Namespace

Sharing code is a good practice. It allows you to build and test once, and then build on top of the code in many other projects. TypeScript uses modules and namespaces to share code, as well as definition file.

WE'LL COVER THE FOLLOWING

Namespace: A Fading Concept

The concept of namespaces in TypeScript is fading away with the second, more powerful way to share code: a module. A namespace is a basic object assigned to the global space. The object, named after the name of the namespace, holds the functions created within this one.

```
namespace myNamespace {
  // code here
}

// Access by using: myNamespace.
```

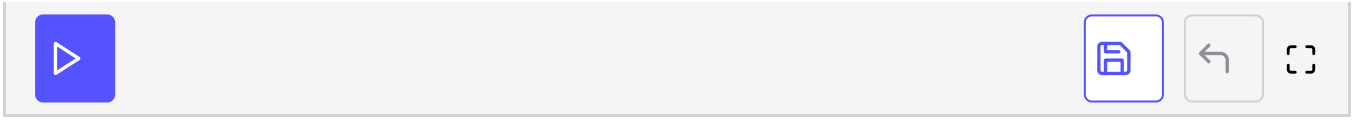
Goal of Using a Namespace

TypeScript allows setting code in the namespace across files as well. The goal of using namespaces is to structure your code into a logical category and to reduce name collision since the uniqueness is only true within one namespace. The following code shows `namespace1` defined twice. The code could be physically in a different location. The code above does not work. The reason is that while being on the same namespace, to access variables these ones need a special identification in front that you will see soon.

> Note: the below code throws an error

```
namespace namespace1 {
    let variable_ns1: number = 1;
}

namespace namespace1 {
    let variable_ns1_also: number = 2;
    console.log(variable_ns1);
}
```



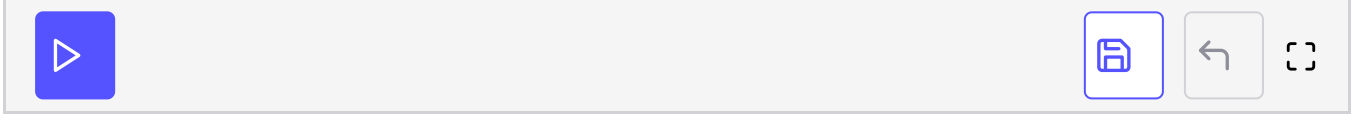
A namespace can contain interfaces, classes, types, functions, and variables. The keyword `export` opens access to outside the boundaries of a namespace. Otherwise, the element will be solely available within the namespace. It's important to notice that the content of a namespace is available across files even when not exported as long as it is within the same namespace.

```
namespace namespace1 {
    export let variable_ns1: number = 1;
}

namespace namespace1 {
    export let variable_ns1_also: number = 2;
}

namespace namespace2 {
    export let variable_ns2: number = 3;
}

console.log(namespace1.variable_ns1);
console.log(namespace1.variable_ns1_also);
console.log(namespace2.variable_ns2);
```



A solution is to embrace the concept of a module, which we will go on to cover in the next lesson.

Issue around Namespace

The issue around namespaces is that as your application grows, the global scope grows as well. It also doesn't offer a way to distribute outside or to break apart an area of your code which you could load dynamically. One

solution is to embrace the concept of a **module**, which we will go on to cover in the next lesson.