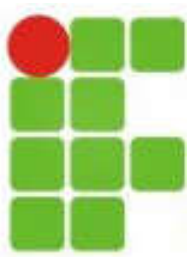
**&& (E)**

| Op 1 | Op 2 | Resultado |
|---|---|---|
| false | false | **false** |
| true | false | **false** |
| false | true | **false** |
| true | true | **true** |

**|| (OU)**

| Op 1 | Op 2 | Resultado |
|---|---|---|
| false | false | **false** |
| true | false | **true** |
| false | true | **true** |
| true | true | **true** |

# Operador Ternário



- Permite atribuir um valor à uma variável dependendo de uma condição;

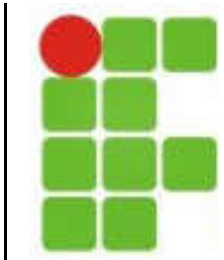
- Condição oriunda de operações de comparação e lógicas;

**Condição ? Valor se verdadeira : Valor se falsa;**

Exemplo:

```
var valor = 10;
var result = (valor > 5) ? “Maior” : “Menor”;
```

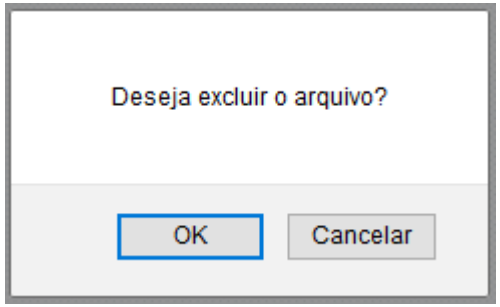
# Caixas de Diálogo



- Em Javascript, é possível gerar caixas de diálogo para requerer informações do usuário;
- Enquanto o usuário não interagir com a caixa, o programa não continuará, i. e., são síncronas.
- **Alert (já aprendido):** Mostra uma informação para notificar o usuário;
- **Confirm:** Solicita uma confirmação do usuário para realizar determinada ação;
- **Prompt:** Solicita uma informação textual do usuário.

# Confirm

- Invocada através da função **confirm(texto);**

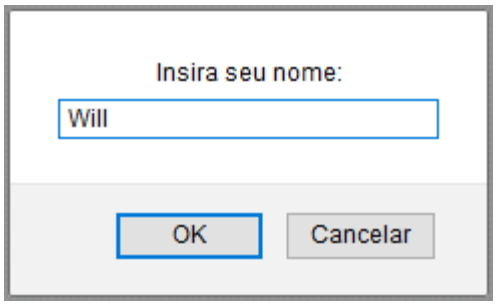



- Retorna **true**, caso o usuário clique em OK, ou **false**, caso o usuário clique em Cancelar;

- Ótimo para ser utilizado em estruturas condicionais.

# Prompt

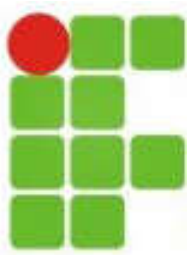
- Invocada através da função **prompt(texto);**




- Retorna uma **string** contendo a informação digitada, caso o usuário clique em OK, ou **null**, caso o usuário clique em Cancelar;

- Ótimo para receber uma única informação do usuário. Para várias utilize um formulário.

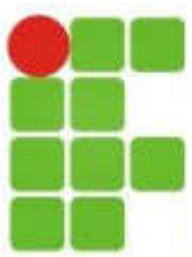
# Estruturas condicionais e repetitivas



- Estruturas condicionais:
  - if/else
  - switch

- Repetitivas:
  - for
  - for ... in
  - while
  - do ... while

# Estrutura condicional if



- Seleção simples

  if () { ... }

- Seleção composta

  if ( ...) { ... }

  else {...}

- Seleção encadeada

  if (...) { ... }

  else if (...) {...}

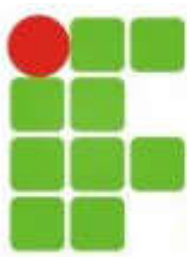
  else {...}

**DICA 1**: Utilize sempre chaves, pois caso realize alguma alteração na lógica, a probabilidade de erro é menor.

**DICA 2**: Procure alinhar a chave de fechamento à instrução condicional que a abriu, pois isso facilita a legibilidade do código.

**Identação sempre.**

# Exercício



- Utilizando JavaScript, construa uma página HTML capaz de receber um número digitado pelo usuário, entre 0 e 100, e verificar se o mesmo é par ou ímpar.

- Caso a informação digitada pelo usuário não se encaixe dentro do esperado, mostre uma notificação ao mesmo;

- Caso seja digitado um número entre 0 e 100, informe o resultado da operação através de uma notificação.

# Estrutura condicional switch

```
switch (<condição>) {
        case valor01:
                <instrução(ões)>
                break;
        case valor02:
                <instrução(ões)>
                break;
         ...
         default:
                <instrução(ões)>
    }
```

O que acontece se não for usado o *break*?

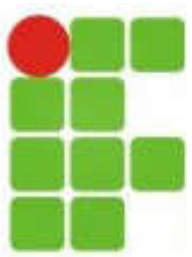
# Exercício

- Qual a mensagem será impressa após a execução do script abaixo?

```
<script>
  var teste = Jovem';
  switch (teste) {
      case 'jovem':
          alert("Um jovem");
      case 'Jovem':
          alert("O jovem");
      default:
          alert("Não é jovem");
  }
</script>
```

Proponha uma solução para que independentemente da forma como a palavra Jovem for escrita apenas um único case será executado.

# Estrutura repetitiva for



**FOR**

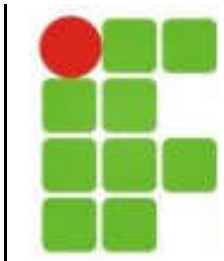
```
for(valor inicial; decisão; operação){
    Instruções
}
```

**Exemplo**

```
for(var i=0; i<10; i++){
    console.log(i);
}
```

- Processa o conjunto de instruções somente se a decisão for **true**;
- Ótimo para repetições com condições de fronteira conhecidas.

# Estrutura repetitiva for...in



**FOR... IN**

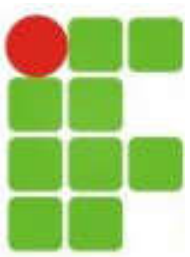
```
for(var elemento in vetor){
    Instruções
}
```

**Exemplo**

```
var vetor = [1, 2, 3, 4, 5];
for(var e in vetor){
    console.log(e);
}
```

- Exclusivo para iterações com elementos de um **array**.
- Executa as instruções dentro do escopo para cada elemento do array.

# Estrutura repetitiva while



**WHILE**

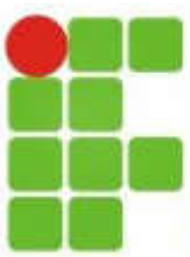
```
while(condição){
    Instruções
}
```

**Exemplo**

```
var n = 0;
while(n < 10){
    console.log(n);
    n++;
}
```

- Processa o conjunto de instruções somente se a condição for **true**;
- Observe que a operação deve ser feita dentro do escopo;
- Ótimo para repetições com condições de fronteira desconhecidas.

# Estrutura repetitiva do... while



**DO... WHILE**

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

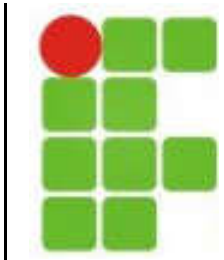
**Exemplo**

```
var n = 0;
do{
    console.log(n);
    n++;
} while(n < 10);
```

- Processa o conjunto de instruções uma vez sem verificação. As demais vezes somente se a condição for **true**;
- Observe que a operação deve ser feita dentro do escopo;

- Ótimo para repetições com condições de fronteira desconhecidas.

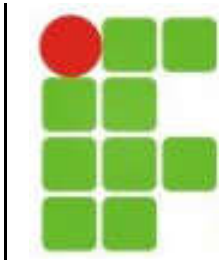
# Exercício



- Utilizando JavaScript, construa uma página HTML capaz de receber um número digitado pelo usuário, e realizar uma somatória de todos os números digitados;

- Caso a informação digitada pelo usuário não se encaixe dentro do esperado, mostre uma notificação ao mesmo, com o resultado, e encerre a aplicação.

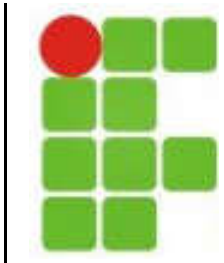
# Exercício



- Utilizando JavaScript, construa uma página HTML capaz de receber um número digitado pelo usuário, e armazenar estes números digitados em um array;

- Caso a informação digitada pelo usuário não se encaixe dentro do esperado, mostre uma notificação ao mesmo, com a média de todos os números presentes no array.

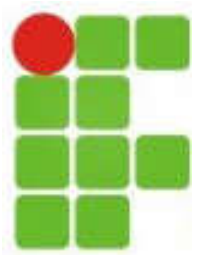
# Funções



- Permitem modularizar as aplicações, agrupando um conjunto de instruções que podem ser invocadas sob demanda;

- Sempre que um conjunto de instruções se fizer presente em mais de uma situação em uma aplicação, coloque-as em uma função.

# Funções declarativas

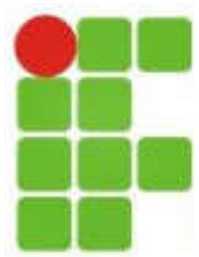


```
function nomeDaFuncao(parametro01, ..., parametroN) {
   ...
}
```

- Passagem de parâmetro:
  - Valores primitivos;
  - Valores armazenados em variáveis/constantes.

- Para retornar valor, utilize a instrução **return valor**;
- **return sempre será a última instrução processada na função.**

# Invocando funções



```
nomeDaFuncao(valor01, ..., valorN);
```

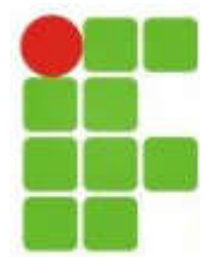
- Utilize o nome da função, e no lugar de cada parâmetro, coloque o respectivo valor desejado;

- É possível utilizar o valor retornado pela função → **return;**

```
var valor = nomeDaFuncao(valor01, ..., valorN);
```

- O retorno de uma função que não possui a instrução **return** será **undefined.**

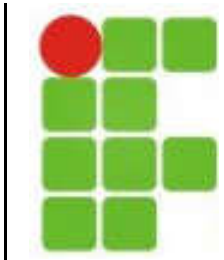
# Funções anônimas



```
var func = function{ parâmetros) {
   ...
}
```

- São funções sem nomes declarados, que podem ser passados por variáveis;
- Utilize o nome da variável para invocar a função: **func(valores);**
- No mais segue as mesmas regras das funções declarativas.

# Exercício



- Utilizando JavaScript, construa uma função para saber se um número e primo ou não;

- Faça uma aplicação para receber 10 números naturais entre 2 e 100, digitados pelo usuário e armazená-los em um vetor. Caso a informação digitada não seja válida, peça a informação novamente, até que seja;

- Após receber todos os números, percorra o vetor verificando quais são primos utilizando a função construída;

- Mostre na tela quantos e quais são os números primos.