

PRACTICAL PROMISES

a pithy promise primer



WILL THIS WORK?

```
fs.readFile("meat.txt", function(err, data){  
  console.log(data)  
})
```

```
fs.readFile("pudding.txt", function(err, data){  
  console.log(data)  
})
```



WILL THIS WORK?

```
fs.readFile("meat.txt", function(err, data){
  console.log(data)

  fs.readFile("pudding.txt", function(err, data) {
    console.log(data)
  })
})
```



DON'T FORGET YOUR ERRORS

```
fs.readFile("meat.txt", function(err, data){
  if (err) {
    console.log('something went wrong:', err)
  } else {
    console.log(data)
  }
})

fs.readFile("pudding.txt", function(err, data) {
  if (err) {
    console.log('something went wrong:', err)
  } else {
    console.log(data)
  }
})
})
```

DON'T FORGET YOUR ERRORS

```
fs.readFile("meat.txt", function(err, data){
  if (err) console.log('something went wrong:', err)
  else console.log(data)

  fs.readFile("pudding.txt", function(err, data) {
    if (err) console.log('something went wrong:', err)
    else console.log(data)

    fs.readFile("afterDinnerMints.txt", function(err, data) {
      if (err) console.log('something went wrong:', err)
      else console.log(data)
    })
  })
})
```



DON'T FORGET YOUR ERRORS

```
fs.readFile("meat.txt", function(err, data){
  if (err) console.log('something went wrong:', err)
  else console.log(data)

  fs.readFile("pudding.txt", function(err, data) {
    if (err) console.log('something went wrong:', err)
    else console.log(data)

    fs.readFile("afterDinnerMints.txt", function(err, data) {
      if (err) console.log('something went wrong:', err)
      else console.log(data)

      fs.readFile("iDunnoGoToDiscoIGuess.txt", function(err, data) {
        if (err) console.log('something went wrong:', err)
        else console.log(data)
      })
    })
  })
})
```

A popular internet meme featuring Grumpy Cat. The cat is sitting on a light blue fabric surface, looking directly at the camera with its characteristic grumpy expression. The text "I HATE JAVASCRIPT" is overlaid at the top in a large, bold, white font with a black outline. In the bottom right corner, the text "memegenerator.net" is visible in a small, white font.

WHAT IS A CALLBACK?



WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors...

- **Blocking**
- **Non-blocking**



BLOCKING CALLBACKS

predicates

e.g. `arr.filter(function predicate (elem) {...});`

comparators

e.g. `arr.sort(function comparator (elemA, elemB) {...});`

iterators

e.g. `arr.map(function iterator (elem) {...});`



NON-BLOCKING CALLBACKS

think: control flow



WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors...

- **Blocking**
- **Non-blocking**



WHAT IS A CALLBACK?

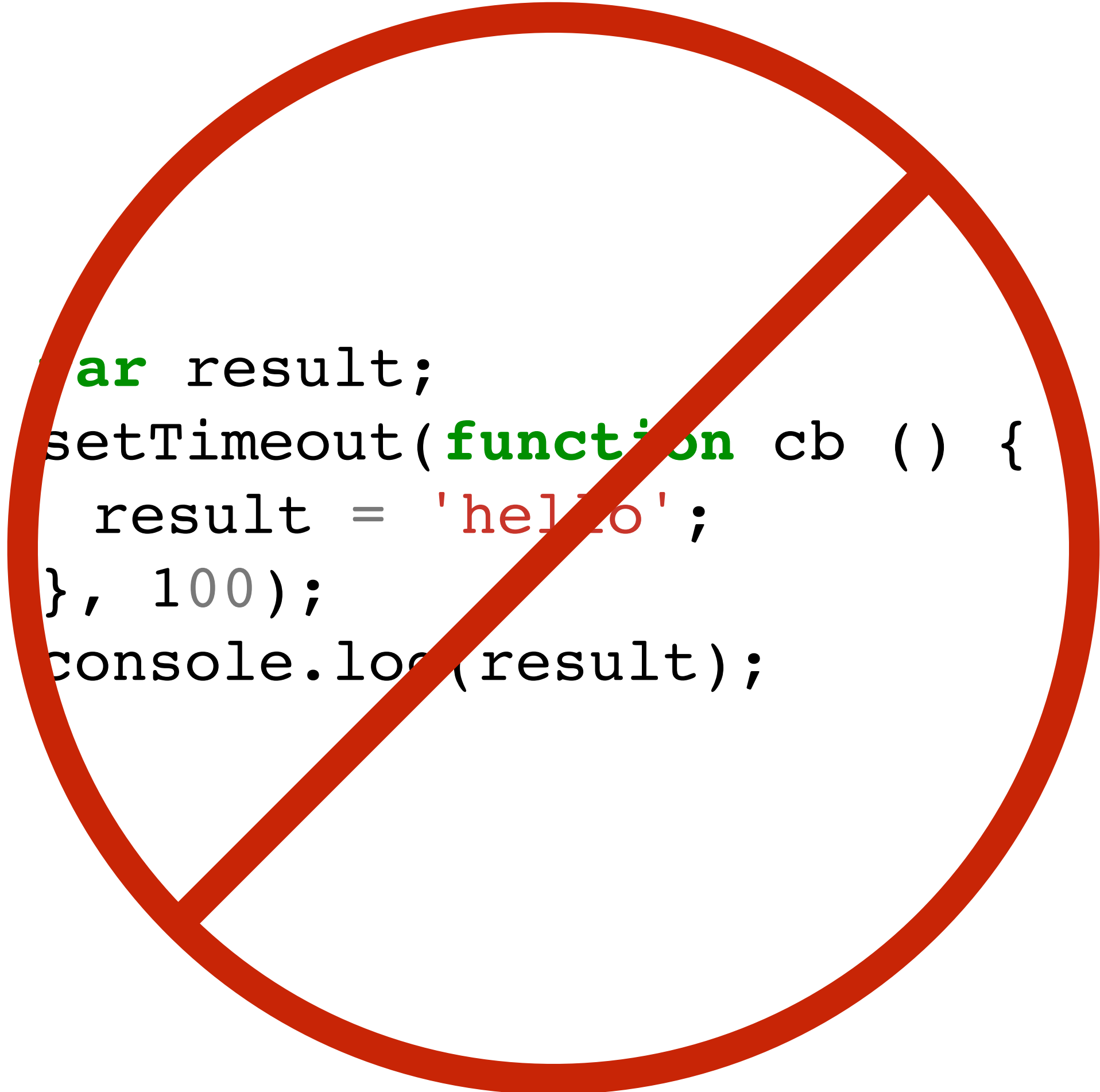
Technically: a function passed to another function

two flavors...

- ◉ Blocking
- ◉ Non-blocking
 - ◉ event handler
 - ◉ middleware
 - ◉ **vanilla async**



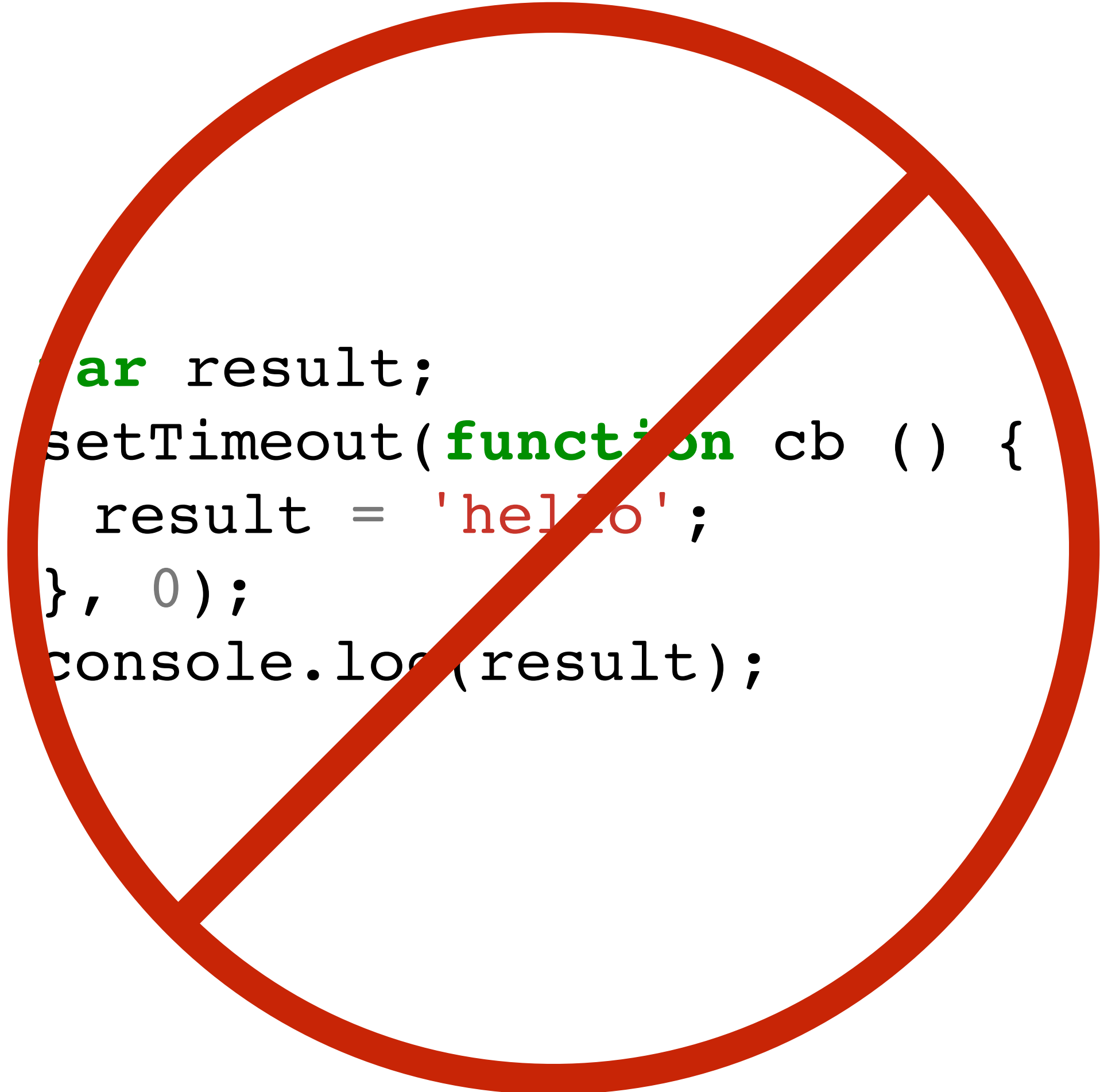
LIKE THIS?



```
var result;  
setTimeout(function cb () {  
  result = 'hello';  
}, 100);  
console.log(result);
```



LIKE THIS?




```
var result;  
setTimeout(function cb () {  
  result = 'hello';  
}, 0);  
console.log(result);
```



LIKE THIS?

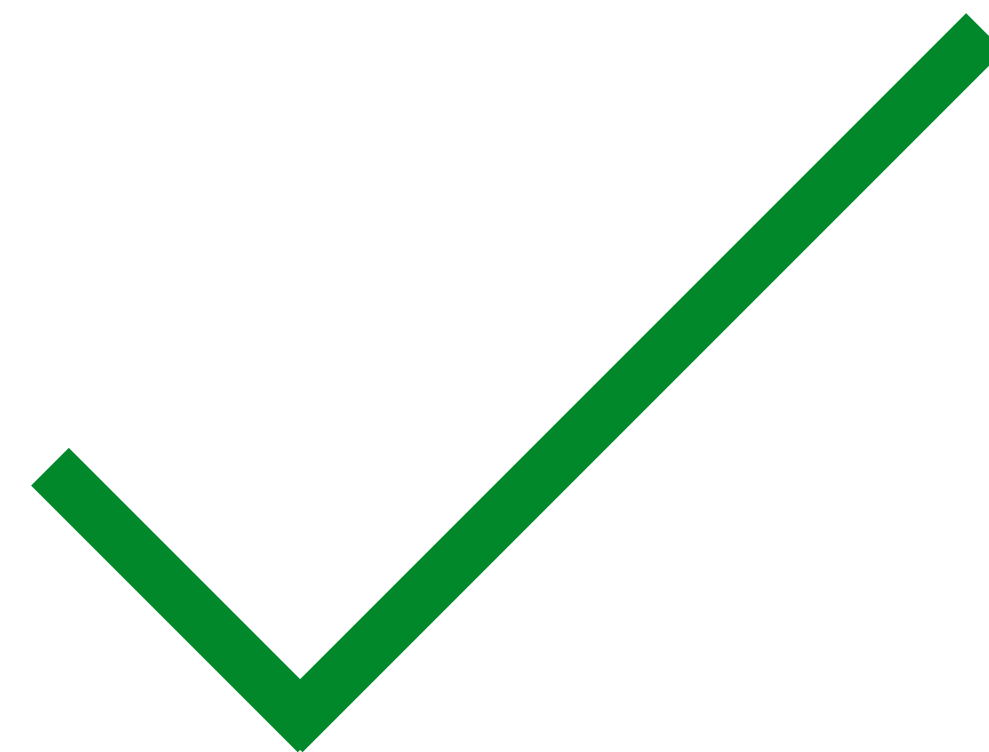
```
var result = setTimeout(function cb () {  
  return 'hello';  
}, 0);  
console.log(result);
```





LIKE THIS?

```
setTimeout(function cb () {  
  var result = 'hello';  
  console.log(result);  
}, 0);
```





PROMISE

“A promise represents the eventual result of an asynchronous operation.”

— THE PROMISES/A+ SPEC



PROMISE

A promise contains some data that you will eventually get



HOW TO GET THAT ASYNC DATA

- A promise implements a method called *then*
- *Promise.then* accepts two callback functions as arguments, one for success, and one for errors.
- Once the promise has the data (or an error), it will invoke the appropriate function



CALLBACK V PROMISE

vanilla async **callback**

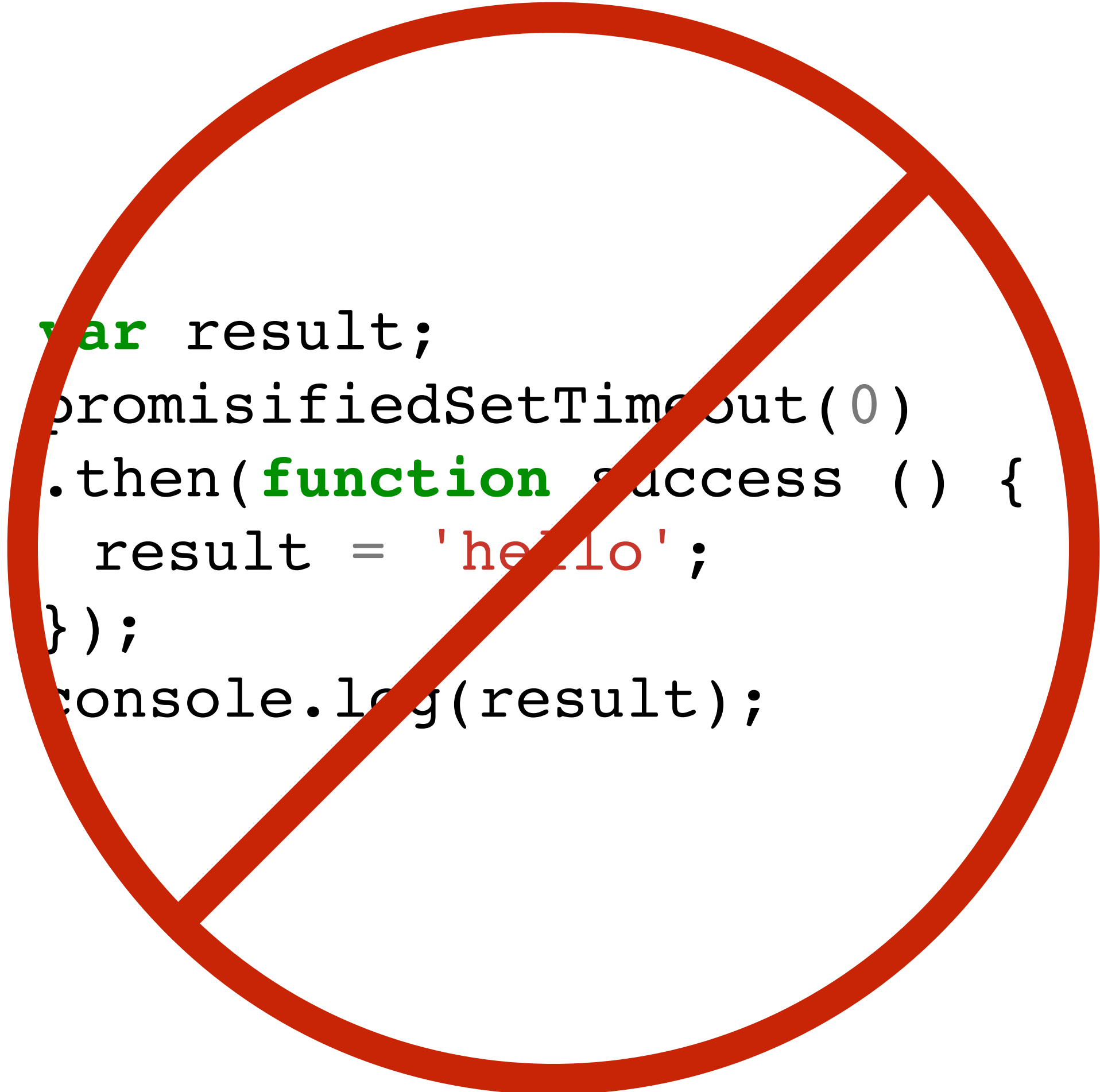
```
fs.readFile('file.txt', function callback (err, data) {  
  
  }  
);
```

async **promise**

```
const myPromise = fs.readFileAsync('file.txt')  
  
myPromise.then(  
  function onSuccess (data) {...},  
  function onError (err) {...}  
);
```



LIKE THIS?




```
var result;  
promisifiedSetTimeout(0)  
.then(function success () {  
  result = 'hello';  
});  
console.log(result);
```



LIKE THIS?

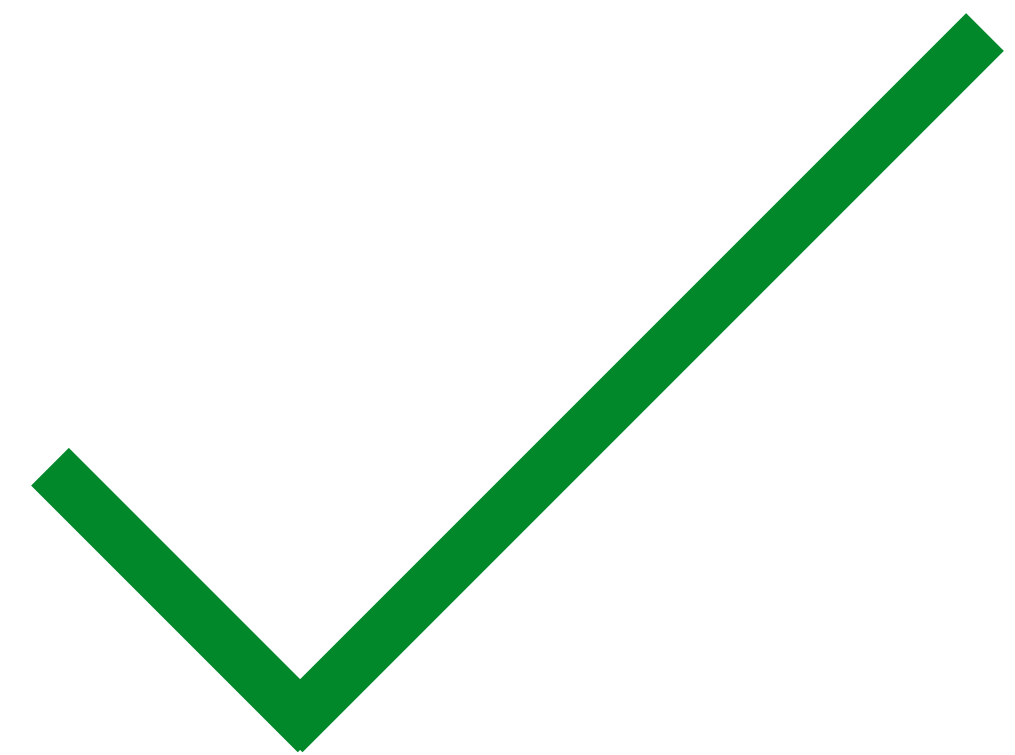
```
var result = promisifiedSetTimeout(0)
  .then(function success () {
    return 'hello';
  });
console.log(result);
```





LIKE THIS?

```
promisifiedSetTimeout(0)
  .then(function success () {
    var result = 'hello';
    console.log(result);
  });
```





**OKAY, LIKE...SO WHAT?
WE STILL NEED CALLBACKS**

PORTABLE

```
let result;

fs.readFile('meat.txt', function(err, data) {
  result = data;
})

// later on... can I do this?
doSomething(result)

// can I export the result?
module.exports = result;
```

PORTABLE

```
const promise = fs.readFileAsync('meat.txt');  
  
// later on, can I do this?  
doSomething(promise);  
  
// can I export the promise?  
module.exports = promise;
```

MULTIPLE HANDLERS

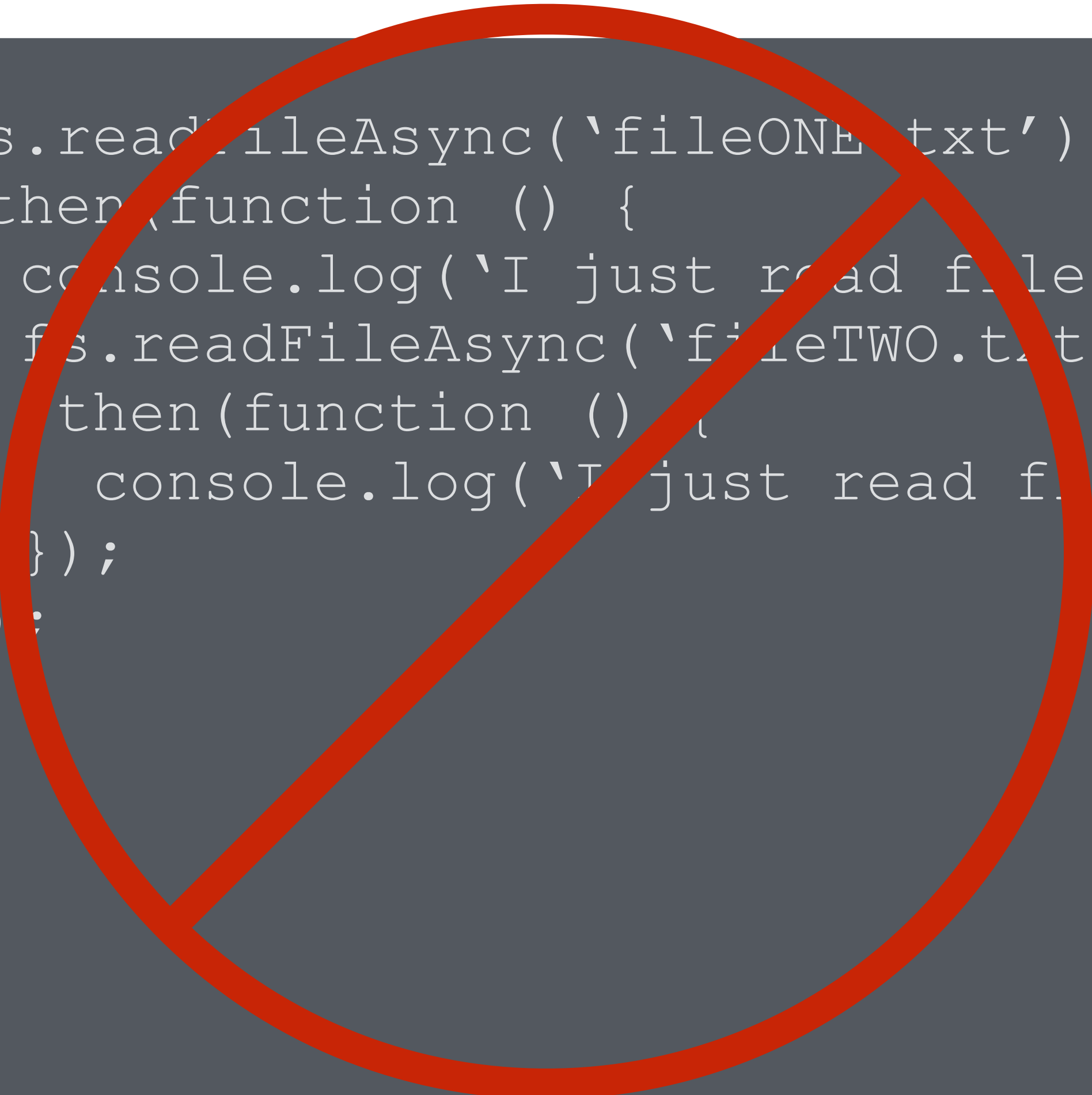
```
const promise = fs.readFileAsync('file.txt');

// do one thing when it finishes
promise
  .then(function (fileContents) {
    console.log(fileContents);
  });

// do another thing when it finishes
promise
  .then(function () {
    // do something else
  });
```

LINEAR/FLAT

```
fs.readFileAsync('fileONE.txt')
  .then(function () {
    console.log('I just read file one');
    fs.readFileAsync('fileTWO.txt')
      .then(function () {
        console.log('I just read file two');
      });
  });
```



LINEAR/FLAT

```
fs.readFileAsync('fileONE.txt')
  .then(function (contents) {
    console.log('I just read file one:', contents);
    return fs.readFileAsync('fileTWO.txt');
  })
  .then(function (contents) {
    console.log('I just read file two:', contents);
  });
```



UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
  .then(function (contents) {
    console.log('I just read file one:', contents);
    return fs.readFileAsync('fileTWO.txt');
  })
  .then(function (contents) {
    console.log('I just read file two:', contents);
    return fs.readFileAsync('fileTHREE.txt')
  })
  .then(null, function (err) {
    console.log('An error occurred at some point');
    console.log(err);
  });
```

UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
  .then(function (contents) {
    console.log('I just read file one:', contents);
    return fs.readFileAsync('fileTWO.txt');
  })
  .then(function (contents) {
    console.log('I just read file two:', contents);
    return fs.readFileAsync('fileTHREE.txt')
  })
  .catch(function(err) {
    console.log('An error occurred at some point');
    console.log(err);
  });
```




UNIFIED ERROR HANDLING

```
fs.readFile("meat.txt", function(err, data){
  if (err) console.log('something went wrong:', err)
  else console.log(data)

  fs.readFile("pudding.txt", function(err, data) {
    if (err) console.log('something went wrong:', err)
    else console.log(data)

    fs.readFile("afterDinnerMints.txt", function(err, data) {
      if (err) console.log('something went wrong:', err)
      else console.log(data)

      fs.readFile("iDunnoGoToDiscoIGuess.txt", function(err, data) {
        if (err) console.log('something went wrong:', err)
        else console.log(data)
      })
    })
  })
})
```

PROMISE ADVANTAGES

- **Portable**
- **Multiple handlers**
- **“Linear” or “flat” chains**
- **Unified error handling**

READING A FILE

SYNCHRONOUS

```
var path = 'demo-poem.txt';
console.log('- I am first -');
var buff = fs.readFileSync(path);
console.log(buff.toString());
}
console.log('- I am last -');
```

ASYNC (CALLBACKS)

```
var path = 'demo-poem.txt';
fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});
console.log('- I am first -');
```

ASYNC (PROMISES)

```
var path = 'demo-poem.txt';
promisifiedReadFile(path)
  .then(function (buff) {
    console.log(buff.toString())
  })
  .then(function () {
    console.log('- I am last -');
  });

console.log('- I am first -');
```

READING A FILE

SYNCHRONOUS

```
var path = 'demo-poem.txt';
console.log('- I am first -');
var buff = fs.readFileSync(path);
console.log(buff.toString());
}
console.log('- I am last -');
```

ASYNC (CALLBACKS)

```
var path = 'demo-poem.txt';
fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});
console.log('- I am first -');
```

ASYNC (PROMISES)

```
var path = 'demo-poem.txt';
promisifiedReadFile(path)
  .then(function (buff) {
    console.log(buff.toString())
  })
  .then(function () {
    console.log('- I am last -');
  });

console.log('- I am first -');
```



IMPLEMENTATIONS

- Aدهun
- avow
- ayepromise
- bloodhound
- bluebird
- broody-promises
- CodeCatalyst
- Covenant
- D
- Deferred
- fate
- ff
- FidPromise
- ipromise
- Legendary
- Lie
- microPromise
- mpromise
- Naive Promesse
- Octane
- ondras
- potch
- P
- Pacta
- Pinky
- PinkySwear
- Potential
- promeso
- promiscuous
- Promis
- Promix
- Promiz
- Q
- rsvp
- Shvua
- Ten.Promise
- then
- ThenFail
- typescript-deferred
- vow
- when
- yapa
- yapi
- Zousan