**No console:**

const a = 2

undefined

function soma (*a*, *b*) { if (typeof *a* === 'number' && typeof *b* === 'number')  return *a* + *b* }

undefined

soma('a', 'b')

undefined

soma (4, 7)

11

**Modificadores de acesso**: public, private, protected e readonly

**TypeScript**

* é como se fosse uma extensão do JavaScript
* adiciona checagem de tipos e várias outras features que o JS não tem nativamente
* ferramenta útil para não ter que escrever muito código

function soma(*a*: *number*, *b*: *number*) {

    return *a* + *b*;

}

soma(1, 2)

//types - define um tipo, uma estrutura de dados para os parâmetros da sua classe, sua função e fazer a validação deles

//interfaces - são contratos para você implementar as suas classes

Tudo o que se faz com a interface, é possível fazer pelo type

**//types - define um tipo, uma estrutura de dados para os parâmetros da sua classe, sua função e fazer a validação deles. fazer junções, para fazer merges das interfaces**

**//interfaces - define contratos de estruturas de dados. são contratos para você implementar às suas classes**

interface *IAnimal* {

    nome : *string*;

    tipo: 'terrestre' | 'aquático';

    domestico: *boolean*;

}

interface *IFelino* extends *IAnimal* {

    visaoNoturna: *boolean*;

}

interface *ICanino* extends *IAnimal* {

    porte: 'pequeno' | 'medio' | 'grande';

}

type *IDomestico* = *IFelino* | *ICanino*; //pode receber tanto um quanto outro

const animal: *IDomestico* = {

    domestico: true,

    nome: 'cachorro',

    porte: 'medio',

    tipo: 'terrestre',

    visaoNoturna: true

}

**Tratar a tag input:**

Arquivo .html

<body>

    <input *id*="input" *type*="text" *placeholder*="Digite os dados"/>

    <script *src*="./index.ts"></script>

</body>

Arquivo .ts

const input = document.getElementById('input') as *HTMLInputElement*; //as inputElement tem que ser depois

input.addEventListener('input', (*event*) => {

    const inpt = *event*.currentTarget as *HTMLInputElement*;

});

**Generic Types**

function adicionaApendiceALista <*T*> (*array*: *any*[], *valor*: *T*) {  //quando vai colocar valores que você não conhece, por padrão se utiliza a letra 'T'

    return *array*.map(*item* => *item* + *valor*);

}

adicionaApendiceALista([1, 2, 3], 1);

**Desenvolvendo Condicionais a partir de parâmetros**

interface *IUsuario* {

    id: *string*;

    email: *string*;

}

interface *IAdmin* extends *IUsuario* {

    cargo: 'gerente' | 'coordenador' | 'supervisor';

}

function redirecione(*usuario*: *IUsuario* | *IAdmin*) {

    if ('cargo' in *usuario*) {

        //redirecionar para a área de administração

    }

    //redirecionar para a área de administração

}

**Utilizando o “?” para variáveis opcionais**

interface *IUsuario* {

    id: *string*;

    email: *string*;

    cargo?: 'funcionario' | 'gerente' | 'coordenador' | 'supervisor'; //o ponto de interrogação transforma o item num DADO OPCIONAL

}

function redirecione(*usuario*: *IUsuario*) {

    if (*usuario*.cargo){

        //redirecionar(usuario.cargo);

    }

    //redirecionar para a área do usuário

}

**Readonly**

interface *Cachorro* {

    nome: *string*;

    idade: *number*;

    parqueFavorito?: *string*;  //opcional, pois o cachorro pode ou não ter um parque favorito

}

type *CachorroSomenteLeitura* = {

    readonly [*K* in keyof *Cachorro*] : *Cachorro* [*K*]; //para cada chave (K) no chaveiro (keyof) de cachorro, será feito um cachorro de chave

}

class MeuCachorro implements *Cachorro* {

        nome;

        idade;

        constructor(*nome*, *idade*) {

*this*.nome = *nome*

*this*.idade = *idade*;

        }

}

const cao = **new** MeuCachorro('Haru', 2);

**Importação de bibliotecas**

Geralmente o padrão é aparecer como @types/algumacoisa. Ex: @types/jquery

**$ npm i --save-dev @types/algumabiblioteca**

**Omit**

interface *Pessoa* {

    nome: *string*;

    idade: *number*;

    nacionalidade: *string*;

}

interface *Brasileiro* extends *Omit*<*Pessoa*, 'nacionalidade'> { //Omit ele basicamente esconde algum tipo de informação

}

const brasileiro: *Brasileiro* = {

}