



# TypeScript

*Material Complementar*



# Douglas Morais

34 anos, atua no mercado de desenvolvimento a cerca de 14 anos atualmente Techlead na Kenlo. Apaixonado por Tecnologia e todas as novidades do segmento.

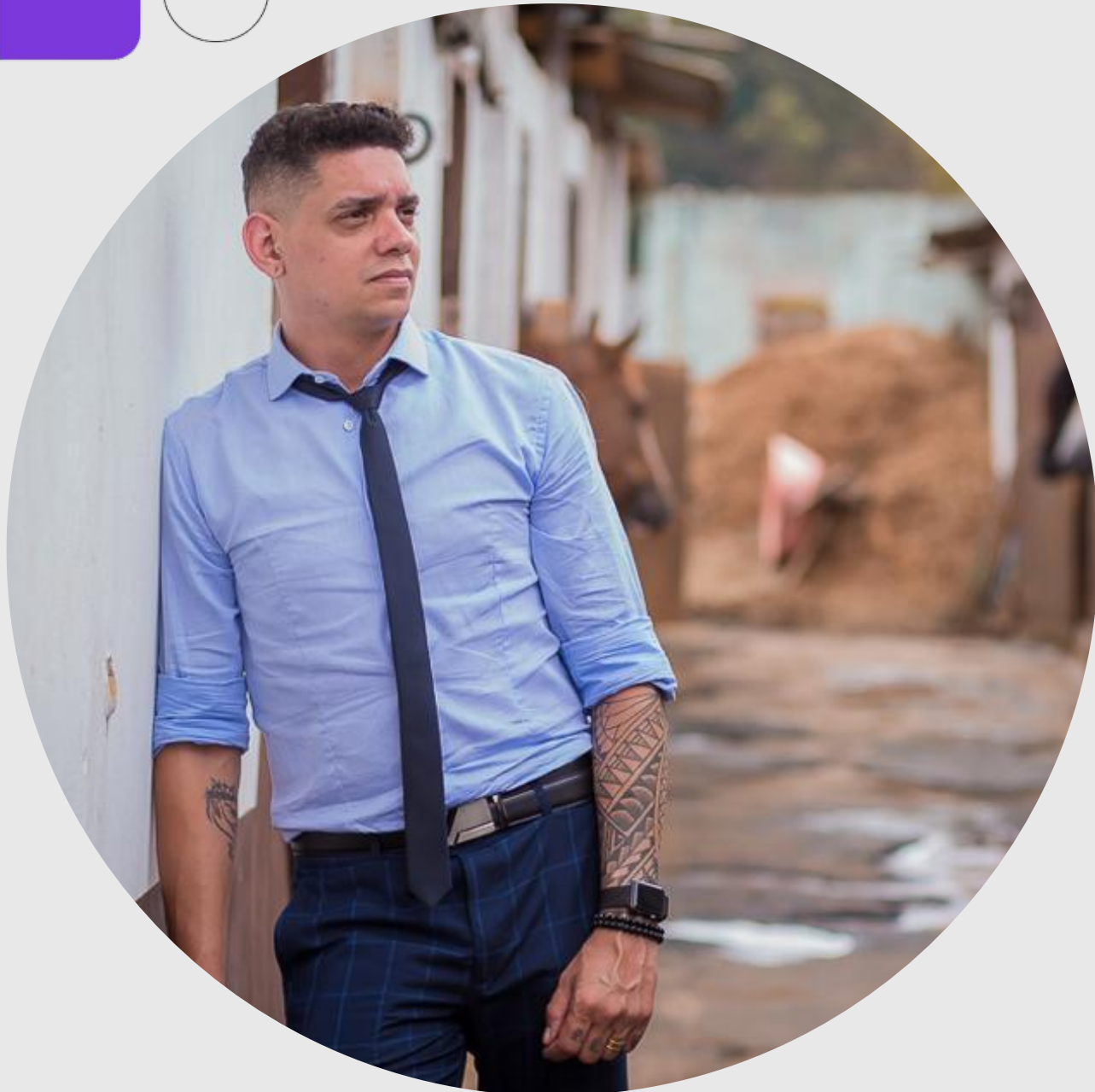
Especialista em desenvolvimento frontend um eterno entusiasta do desenvolvimento mobile.



[linkedin.com/in/douglasmoraisdev/](https://www.linkedin.com/in/douglasmoraisdev/)



<https://github.com/mrdouglasmorais>



#PraCegoVer: Fotografia do autor Douglas  
Morais.



# Índice

- + [O que é TypeScript?](#)
- + [Tipos](#)
- + [Interfaces](#)
- + [Playground](#)
- + [Criando um projeto](#)
- + [Consumindo dados de API](#)

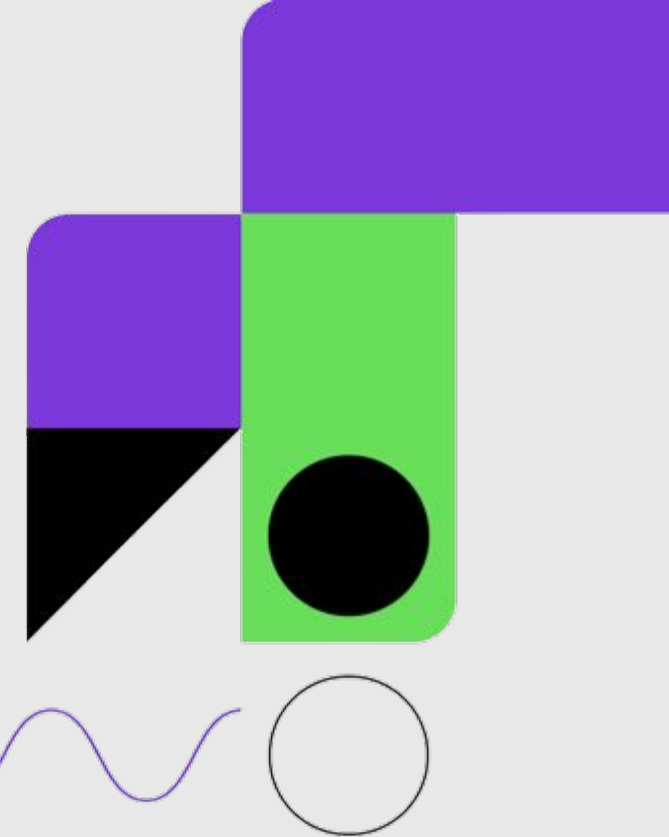
# Introdução

Neste módulo vamos aprender sobre a linguagem que vem ganhando muito espaço no mercado.

Qual a importância de saber **TypeScript**?

Teve inicialmente uma forte adoção no mercado, principalmente no backend, porém vem sendo muito utilizado no frontend também no frontend. Um bom exemplo é o Framework Angular sendo o primeiro a implementar o TypeScript no frontend.

Segundo o *Stack Share* esta linguagem é utilizada por mais de 4.000 empresas espalhadas pelo mundo, saiba mais clicando [aqui](#).



# TypeScript o que é?

*Entendendo um pouco melhor sobre esta linguagem.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.



# TypeScript o que é?

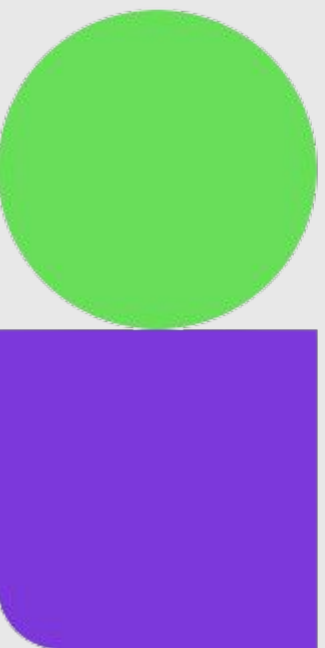
O que é **TypeScript**?

**TypeScript** é uma linguagem de programação desenvolvido pela **Microsoft**, mais especificamente um conjunto super sintático do **Javascript** ou seja ele trata e adiciona tipos ao **Javascript** que é uma linguagem dinâmica.

E qual o ganho de utilizar o **TypeScript**?

Ele te fornece mais controle sobre sua aplicação garantindo mais segurança desde o desenvolvimento até o controle de tipos e retornos de dados de seu projeto.

Não estranhe a sintaxe um pouco diferente e verbosa, nas páginas seguintes vamos explicar as principais diferenças e peculiaridades desta linguagem.





#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Tipos

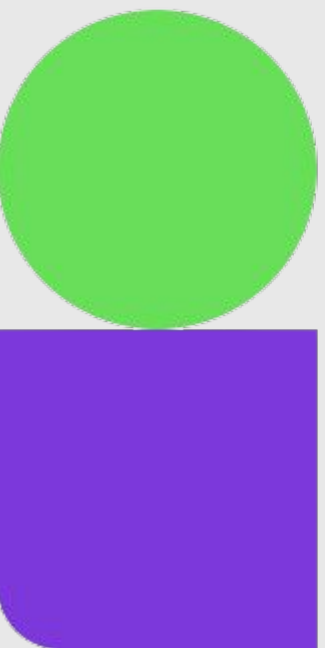
*Vamos entender um pouco mais sobre os  
tipos*

# Os tipos

Neste módulo vamos falar um pouco sobre tipos de dados e sendo assim entender inicialmente como implementar em nosso dia a dia.

Quando utilizar valores implícitos e tipos compostos.

Para este módulo não é necessário ainda utilizar editor de códigos ou github.





# Tipos

Os principais tipos são:

**String:** Valores textuais podendo conter até mesmo números.

**Boolean:** Valores onde existem apenas 2 opções, sendo elas true ou false.

**Number:** São valores numéricos, sendo eles fracionados ou inteiros.

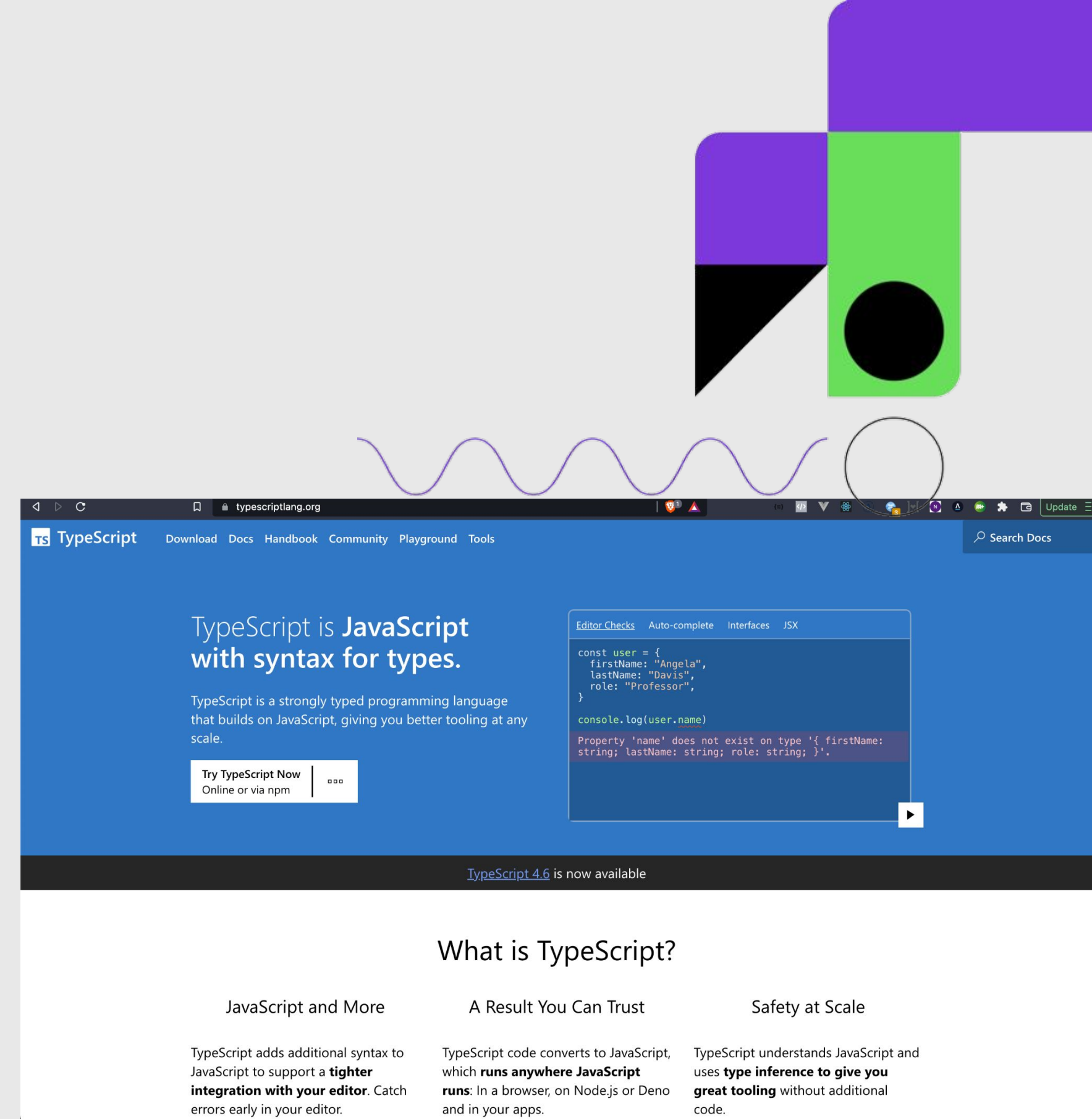
**Array:** São vetores compondo qualquer outro tipo mencionado acima.

**Tuple:** São valores numéricos dentro de um Array.

**Enum:** São tipos de atribuições parametrizados por chaves.

**Any:** Pode ser qualquer valor ou tipo.

**Void:** Muito utilizado em funções, onde é informado um valor vazio.

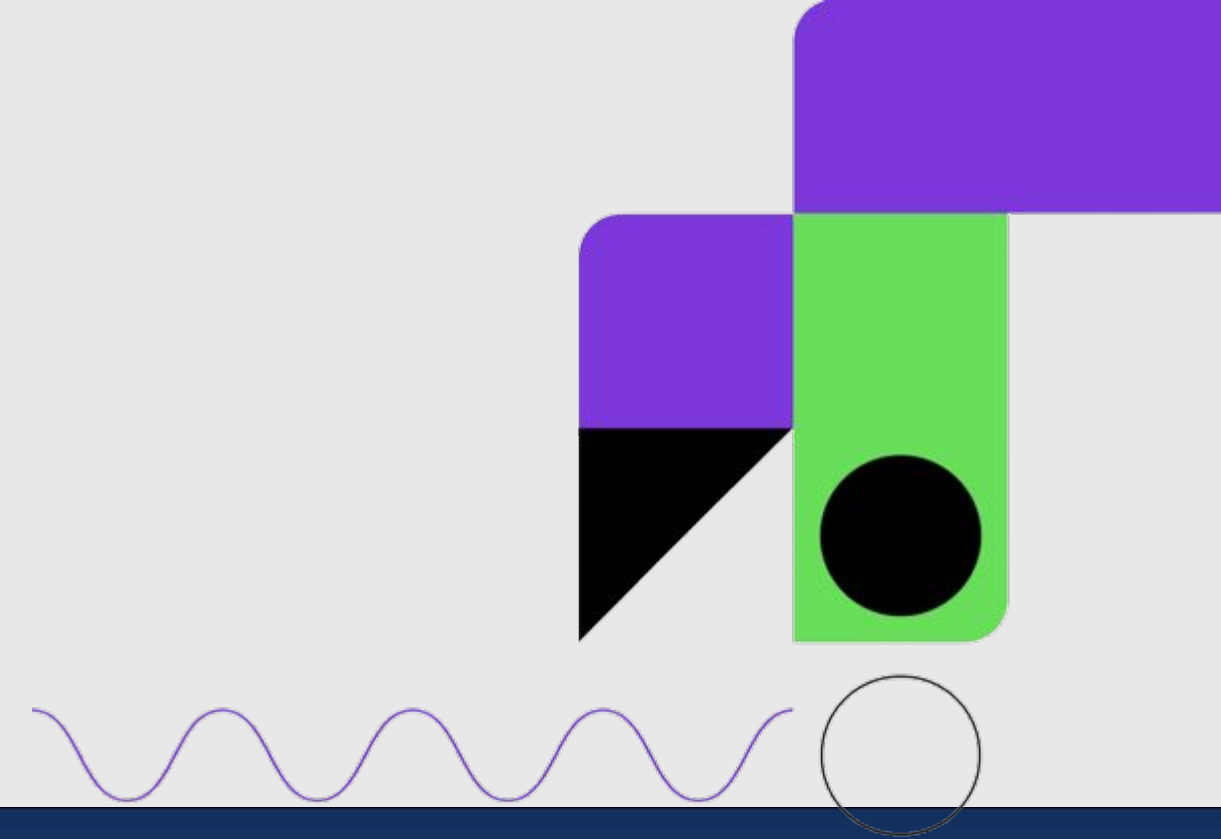


#PraCegoVer: Site oficial do TypeScript.

# String

Exemplos mais claros sobre as strings:

```
let qualquer = '1';  
console.log(qualquer, typeof qualquer );  
  
qualquer = 'String';  
console.log(qualquer, typeof qualquer );
```

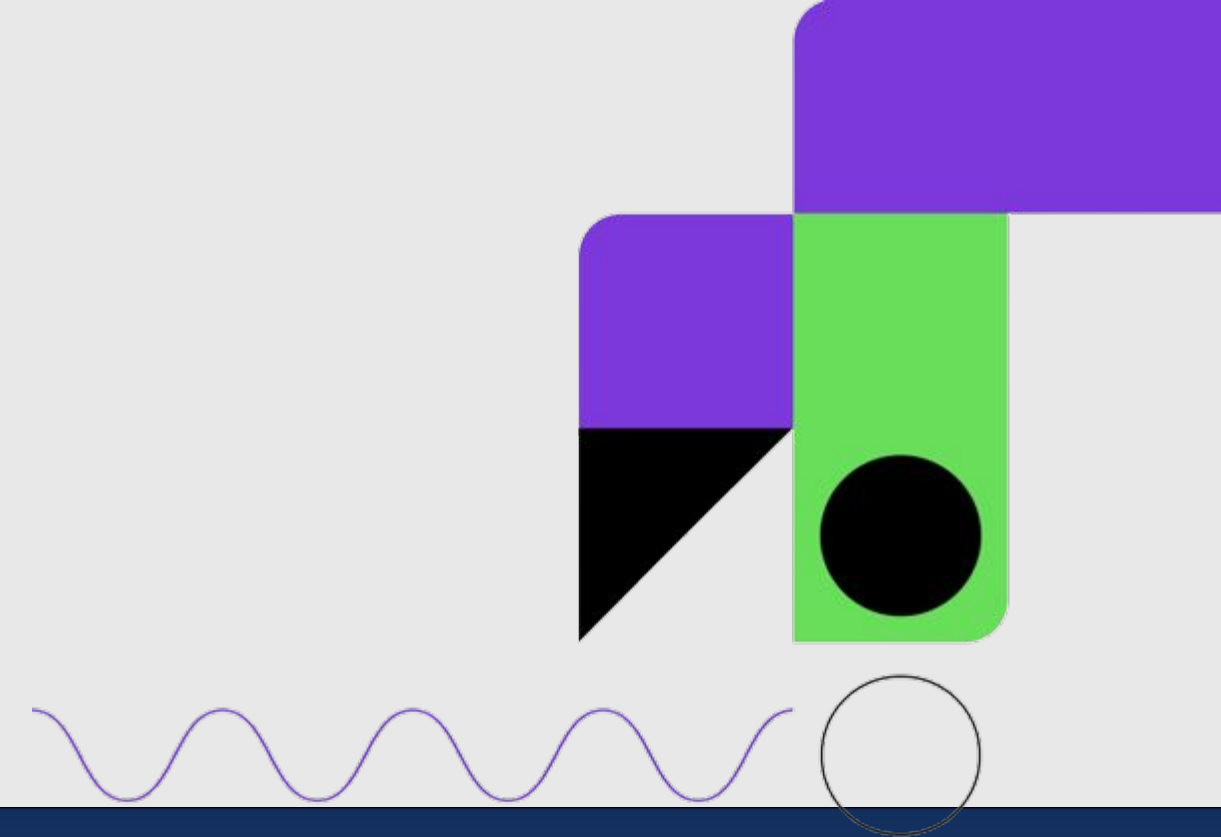


#PraCegoVer: Logomarca TypeScript.

# Boolean

Exemplos mais claros sobre os booleanos:

```
let verdadeiro: boolean  
  
verdadeiro = false  
  
console.log(verdadeiro, typeof verdadeiro)
```



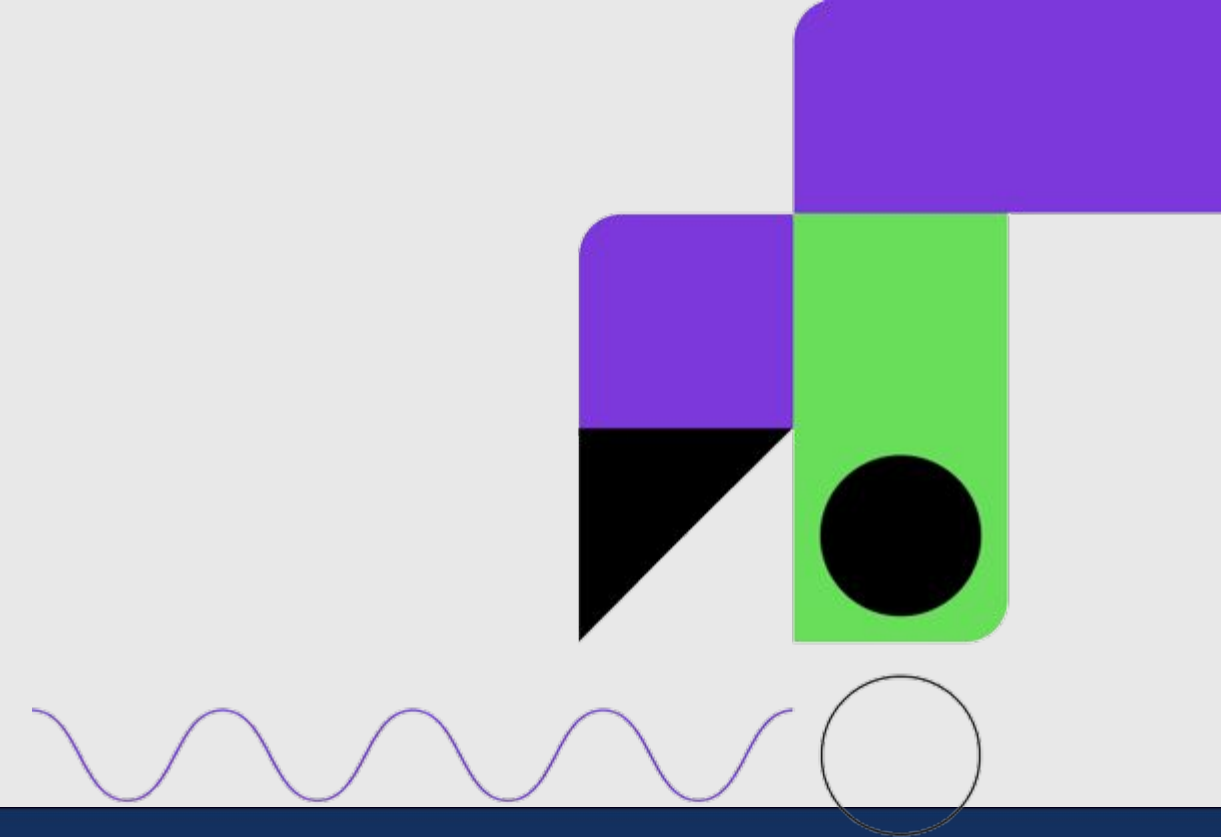
#PraCegoVer: Logomarca TypeScript.



# Number

Exemplos mais claros sobre as strings:

```
let valorNumerico: number;  
  
valorNumerico = 11.5;  
  
console.log(valorNumerico, typeof valorNumerico);
```

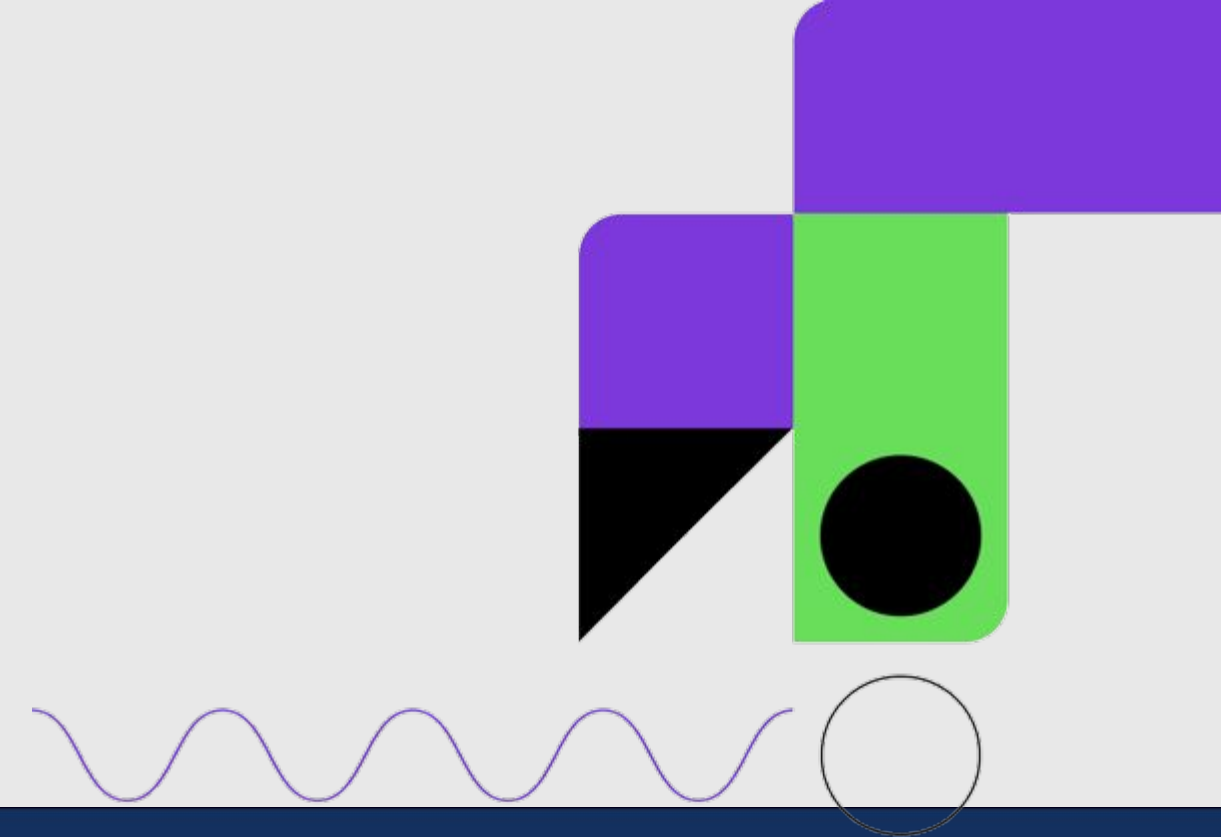


#PraCegoVer: Logomarca TypeScript.

# Array

Exemplos mais claros sobre as strings:

```
let arr: string[] = ['Arroz', 'Feijão', 'Couve', 'Bisteca de Porco'];  
  
console.log(arr, typeof arr)
```



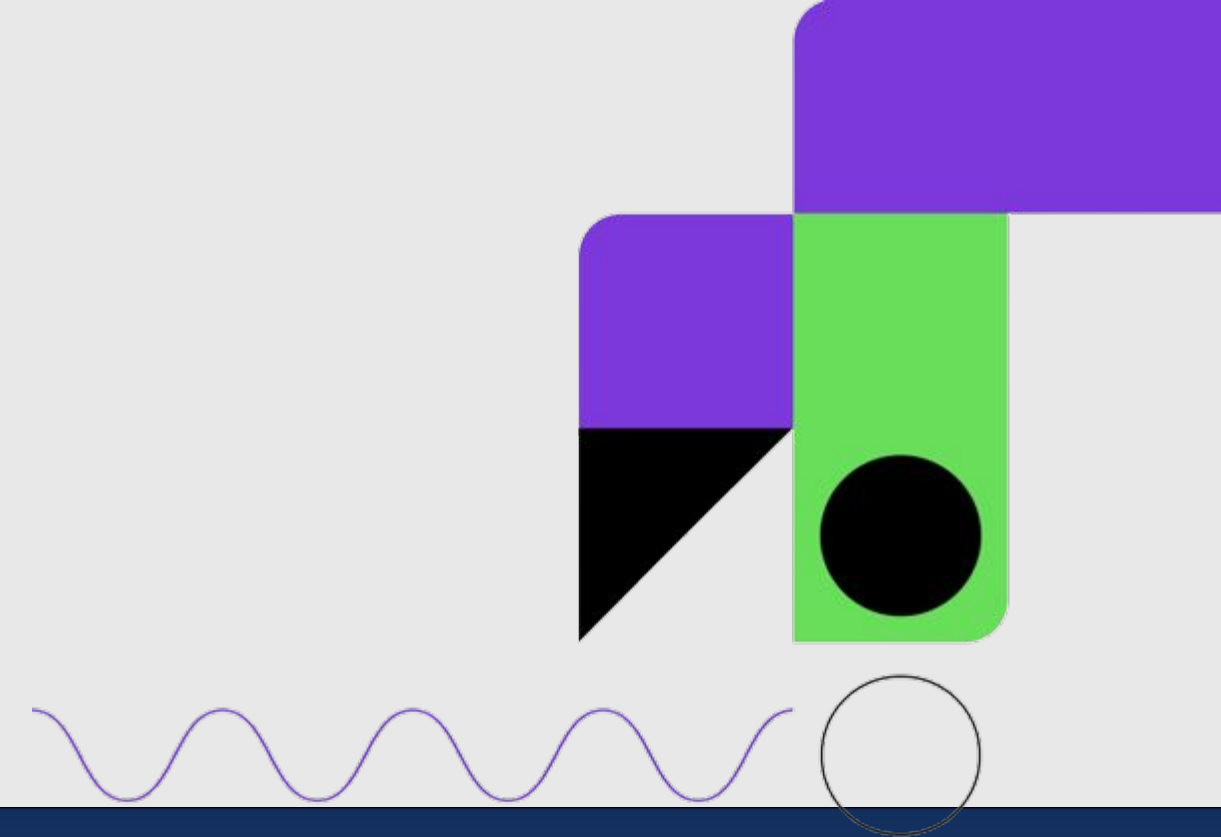
**TS**  
**TypeScript**

#PraCegoVer: Logomarca TypeScript.

# Tuple

Exemplos mais claros sobre o tuple:

```
let tuple: [string, number, string, number];  
tuple = ['hello', 1, 'world', 2];  
  
console.log(tuple[0]);  
console.log(tuple[1]);
```



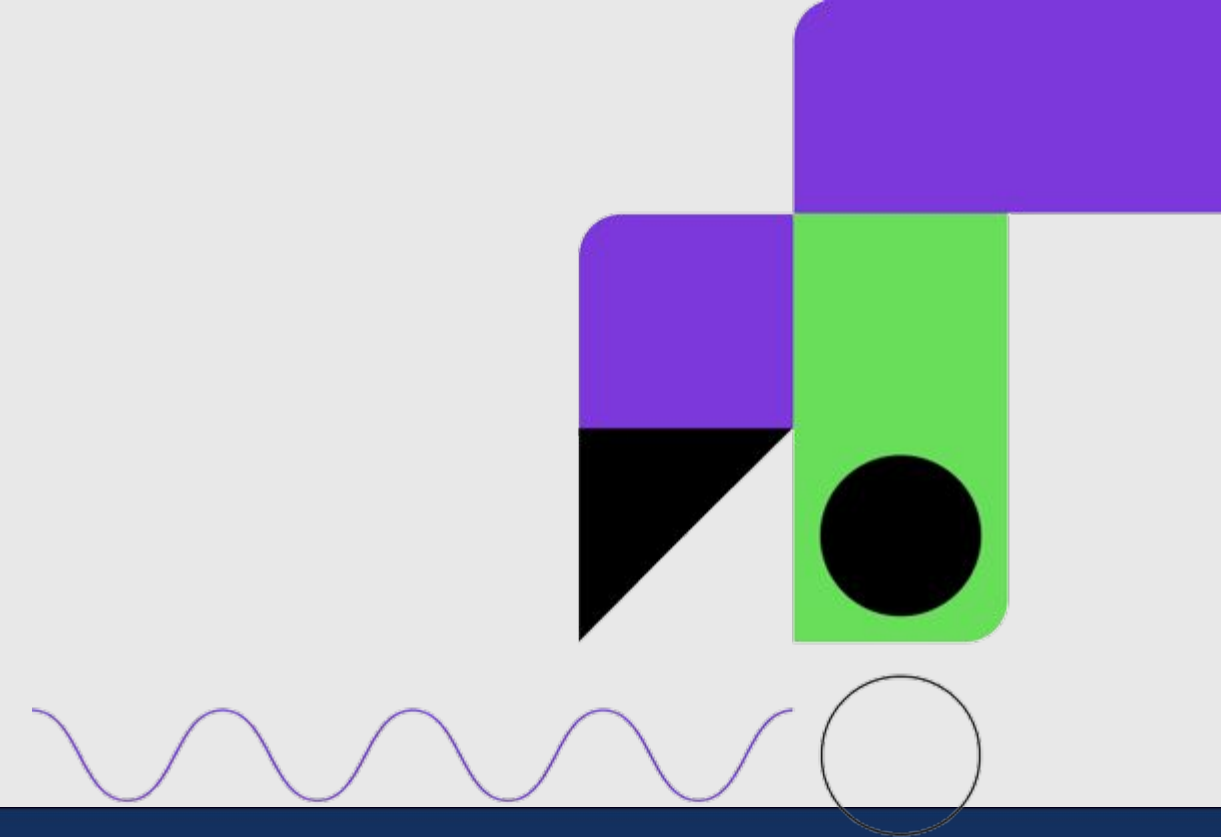
#PraCegoVer: Logomarca TypeScript.



# Enum

Exemplos mais claros sobre o Enum:

```
enum Color {Red, Green, Blue, Yellow};  
enum AnotherColor {Red = 1, Green = 2, Blue = 4, Yellow};  
  
console.log(Color)  
console.log(AnotherColor.Red)
```

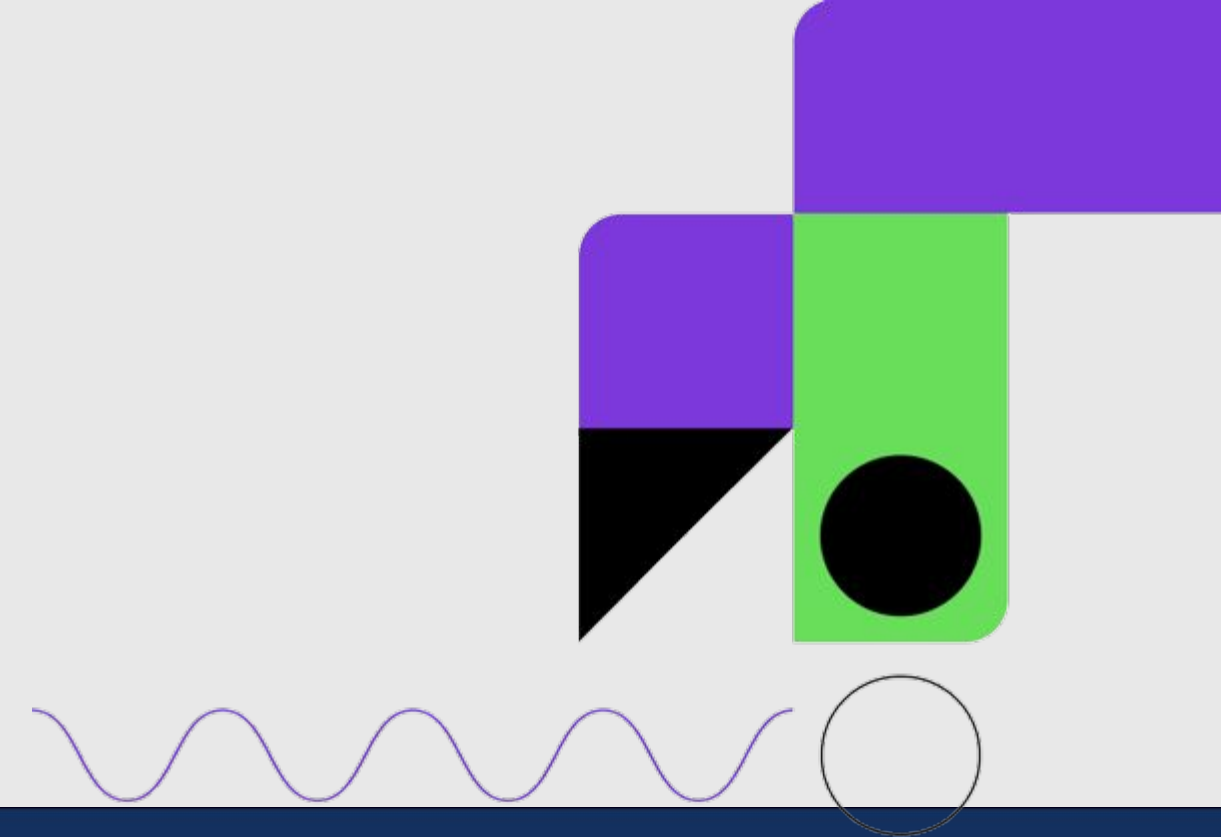


#PraCegoVer: Logomarca TypeScript.

# Any

Exemplos mais claros sobre o any :

```
let qualquer: any;  
console.log(qualquer);  
  
qualquer = 'String';  
console.log(qualquer);  
  
qualquer = 4;  
console.log(qualquer);  
  
qualquer = true;  
console.log(qualquer);
```

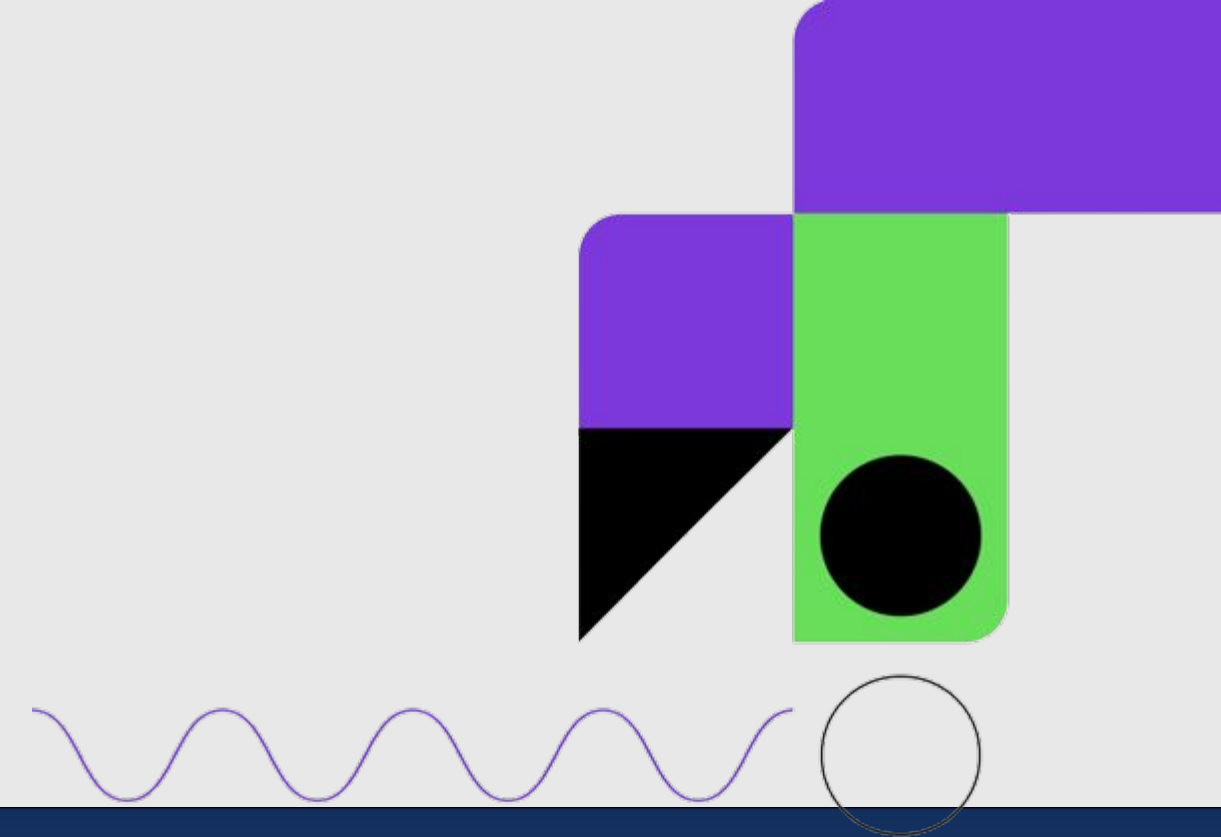


#PraCegoVer: Logomarca TypeScript.

# Void

Exemplos mais claros sobre o void:

```
function print(msg: string ): void {  
  console.log(`Função sem retorno: ${msg}`);  
}  
  
print('Escreve algo na tela');
```



#PraCegoVer: Logomarca TypeScript.



# Os tipos

Neste módulo aprendemos sobre tipos e tenha plena certeza que este conhecimento é de suma importância para os próximos passos.

Agora é hora de colocar em prática o aprendizado, até o próximo capítulo.





#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.

# Interfaces

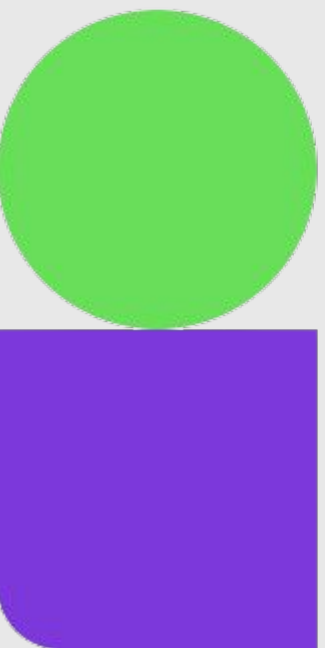
*O que são interfaces e como utilizar em  
nosso dia a dia.*

# Interfaces

O que são interfaces?

São objetos tipados de nomenclatura reservada do TypeScript (interface), serve para implementar e informar tipos associados a objetos, como se fosse uma matriz.

Tendo claro este exemplo nós vamos aos exemplos na prática.



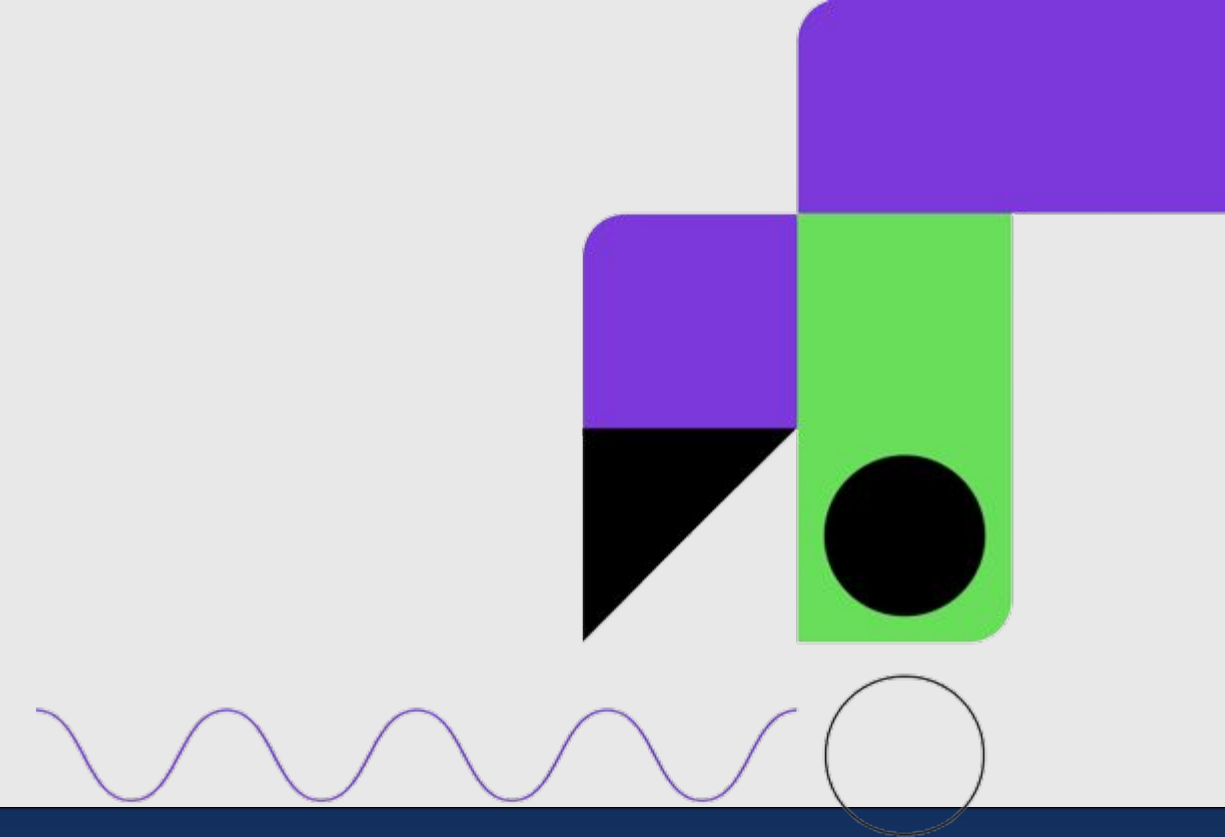


# Interface

A seguir vamos criar uma interface onde vamos tratar de valores para pets, sendo eles de qualquer tipo, sendo assim a interface vai ser responsável por tipar um objeto no qual vamos manipular.

Teremos os seguintes tipos de Pet:  
Dog, Cat e Guinea pig.

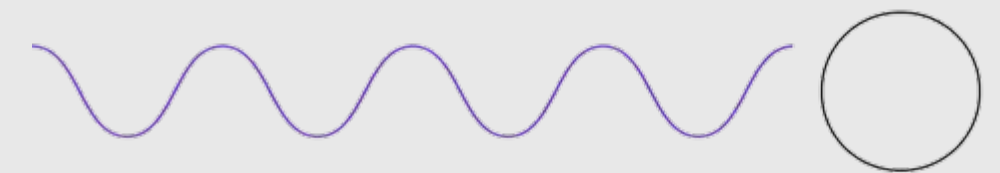
```
interface IPet{  
  name: string  
  type: string  
  age: number  
  food: []  
}
```



#PraCegoVer: Logomarca TypeScript.

# Implementando uma interface

Agora implementando a interface, observe que ela é reutilizável para alguns outros objetos.



```
interface IPet{
  name: string
  type: string
  age: number
  food: string[]
}

let Dog: IPet = {
  name: 'Bolt',
  type: 'Dog',
  age: 1,
  food: ['Bone', 'Beef']
}

let Cat: IPet = {
  name: 'Luna',
  type: 'Dog',
  age: 3,
  food: ['Fish']
}

let GuineaPig: IPet = {
  name: 'Hurley',
  type: 'Guinea pig',
  age: 1,
  food: ['Fruits']
}
```

# Interfaces

Neste módulo aprendemos a implementar interfaces, estamos em fluxo de aprendizado bem interessante e não se esqueça de após fazer a leitura desta apostila, colocar em prática cada aprendizado.



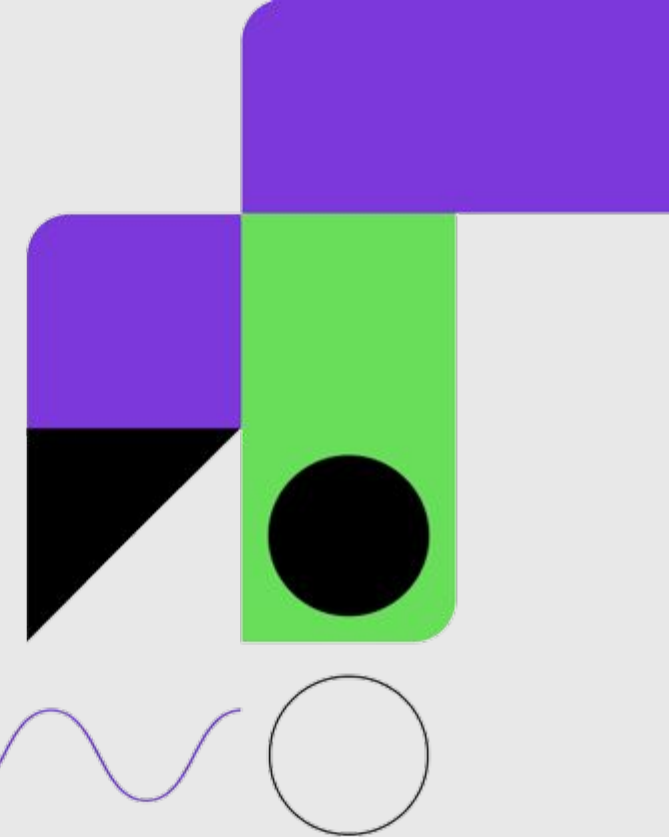


# Valores implícitos

*Entendendo como funciona e quando utilizar.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.





# PlayGround

*Entendendo um pouco mais sobre o  
TypeScript.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.



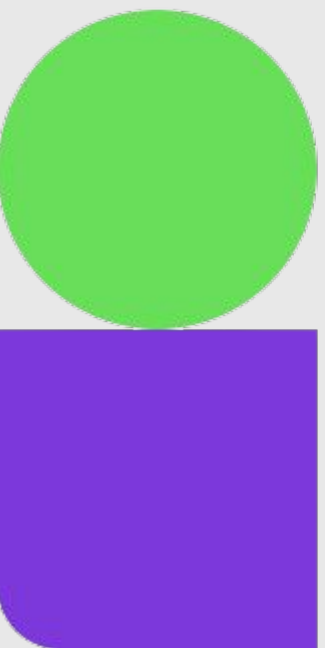
# PlayGround

O que é o PlayGround?

Na documentação oficial do TypeScript ele nos fornece um local onde podemos implementar e visualizar os resultados de nosso código, sem a necessidade de criar uma estrutura de projeto e afins.

Aqui vamos aprender um pouco mais sobre as permissividade desta linguagem.

Acesse o playground via navegador clicando [aqui](#).

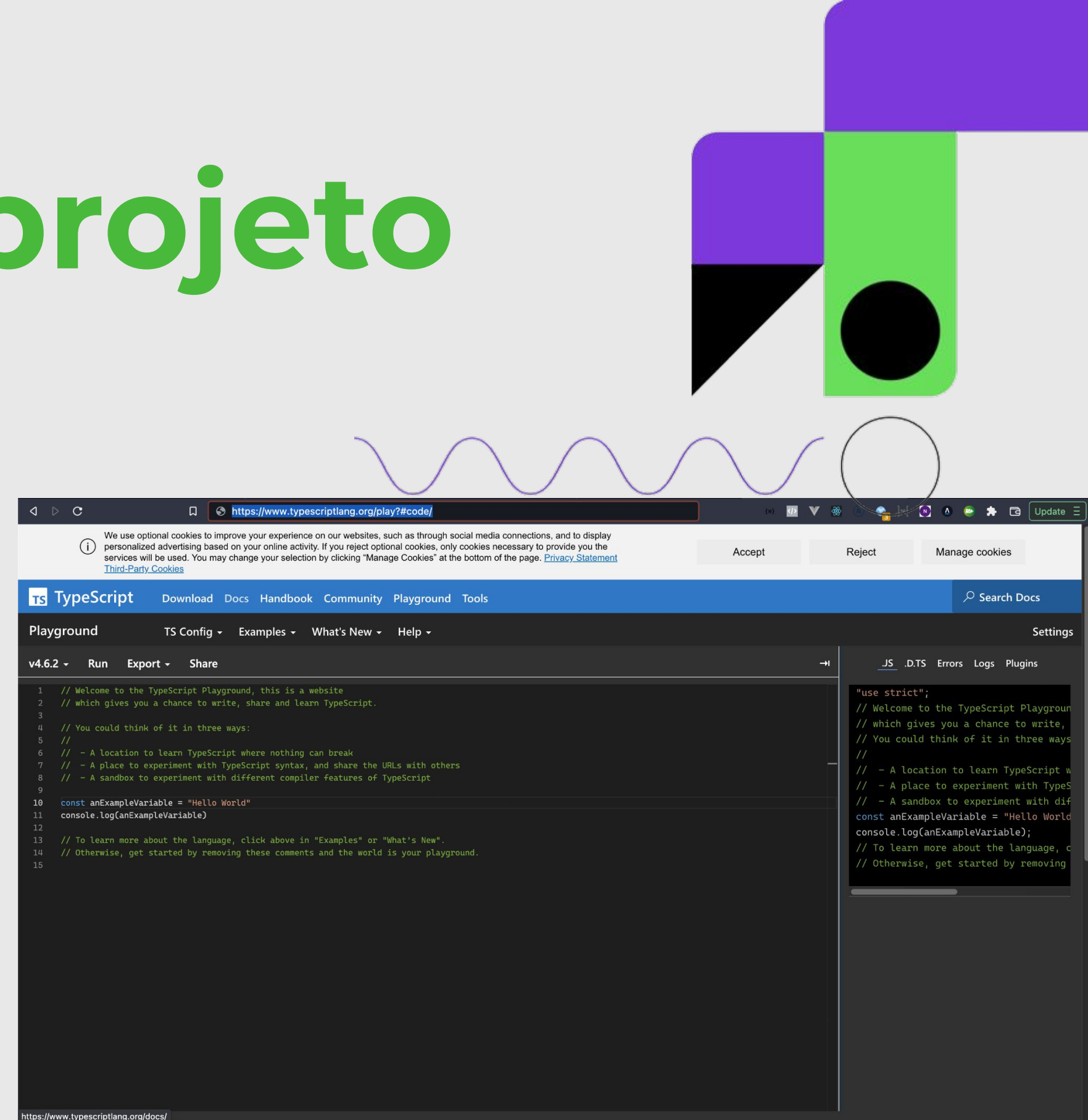


# Iniciando um projeto

Para colocar a mão na massa podemos implementar códigos e estruturas simples utilizando o log de nosso PlayGround.

Inicialmente uma estrutura de um "Hello world" é apresentada, e para este módulo o ideal é você entender o poder desta stack.

Nosso desafio é compor adicionar no log do PlayGround variáveis com notação de tipos setados e implícitos.



#PraCegoVer: PlayGround aberto no navegador.



# PlayGround

Este módulo tratou de apresentar um ferramenta onde você pode iniciar e entender um pouco mais sobre as possibilidades utilizando TypeScript.

E claro que para apimentar e trazer dinâmica para nossas atividades ficou um pequeno desafio pessoal, lembre-se pequenos desafios compõe grandes vitórias.

Até o próximo módulo.



# Criando um projeto

*Agora é a hora de colocar a mão na massa.*

#PraCegoVer: Dois notebooks  
o primeiro apresentando o teclado  
o segundo apresentando o  
monitor.



# Criando um projeto

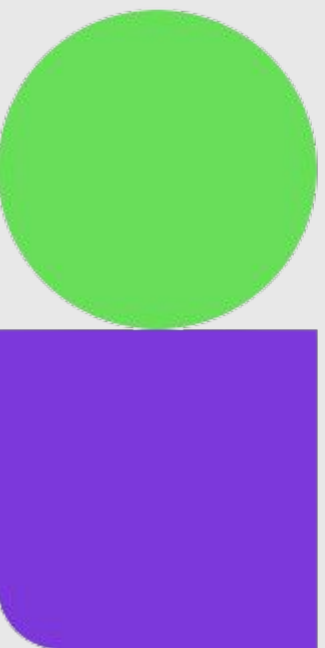
Para este módulo, vamos utilizar o TypeScript utilizando o NodeJS para transpilar o nosso código, lembrando que o navegador não reconhece o TypeScript e sim o Javascript.

O que vais ser necessário para iniciar o nosso projeto?

É necessário ter o NodeJS instalado em seu equipamento, caso ainda não tenha instalado em seu equipamento siga o passo a passo no site oficial, clique [aqui](#) recomendo utilizar a versão LTS.

Um editor de códigos de sua preferência, eu utilizo e recomendo o VSCode, para baixar clique [aqui](#).

Ter o Git devidamente configurado e conectado em seu github.





# Iniciando um projeto

Vamos precisar ter instalado de maneira global o TypeScript e para isto devemos instruir o comando a seguir

```
npm i -g typescript
```

A seguir vamos criar um diretório onde vamos armazenar o nosso projeto, em meu caso nomeei como *typescript-gama* e adicionei ao seguinte repositório, clique [aqui](#).

Inicialmente temos os nossos diretórios organizados.



#PraCegoVer: Terminal com o nosso diretório criado e adicionado ao repositório no github.

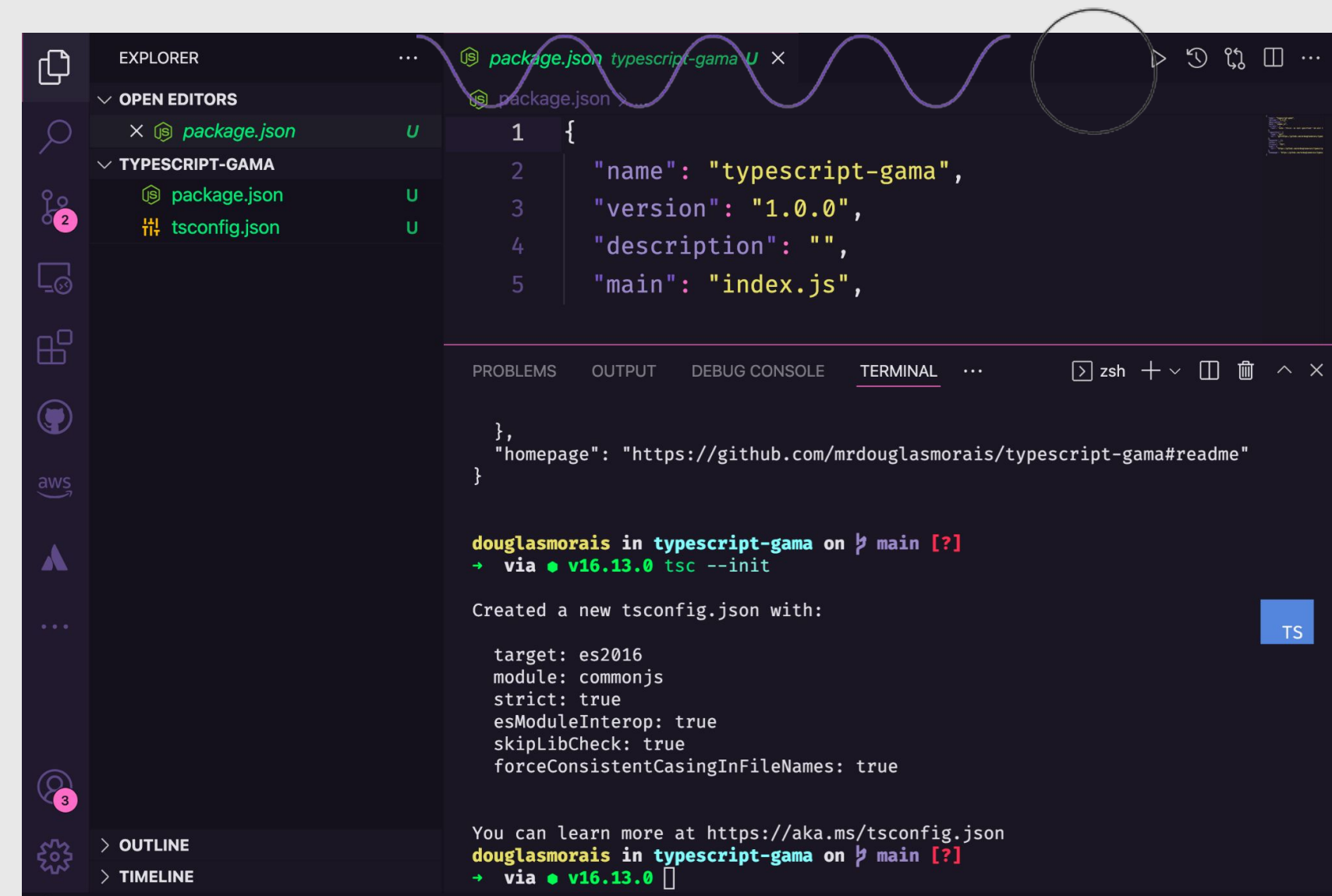
# Iniciando um projeto

A seguir vamos iniciar instruir os seguintes comandos.

```
npm init -y  
tsc --init
```

Assim que os comandos acima forem instruídos o nosso projeto deve conter 2 arquivos, o package.json e tsconfig.json.

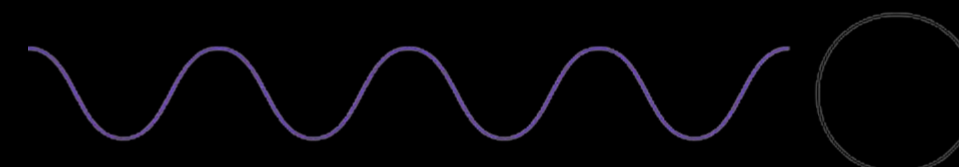
Agora vamos precisar criar um arquivo com o nome index.ts para adicionar o nosso projeto e por fim adicionar um script para facilitar o nosso log no package.json...



#PraCegoVer: VSCode com estrutura inicial do nosso projeto.

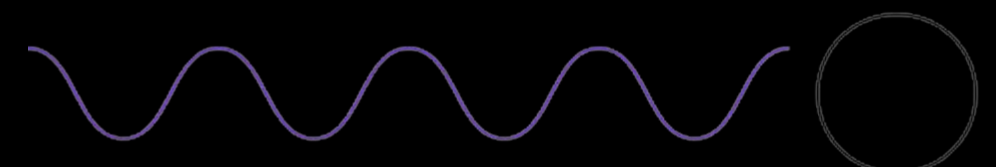
# index.ts

```
console.log('Hello Gama!!!')
```



# package.json

```
{
  "name": "typescript-gama",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.ts",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/mrdouglasmorais/typescript-gama.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/mrdouglasmorais/typescript-gama/issues"
  },
  "homepage": "https://github.com/mrdouglasmorais/typescript-gama#readme"
}
```



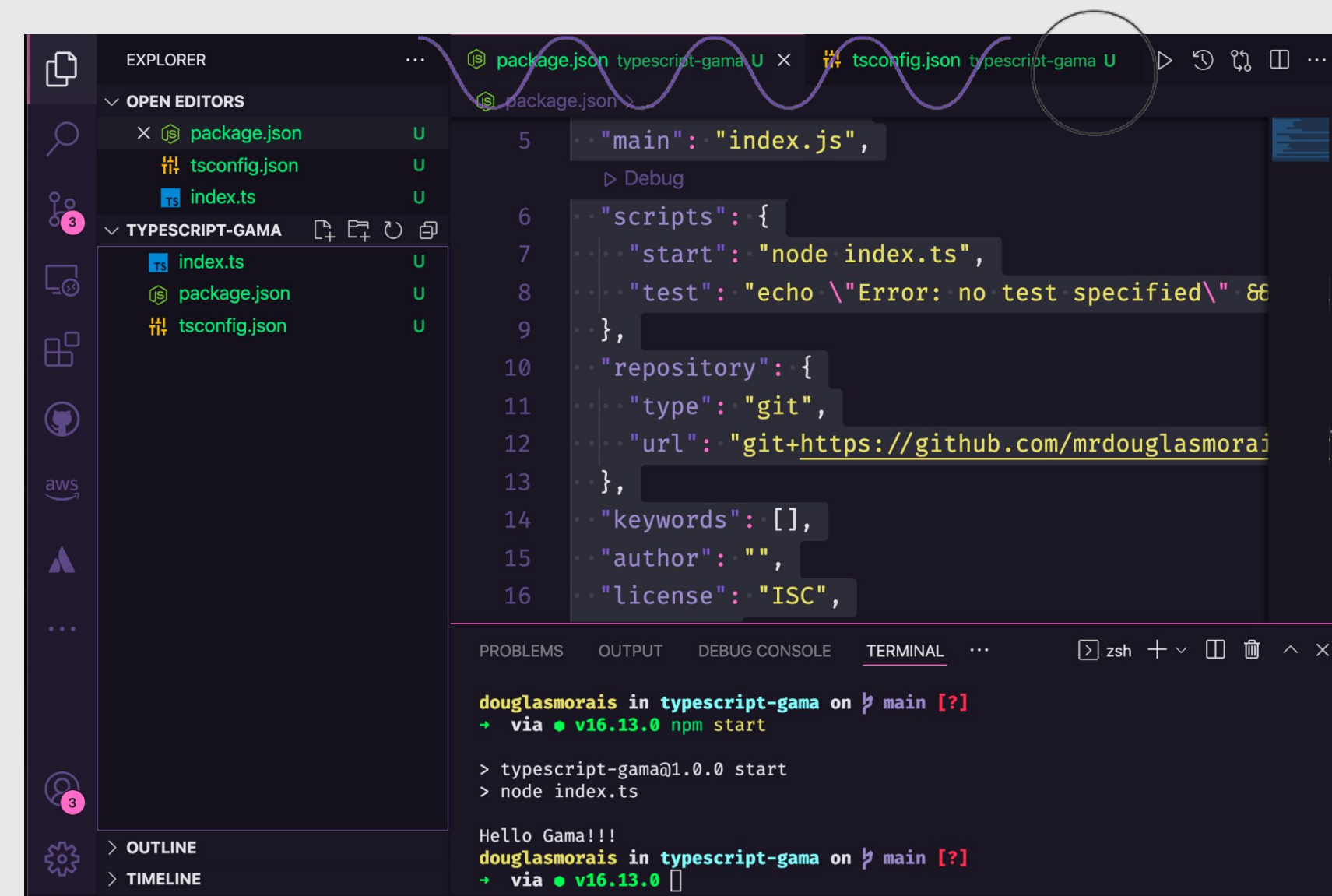
# Configurando

Ao rodar o seguinte script:

```
npm start
```

Ao rodar o comando mencionado, o nosso terminal deve aparecer a mensagem adicionada ao nosso arquivo index.ts "Hello Gama!!!", caso a mensagem não apareça em seu terminal, confira a localização de seus arquivos.

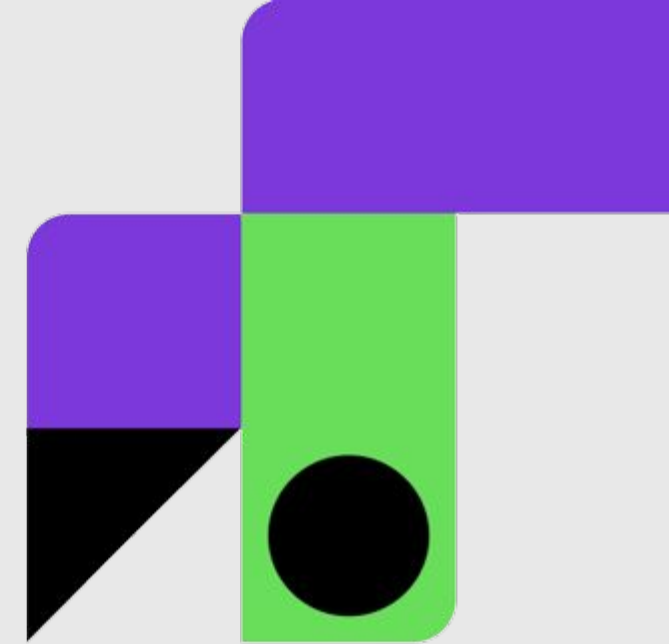
A seguir vamos criar uma pasta *src* dentro da raiz de nosso projeto.



#PraCegoVer: VSCode com estrutura inicial do nosso projeto.



# Arquivos do projeto



Agora vamos criar dentro de src os seguintes arquivos:

function.ts

interface.ts

object.ts

Nós vamos criar um array contendo personas, uma interface destas personas e por fim criar uma função que retorna por filtro o gênero cada uma delas.

A seguir os arquivos devem ficar conforme as páginas a seguir.

The screenshot shows the VS Code interface with the Explorer sidebar on the left. The Explorer sidebar shows the project structure: 'package.json', 'index.ts', 'src' (containing 'function.ts', 'interface.ts', 'object.ts', and 'index.ts'), 'package.json', and 'tsconfig.json'. The main editor area shows the content of 'index.ts' with the following code:

```
1 import getPersonType from './src/function';
2 import AllPersons from './src/object';
3
4 console.log(AllPersons);
5 console.log('Retorna genero feminino', getPersonType);
6 console.log('Retorna genero masculino', getPersonType);
```

The bottom panel shows the TERMINAL output with the following text:

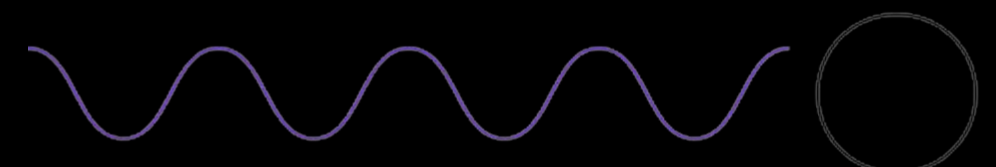
```
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 3.13 KiB | 3.13 MiB/s, done.
Total 13 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mrdouglasmorais/typescript-gama
* [new branch]    main -> main
dougasmorais in typescript-gama on main took 3s
→ via v16.13.0
```

#PraCegoVer: VSCode com estrutura inicial, contendo arquivos do nosso projeto.

# interface.ts

```
export interface IPerson{  
  name: string;  
  age: number;  
  occupation: string;  
  gender: string;  
}
```

```
// Interface com tipos obrigatórios
```



# object.ts

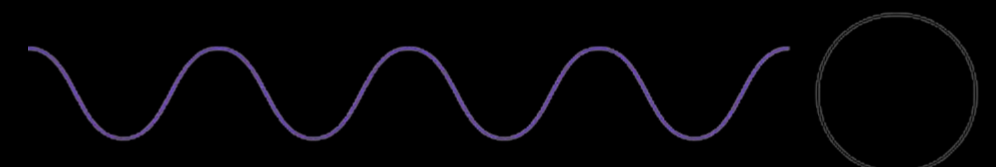
```
import { IPerson } from './interface';
// importação de interface para tipagem de nossos objetos

const PersonA: IPerson = {
  name: 'John',
  age: 22,
  occupation: 'Developer',
  gender: 'Male'
}
// Objeto persona A

const PersonB: IPerson = {
  name: 'Kath',
  age: 25,
  occupation: 'Product Manager',
  gender: 'Female'
}
// Objeto persona B

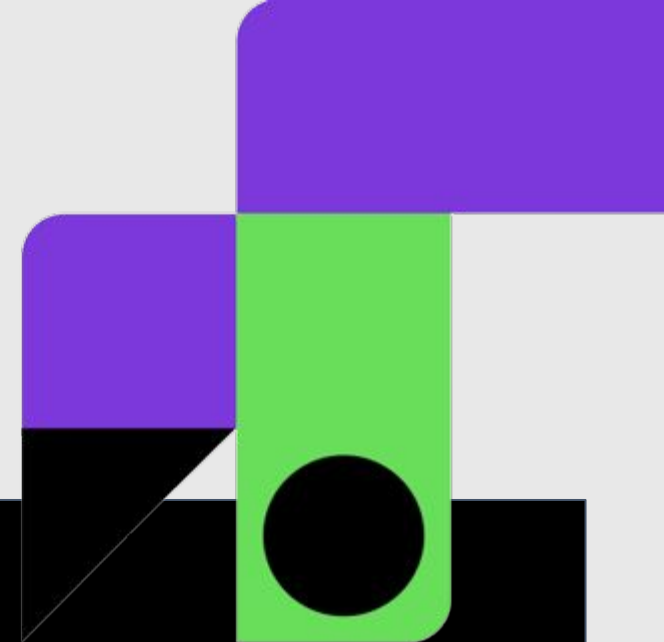
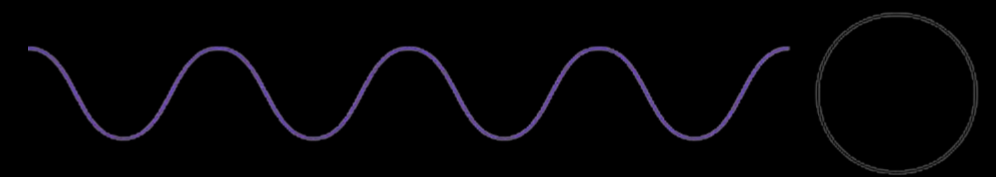
const AllPersons: IPerson[] = [PersonA, PersonB]
// array comendo 2 objetos e interface comendo array

export default AllPersons
// exportando array por padrão em nosso arquivo
```



# function.ts

```
import { IPerson } from './interface';  
// importando a interface em nossa função  
  
export default function getPersonType(info: IPerson[] ,type: string){  
  const findPerson = info.find( item => item.gender === type )  
  return findPerson  
}  
  
// função com exportação padrão de nosso arquivo
```



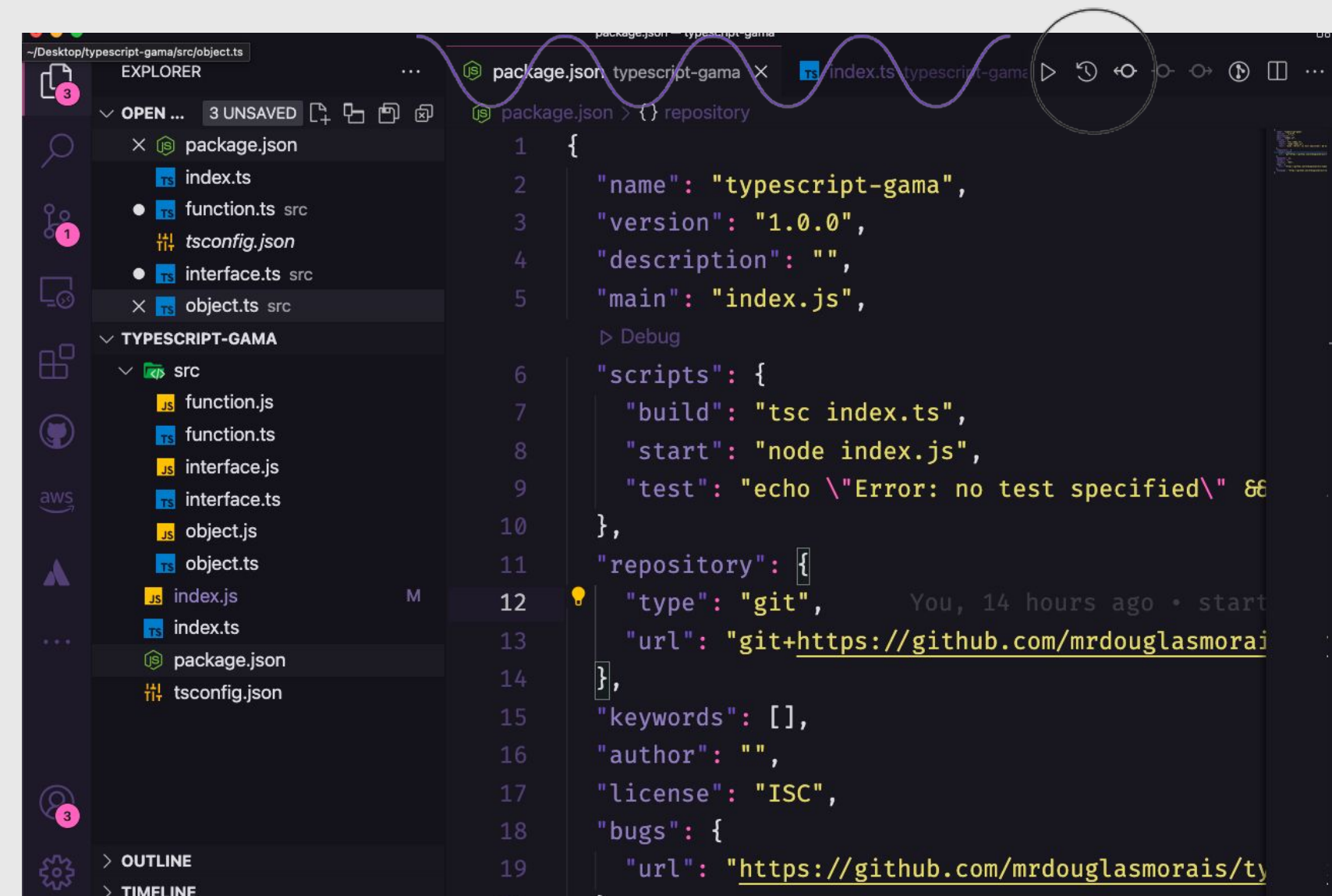
# Configurando scripts

Em seguida devemos configurar os nossos scripts para que os nossos arquivos TypeScript sejam transpilados para Javascript e nossa engineer NodeJS interprete os resultados.

E por que é necessária esta ação?

Vamos entender que o NodeJS é uma Runtime que utiliza a V8 do Google Chrome como core, sendo assim ele interpreta apenas o Javascript.

\*Nota a Runtime *Deno* uma nova proposta de evolução do NodeJS interpreta por padrão os códigos em TypeScript, saiba mais clicando [aqui](#).



The screenshot shows the VS Code interface with the 'package.json' file open. The Explorer sidebar on the left shows the project structure: 'package.json', 'index.ts', 'function.ts', 'tsconfig.json', 'interface.ts', and 'object.ts' in the 'src' directory. The main editor displays the 'package.json' content:

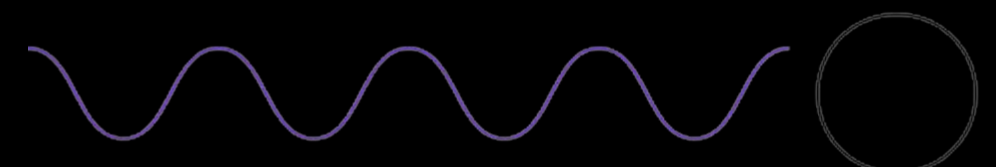
```
{
  "name": "typescript-gama",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "tsc index.ts",
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/mrdouglasmorais/ty"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/mrdouglasmorais/ty"
  }
}
```

#PraCegoVer: VSCode com package.json aberto, mostrando as configurações de nosso projeto.



# package.json

```
{
  "name": "typescript-gama",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "tsc index.ts",
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/mrdouglasmorais/typescript-gama.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/mrdouglasmorais/typescript-gama/issues"
  },
  "homepage": "https://github.com/mrdouglasmorais/typescript-gama#readme"
}
```



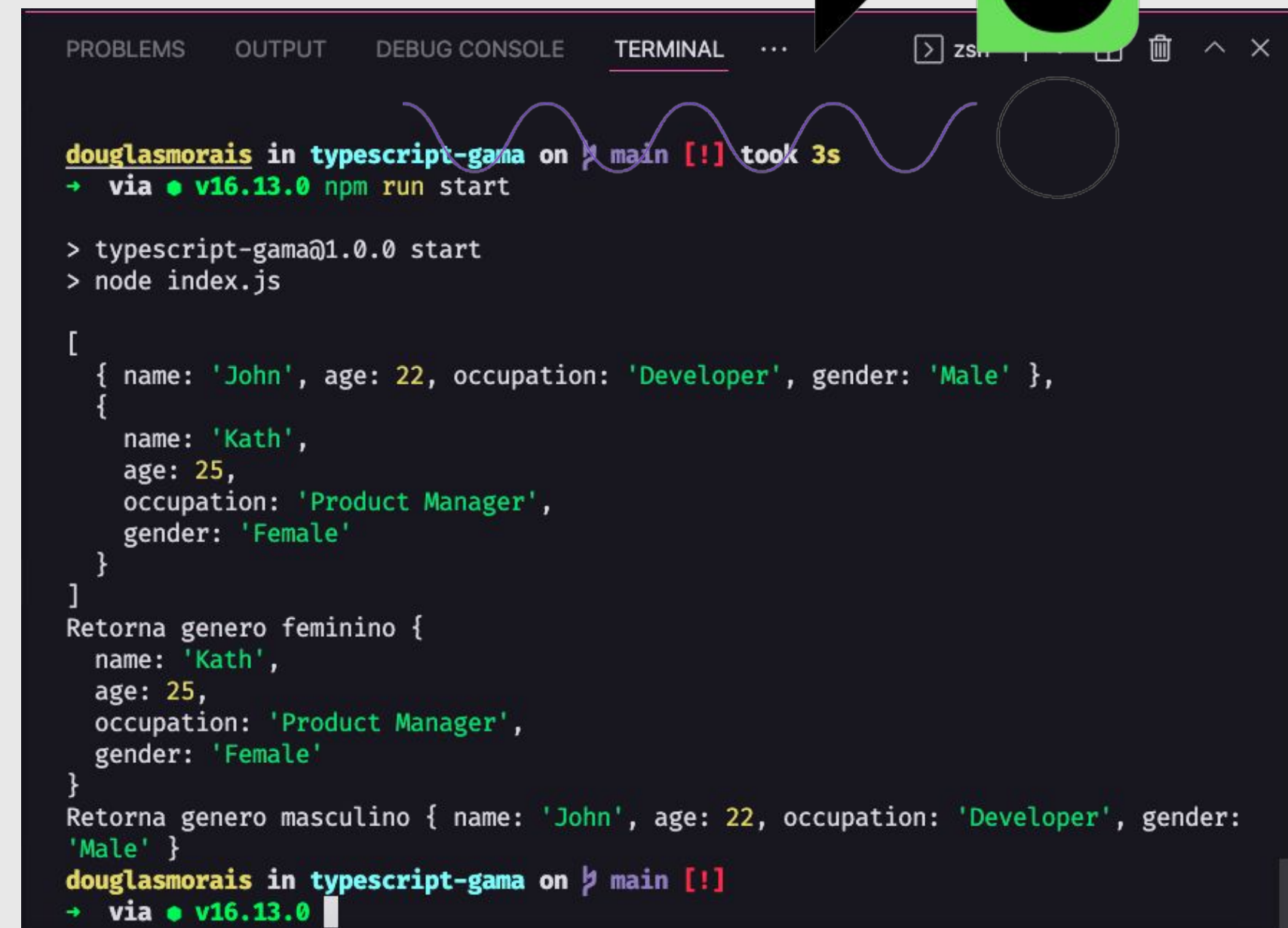
# Resultado

Para os passos finais devemos instruir dois comandos, sendo o primeiro responsável por converter nosso código para Javascript e segundo para iniciá-lo.

```
npm run build  
npm start
```

Em nosso log teremos o resultado de nossa função, conforme imagem ao lado.

Vale a pena popular ainda mais estes dados para canalizar e criar mais resultados dentro do exercício.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ...  
zsh  
dougasmorais in typescript-gama on main [!] took 3s  
→ via v16.13.0 npm run start  
  
> typescript-gama@1.0.0 start  
> node index.js  
  
[  
  { name: 'John', age: 22, occupation: 'Developer', gender: 'Male' },  
  {  
    name: 'Kath',  
    age: 25,  
    occupation: 'Product Manager',  
    gender: 'Female'  
  }  
]  
Retorna genero feminino {  
  name: 'Kath',  
  age: 25,  
  occupation: 'Product Manager',  
  gender: 'Female'  
}  
Retorna genero masculino { name: 'John', age: 22, occupation: 'Developer', gender:  
'Male' }  
dougasmorais in typescript-gama on main [!]  
→ via v16.13.0
```

#PraCegoVer: Terminal do VSCode com o resultado de nosso exercício.

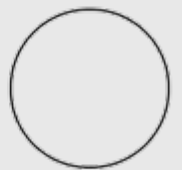


# Criando um projeto

Neste módulo aprendemos a criar nosso primeiro micro projeto utilizando o TypeScript e funcionalidades importantes.

Lembrando que o projeto está no github e os commits estão sendo disponibilizados, clique [aqui](#) para visualizar.

Vale muito a pena enfatizar, que este conteúdo é introdutório, e que as possibilidades fornecidas por esta linguagem são enormes.



# Consumindo API

*Consumindo api e controlando retorno de dados.*

#PraCegoVer: Dois notebooks  
o primeiro apresentado o teclado  
o segundo apresentando o  
monitor.



# Consumindo API

O que é **TypeScript**?

**TypeScript** é uma linguagem de programação desenvolvido pela **Microsoft**, mais especificamente um conjunto super sintático do **Javascript** ou seja ele trata e adiciona tipos ao **Javascript** que é uma linguagem dinâmica.

E qual o ganho de utilizar o **TypeScript**?

Ele te fornece mais controle sobre sua aplicação garantindo mais segurança desde o desenvolvimento até o controle de tipos e retornos de dados de seu projeto.

Não estranhe a sintaxe um pouco diferente e verbosa, nas páginas seguintes vamos explicar as principais diferenças e peculiaridades desta linguagem.





# Consumindo API

Neste módulo tivemos uma breve introdução ao **React** fazendo uso do create-react-app e iniciando o projeto em nosso ambiente local.

Lembrando que existem inúmeras maneiras de "iniciar um projeto do zero" a maneira que iniciamos o nosso é a mais rápida a mérito de aprendizado.

Não se esqueçam existem várias e várias maneiras de implementar **React** em projeto neste módulo aprendemos a criar um projeto dedicado.

# Fechamento

## Sobre o nosso aprendizado.

Agora que entendemos conceitos e funcionalidades extrema importância do React é hora de avançar para o próximo nível.

Tenha clareza que conceitos de suma importância para o seu projeto estão em torno do conhecimento da linguagem que faz parte do contexto de uma solução, em nosso caso Javascript e o superset Typescript.

Siga-nos no LinkedIn clique [aqui](#).



*E os estudos não param por aqui.*

*Aproveite ao máximo para praticar lembrando que...*

*"a repetição sem exaustão leva a perfeição" - Autor desconhecido.*

*Prof Douglas Moraes*

# Referência Bibliográfica

TYPESCRIPT, Documentação oficial.

