No Implicit `any`

This lesson introduces the `noImplicitAny` compiler flag.

WE'LL COVER THE FOLLOWING

Definition

The first and the most important flag is `noImplicitAny`. Let's look at its definition from the documentation:

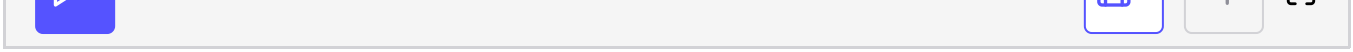
> Raise error on expressions and declarations with an implied any type.

In other words, when you enable the flag, TypeScript will throw an error whenever it's impossible to infer the type of something. Without the flag, TypeScript would automatically assume that the type is `any`. With the flag enabled, you have to provide type annotation. You can still use `any`, but now you have to do this explicitly.

`noImplicitAny` in practice

Let's look at the example of untyped function arguments. It's not possible for TypeScript to infer their types unless you provide them with default values; so if you don't type them explicitly, it will throw an error.

```
// 🔘 Parameter 'person' implicitly has an 'any' type.(7006)
function sayHello(person) {
  console.log(`Hello, ${person.name}`);
}
```

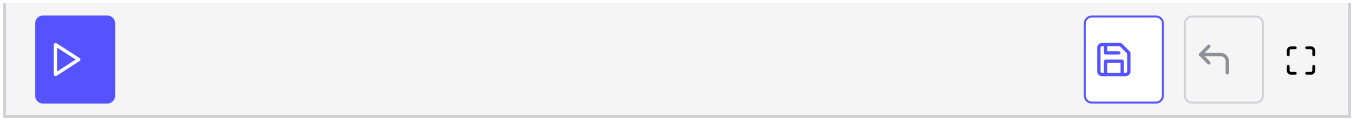


Run the code to see the compile error. `noImplicitAny` flag enabled.

Adding the type annotation fixes the error.

```
interface Person {
    name: string;
}

function sayHello(person: Person) {
  console.log(`Hello, ${person.name}`);
}
```



Run the code to verify there are no errors. `noImplicitAny` enabled.

Why is `noImplicitAny` a good thing? Having `person` typed as `any` essentially breaks type safety for the whole function. You cannot rely on TypeScript to check whether the reference to the `name` property is correct. While this example is trivial, imagine having a 100-line function; losing type-safety in such a huge chunk of code is a big deal.

Furthermore, `any` can propagate. In the example below, the return type of `getName` is inferred as `any`. This means that you can break type safety not only inside the function but also in the places it's being used.

```
function getName(person) {
  return person.name;
}
```

Hover over `getName` to see that it's type is inferred to `any`

In the next lesson, we'll discuss how to deal with type errors that materialize when `noImplicitAny` is enabled.