Raquel Moreira, 3A
código no github

TYPESCRIPT

→ É um superset para o JavaScript. Ou seja, é uma pequena extensão para o js. Portanto, não executamos typescript, mas compilamos para js.
→ Alguns servidores não leem Node diretamente, ele é convertido em typescript.
→ A principal diferença entre esses dois é que o typescript tem tipagens que mostram possíveis erros no código. O js não tem esse limite, ele é bem livre, ignorando as tipagens.

BÁSICO

→ Não especificamos para o js se a variável é do tipo string, number ou boolean. No typescript, precisamos especificar.
Exemplo: `let car: string = 'Ferrari'`
→ Para converter um arquivo ts para js, digitamos na terminal `npx tsc nomedoarquivotypescript`

ARRAY

→ Modos de fazer um array:
(1) `let fruits: string[] = ['banana','apple','orange']`
(2) `let numbers: Array<number> = [1,2,3,4,5]`

→ A única diferença entre o array de typescript e js, é que no type declaramos se é tipo string, number, etc.

TUPLAS

→Determina que tipo será o objeto.
Exemplo: `let plane: [string, boolean, number, string]`
`plane = ['Boeing 737', true, 5, '2 turbinas']`
→Quando passado pra js, ele simplesmente ignora os tipos, deixando você colocar um número no lugar da string, por exemplo.

ENUM

→Todos os objetos serão number.
Exemplo:
```
enum boat {
  engine = 2,
  bow = 1,
```

```
    stern = 1
}
```
→ Ele é do tipo leitura, por isso não podemos colocar valores.
Exemplo: `boat.engine = '2 engines'`

ANY
→ Podemos colocar qualquer tipo de dado, sem ficar restrito a colocar somente o que se pede. Não tipamos as variáveis.

VOID
→ Permite fazer funções que não retornam nada ou funções que não tem parâmetros. Por serem vazios, eles são indeclaráveis.
→ É o oposto do tipo any.
→ Quando passamos para a linguagem js, ele ignora completamente o void, a palavra some.

INTERFACES
→Funcionam como uma classe pai, como um contrato. Uma classe que implementa uma interface é obrigada a implementar todos os seus membros, com exceção dos que forem acompanhados de um (?).
Exemplo:
```
interface Person {
  name: string;
  weight? : number;
  age: number;
  hairColor? : string;
   height: number;
}
```

→ O *name*, *age* e *height* são obrigatórios. *Weight* e *hairColor* são opcionais.

HERANÇA
→Funciona igual entre ts e js. Ela serve para que as classes filhos puxem as características principais da classe pai (que chamamos no ts de interface).
→Evita a repetição de informações, diminuindo a quantidade de código e tempo.

FUNÇÕES

→ Podemos colocar parâmetros na função, dizendo se é string, number, etc. Na verdade, passar os parâmetros em uma função é recomendado, pois é mais seguro.

Exemplo:

```
function bemVindo(saudacao?:string, nome:string){
  if(saudacao){
    console.log(`Olá $(saudacao) $(nome)`);
  } else {
    console.log(`Olá $(nome)`);
  }
}
```

GENERIC

→ Ele cria componentes que podem receber vários tipos (string, number ou boolean), em vez de apenas um. Você pode determinar o tipo ao chamar a função.

Exemplo:

```
console.log(identity<number>('asd'));  //aqui dá erro
console.log(identity<number>(1));  //aqui ele funciona
```

→Só podem ser criados com interfaces ou classes.