Disciplina: Linguagem Web – Linguagem JavaScript
Professora: Sandra Hedler de Amorim

Em relação aos métodos de manipulação de string, os de manipulação de números são bem menos. São 4 os principais métodos de manipulação de números.

Primerio a função **toString()**

Exemplo no arquivo programa1.html

```
<!DOCTYPE html>

<html>

        <head>

                <meta charset="UTF-8"/>

                <title> Curso JavaScript </title>

        </head>

<body>

                <h1> Métodos de Manipulando de Números </h1>

                <script src="script1.js"></script>

</body>

</html>
```

Arquivo script1.js

```
let n = 10;

let res = n;

console.log(res);
```

Observem o resultado no console, o valor da variável vai estar na cor azul - digo o valor 10, indicando que é um número de verdade.

Se mudarmos as linhas de códigos no *script1.js* conforme a seguir:

```
let n = 10;

let res = n.toString();

console.log(res);
```

Observem que o resultado no console mostrará o valor 10 da variável em preto. O que isso significa? Que agora esse valor é uma string, não mais um número. Essa função toString, ela existe em todas as variáveis, objetos, arrays, mas ela é mais útil se tratando de números.

Segunda função **toFixer()**

A função toFixed, permite arredondar para mais ou para menos o número de casas decimais de um determinado valor.

Exemplo no arquivo programa2.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Manipulando de Números </h1>
        <script src="script2.js"></script>
</body>
</html>
```

Arquivo script2.js

```
let n = 10.655632566;
let res = n.toFixed(3);
console.log(res);
```

Ou

```
let n = 10.6545678;
let res = n.toFixed(2);
console.log(res);
```

Exemplo se usa para resultados de valores em reais

```
let n = 10.65532566
let res = 'R$ '+n.toFixed(2);
console.log(res);
```

Terceira função **parseInt()**

A função parseInt, transforma uma string em um número.

Exemplo no arquivo programa3.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Manipulando de Números </h1>
        <script src="script3.js"></script>
</body>
</html>
```

Arquivo script3.js

```
let n = '20';
let res = n + 5;
console.log(res);
```

Quando for executado o programa 3 no console o resultado de saída vai ser 205, ou seja, o "+" fez uma concatenação da variável tipo string com o valor 5.

Para transformar essa string em um número aplica a função parseInt().

```
let n = '20';
let res = parseInt(n) + 5;
console.log(res);
```

Observando que a função parseInt(), a letra " i " tem que estar em maiúscula.

Essa função é importante quando se recebe um valor do usuário e quer realizar um calculo.

Quarta função **parseFloat()**

A função parseFloat, transforma uma string em um número, preservando as casas decimais se estiver.

Exemplo no arquivo programa4.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
```

```
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Manipulando de Números </h1>
        <script src="script4.js"></script>
</body>
</html>
```

Arquivo script4.js

```
let n = '20';
let res = parseFloat(n) + 5;
console.log(res);
```

Exemplo com casas decimais

```
let n = '20.56';
let res = parseFloat(n) + 5;
console.log(res);
```

Então quando se está trabalhando com valores reais, precisa usar essa função em vez de parseInt.

Métodos de Arrays

Exemplo no arquivo programa5.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Arrays </h1>
```

```
        <script src="script5.js"></script>
</body>
</html>
```

Arquivo script5.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
let res = lista;
console.log(res);
```

Primeiro método **toString().**

Esse método transforma um array em uma única string, juntando tudo. Lembrando no método de números que foi colocado que essa função se usava também em arrays.

No código a seguir mostra a aplicação dessa função no arquivo script5.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
let res = lista.toString();
console.log(res);
```

O método toString, coloca uma virgula entre cada elemento do array, transformando em um única string.

Segundo método **join()**

Quando foi explicado o método split, nos método das strings, que transformava uma string em um array. O método join, é um processo inverso do método split, pegando o array e transformar em uma string, separando os itens pelo parâmetro que for usado na função.

Exemplo no arquivo programa6.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Arrays </h1>
        <script src="script6.js"></script>
</body>
```

</html>

Arquivo script6.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

let res = lista.join('-');

console.log(res);
```

Conforme código acima, vai colocar cada item separado por um traço (ovo-farinha-fermento-chocolate).

Se substituir as linhas de códigos do script6.js por

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

let res = lista.join(',');

console.log(res);
```

O resultado vai ser ovo,farinha,fermento,chocolate.

Agora para fazer uma comparação com o método toString, observem a nova alteração no arquivo:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

let res = lista.toString();

console.log(res);
```

O resultado no console vai ser ovo,farinha,fermento,chocolate. Podem perceber que toString é um join() separado por vírgula, foi só um exemplo para perceberem a comparação entre as duas funções.

Terceiro método indexOf()

O método indexOf, também existente nos métodos de strings, mas com a forma de funcionamento diferença.

O método permite procurar um item específico no array e vai retornar a posição onde está este item e se não encontrar retorna -1.

Exemplo no arquivo programa7.html

```
<!DOCTYPE html>

<html>

    <head>

        <meta charset="UTF-8"/>

        <title> Curso JavaScript </title>

    </head>
```

```
<body>
        <h1> Métodos de Arrays </h1>
        <script src="script7.js"></script>
</body>
</html>
```

Arquivo script7.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
let res = lista.indexOf('fermento');
console.log(res);
```

No console vai aparecer valor 2.

Outro exemplo:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
let res = lista.indexOf('açucar');
console.log(res);
```

No console vai mostrar o valor -1, ou seja, conforme parâmetro, não existe esse item no array lista.

Quarto método **pop()**

O método pop(), remove o último item do array.

Exemplo no arquivo programa8.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Arrays </h1>
        <script src="script8.js"></script>
</body>
</html>
```

Arquivo script8.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.pop();
let res = lista;
console.log(res);
```

Conforme exemplo vai remover o item chocolate.

Quinto método **shift()**

O método shift(), remove o primeiro item do array.

Exemplo no arquivo programa9.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos de Arrays </h1>
		<script src="script9.js"></script>
</body>
</html>
```

Arquivo script9.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.shift();
let res = lista;
console.log(res);
```

Conforme exemplo abaixo vai remover o item ovo.

Exemplo usando as duas funções pop e shift juntos:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.shift();
```

```
lista.pop();
let res = lista;
console.log(res);
```

Sexto método **push()**

A função adiciona um novo item no array. É a mais usada em arrays.

Exemplo no arquivo programa10.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos de Arrays </h1>
		<script src="script10.js"></script>
</body>
</html>
```

Arquivo script10.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.push('manteiga');
let res = lista;
console.log(res);
```

Uma outra forma de fazer alterações em um array, é utilizando o próprio índice do item.

Exemplo, alterar o primeiro item no array lista.

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista[0] = 'ovos';
let res = lista;
console.log(res);
```

Assim, o elemento (item) na primeira posição do array lista foi alterado para ovos.

Observação: Se colocar um índice que não existe no array, dessa forma vai criar esse índice, acrescentando no array.

Exemplo inserir índice 4 no array lista

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

lista[4] = 'manteiga';

let res = lista;

console.log(res);
```

Observem, se não sabemos o número de itens no array, pode ser adicionado dessa forma a seguir:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

lista[ lista.length ] = 'manteiga';

let res = lista;

console.log(res);
```

Sétimo método **splice()**

Remove um item do array. Nesse método existe 2 parâmetros. O primeiro qual é o índice que se quer remover e o segundo parâmetro quantos itens se quer remover.

Exemplo no arquivo programa11.html

```
<!DOCTYPE html>

<html>

	<head>

		<meta charset="UTF-8"/>

		<title> Curso JavaScript </title>

	</head>

<body>

		<h1> Métodos de Arrays </h1>

		<script src="script11.js"></script>

</body>

</html>
```

Arquivo script11.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
```

```
lista.splice(1,1);

let res = lista;

console.log(res);
```

Conforme resultado visto no console remove o item farinha nesse exemplo.

Outro exemplo, para remover dois itens do array, a partir do índice 1:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

lista.splice(1,2);

let res = lista;

console.log(res);
```

Conforme exemplo acima, remove os itens farinha e fermento. Observem que a remoção desses 2 itens foi a partir do índice 1.

Outro exemplo referente a função splice(). Se colocar só um parâmetro conforme exemplo abaixo, vai remover todos os elementos a partir do índice 1, ficando somente o elemento ovo do array lista.

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];

lista.splice(1);

let res = lista;

console.log(res);
```

Oitavo método **concat()**

Vai concatenar 2 ou mais arrays.

Exemplo no arquivo programa12.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Arrays </h1>
        <script src="script12.js"></script>
</body>
```

```
</html>
```

Arquivo script12.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
let lista2 = ['forma', 'batedeira', 'liquidificador', 'fogão'];
let res = lista.concat(lista2);
console.log(res);
```

Nono método **sort()**

O método sort() ordena os itens por ordem alfabética.

Exemplo no arquivo programa13.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos de Arrays </h1>
		<script src="script13.js"></script>
</body>
</html>
```

Arquivo script13.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.sort();
let res = lista;
console.log(res);
```

Décimo método **reverse()**

O método reverse() ordena os itens por ordem alfabética decrescente.

Exemplo no arquivo programa14.html

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos de Arrays </h1>
        <script src="script14.js"></script>
</body>
</html>
```

Arquivo script14.js

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.sort();
lista.reverse();
let res = lista;
console.log(res);
```

Ou seja, primeiro ordena com o método sort(0 e depois inverte essa ordem com reverse().

Se usar o método reverse() sem usar o método sort(), só vai inverte da forma que está no array.

Exemplo:

```
let lista = ['ovo', 'farinha', 'fermento', 'chocolate'];
lista.reverse();
let res = lista;
console.log(res);
```

A partir de agora vai ser visto métodos mais avançados de array.

Método **map()**

Essa função map() mapeia, percorre o array, e executa uma função em cada item do array. Mapeia e gera um outro array.

Exemplo no arquivo programa15.html

```
<!DOCTYPE html>
<html>
```

```
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos Avançados de Arrays </h1>
		<script src="script15.js"></script>
</body>
</html>
```

Arquivo script15.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.map(function(item){
		return item * 2;
		});
let res = lista2;
console.log(res);
```

Outra forma de poder fazer o mesmo exemplo:

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
for(let i in lista){
	lista2.push(lista[i]*2);
}
let res = lista2;
console.log(res);
```

Nesse exemplo foi preciso usar um for.

Método **filter()**

Esse método vai rodar uma função e essa vai retornar true ou false. Se retornar false não aporoveita o item do array.

Exemplo no arquivo programa16.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos Avançados de Arrays </h1>
		<script src="script16.js"></script>
</body>
</html>
```

Arquivo script16.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.filter(function(item){
		if(item < 20){
			return true;
		}else{
			return false;
		}
	});
let res = lista2;
console.log(res);
```

Nesse exemplo acima gerou um novo array com apenas os números menores que 20.

A função filter() é muito útil quando se quer fazer uma filtragem referente uma postagem em um sistema, por exemplo, comentários deixados pelos usuários até determinada data.

Método **every()**

Esse método vai rodar uma função e essa vai retornar true ou false, só não vai gerar um novo array. Essa função vai retornar se todos os itens estão de acordo com uma determinada condição, sim ou não. Ou seja, retorna true ou false.

Exemplo no arquivo programa17.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos Avançados de Arrays </h1>
		<script src="script17.js"></script>
</body>
</html>
```

Arquivo script17.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.every(function(item){
		if(item > 20){
			return true;
		}else{
			return false;
		}
	});
let res = lista2;
console.log(res);
```

Nesse exemplo retorna false, ou seja, não são todos os itens maiores que 20.

Método **some()**

Esse método vai rodar uma função e essa vai retornar true ou false. Se pelo menos um item estiver de acordo com a condição vai retorna true.

Exemplo no arquivo programa17.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos Avançados de Arrays </h1>
		<script src="script17.js"></script>
</body>
</html>
```

Arquivo script17.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.some(function(item){
		if(item > 20){
			return true;
		}else{
			return false;
		}
	});
let res = lista2;
console.log(res);
```

Nesse exemplo retorna true, ou seja, existe 2 itens maiores que 20 no array lista. Lembrando, basta existir um elemento de acordo com a condição para retornar true.

Outra forma de simplifica esse mesmo exemplo:

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.some(function(item){
		return (item > 20) ? true : false;
```

```
    });
let res = lista2;
console.log(res);
```

Método **find()**

Esse método vai rodar uma função e retornar o primeiro item que satisfaça uma condição. Se achar o item conforme condição retorna true, caso contrário false.

Exemplo no arquivo programa18.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title> Curso JavaScript </title>
    </head>
<body>
        <h1> Métodos Avançados de Arrays </h1>
        <script src="script18.js"></script>
</body>
</html>
```

Arquivo script18.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.find(function(item){
        return (item == 8)? true : false;
    });
let res = lista2;
console.log(res);
```

Conforme exemplo vai retornar o número 8, pois existe no array lista.

Método **findIndex()**

Esse método vai rodar uma função e retornar o índice do primeiro item que satisfaça uma condição. Ou seja, em vez de retornar o item (elemento), retorna a posição.

Exemplo no arquivo programa19.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8"/>
		<title> Curso JavaScript </title>
	</head>
<body>
		<h1> Métodos Avançados de Arrays </h1>
		<script src="script19.js"></script>
</body>
</html>
```

Arquivo script19.js

```
let lista = [10, 42, 8, 6, 36];
let lista2 = [];
lista2 = lista.findIndex(function(item){
		return (item == 8)? true : false;
	});
let res = lista2;
console.log(res);
```

Conforme exemplo vai retornar 2, pois o item 8 está na posição 2. Lembrando que o índice de um array começa pela posição zero.

Observem o exemplo a seguir, tem um array chamado lista e dentro tem objetos.

```
let lista = [
	{id: 1, nome:'Sandra', sobrenome:'Amorim'},
	{id:2, nome:'Maria', sobrenome:'Costa'},
	{id:3, nome:'Valquiria', sobrenome:'Moraes'}
];
let pessoa = lista.find(function(item){
		return (item.sobrenome == 'Costa')? true : false;
```

```
    });
let res = pessoa;
console.log(res);
```

Com uso da função find() foi feita um consulta dentro desse array pela condição do sobrenome. Poderia ser pelo id ou ainda pelo nome.

O exemplo a seguir, busca pelo id = 3 do array lista.

```
let lista = [
    {id: 1, nome:'Sandra', sobrenome:'Amorim'},
    {id:2, nome:'Maria', sobrenome:'Costa'},
    {id:3, nome:'Valquiria', sobrenome:'Moraes'}
];
let pessoa = lista.find(function(item){
        return (item.id == 3)? true : false;
    });
let res = pessoa;
console.log(res);
```

Objeto Date

Esse objeto do javascript existe várias funções, formas de trabalhar em relação a data, ou seja, dia, mês, ano, hora, minutos, segundos e milésimos.

Exemplos

O exemplo a seguir vai mostrar informação completa do dia da semana, mês, dia, hora, minutos, segundos, fuso horário.

```
let d= new Date(); // aqui foi criado o objeto da classe Date(), pega a data do
        // dispositivo que está acessando
console.log(d);
```

O exemplo a seguir vai mostra as informações um pouco mais resumida.

```
let d = new Date();
console.log(d.toDateString());
```

Similar a esse exemplo anterior, pode ser feito com a função toString().

```
console.log(d.toString());
```

O próximo exemplo vai mostrar a hora em GMT, por exemplo, na Europa, vai tirar o fuso horário. Quem faz isso é a função toUTCString().

```
let d = new Date();

console.log(d.toUTCString());
```

A classe Date permite colocar até 7 parâmetros ou um somente.

No exemplo a seguir, na função Date() está com 6 parâmetros. O primeiro é o ano, segundo o mês, e observem que o mês começa com zero (0), que se refere a janeiro, e assim por diante até onze (11) que se refere a dezembro. Terceiro parâmetro dia, quarto hora, quinto minutos e sexto segundos.

```
let d = new Date(2020,0,1, 12, 30, 12); //obs os parâmetros

console.log(d.toString());
```

Mas existem outras formas de gerar a mesma data através do que chamam da datestring, observem:

```
let d = new Date('2020-01-01 12:30:12'); //aqui janeiro é 01

console.log(d.toString());
```

Outro exemplo com ano e mês, que é o mínimo para gerar uma data:

```
let d = new Date(2021, 2); //aqui o número 2 significa o mês de março.

console.log(d.toString());
```

Observem o exemplo a seguir que Date tem só o parâmetro zero.

```
let d = new Date(0);

console.log(d.toString());
```

Nesse exemplo foi para data 31 dezembro de 1969.

No exemplo a seguir com a função toUTCString() ou seja em GMT a data vai para 01 de janeiro de 1970. Por quê?

Por que a data no javascript começa a contar os milissegundos a partir de 01 de janeiro de 1970. é um padrão da linguagem javascript e também da linguagem PHP.

```
let d = new Date(0);

console.log(d.toUTCString());
```

No próximo exemplo veja como aplicar na função o parâmetro com milissegundos:

```
let d = new Date(1234545667); //números quaisquer

console.log(d.toUTCString());
```

Agora, como colocar datas anteriores a 1970? Para datas anteriores a 1970 basta colocar em negativo os milissegundos. Veja exemplo:

```
let d = new Date(-1234545667);

console.log(d.toUTCString());
```

São raras as vezes que o programador vai usar a data em milissegundos, mas é bom saberem como proceder se for o caso.

No exemplo a seguir vai retornar somente o ano conforme essa data.

```
let d = new Date();

let novoValor = d.getFullYear(); //Obs. que o F e Y em maiúsculo

console.log(novoValor);
```

Como pegar o mês atual ? Função getMonth()

```
let d = new Date();

let novoValor = d.getMonth(); //Obs que vai retornar em numeral e começa o mês
                              //por zero não esquecer.

console.log(novoValor);
```

Como pegar o dia da semana? Função getDay()

```
let d = new Date();

let novoValor = d.getDay(); //Retorna também de 0 a 6.  Não esquecer começa no domingo que é zero e assim por diante

console.log(novoValor);
```

Como pegar o dia atual ? Função getDate()

```
let d = new Date();

let novoValor = d.getDate();

console.log(novoValor);
```

Como pegar a hora ? Função getHours()

```
let d = new Date();

let novoValor = d.getHours();

console.log(novoValor);
```

Como pegar os minutos ? Função getMinutes()

```
let d = new Date();
let novoValor = d.getMinutes();
console.log(novoValor);
```

Como pegar os segundos ? Função getSeconds()

```
let d = new Date();
let novoValor = d.getSeconds();
console.log(novoValor);
```

Como pegar os milissegundos ? Função getMilliseconds()

```
let d = new Date();
let novoValor = d.getMilliseconds();
console.log(novoValor);
```

Função getTime()

Vai pegar os milissegundos de 01 de janeiro de 1970 até o momento atual

```
let d = new Date();
let novoValor = d.getTime();
console.log(novoValor);
```

Date.now()

Define milissegundo do momento atual, sem precisar definir uma variável, objeto.

```
//let d = new Date();
let novoValor = Date.now();
console.log(novoValor);
```

Alterando informações sobre datas

Função setFullYear()

Altera o ano, mas pode ser colocado como parâmetro o mês e o dia. Como o próprio nome já se refere a ano, o ideal é usar somente para alterar o ano.

```
let d = new Date();

d.setFullYear(2022);

let novoValor = d;

console.log(novoValor);
```

Como trocar o mês? Função setMonth()

```
let d = new Date();

d.setMonth(5); // lembrando que os meses vão de 0 a 11.

let novoValor = d;

console.log(novoValor);
```

Como alterar o dia ? Função setDate()

```
let d = new Date();

d.setDate(20);

let novoValor = d;

console.log(novoValor);
```

Como aumentar X dias pra frente da data atual? Função setDate() e embutida nessa mesma **função getDate**. Exemplo a seguir aumenta 5 dias em relação a data atual.

```
let d = new Date();

d.setDate( d.getDate() + 5 );

let novoValor = d;

console.log(novoValor);
```

Se quiser saber em que dia da semana vai ser essa alteração:

```
let d = new Date();

d.setDate( d.getDate() + 5 );

let novoValor = d.getDay(); //Lembrando que o dia vai de 0 a 6

console.log(novoValor);
```

Se quiser alterar para uns 90 dias a partir da data atual?

```
let d = new Date();

d.setDate( d.getDate() + 90 );

let novoValor = d;

console.log(novoValor);
```

Só alterar para 90, pois o próprio sistema já atualiza o mês, ano sem problemas.

E se quiser fazer alterações em horas, minutos e segundos?

```
let d = new Date();

d.setHours( d.getHours() + 24 );

let novoValor = d;

console.log(novoValor);
```

Obs.: Todos esses exemplos podem ser feitos também para diminuição de horas, dias, somente colocando o sinal de subtração.

Exemplo:

```
let d = new Date();

d.setHours( d.getHours() - 12 );

let novoValor = d;

console.log(novoValor);
```