```
//OPERADORES RELACIONAIS:
//Operadores relacionais são usados para fazer comparações entre 2 valores e trazer uma resposta "true" ou "false", e esse é um detal
he muito importante do Javascript, ele sempre vai trazer uma mensagem "true" ou "false", nunca irá trazer qualquer outro tipo de resp
osta.
//No Javascript temos 2 formas de fazer comparações relacionais, podemos comparar por: valor, usando os operadores convencionais de r
elação, ou podemos comparar por estritamente, quando comparamos se não somente o valor como também os tipos de dados são iguais. O Ja
vascript tem essas 2 formas de comparar por que ele é uma linguagem fracamente tipada, ou seja, comparação entre valores que signific
am a mesma coisa mas são de tipos diferentes é possível.

//OPERADORES RELACIONAIS CONVENCIONAIS NO JAVASCRIPT:
//Note nos exemplos abaixo que comparamos strings com numbers, e ainda assim não ocorre erro pelo fato da linguagem ser fracamente ti
pada, porém as comparações são somente entre os valores, por que os tipos são diferentes...
const [a, b, c, d] = [1, '1', 2, '2']; //Usamos um destructuring para facilitar a declaração de variáveis e valores...
console.log(`1) Operador de Igualdade (==): ${a} == '${b}'? \t\tResultado: ${a == b}`);
console.log(`1) Operador de Diferença (!=): ${a} != ${c}? \t\t\tResultado: ${a != c}`);
console.log(`1) Operador de Maior Que (>): '${d}' > '${b}'? \t\tResultado: ${d > b}`);
console.log(`1) Operador de Menor Que (<): ${a} < ${c}? \t\t\tResultado: ${a < c}`);
console.log(`1) Operador de Maior ou Igual (>=): '${d}' >= ${a}? \tResultado: ${d >= a}`);
console.log(`1) Operador de Menor ou Igual (<=): '${b}' <= ${c}? \tResultado: ${b <= c}`);



//OPERAÇÕES RELACIONAIS ESTRITAS NO JAVASCRIPT:
//Podemos fazer também comparações estritas que não comparam apenas os valores como também os tipos de dados...
console.log(`\n2) Operador de Igualdade Estrita (===): ${a} === '${b}'? \t\t\tResultado: ${a === b}`); //Perceba que embora os valore
s sejam iguais, o resultado é false por que os tipos são diferentes...
console.log(`2) Operador de Desigualdade Estrita (!==): ${a} !== '${b}'? \t\tResultado: ${a !== b}`); //Perceba que embora os valores
 sejam iguais, o resultado é true, por que os tipos são diferentes...
console.log(`2) Operador de Igualdade Estrita (===): ${a} === ${c}? \t\t\tResultado: ${a === c}`); //Perceba que mesmo sendo de tipos
 iguais, se os valores forem diferentes o resultado é false...
console.log(`2) Operador de Desigualdade Estrita (!==): '${b}' !== '${d}'? \t\tResultado: ${b !== d}`); //Perceba que embora os tipos
 sejam strings, o resultado é true, por que os valores são diferentes...



//OPERADORES RELACIONAIS EM REFERÊNCIAS DE MEMÓRIA:
```

```
//Note que quando usamos variáveis que se transformação em endereços de memória temos que tomar cuidado ao usar operadores relacionais...
const d1 = new Date(0); //Perceba que instanciamos 2 constantes sobre a classe interna do Javascript Date...
const d2 = new Date(0);
console.log(`\n3) ${d1} == ${d2}  Resultado: ${d1 == d2}`); //Perceba que quando fazemos isso, d1 e d2 viram ponteiros para um endereço o valor da data no marco "0", embora os valores sejam iguais as instâncias ocupam espaços de memória diferentes, o que significa que ao fazermos a comparação de é igual ou estritamente igual, o resultado é "false"...
console.log(`3) ${d1} === ${d2}  Resultado: ${d1 === d2}`);
console.log(`3) ${d1.getDate()} == ${d2.getDate()}  Resultado: ${d1.getDate() == d2.getDate()}`); //Agora, quando usamos a função getDate() para retornar para nós um valor na forma de um number, a comparação passa a ser entre os valores e não mais entre espaços de memória. Tanto a igualdade quanto a igualdade estrita gerarão valores "true"...
console.log(`3) ${d1.getDate()} === ${d2.getDate()}  Resultado: ${d1.getDate() === d2.getDate()}`);



//TESTANDO IGUALDADE EM NULL E UNDEFINED:
//Podemos ver também como a diferença conceitual entre null e undefined pode ser mostrada quando usamos operadores de igualdade comum e igualdade estrita...
console.log(`\n4) ${undefined} == ${null}?  Resultado: ${undefined == null}`); //Perceba que, quando fazemos uma comparação de valor entre "null" e "undefined" elas aparentam ser iguais...
console.log(`4) ${undefined} === ${null}?  Resultado: ${undefined === null}`); //Mas quando a comparação é feita de forma estrita, podemos ver que null e undefined não são a mesma coisa...
```

RESULTADO NO CONSOLE...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\tempCodeRunnerFile.js"
1) Operador de Igualdade (==): 1 == '1'?        Resultado: true
1) Operador de Diferença (!=): 1 != 2?          Resultado: true
1) Operador de Maior Que (>): '2' > '1'?        Resultado: true
1) Operador de Menor Que (<): 1 < 2?            Resultado: true
1) Operador de Maior ou Igual (>=): '2' >= 1?   Resultado: true
1) Operador de Menor ou Igual (<=): '1' <= 2?   Resultado: true

2) Operador de Igualdade Estrita (===): 1 === '1'?          Resultado: false
2) Operador de Desigualdade Estrita (!==): 1 !== '1'?       Resultado: true
2) Operador de Igualdade Estrita (===): 1 === 2?            Resultado: false
```

```
2) Operador de Desigualdade Estrita (!==): '1' !== '2'?         Resultado: true

3) Wed Dec 31 1969 21:00:00 GMT-0300 (Horário Padrão de Brasília) == Wed Dec 31 1969 21:00:00 GMT-
0300 (Horário Padrão de Brasília)  Resultado: false
3) Wed Dec 31 1969 21:00:00 GMT-0300 (Horário Padrão de Brasília) === Wed Dec 31 1969 21:00:00 GMT
0300 (Horário Padrão de Brasília)  Resultado: false
3) 31 == 31  Resultado: true
3) 31 === 31  Resultado: true

4) undefined == null?  Resultado: true
4) undefined === null?  Resultado: false

[Done] exited with code=0 in 0.17 seconds
```