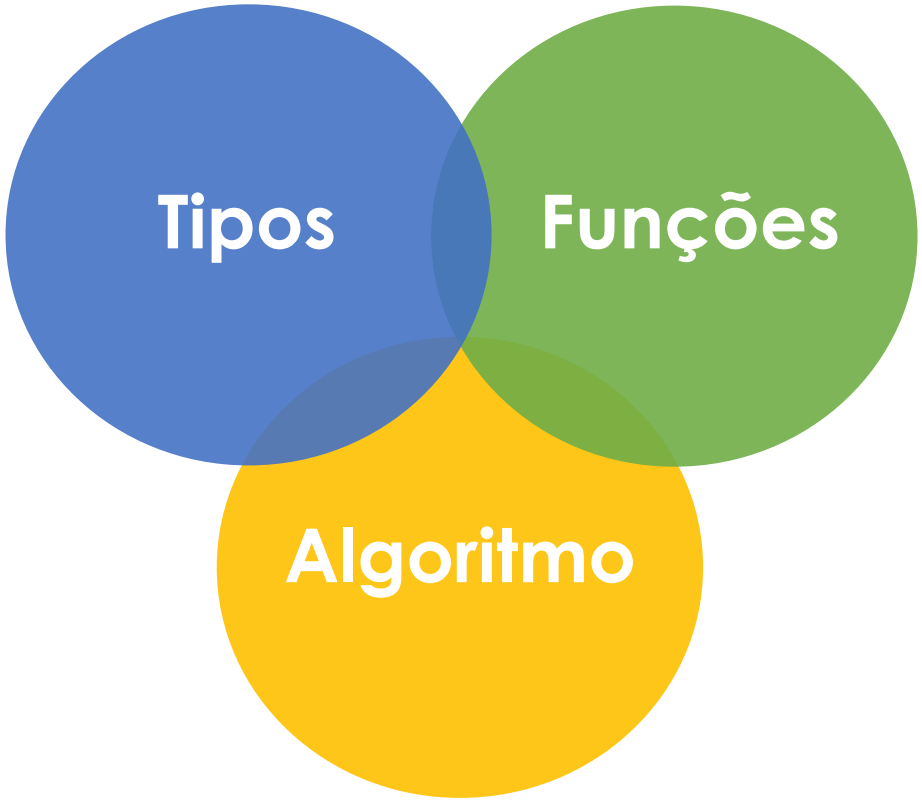
Nossa Senhora de Fátima

10.2 | 5 | 'A'

"Oie" | true

01/12/2020

{ 1, 2, 3, 4 }



+ | - | * | / | % | > | < | >= | <=

[ ] | ? : | | | | && | ! | ==

Math.Pow | Replace

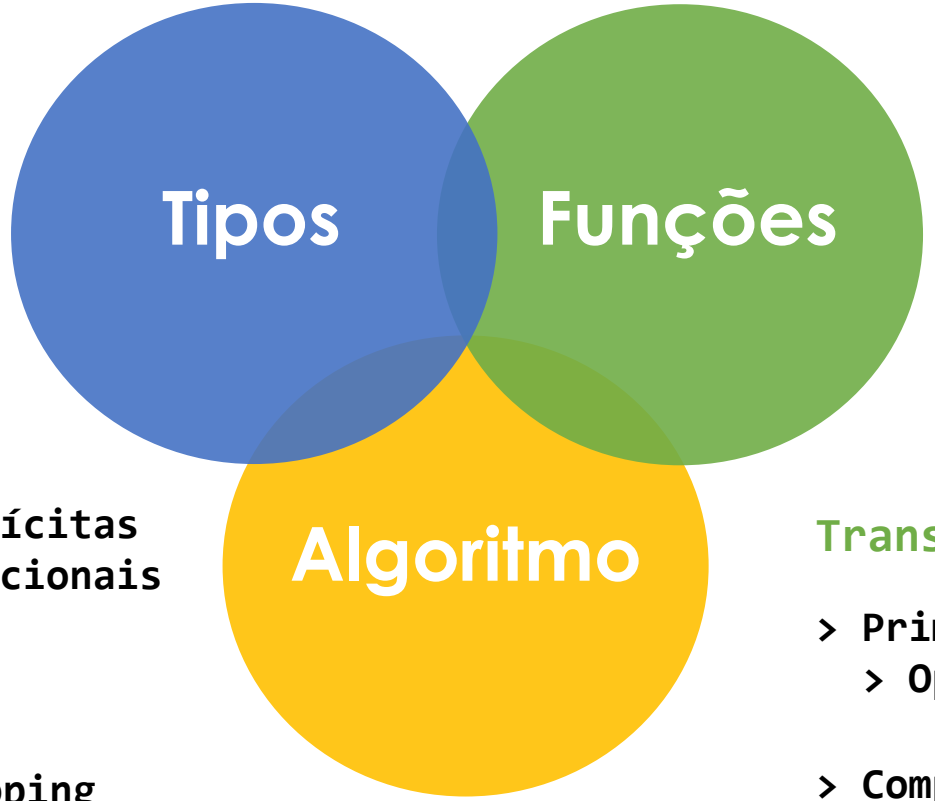
Where | AddDays

15

6

true

true



Tipagem
> Fraca vs Forte
> Estática vs Dinâmica
> Valor vs Referência

Expressões
> Ordem
> Precedência
> Sequencialidade

Transição
> Operativa
> Aninhado
> Fluente

Abstrações
> Variável
> Funções
  > Nomeadas
  > Anônimas
  > Recursivas
  > HighOrder
> Objetos
  > Modelo
  > Serviço
  > Mixed

Valores
> Primitivos
> Compostos
  > Builtin
  > Recursivos
  > User Defined

Relacionamento
> Associação
> Herança
> Dependência
> Agregação
> Composição

Escopo
> Blocos
> Ambiente

Conversões
> Implícitas vs Explícitas
> Categóricas vs Funcionais

Rotas
> Única
> Condicional
  > Simples
  > Multidirecional
  > Opcional
  > Interrupção
  > Aninhada
> Looping
  > Simples
  > Condicional
  > Interrupção
  > Pulo
  > Aninhado

Estruturas
> Seleção
> Repetição

Transformações
> Primitivas
  > Operadores
> Compostas
  > Builtin
  > Community
  > User Defined

Aridade
> Unário
> Binário
> Ternário
> N

Notação
> Prefix
> Postfix
> Infix

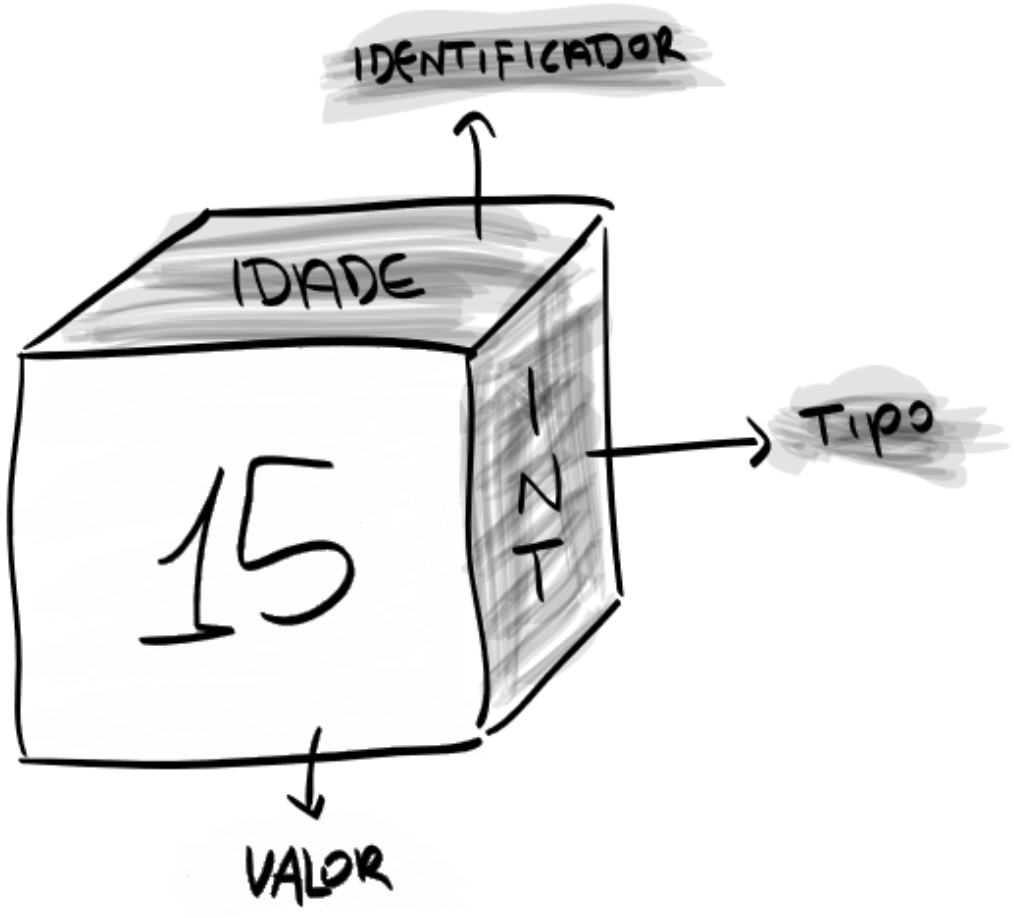
Associatividade
> Esquerda -> Direita
> Direita -> Esquerda

Valores | Expressar ideias

```
> 15
< 15

> 73.5
< 73.5

> -1000
< -1000

> "Oie fofuras s2"
< "Oie fofuras s2"

> 'Oie fofuras s2’
< "Oie fofuras s2"

> 'A'
< 'A'

> "A"
< "A"
```

```
> true
< true

> false
< false

> new Date()
< Mon May 22 2020 08:00:00 GMT-0300

> new Date(2020, 11, 25)
< Mon Dec 25 2020 00:00:00 GMT-0300

> [ 4, 6, 10 ]
< (3) [4, 6, 10]

> [ "Luiza", "Ingrid", "Junior" ]
< (3) ["Luiza", "Ingrid", "Junior"]
```

Variável | Diagrama




```
let idade = 15;
```

- `let` — Criar nova variável
- `idade` — Nome
- `=` — Operador Atribuição
- `15` — Valor (tipo)
- `;` — Fim

Nomeação | Expressar ideias

```
> let idade = 15;
> idade
< 15

> let peso = 73.5;
> peso
< 73.5

> let saldoBancario = -1000;
> saldoBancario
< -1000

> let mentira = "Oie fofuras s2";
> mentira
< "Oie fofuras s2"

> let inicialCrush = 'A';
> inicialCrush
< 'A'

> let profLindo = true;
> profLindo
< true
```

```
> let profCareca = false;
> profCareca
< false

> let diaPraDarMeuMehor = new Date();
> diaPraDarMeuMelhor
< Mon May 22 2020 08:00:00 GMT-0300

> let melhorNiver = new Date(2020, 11, 25);
> melhorNiver
< Mon Dec 25 2020 00:00:00 GMT-0300

> let notas = [ 4, 6, 10 ];
> notas
< (3) [4, 6, 10]

> let familia = ["Luiza", "Ingrid", "Junior"];
> familia
< (3) ["Luiza", "Ingrid", "Junior"]
```

```
> let x = 10;
> x
< 10

> x = 15;
> x
< 15

> x = "Joga dez!";
> x
< "Joga dez!"

> x = true;
> x
< true

> x = ["Eu", "não", "minto"];
> x
< ["Eu", "não", "minto"]
```

Valores

boolean

```
let b1 = true;
let b2 = false;
```

string

```
let t1 = "Oie";
let t2 = 'O';
let t3 = 'Oie';
let t4 = t3[0];
```

number

```
let n1 = 10;
let n2 = -10;
let n3 = 10.5;
let n4 = 2000000000000;
```

object

```
let o1 = new Date();
let o2 = [ 1, 2, 3 ];
let o3 = {
  nome: "Bruno",
  idade: 15
};
```

function

```
let f1 = function(a,b) {
   return a + b;
}

let f2 = (a,b) => a + b;
```

null e undefined

```
let x1 = null;
let x2 = undefined;
```

Tipagem Implícita

- number
- string
- boolean
- object
- function
- undefined
- null

Tipagem Fraca

var Não garante escopo

let Garante escopo

const Valor constante

Coleções

mix

```
let l1 = [];
let l2 = [ 1, 1.5, "js", true, { nome: "Mon" } ];
```

number

```
let l1 = [];
let l2 = [1, 2, 3];
```

string

```
let l1 = [];
let l2 = ["c#", "js"];
```

boolean

```
let l1 = [];
let l2 = [true, false];
```

object

```
let l1 = [];
let l2 = [
    { nome: "Chandler" },
    { nome: "Ross" }
];
```

Composição

Modelo simples

```
let modelo = {
     nome: "Bruno",
      job: "Dev"
};
```

Modelo + Coleção Primitiva

```
let modelo = {
     nome: "Bruno",
      job: "Dev",
     ling: [ "C#", "JS" ]
};
```

Modelo composto

```
let modelo = {
  pessoa: {
        nome: "Bruno",
        idade: 15
  },
  job: {
        nome: "Dev",
        Salario: 10000
  }
};
```

Modelo + Coleção Objeto

```
let modelo = {
     nome: "Bruno",
      job: "Dev",
     ling: [
          { nome: "C#" },
          { nome: "JS" }
     ]
};
```

Alteração | Modelos

```
> let m1 = {
    nome: "Bruno",
     job: "Dev"
  };

> m1.nome = "Phoebe";
> m1.job = "Música";
> m1
< {nome: "Phoebe", job: "Música"}
```

```
> let m2 = {
    nome: "Bruno",
     job: "Dev",
    ling: [ "C#", "JS" ]
  };

> m2.ling = [ "Python", "Ruby" ];
> m2.ling[0] = "Java";
> m2.ling[1] = "C++";
```

```
> let m3 = {
    nome: "Bruno",
     job: "Dev",
    ling: [
        { nome: "C#" },
        { nome: "JS" }
    ]
  };

> m3.nome = "Bill";
> m3.job = "Eng.Software";
> m3.ling[0].nome = "Haskell";
> m3.ling[1].nome = "R";

> let a = m3.nome;
> let b = m3.ling;
> let c = m3.ling[0];
> let d = m3.ling[0].nome;
```

Valor | Atribuição Por..

```
> let a = 10;
> let b = 10;
>
> a == b
< true

> b = 15;

> a == b
< false
```

```
> let a = 10;
> let b = a;
>
> a == b
< true

> a = 15;

> a == b
< false
```

Valor | Atribuição Por..

```
> let a = "oie";
> let b = "oie";
>
> a == b
< true

> b = "xau";

> a == b
< false
```

```
> let a = "oie";
> let b = a;
>
> a == b
< true

> a = "xau";

> a == b
< false
```

Referência | Atribuição Por..

```
> let pessoa1 = {
    nome: "João",
    idade: 15,
  };

> let pessoa2 = pessoa1;

> pessoa1 == pessoa2
< true

> pessoa1.nome == pessoa2.nome
< true


> pessoa1.nome = "Maria";

> pessoa1 == pessoa2
< true

> pessoa1.nome == pessoa2.nome
< true

> pessoa1
< { nome: "Maria", idade: 15 }

> pessoa2
< { nome: "Maria", idade: 15 }
```

```
> let pessoa1 = {
    nome: "João",
    idade: 15,
  };

> let pessoa2 = {
    nome: "João",
    idade: 15,
  };

> pessoa1 == pessoa2
< false

> pessoa1.nome == pessoa2.nome
< true
```

Referência | Atribuição Por..

```
> let notas1 = [ 4, 6, 8, 9 ];

> let notas2 = notas1;

> notas1 == notas2
< true

> notas1[0] = 5;

> notas1
< (4) [5, 6, 8, 9]

> notas2
< (4) [5, 6, 8, 9]
```

```
> let notas1 = [ 4, 6, 8, 9 ];

> let notas2 = [ 4, 6, 8, 9 ];

> notas1 == notas2
< false

> notas1[0] = 5;

> notas1
< (4) [5, 6, 8, 9]

> notas2
< (4) [4, 6, 8, 9]
```

Explícita | Conversão

```
> let a = "10";

> let b = Number(a);
> b
< 10

> let c = Boolean(a);
> c
< true
```

```
> let a = 10;

> let b = String(a);
> b
< "10"

> let c = Boolean(a);
> c
< true
```

```
> let a = true;

> let b = String(a);
> b
< "true"

> let c = Number(a);
> c
< 1
```

Implícita | Conversão

```
> let a = 10 + 10;
> a
< 20

> let b = "10" + "10";
> b
< "1010"

> let c = 10 + "10";
> c
< "1010"

> let d = "10" + 10;
> d
< "1010"

> let e = 10 + true;
> e
< 11

> let f = "10" + true;
> f
< "10true"

> let g = true + true;
> g
< 2
```

```
> let a = 10 - 10;
> a
< 0

> let b = "10" - "10";
> b
< 0

> let c = 10 - "10";
> c
< 0

> let d = "10" - 10;
> d
< 0

> let e = 10 - true;
> e
< 9

> let f = "10" - true;
> f
< 9

> let g = true - true;
> g
< 0
```

Limites

```
> let a = Number.MAX_VALUE;
> a
< 1.7976931348623157e+308

> let b = Number.MIN_VALUE;
> b
< 5e-324

> let c = Number.MAX_SAFE_INTEGER;
> c
< 9007199254740991

> let d = Number.MIN_SAFE_INTEGER;
> d
< -9007199254740991
```

Constantes

```
> const a = 10;
> a = 15;
< Uncaught TypeError:
  Assignment to constant variable.

> const b = "Oie";
> b = "Xau";
< Uncaught TypeError:
  Assignment to constant variable.

> const c = true;
> c = false;
< Uncaught TypeError:
  Assignment to constant variable.
```

```
> let a = 1;
  {
     let b = a + 1;
     {
        let c = a + b + 1;
     }
  }

> c
< Uncaught ReferenceError:
  c is not defined.

> b
< Uncaught ReferenceError:
  b is not defined.

> a
< 1
```

Operadores

Operador Leitura []

```
let t  = "Oiee fofos";
let x1 = t[0];
let x2 = t[t.length - 1];
```

+ - * / % ** Operadores Matemáticos

```
let x1 = 2 + 2 * 2;
let x2 = (2 + 2) * 2;
let x3 = 4 % 2;
let x4 = 4 ** 2;
```

Operadores Condicionais ? :

```
let x1 = 10 > 5
         ? "A"
         : "B";
```

> >= < <= == === != !== ! Operadores Relacionais

```
let x1 = 2 > 2;
let x2 = 2 >= 2;
let x3 = 2 >= 2 + 2;
let x4 = 2 * 2 >= 2 + 2;
let x5 = 10 == "10";
let x6 = 10 === "10";
let x7 = 10 !== "10";
let x8 = !(10 === 10);
```

Operadores Lógicos && ||

```
let x1 = 4 > 2 && 5 > 3;
let x2 = 4 > 2 && 5 > 30;
let x3 = 4 > 2 || 5 > 3;
let x4 = 4 > 2 || 5 > 30;
```

Operadores

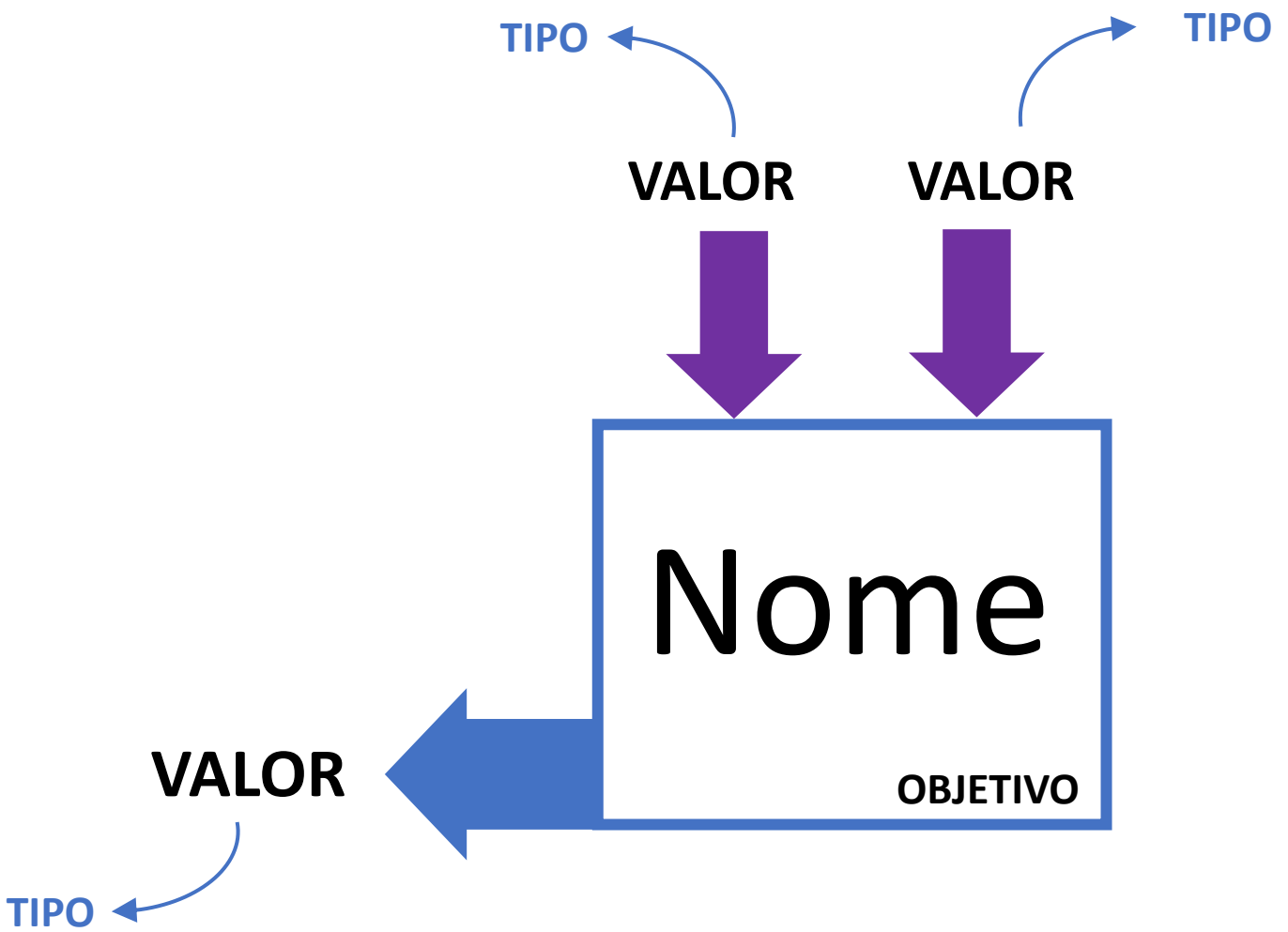
Operador Desconstrução { }

```
let computador = {
  marca: "Dell",
  ram: {
    marca: "Kingston",
    valor: 8
  }
};

let { marca } = computador;
let { marca: empresa } = computador;

let { ram: { valor } } = computador;
let { marca: empresa, ram: { marca } } = computador;
```

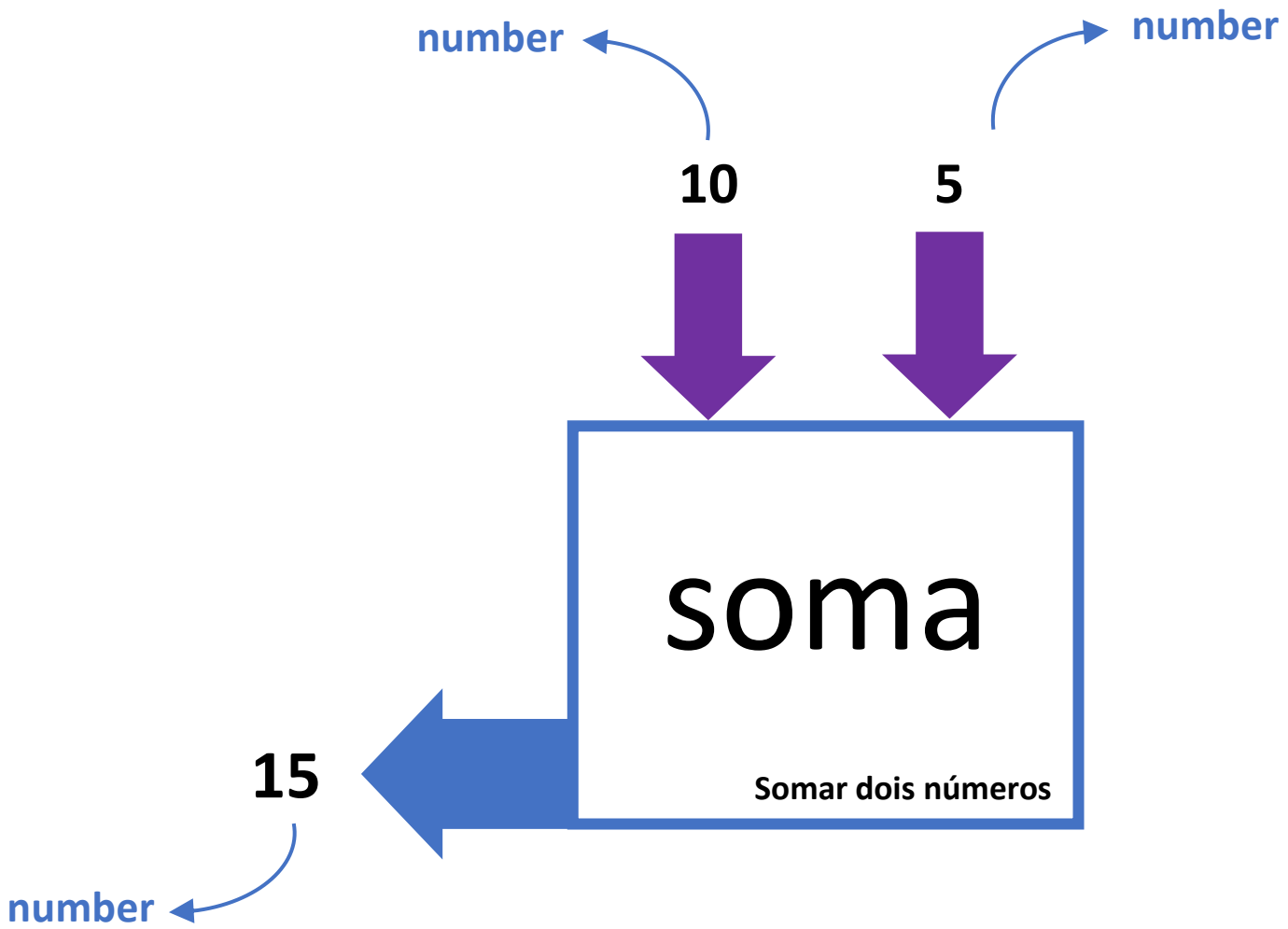
Operador Spread ...

```
let colecao = [10, 15, 20];
let x2 = [5, ...colecao];
let x3 = [5, ...colecao, 25];


function funcaoVazia(a, b, c) {

}

funcaoVazia(colecao);

funcaoVazia(...colecao);
```

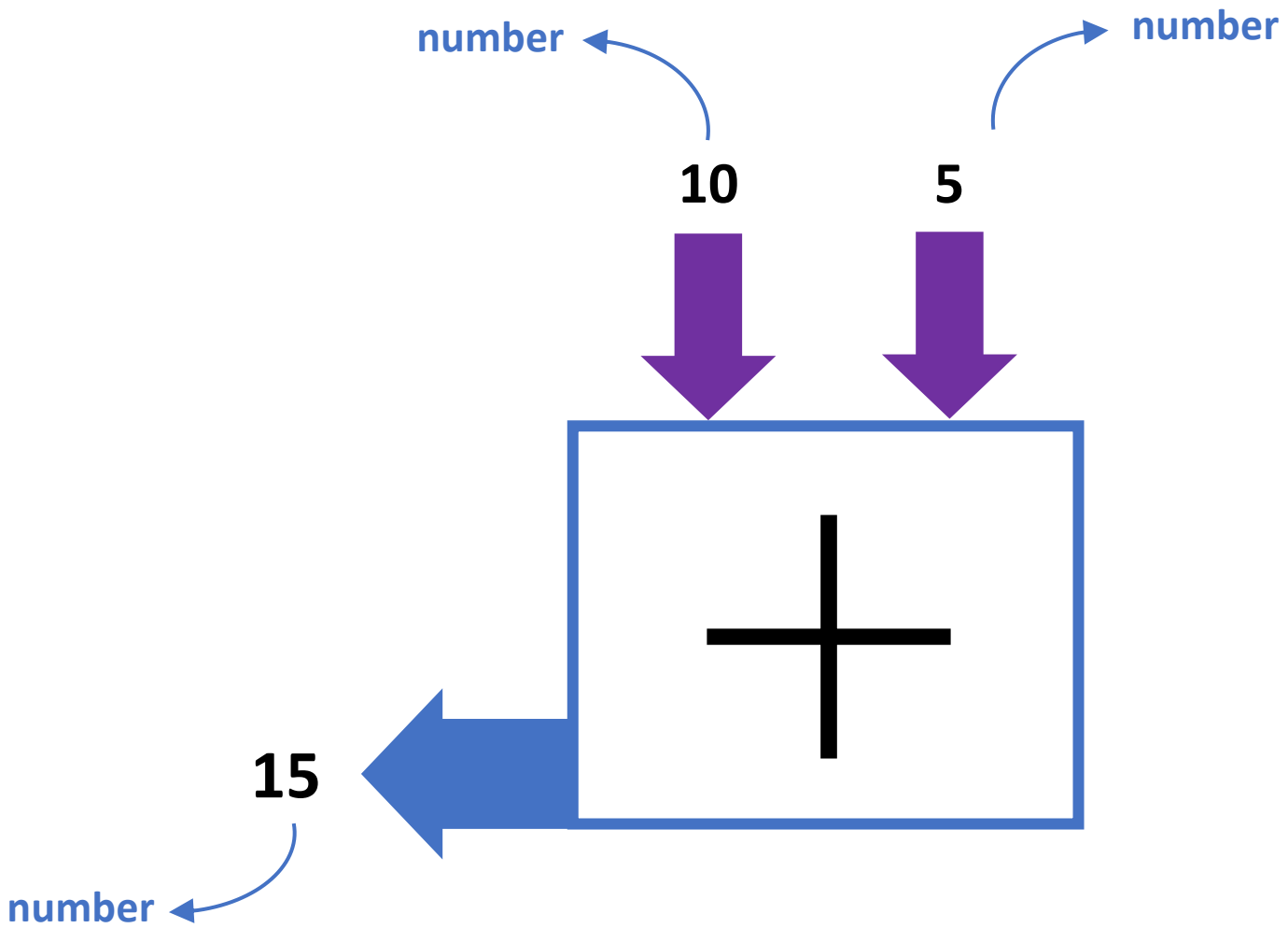
Máquina de Função



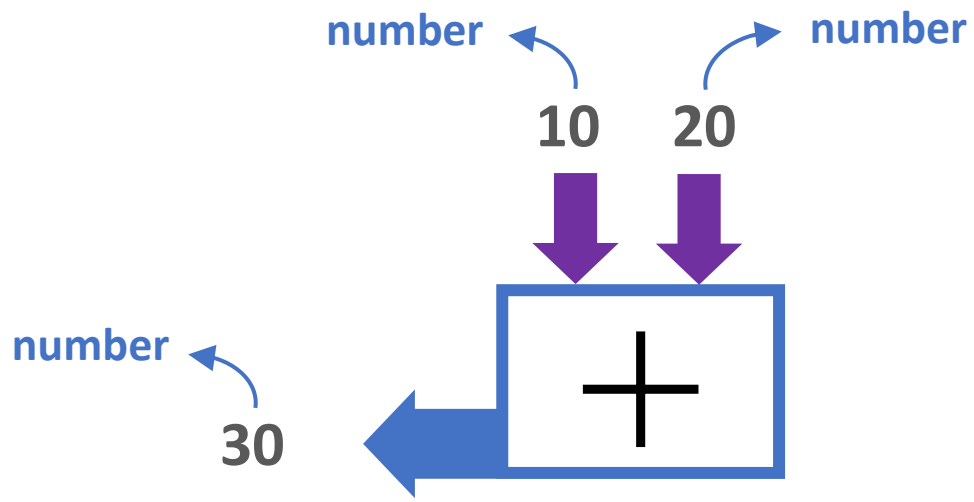
Máquina de Função

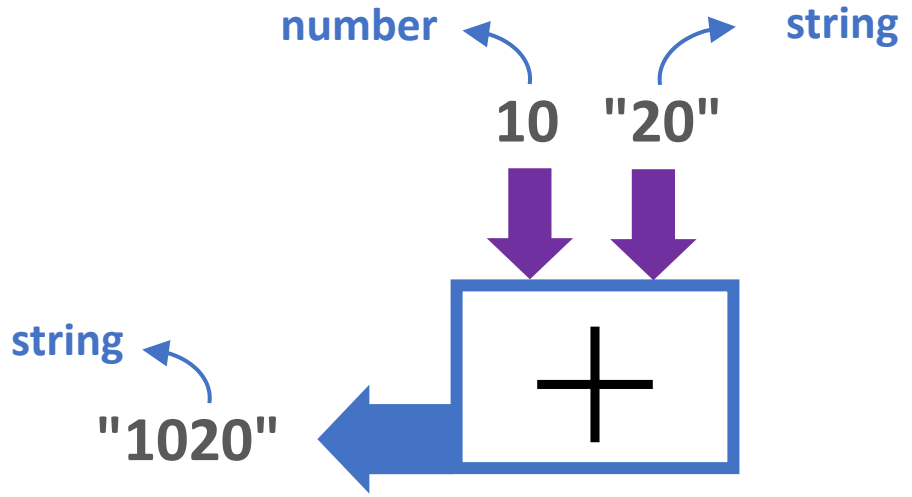


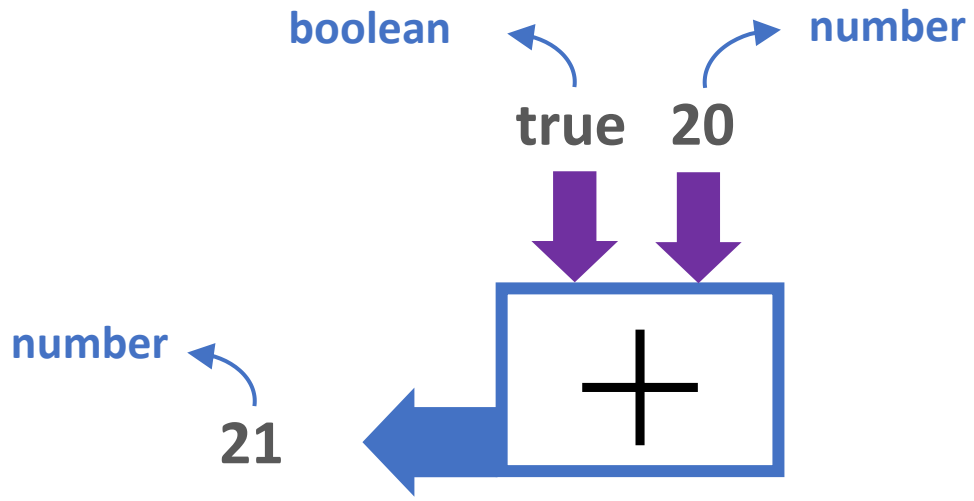
Máquina de Função

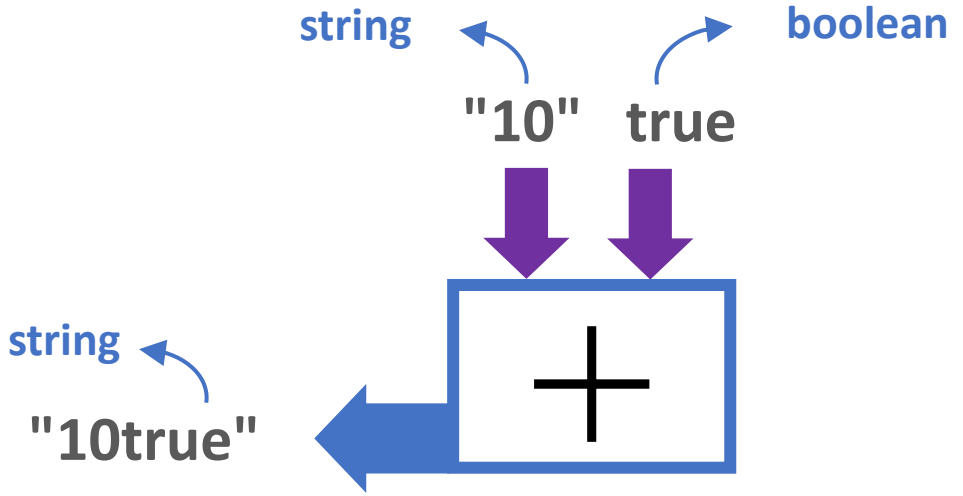


Máquina de Função

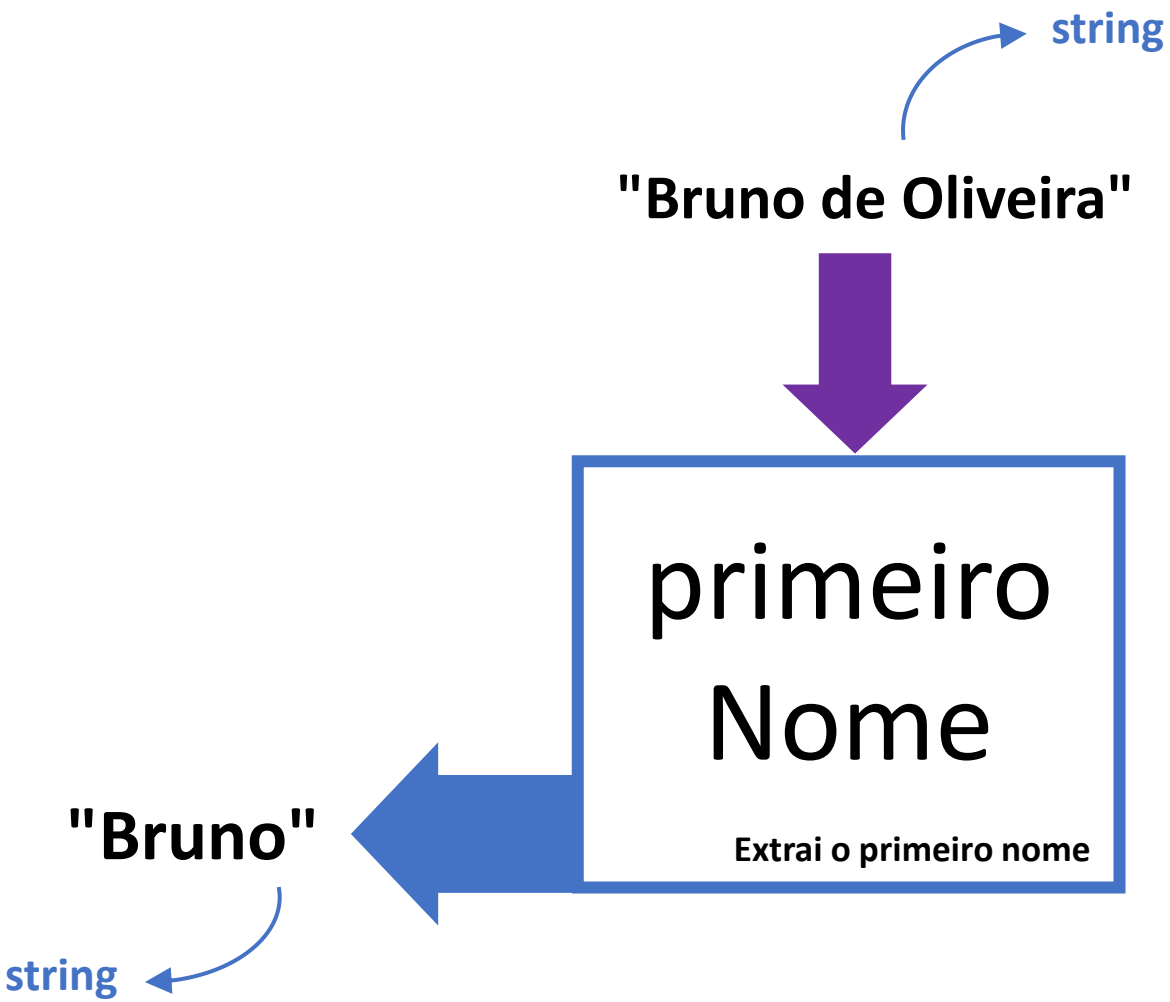






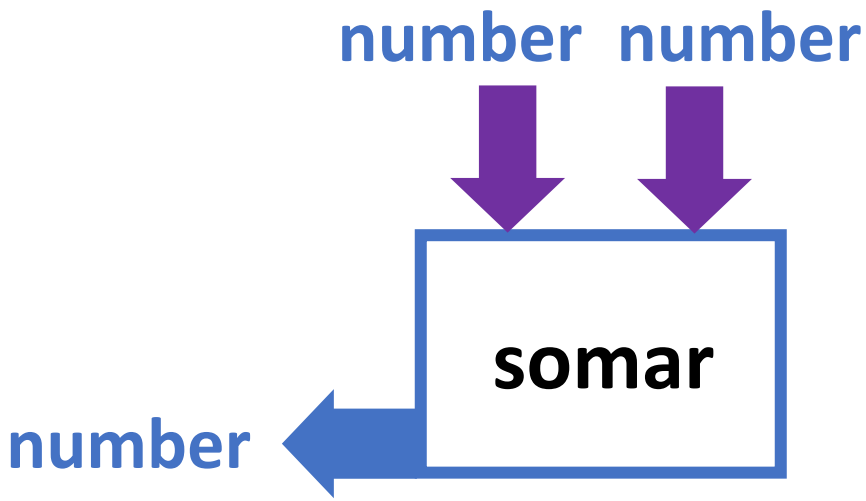


Máquina de Função



Máquina de Função

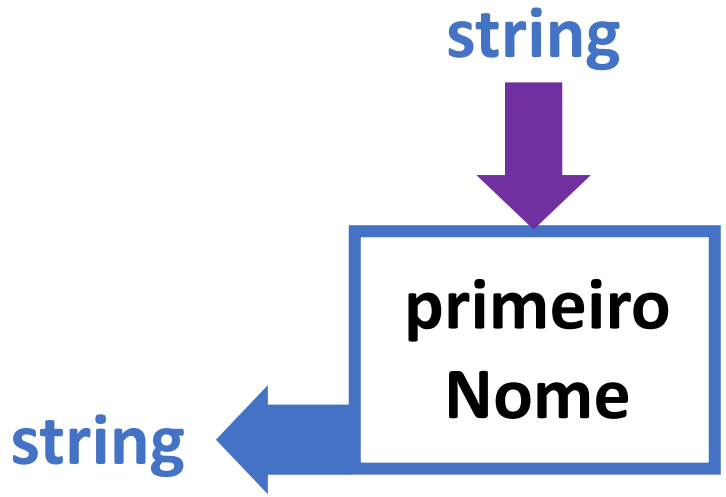
```
function somar(n1, n2) {
      let x = n1 + n2;
      return x;
}
```




```
> let n1 = 10;
> let n2 = 5;
> let x  = somar(n1, n2);
> x
< 15
```

Máquina de Função

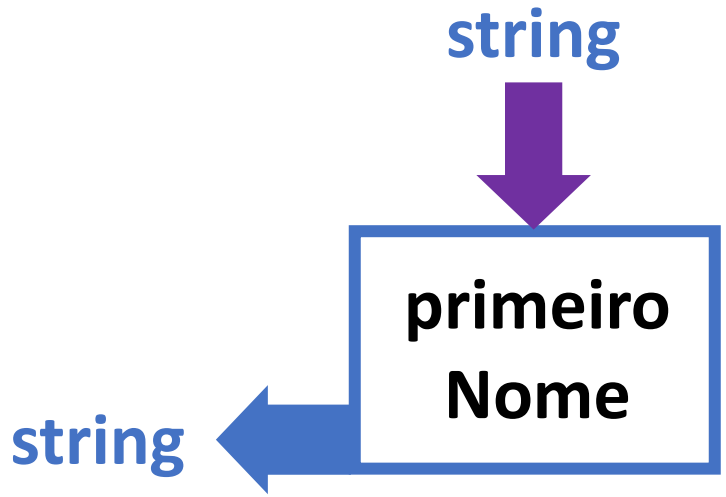
```
function primeiroNome(nome) {
   let a = nome.indexOf(" ");
   let x = nome.substr(0, a);
   return x;
}
```




```
> let nome = "Bruno de Oliveira";
> let x = primeiroNome(nome);
> x
< "Bruno"
```

Máquina de Função

```
function primeiroNome(nome) {
  let a = nome.indexOf(" ");
  console.log("a: " + a);

  let x = nome.substr(0, a);
  console.log("x: " + x);

  return x;
}
```

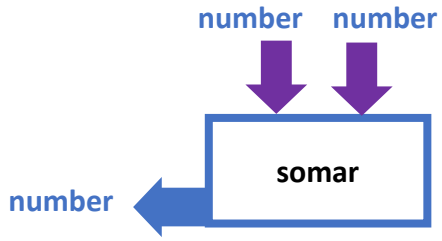



```
> let nome = "Bruno de Oliveira";
> let x = primeiroNome(nome);
  a: 5
  x: Bruno
> x
< "Bruno"
```

Exemplos

Simples

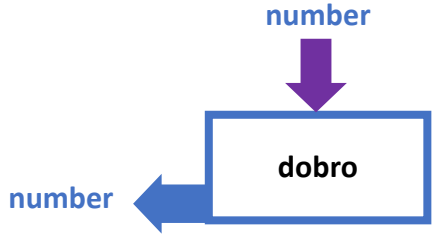
```
function soma(n1, n2) {
  let x = n1 + n2;
  return x;
}
```




```
> let n1 = 10;
> let n2 = 5;
> let x = soma(n1, n2);
> x
< 15
```

Simples

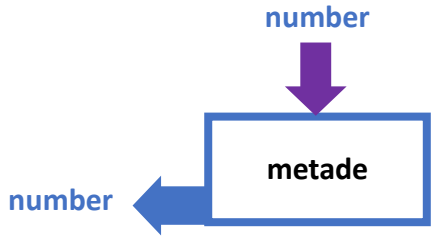
```
function dobro(n) {
  let x = n * 2;
  return x;
}
```




```
> let a = 10;
> let x = dobro(a);
> x
< 20
```

Simples

```
function metade(n) {
  let m = n / 2;
  return m;
}
```

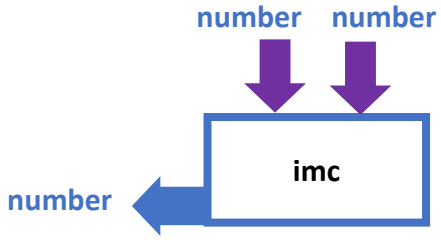



```
> let a = 10;
> let b = metade(a);
> x
< 5
```

Exemplos

Expressão

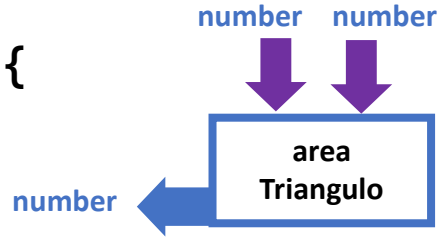
```
function imc(peso, altura) {
  let x = peso / (altura * altura);
  return x;
}
```




```
> let a = 74, b = 1.8;
> let x = imc(a, b);
> x
< 22.83
```

Expressão

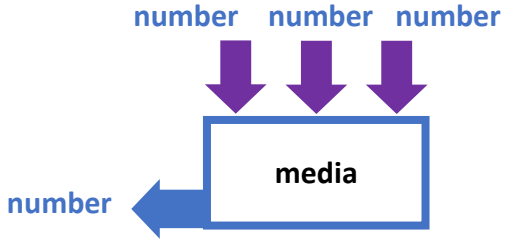
```
function areaTriangulo(base, altura) {
  let x = (base * altura) / 2;
  return x;
}
```




```
> let a = 4;
> let b = 5;
> let x = areaTriangulo(a, b);
> x
< 10
```

Expressão

```
function media(n1, n2, n3) {
  let m = (n1 + n2 + n3) / 3;
  return m;
}
```

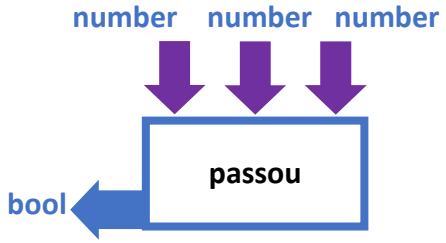



```
> let a = 6, b = 8, c = 10;
> let x = media(a,b,c);
> x
< 8
```

Exemplos

Op. Relacional

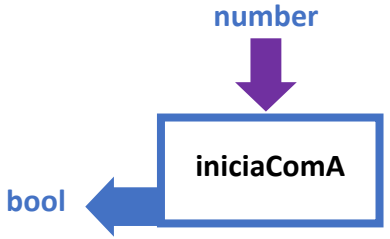
```
function passou(n1, n2, n3) {
  let m = (n1 + n2 + n3) / 3;
  let x = m >= 5;
  return x;
}
```




```
> let a = 6, b = 8, c = 10;
> let x = media(a,b,c);
> x
< true
```

Op. Relacional

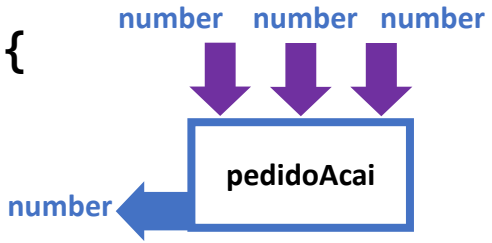
```
function iniciaComA(nome) {
  let primeiraLetra = nome[0];
  let x == 'A';
  return x;
}
```




```
> let a = "Programação";
> let x = iniciaComA(a);
> x
< false
```

Variáveis auxiliares

```
function pedidoAcai(qtd240, qtd300, qtd500) {
  let total240 = qtd240 * 10;
  let total360 = qtd360 * 12;
  let total500 = qtd500 * 14;
  let total = total240 + total360 + total500;
  return total;
}
```

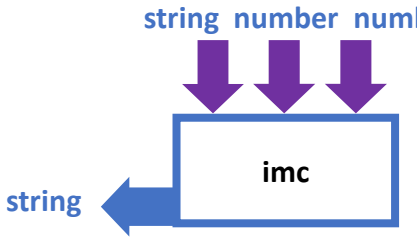



```
> let a = 2, b = 2, c = 1;
> let x = pedidoAcai(a,b,c);
> x
< 58
```

Exemplos

Concatenação

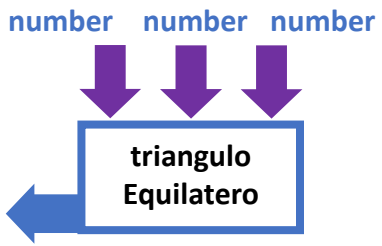
```
function imc(nome, peso, altura) {
  let r = peso / (altura * altura);
  let x = nome + ", seu imc é " + r;
  return x;
}
```




```
> let a = "Bruno";
> let b = 74, c = 1.8;
> let x = imc(a, b, c);
> x
< "Bruno, seu imc é 22.83"
```

Op. Lógico

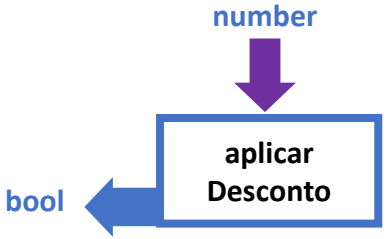
```
function trianguloEquilatero(ladoA, ladoB, ladoC) {
  let x = ladoA == ladoB && ladoA == ladoC;
  return x;
}
```




```
> let a = 5, b = 5, c = 5;
> let x =
   trianguloEquilatero(a,b,c);
> x
< true
```

Op. Ternário

```
function aplicarDesconto(total) {
  total = total > 100
            ? total * 0.9
            : total;
  return x;
}
```

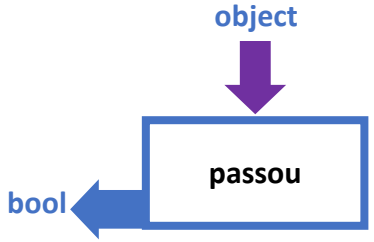



```
> let a = 200;
> let x = aplicarDesconto(a);
> x
< 180
```

Exemplos

Modelo

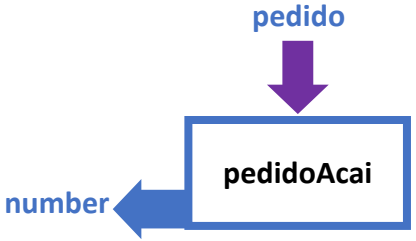
```
function passou(aluno) {
  let m = (aluno.n1 +
           aluno.n2 +
           aluno.n3) / 3;
  let x = m >= 5;
  return x;
}
```




```
> let a = {
    n1: 6,
    n2: 8,
    n3: 10
  };
> let x = passou(a);
> x
< true
```

Modelo

```
function pedidoAcai(pedido) {
  let total240 = pedido.qtd240 * 10;
  let total360 = pedido.qtd360 * 12;
  let total500 = pedido.qtd500 * 14;
  let total = total240 + total360 + total500;
  return total;
}
```




```
> let a = {
    qtd240: 2,
    qtd360: 2,
    qtd500: 1
  };
> let x = pedidoAcai(a);
> x
< 58
```

Funções Builtin

NÚMERO

```
let n = 10.58;
```

```
number Math.floor(n)
number Math.ceil(n)
number Math.round(n)

number Math.pow(n,n)
number Math.sqrt(n)

number Math.abs(n)
number Math.random()

string toPrecision(n)
string toFixed(n)
string toString()
```

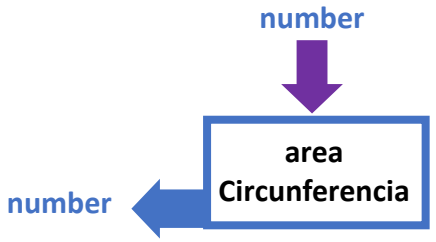
```
let x1 = Math.floor(n);
let x2 = Math.ceil(n)
let x3 = Math.round(n)

let x4 = Math.pow(2, 3)
let x6 = Math.sqrt(25)

let x5 = Math.abs(-10)
let x7 = Math.random()

let x8 = n.toPrecision(3)
let x9 = n.toFixed(5)
let x10 = toString()
```

```
= 10
= 11
= 11

= 8
= 5

= 10
= ?

= "10.580"
= "10.580"
= "10.58"
```

Exemplos

Simples

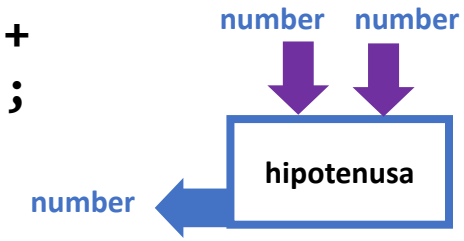
```
function areaCircunferencia(raio) {
  let area = Math.PI * Math.pow(raio, 2);

  return area;
}
```




```
> let r = 5;
> let x = areaCircunferencia(r);
> x
< 78.53
```

Simples

```
function hipotenusa(catetoAdj, catetoOpo) {
  let hip = Math.sqrt(Math.pow(catetoAdj, 2) +
                      Math.pow(catetoOpo, 2));

  return hip;
}
```

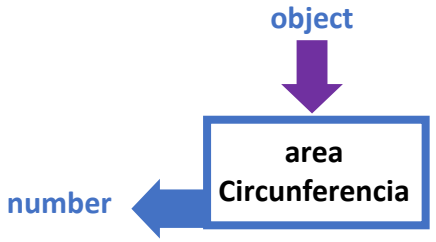



```
> let a = 3, b = 4;
> let x hipotenusa(a,b);
> x
< 5
```

Exemplos

Modelo

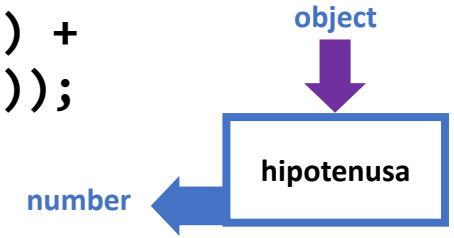
```
function areaCircunferencia(circ) {
  let area = Math.PI * Math.pow(circ.raio, 2);

  return area;
}
```




```
> let c = { raio: 5 };
> let x = areaCircunferencia(c);
> x
< 78.53
```

Modelo

```
function hipotenusa(catetos) {
  let hip = Math.sqrt(Math.pow(catetos.adj, 2) +
                      Math.pow(catetos.opo, 2));

  return hip;
}
```




```
> let catetos = {
    adj: 3,
    opo: 4
  };
> let x hipotenusa(catetos);
> x
< 5
```

Funções Builtin

DATA e HORA

```
object new Date()
object new Date(n, n, n)
object new Date(n)
object new Date(s)

number getFullYear()
number getMonth()
number getDate()
number getDay()

number setFullYear()
number setMonth()
number setDate()
```

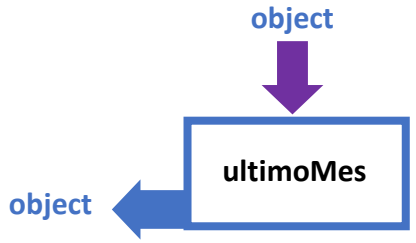
```
let x1 = new Date();
let x2 = new Date(1589499651691)
let x3 = new Date("2020-05-14T20:40:00Z")
let x4 = new Date(2020, 04, 14)

let x5 = x4.getFullYear()
let x6 = x4.getMonth()
let x7 = x4.getDate()
let x8 = x4.getDay()

let x9  = x4.setFullYear(2021)
let x10 = x4.setMonth(11)
let x11 = x4.setDate()
```

```
?
14/05/2020 20:40:00
14/05/2020 20:40:00
14/05/2020 00:00:00

2020
04
14
4

1620961200000 (14/05/2021)
1639450800000 (14/12/2021)
1640401200000 (25/12/2021)
```

Exemplos

Simples

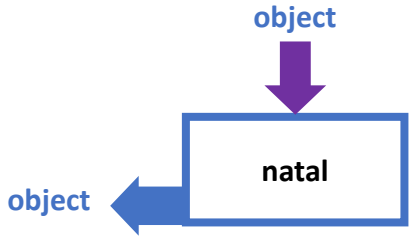
```
function ultimoDiaMes(dia) {
  let ult = new Date(dia.getFullYear(),
                     dia.getMonth() + 1,
                     0);

  return ult;
}
```




```
> let a = new Date(2020,6,1);
> let x = ultimoMes(a);
> x
< 31/07/2020
```

Simples

```
function natal(dia) {
  let diaAtual = new Date();
  let diaNatal = new Date(diaAtual.getFullYear(), 11, 25);

  let x = dia == diaNatal;
  return x;
}
```

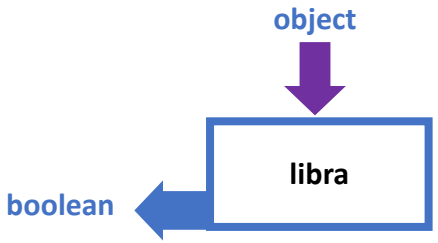



```
> let a = new Date(2020,11,24);
> let x = natal(a);
> x
< false
```

Exemplos

Modelo

```
function libra(pessoa) {
  let a = pessoa.nasc.getMonth() == 8 && pessoa.nasc.getDate() > 22;
  let b = pessoa.nasc.getMonth() == 9 && pessoa.nasc.getDate() <= 22;
  let x = a || b
          ? pessoa.nome + ", seu signo é libra."
          : pessoa.nome + ", seu signo não é libra.";
    return x;
}
```




```
> let pessoa = {
    nome: "Bruno",
    nasc: new Date(2020,9,22)
  };
>
> let x = libra(pessoa);
> x
< "Bruno, seu signo é libra."
```

Funções Builtin | TEXTO

```
let t = "Friends é topper";
```

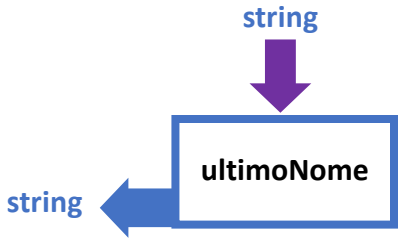
```
string toUpperCase()
string toLowerCase()
string trim()

string replace(s, s)
string replace(regex, s)

string substring(n, [n])
string substr(n, [n])
string slice(n, [n])

number search(s)
number search(regex)

number indexOf(s, [n])
number lastIndexOf(s, [n])
number charCodeAt(n)
number charAt(n)

number length
object split(s)
```

```
let x1 = t.toUpperCase()          = "FRIENDS É TOPPER"
let x2 = t.toLowerCase()          = "friends é topper"
let x3 = t.trim()                 = "Friends é topper"

let x4 = t.replace('e', '&')      = "Fri&nds é topper"
let x5 = t.replace(/e/g, '&')     = "Fri&nds é topp&r"

let x6 = t.substring(3, 7)        = "ends"
let x7 = t.substr(3, 4)           = "ends"
let x8 = t.slice(-6, -3)          = "top"

let x9  = t.search('é')           = 8
let x10 = t.search(/[aei]/)       = 2

let x11 = t.indexOf('e', 5)       = 14
let x12 = t.lastIndexOf('e')      = 14
let x13 = t.charCodeAt(0)         = 'F'
let x14 = t.charAt('F')           = 0

let x15 = t.length                = 16
let x16 = t.split(' ')            = ['Friends', 'é', 'topper']
```

Exemplos

Simples

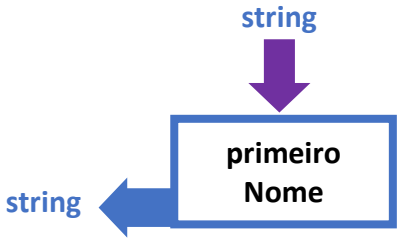
```
function ultimoNome(nome) {
  let ult = nome.substr(" ") + 1;
  return ult;
}
```




```
> let a = "Bruno de Oliveira;
> let x = ultimoNome(a);
> x
< "Oliveira"
```

Simples

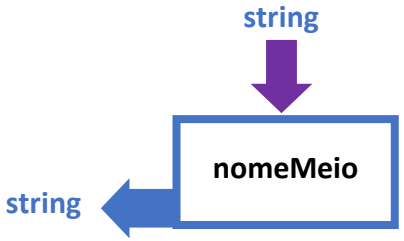
```
function primeiroNome(nome) {
  let primeiro = nome.substr(0,
                    nome.substr(" "));
  return primeiro;
}
```




```
> let a = "Bruno de Oliveira;
> let x = primeiroNome(a);
> x
< "Bruno"
```

Simples

```
function nomeMeio(nome) {
  let primEsp  = nome.substr(" ");
  let proxEsp  = nome.substr(" ", primEsp);
  let nomeMeio = nome.substring(proxEsp + 1, proxEsp);
  return nomeMeio;
}
```

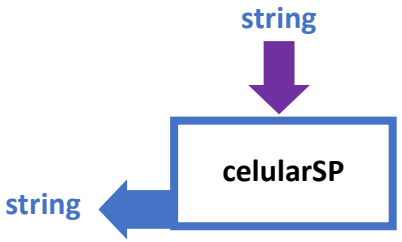



```
> let a = "Bruno de Oliveira;
> let x = nomeMeio(a);
> x
< "de"
```

Exemplos

Simples

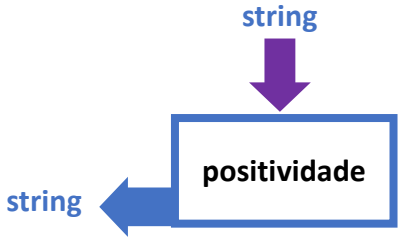
```
function celularSP(nome) {
  let ddd = nome.substr(0, 4);
  return ddd == "(11)";
}
```




```
> let a = "(11) 9888-7666";
> let x = celularSP(a);
> x
< true
```

Simples

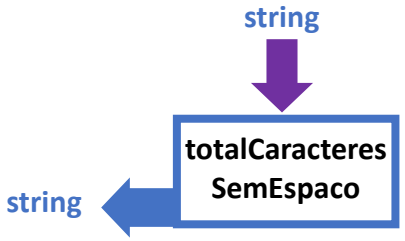
```
function positividade(frase) {
  let x = frase.replace("odeio", "amo");
  return x;
}
```




```
> let a = "Eu odeio lava-louça";
> let x = positividade(a);
> x
< "Eu amo lava-louça"
```

Simples

```
function totalCaracteresSemEspaco(frase) {
  frase = frase.replace(/ /g, "");
  let x = frase.length;
  return x;
}
```

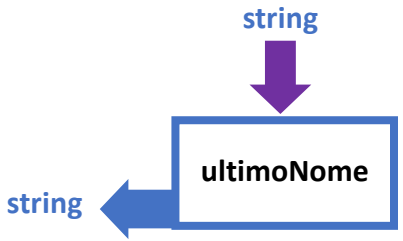



```
> let a = "Eu quero um play4";
> let x =
    totalCaracteresSemEspaco(a);
> x
< 14
```

Exemplos

Modelo

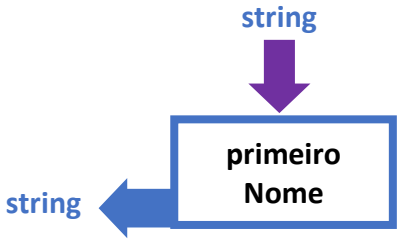
```
function ultimoNome(pessoa) {
  let ult = pessoa.nome.substr(" ") + 1;
  return ult;
}
```




```
> let a = {
    nome: "Bruno de Oliveira“
  };
> let x = ultimoNome(a);
> x
< "Oliveira"
```

Modelo

```
function primeiroNome(pessoa) {
  let primeiro = pessoa.nome.substr(0,
                    pessoa.nome.substr(" "));
  return primeiro;
}
```




```
> let a = {
    nome: "Bruno de Oliveira“
  };
> let x = primeiroNome(a);
> x
< "Oliveira"
```

Funções Builtin

Coleção

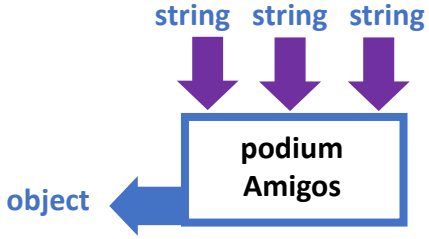
```
number push(v)
number unshift(v)

tipo pop()
tipo shift()

tipo lista[n]
number length

object splice(n, n, [n], [n])

object slice(n, [n])
```

```
let lista    = [1, 2, 3, 4, 5];

let tamanho1 = lista.push(10);          = 6  // [1,2,3,4,5,10]
let tamanho2 = lista.unshift(11);       = 7  // [11,1,2,3,4,5,10]

let ultimo   = lista.pop();             = 10 // [11,1,2,3,4,5]
let primeiro = lista.shift();           = 11 // [1,2,3,4,5]

let posicao0 = lista[0];                = 1
let tamanho3 = lista.length;            = 5

let removidos = lista.splice(0, 1);     = [1]    // [2,3,4,5]
let removidos = lista.splice(1, 2);     = [3,4]  // [2,5]
lista.splice(0, 0, 10, 11);             =        // [10,11,2,5]
lista.splice(3, 0, 12);                 =        // [10,11,2,12,5]

let novaLista1 = lista.slice(1);        = [11,2,12,5]
let novaLista2 = lista.slice(2, 4);     = [2,12]
```

Exemplos

Simples

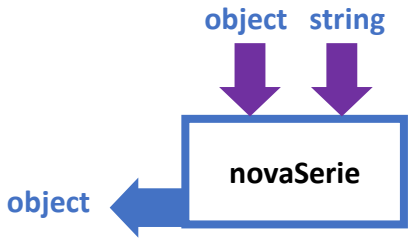
```
function podiumAmigos(a1, a2, a3) {
  let podium = [];
  podium.push(a1);
  podium.push(a2);
  podium.push(a3);

  return podium;
}
```




```
> let a = "Ross";
> let b = "Rachel";
> let c = "Chandler";
> let x = podiumAmigos(a, b, c);
> x
< (3) ["Ross", "Rachel", "Chandler"]
```

Simples

```
function novaSerie(series, nova) {
  series.push(nova);
  return series;
}
```




```
> let a = [];
> a = novaSerie(a, "Friends");
> a = novaSerie(a, "Outlander");
> a = novaSerie(a, "BlackList");
> a
< (3) ["Ross", "Rachel", "Chandler"]
```

Funções Builtin

Lambda

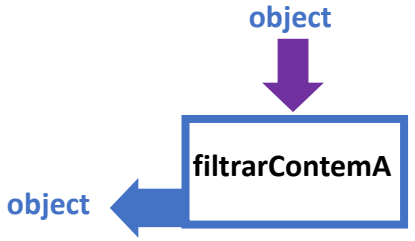
```
object Math.max.apply (null, o)
object Math.min.apply (null, o)

object sort ()
object sort (=>)

number indexOf (o,[n])
number lastIndexOf (o,[n])

object filter (=>)
object find (=>)
object findIndex (=>)

boolean some (=>)
boolean every (=>)

object map (=>)
object reduce (=>)
```

```
let lista = [3, 2, 1, 2];

let x1 = Math.max.apply(lista);                    = 3
let x2 = Math.min.apply(lista);                    = 1

let x3 = lista.sort();                             = [1,2,2,3]
let x4 = lista.sort(item => item);                 = [1,2,2,3]

let x5 = lista.indexOf(2, 2);                      = 2
let x6 = lista.lastIndexOf(2);                     = 2

let x7 = lista.filter(item => item >= 2);          = [2,2,3]
let x8 = lista.find(item => item > 1);             = 2
let x8 = lista.findIndex(item => item > 1);        = 1

let x9  = lista.some(item => item >= 3);           = true
let x10 = lista.every(item => item >= 3);          = false

let x11 = lista.map(item => item * 2);             = [2,4,4,6]
let x12 = lista.reduce(tot,item => tot + item);    = 8
```

Exemplos

Simples

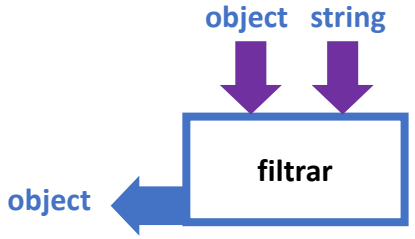
```
function filtrarContemA(series) {
  let x = series.filter(
           x => x.indexOf("a") != -1);

  return x;
}
```




```
> let a = ["Ross", "Rachel", "Chandler"];
> let x = filtrarContemA(a);
> x
< (2) ["Rachel", "Chandler"]
```

Simples

```
function filtrar(series, busca) {
  let x = series.filter(
           x => x.indexOf(busca) != -1);

  return x;
}
```

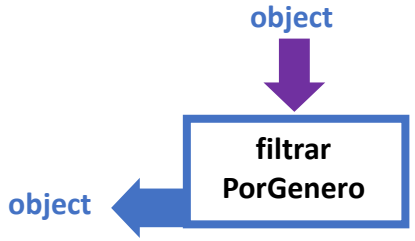



```
> let a = ["Friends", "Outlander", "BlackList"];
> let x = filtrar(a, "and");
> x
< (1) ["Outlander"]
```

Exemplos

Modelo

```
function filtrarPorGenero(series, genero) {
  let x = series.filter(
           x => x.genero == genero);

  return x;
}
```




```
> let a = [
    { serie: "Friends", genero: "Comédia" },
    { serie: "Outlander", genero: "Drama" },
    { serie: "TBBT", genero: "Comédia" }
  ];
>
> let x = filtrarPorGenero(a, "Comédia");
> x
< (2) [{ serie: "Friends", genero: "Comédia" },
       { serie: "TBBT", genero: "Comédia" }]
```

Nomeadas

```
function media(n1, n2, n3) {
     let x = (n1 + n2 + n3) / 3;
     return x;
}
```

Anônima

```
let media = function(n1, n2, n3) {
     let x = (n1 + n2 + n3) / 3;
     return x;
}
```

ArrowFunction

```
let media = (n1, n2, n3) => {
     let x = (n1 + n2 + n3) / 3;
     return x;
}

let media = (n1, n2, n3) => (n1 + n2 + n3) / 3;
```

Composta

```
function passou(n1, n2, n3) {
     let m = media(n1 + n2 + n3);
     let x = m >= 5;
     return x;
}
```

Invocação

```
let final = media(8.0, 7.0, 6.5);
```

Funções UserDefined

Funções UserDefined

Recursividade

```
function fatorial(numero) {
     if (numero == 0)
          return 1;

     return numero * fatorial(numero - 1);
}
```

```
> let numero = 5;
> let fat = fatorial(5);
> fat
< 120
```

Funções UserDefined

High Order | Callback

```
function pagarIngresso(preco, qtd, funcaoRetorno) {
     let total = preco * qtd;
     if (total >= 100)
         total = total * 0.9;

     funcaoRetorno(total);
}


> let preco = 20;
> let quantidade = 5;
>
> let funcao = total => {  console.log("Total da compra ficou: " + total); };
>
> pagarIngresso(preco, quantidade, funcao);
```

Funções UserDefined

High Order | Callback

```
function pagarIngresso(preco, qtd, funcaoRetorno) {
     let total = preco * qtd;
     if (total >= 100)
         total = total * 0.9;

     funcaoRetorno(total);
}
```

```
> let preco = 20;
> let quantidade = 5;
>
> pagarIngresso(preco, quantidade, total => {
     console.log("Total da compra ficou: " + total);
  });
```

Funções UserDefined

High Order | Closure

```
function visualizadorFriends() {
     let contador = 0;

     return function() {
          contador += 1;
          return contador;
     };
}


> let assistir = visualizadorFriends();
> let x = assistir();
> x
< 1
> x = assistir();
> x
< 2
```

Classes UserDefined

Modelo

```
class Pessoa {
      set nome(n) {
          this._nome = n;
      };
      get nome() {
          return this._nome;
      };
}
```

```
> let pessoa = new Pessoa();
> pessoa.nome = "Bruno";
> let n = pessoa.nome;
```

```
// pessoa = { }
// pessoa = { _nome: "Bruno" }
// n = "Bruno"
```

Classes UserDefined

Modelo | Construtor

```
class Pessoa {
    constructor(n) {
        this._nome = n;
    }
    set nome(n) {
        this._nome = n;
    }
    get nome() {
        return this._nome;
    }
}
```

```
> let pessoa = new Pessoa("Ada");   // pessoa = { _nome: "Ada" }
> pessoa.nome = "Grace";            // pessoa = { _nome: "Grace" }
> let n = pessoa.nome;              // n = "Grace"
```

Classes UserDefined

Serviço

```
class Escola {

   media(n1, n2, n3) {
      let x = (n1 + n2 + n3) / 3;
      return x;
   }

   passou(n1, n2, n3) {
      let m = this.media(n1, n2, n3);
      let x = m >= 5;
      return x;
   }

}
```

```
> let escola = new Escola();          // escola = { }
> let m = escola.media(8, 7, 3);      // m = 6
> let p = escola.passou(8, 7, 3);     // p = true
```

Classes UserDefined

Serviço

```
class Desconto {
   aplicar(total) {
      if (total >= 100)
          total = total * 0.9;
      return total;
   }
}

class Ingresso {
   calcular(preco, valor) {
      let total = preco * valor;

      let desc = new Desconto();
      total = desc.aplicar(total);

      return total;
   }
}
```

```
> let ingresso = new Ingresso();          // ingresso = {}
> let total = ingresso.calcular(20, 5);   // total = 90
```

Mixed

Classes UserDefined

```
class Pessoa {
  constructor() {
    this._peso = 0;
    this._altura = 0;
  }

  set peso(p) {
    this._peso = p;
  }
  get peso() {
    return this._peso;
  }
  set altura(a) {
    this._altura = a;
  }
  get altura() {
    return this._altura;
  }

  calcularIMC() {
    let imc = this.peso / Math.pow(this.altura, 2);
    return imc;
  }
}
```

```
> let p = new Pessoa();          // pessoa={_peso: 0,.. }
> p.peso = 74;                   // pessoa={_peso:74,.. }
> p.altura = 1.8;                // pessoa={..,_altura:1.8}
> let imc = p.calcularIMC();     // 22.83
```

Mixed

Classes UserDefined

```
class Pessoa {
  constructor() {
    this._nome = "";
  }

  set nome(n) {
    this._nome = n;
  }
  get nome() {
    return this._nome;
  }

  primeiroNome() {
    let x = this.nome.substr(0,
              this.nome.indexOf(" "));
    return x;
  }

  criptografar() {
    this.nome = this.nome.replace(/a/g, "@")
                         .replace(/e/g, "&");
    return this.nome;
  }
}
```

```
> let p = new Pessoa();
> p.nome = "Ana Lice";
> let pri = p.primeiroNome();
> p
> let cpt = p.criptografar();
> p
```

```
// p={_nome: ""}
// p={_nome: "Ana Lice"}
// pri="Ana"
// p={_nome: "Ana Lice"}
// cpt="@n@ Lic&"
// p={_nome: "@n@ Lic&"}
```

Estrutura seleção

| IF |

Simples

```
let total = 120;

if (total >= 100) {
    total = total * 0.9;
}
```

Com expressão

```
let tel = "(11) 98877-6655";
let sp = false;

if (tel.substr(0, 4) == "(11)") {
    sp = true;
}
```

Com Operador Lógico

```
let nasc  = new Date(2000, 10, 22);
let signo = "";

if (nasc.getMonth() == 9 && nasc.getDate() <= 22 ||
    nasc.getMonth() == 8 && nasc.getDate() >= 1) {
    signo = "Libra";
}
```

Estrutura seleção

| IF – ELSE |

Comparando número

```
let media = 7.5;
let situacao = "";

if (media >= 5) {
    situacao = "Aprovado";
} else {
    situacao = "Reprovado";
}
```

Comparando texto

```
let genero = "forró";
let musica = "";

if (genero == "forró") {
    musica = "Xenhenhém";
} else {
    musica = "Godzilla";
}
```

Comparando data

```
let dia = new Date();
let semestre;

if (dia.getMonth() <= 5) {
    semestre = 1;
} else {
    semestre = 2;
}
```

Estrutura seleção

| IF – ELSE IF – ELSE |

Comparando número

```
let media = 7.5;
let situacao = "";

if (media >= 5) {
    situacao = "Aprovado";
} else if (media >= 3) {
    situacao = "Recuperação";
} else {
    situacao = "Reprovado";
}
```

Comparando texto

```
let estado = "MG";
let cidade = "";

if (estado == "SP") {
    cidade = "São Paulo";
} else if (estado == "RJ") {
    cidade = "Rio de Janeiro";
} else if (estado == "BH") {
    cidade = "Salvador";
} else if (estado == "MG") {
    cidade = "Belo Horizonte";
}
```

IF | Estrutura seleção

Simples

```
function aplicarDesconto(total) {
  if (total > 100) {
    total = total * 0.9;
  }
  return total;
}
```

```
> let total = 100;
> let x = aplicarDesconto(total);
> x
< 90
```

Complemento Else

```
function passou(n1, n2, n3) {
  let media = (n1 + n2 + n3) / 3;
  let situacao = "";
  if (media >= 5) {
    situacao = "Aprovado";
  } else {
    situacao = "Reprovado";
  }
  return situacao;
}
```

```
> let a = 6, b = 8, c = 10;
> let x = passou(a, b, c);
> x
< "Aprovado"
```

IF | Estrutura seleção

Operador Lógico

```
function diaEscolar(dia) {
  let tipo = "";
  if (dia.getMonth() == 0 ||
      dia.getMonth() == 6 ||
      dia.getMonth() == 11) {
    tipo = "Férias";
  } else {
    tipo = "Dia Letivo";
  }
  return tipo;
}
```

```
> let dia = new Date(2020, 3, 10);
> let x = diaEscolar(dia);
> x
< "Dia Letivo"
```

Complemento Else

```
function validarEmail(email) {
  let validado = false;
  if (email.indexOf("@") != -1) {
    validado = true;
  }
  return validado;
}
```

```
> let a = "bruno@gmail.com";
> let x = validarEmail(a);
> x
< true
```

IF | Estrutura seleção

Simples

```
function operacao(numero) {
  if (total > 100) {
    total = total * 0.9;
  }
  return total;
}
```

```
> let a = 80;
> let x = operacao(a);
> x
< 80
```

Complemento else if

```
function calcular(operacao, numero) {
  let x = 0;
  if (operacao == "metade") {
    x = numero / 2;
  } else if (operacao == "dobro") {
    x = numero * 2;
  } else {
    x = 0;
  }
  return x;
}
```

```
> let a = "triplo", b = 10;
> let x = calcular(a, b);
> x
< 0
```

IF | Estrutura seleção

Conversão implícita para bool

```
function ultimoNome(nomeCompleto) {
  if (!nomeCompleto) {
    return "Nome não definido.";
  }
  let ult = nomeCompleto.substr(
              nomeCompleto.lastIndexOf(" "));
  return ult;
}
```

```
> let a;
> let x = ultimoNome(a);
> x
< "Nome não definido."
```

Funções lambda

```
function todosPares(numeros) {
  if (numeros.every(x => x % 2 == 0)) {
    return "sim";
  } else {
    return "não";
  }
}
```

```
> let a = [2, 4, 9];
> let x = todosPares(a);
> x
< "não"
```

IF | Estrutura seleção

Aninhado

```
function oqueFazer(tempo, comAmigos) {
  let x = "";
  if (tempo == "sol") {

    if (comAmigos)
      x = "Parque";
    else
      x = "Bicicleta";

  } else {

    if (comAmigos)
      x = "Tabuleiro";
    else
      x = "Netflix";

  }
  return x;
}
```

```
> let a = "chuva", b = false;
> let x = oqueFazer(a, b);
> x
< "Netflix"
```

Estrutura seleção

| SWITCH |

Simples

```
let totalAcertos = 3;
let msg = "";

switch (totalAcertos) {
  case 1:
    situacao = "Treine mais";
  break;

  case 2:
    situacao = "Você foi bem";
  break;

  case 3:
    situacao = "Acertou todas";
  break;
}
```

Aninhado

```
let cor = "azul";
let primaria = false;

switch (cor) {
  case "azul":
  case "amarelo":
  case "vermelho":
    primaria = true;
  break;
}
```

Seleção padrão

```
let dia = new Date();
let diaSemana = dia.getDay();

let oqueFazer;

switch (diaSemana) {
  case 6:
    oqueFazer = "Diversão";
  break;

  case 7:
    oqueFazer = "Família";
  break;

  default:
    oqueFazer = "Estudo";
  break;
}
```

SWITCH | Estrutura seleção

Simples

```
function semaforo(cor) {
 let acao = "";
  switch (cor) {
    case "vermelho":
      acao = "Pare";
    break;

    case "amarelo":
      acao = "Espere";
    break;

    case "verde":
      acao = "Atravasse";
    break;
  }
  return acao;
}
```

```
> let a = "verde";
> let x = semaforo(a);
> x
< "Atravesse"
```

SWITCH | Estrutura seleção

Seleção padrão

```
function calculo(operacao, numero) {
  let x = 0;
  switch (operacao) {
    case "metade":
      x = numero / 2;
    break;

    case "dobro":
      x = numero * 2;
    break;

    default:
      x = numero * 2;
    break;
  }
  return x;
}
```

```
> let a = "triplo", b = 5;
> let x = calculo(a, b);
> x
< 10
```

SWITCH | Estrutura seleção

Seleção padrão

```
function corPrimaria(cor) {
 let primaria = false;
  switch (cor) {
    case "vermelho":
    case "amarelo":
    case "azul":
      acao = true;
    break;

    default:
      acao = false;
    break;
  }
  return acao;
}
```

```
> let a = "amarelo";
> let x = corPrimaria(a);
> x
< true
```

SWITCH | Estrutura seleção

Aninhado

```
function oqueFazer(tempo, comAmigos) {
  let x = true;
  switch (tempo) {
    case "sol":
        if (!comAmigos)
            x = "Bicicleta";
        else
            x = "Parque";
    break;

    case "chuva":
        if (!comAmigos)
            x = "Netflix";
        else
            x = "Tabuleiro";
    break;
  }
  return x;
}
```

```
> let a = "sol", b = false;
> let x = oqueFazer(a, b);
> x
< "Bicicleta"
```

Estrutura repetição

| FOR |

Crescente

```
let colecao = [];

for (let i = 0; i <= 5; i++) {
  colecao.push(i);
}
```

Crescente 2 em 2

```
let colecao = [];

for (let i = 0; i <= 5; i+=2) {
  colecao.push(i);
}
```

Decrescente

```
let colecao = [];

for (let i = 5; i >= 0; i--) {
  colecao.push(i);
}
```

Crescente com volta iniciada

```
let colecao = [];

for (let i = 3; i <= 5; i++) {
  colecao.push(i);
}
```

FOR | Estrutura repetição

Crescente

```
function sequencia(fim) {
  let colecao = [];
  for (let i = 0; i <= fim; i++) {
    colecao.push(i);
  }
  return colecao;
}
```

```
> let a = 5;
> let x = sequencia(a);
> x
< [0,1,2,3,4,5]
```

Crescente

```
function somarPares(fim) {
  let colecao = [];
  for (let i = 0; i <= fim; i++) {
    if (i % 2 == 1)
      colecao.push(i);
  }
  return colecao;
}
```

```
> let a = 4;
> let x = somarPares(a);
> x
< 6
```

FOR | Estrutura repetição

Crescente

```
function quadrados(inicio, fim) {
  let colecao = [];
  for (let i = inicio; i <= fim; i++) {
    let q = Math.pow(i, 2);
    colecao.push(q);
  }
  return colecao;
}
```

```
> let a = 2, b = 4;
> let x = quadrados(a);
> x
< [4,9,16]
```

Crescente

```
function separarCaracteres(frase) {
  let novaFrase = "";
  for (let i = 0; i <= frase.length; i++) {
    let letra = frase[i];
    novaFrase += letra + "-";
  }
  return novaFrase;
}
```

```
> let a = "Bora ser ninja";
> let x = separarCaracteres(a);
> x
< B-o-r-a- -s-e-r- -n-i-n-j-a
```

FOR | Estrutura repetição

Crescente

```
function proximosDias(dia, qtd) {
  let colecao = [];
  for (let i = 1; i <= qtd; i++) {
    dia = new Date(dia.getFullYear(),
                   dia.getMonth(),
                   dia.getDate() + 1);
    colecao.push(dia);
  }
  return colecao;
}
```

```
> let a = new Date(2020, 11, 30);
> let b = 3;
> let x = proximosDias(a, b);
> x
< [30/12/2020, 31/12/2020, 01/01/2021]
```

Crescente

```
function inverter(frase) {
  let novaFrase = "";
  for (let i = 0; i <= frase.length; i++) {
    let letra = frase[i];
    novaFrase = letra + novaFrase;
  }
  return novaFrase;
}
```

```
> let a = "Ninja";
> let x = inverter(a);
> x
< "ajniN"
```

FOR | Estrutura repetição

Crescente

```
function sequenciaPares(fim) {
  let colecao = [];
  for (let i = 0; i <= fim; i++) {
    if (i % 2 != 0)
      continue;
    colecao.push(i);
  }
  return colecao;
}
```

```
> let a = 6;
> let x = sequenciaPares(a);
> x
< [0,2,4,6]
```

Crescente

```
function sequenciaPares(fim) {
  let colecao = [];
  for (let i = 0; i <= fim; i++) {
    if (i % 2 != 0)
      break;
    colecao.push(i);
  }
  return colecao;
}
```

```
> let a = 6;
> let x = sequenciaPares(a);
> x
< [0]
```

Estrutura repetição

| FOR OF |

Crescente

```
let colecao = [1, 2, 3];
let soma = 0;
for (let item of colecao) {
  soma += item;
}
```

// 6

Crescente 2 em 2

```
let colecao = ["oie", "tudo", "bem?"];
let frase = "";
for (let item of colecao) {
  frase += item + " ";
}
```

Crescente com volta iniciada

```
let colecao = [ {nome: "Junior"}, {nome: "Ingrid"}, {nome: "Luiza"}];
let frase = "";
for (let item of colecao) {
  frase += item.nome + ",";
}
```

FOR | Estrutura repetição

Crescente

```
function somarPares(numeros) {
  let soma = 0;
  for (let item of numeros) {
    if (item % 2 == 0)
      soma += item;
  }
  return soma;
}
```

```
> let a = [2, 5, 8];
> let x = somarPares(a);
> x
< 10
```

Crescente

```
function raiz(numeros) {
  let x = [];
  for (let item of numeros) {
    let r = Math.sqrt(item);
    x.push(r);
  }
  return x;
}
```

```
> let a = [4, 25, 81];
> let x = raiz(a);
> x
< [2, 5, 9]
```

FOR | Estrutura repetição

Crescente

```
function separar(frase) {
  let novaFrase = "";
  for (let item of frase) {
    novaFrase += item + "-";
  }
  return novaFrase;
}
```

```
> let a = "Desistir jamais";
> let x = separar(a);
> x
< D-e-s-i-s-t-i-r- -j-a-m-a-i-s
```

Crescente

```
function qtdVogais(frase) {
  let qtd = 0;
  for (let item of frase) {
    if (item == "a" || item == "e" || item == "i" ||
        item == "o" || item == "u")
        qtd++;
  }
  return qtd;
}
```

```
> let a = "Força do ódio!";
> let x = qtdVogais(a);
> x
< 5
```

FOR | Estrutura repetição

Crescente

```
function ultimosDiasMes(dias) {
  let colecao = [];
  for (let item of dias) {
    let ult = new Date(item.getFullYear(),
                       item.getMonth() + 1,
                       0);
    colecao.push(ult);
  }
  return colecao;
}
```

```
> let a = [ new Date(2020,4,12),
            new Date(2020,5,07)];
> let x = ultimosDiasMes(a);
> x
< [31/05/2020, 30/06/2020]
```

Crescente

```
function inverter(frase) {
  let novaFrase = "";
  for (let item of frase) {
    novaFrase = item + novaFrase;
  }
  return novaFrase;
}
```

```
> let a = "Me salva";
> let x = sequenciaPares(a);
> x
< "avlas eM"
```

FOR | Estrutura repetição

Crescente

```
function somarPares(numeros) {
  let soma = 0;
  for (let item of numeros) {
    if (item % 2 != 0)
        continue;
    soma += item;
  }
  return soma;
}
```

```
> let a = [2, 4, 5, 8];
> let x = somarPares(a);
> x
< 14
```

Crescente

```
function somarPrimeirosPares(numeros) {
  let soma = 0;
  for (let item of numeros) {
    if (item % 2 != 0)
        break;
    soma += item;
  }
  return soma;
}
```

```
> let a = [2, 4, 5, 8];
> let x = somarPrimeirosPares(a);
> x
< 6
```

Estrutura repetição

| WHILE |

Simulando o FOR

```
let soma = 0;

let i = 0;
while (i <= 10) {
  soma += i;
  i++;
}
```

Simulando o FOREACH

```
let colecao = [10, 20, 30];

let soma = 0;
let i = 0;
while (i < colecao.length) {
  let item = colecao[i];
  soma += item;
  i++;
}
```

WHILE | Estrutura repetição

Com Pulo

```
function somarPares(numeros) {
  let soma = 0;
  let i = 0;
  while (i < numeros.length) {
    if (i % 2 != 0) {
      i++;
      continue;
    }
    soma += numeros[i];
    i++;
  }
  return soma;
}
```

```
> let a = [2, 4, 5, 8];
> let x = somarPares(a);
> x
< 14
```

WHILE | Estrutura repetição

Com Parada

```
function somarPrimeirosPares(numeros) {
  let soma = 0;
  let i = 0;
  while (i < numeros.length) {
    if (i % 2 != 0) {
      break;
    }
    soma += numeros[i];
    i++;
  }
  return soma;
}
```

```
> let a = [2, 4, 5, 8];
> let x = somarPrimeirosPares(a);
> x
< 6
```

WHILE | Estrutura repetição

Sequência com voltas

```
function sequencia(inicio, fim) {
  let seq = [];
  let i = inicio;
  while (i <= fim) {
    seq.push(i);
    i++;
  }
  return seq;
}
```

```
> let a = 1;
> let b = 5;
> let x = sequencia(a, b);
> x
< [1, 2, 3, 4, 5]
```

Sequência sem voltas

```
function sequencia(inicio, fim) {
  let seq = [];
  let i = inicio;
  while (i <= fim) {
    seq.push(i);
    i++;
  }
  return seq;
}
```

```
> let a = 10;
> let b = 5;
> let x = sequencia(a, b);
> x
< []
```

WHILE | Estrutura repetição

Sequência com voltas

```
function sequencia(inicio, fim) {
  let seq = [];
  let i = inicio;
  do {
    seq.push(i);
    i++;
  } while (i <= fim);
  return seq;
}
```

```
> let a = 1;
> let b = 5;
> let x = sequencia(a, b);
> x
< [1, 2, 3, 4, 5]
```

Sequência com 1 volta

```
function sequencia(inicio, fim) {
  let seq = [];
  let i = inicio;
  do {
    seq.push(i);
    i++;
  } while (i <= fim);
  return seq;
}
```

```
> let a = 10;
> let b = 5;
> let x = sequencia(a, b);
> x
< [10]
```

WHILE | Estrutura repetição

Com Parada

```
function somarPrimeirosPares(numeros) {
  let soma = 0;
  let i = 0;
  while (i < numeros.length) {
    if (i % 2 != 0) {
      break;
    }
    soma += numeros[i];
    i++;
  }
  return soma;
}
```

```
> let a = [2, 4, 5, 8];
> let x = somarPrimeirosPares(a);
> x
< 6
```

WHILE | Estrutura repetição

Apenas o while faz

```
function proximaRaizInteira(numero) {
  let resto = -10;
  while (resto != 0)
    numero++;
    resto = Math.sqrt(numero) % 1;
  }
  return numero;
}
```

```
> let a = 10;
> let x = proximaRaizInteira(a);
> x
< 16
```

Apenas o while faz

```
function diasParaSextar() {
  let dias = [];
  let dia = new Date();
  while (dia.getDay() != 6) {
    dia = new Date(dia.getFullYear(),
                   dia.getMonth(),
                   dia.getDate() + 1);
    dias.push(dia);
  }
  return dias;
}
```

```
// hoje sendo 01/01/2020
> let x = diasParaSextar();
> x
< [02/01/2020, 03/01/2020]
```

WHILE | Estrutura repetição

Apenas o while faz

```
function gerarMaioresQueTres() {
  let numeros = [];
  let x = Number.MAX_VALUE;
  while (x >= 3) {
    x = Math.floor(Math.random() * 10) + 1;
    numeros.push(x);
  }
  x.pop();
  return dias;
}
```

```
> let x = gerarMaioresQueTres();
> x
< [5, 7, 8, 5, 4]
```

Nossa Senhora de Fátima