Solution

1. The `promisify` function takes a single argument `func`, which is the callback-based function you want to promisify.
2. The `return` statement returns a new function that wraps `func`. This new function is the promisified version.
3. Inside the returned function, we use the spread operator `...args` to capture any arguments passed to the promisified function.
4. We create a new `Promise` that wraps the original callback-based function. The Promise constructor takes a function with two arguments: `resolve` and `reject`. These are functions we call based on the outcome of the asynchronous operation.
5. Inside the Promise's function, we invoke `func` with the provided arguments `(...args)` and pass a callback function as its last argument as that's what `func` expects.
6. The callback function takes two arguments: `err` (error) and `result` (success value). If `err` is truthy, we reject the `Promise` with the `err`. Otherwise, we resolve the `Promise` with the `result`.

With the `promisify` function, you can convert any callback-based function into a `Promise`-based function, making it easier to work with asynchronous operations using modern Promise syntax.

To preserve the `this` value, the returned function should not be defined using arrow functions and `func` should be invoked with `call`/`apply` and the correct `thisArg` value.

JavaScript TypeScript

```javascript
/**
 * @callback func
 * @returns Function
 */
export default function promisify(func) {
  return function (...args) {
    return new Promise((resolve, reject) => {
      func.call(this, ...args, (err, result) =>
        err ? reject(err) : resolve(result),
      );
    });
  };
}
```

Notes

Notes

The `promisify` function assumes the callback is the last argument and that the callback uses an error-first format. If the function you are trying to promisify is not the last argument or has a different format, you cannot use this. Node.js provides a custom promisify function `util.promisify.custom` that you can use for such cases.

Not every function that accepts callbacks can/should be promisified! A promise can have only one result, but a callback can be called many times (e.g. `setInterval`). Hence promisification is only meant for functions that call the callback once because further calls will be ignored.