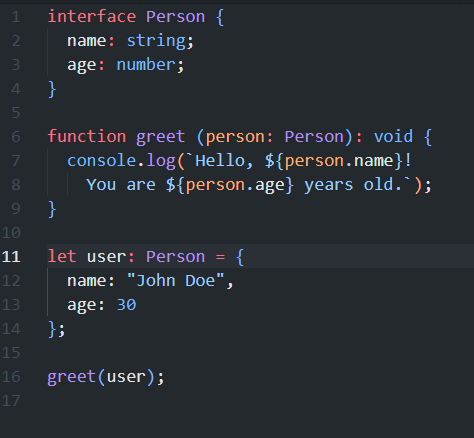
Break Free from the 'any'thing Goes Mentality, adopt INTERFACES

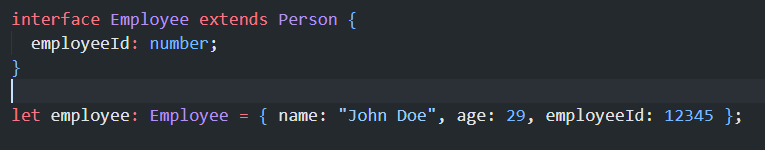
Interfaces play a crucial role in defining custom types and ensuring that objects adhere to a specific structure. They provide a blueprint for classes and objects, making code more maintainable and less error-prone.

BASIC Interface:



Here, the Person interface defines the structure of an object with name and age properties. The greet function takes an argument of type Person, ensuring that the object passed to it conforms to the specified structure.

EXTENDED Interface:



The Employee interface extends the Person interface, inheriting its properties while adding the employeeId property. This promotes code reusability and maintains a clear structure.

BASIC rules - why not to use ANY:

1. **Type Safety** net:

TypeScript's primary goal is to provide a statically typed environment for JavaScript. Using 'any' undermines this goal, as it essentially opts out of type checking. Avoiding 'any' promotes stronger type safety and helps catch potential issues during development.

2. Work in Harmony - **Interface Consistency**:

Interfaces encourage code consistency and contract adherence. By defining clear interfaces, you ensure that your functions and objects follow a standardized structure, making your codebase more readable and maintainable.

3. keep it tidy - **Maintainability**:

Using 'any' can lead to code that is difficult to maintain. As your codebase grows, the lack of type information makes it challenging to understand the expected shapes of variables and objects, increasing the likelihood of bugs.

***Para concluir***, by adhering to best practices and avoiding the use of 'any,' developers can leverage the full potential of TypeScript's static typing capabilities, catching errors early in the development process and fostering a more reliable and scalable codebase.