# Closures

A `function` can access all variables defined **inside** the function, like this

```
function myFunction() {
  let a = 4;
  return a * a;
}
```

But a `function` can also access variables defined **outside** the function, like this

```
let a = 4;
function myFunction() {
  return a * a;
}
```

---

## A callback function

is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

The consumer of a callback-based API writes a function that is passed into the API. The provider of the API (called the *caller*) takes the function and calls back (or executes) the function at some point inside the caller's body. The caller is responsible for passing the right parameters into the callback function. The caller may also expect a particular return value from the callback function, which is used to instruct further behavior of the caller.

---

## Higher Order Function?

A higher order function is a function that takes one or more functions as arguments, or returns a function as its result.

There are several different types of higher order functions like map and reduce. We will discuss these later in this tutorial. But before that let's first dive deep into what higher order functions are.

```
function callbackFunction(){
    console.log('I am  a callback function');
}

// higher order function
function higherOrderFunction(func){
    console.log('I am higher order function')
    func()
}

higherOrderFunction(callbackFunction);
```

- **Conditional rendering**: Conditionally render components based on certain logic, such as user authentication or permission checks. A HOC can determine whether a component should be displayed and then wrap components with this HOC to make rendering decisions based on certain conditions
- **Authentication**: Implement user authentication and authorization. A HOC can protect routes or components, ensuring that only authenticated users have access. You can create an AuthHOC that checks the user's authentication status and role. Wrap components or routes with this HOC to conditionally render contents based on user authentication and authorization
- **Data fetching**: Handle data fetching and loading states. A HOC can fetch data and pass it as props

to the wrapped component, handling loading and error states

- **Styling**: Apply CSS styles or themes to components. A HOC can pass styling information as props to customize the appearance of components
- **State management**: Manage and share state, such as global app state or Redux store data, with multiple components using a HOC
- **Logging and analytics**: Implement logging, error tracking, or analytics by wrapping components with a HOC that reports events or errors
- **Caching and memoization**: Cache expensive computations or memoize functions to improve performance by using a HOC