Solution

We can use closures to capture state required by the returned function. The returned function's closure has access to the following variables:

- `func`: The argument to `once()`, the callback function to be invoked.
- `ranOnce`: A boolean variable indicating if the callback function has been invoked before. If the value is `true`, `func` will not be invoked again.
- `value`: Since we need to return the value of the first invocation for subsequent invocations, we have to save the result of the first invocation and return it directly for subsequent invocations.

JavaScript TypeScript

```
/**
 * @callback func
 * @return {Function}
 */
export default function once(func) {
  let ranOnce = false;
  let value;

  return function (...args) {
    if (!ranOnce) {
      value = func.apply(this, args);
      ranOnce = true;
    }

    return value;
  };
}
```

Note that the callback function is invoked with the `this` binding and arguments of the created function, hence we cannot return an arrow function if we want the value of `this` to refer to the object calling the created function.

Edge cases

- Functions which access `this`.

Techniques

- Closures.
- How `this` works.
- Invoking functions via `Function.prototype.apply()` / `Function.prototype.call()`.