

PRACTICAL PROLONGED PROCESS PROGRAMMING

a prompt promise primer

WHAT IS A CALLBACK?



WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors...

- Blocking
- Non-blocking



BLOCKING CALLBACKS

think: portable code

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



NON-BLOCKING CALLBACKS

think: control flow

event handlers

e.g. `button.on('click', function handler (data) {...});`



NON-BLOCKING CALLBACKS

think: control flow

event handlers

e.g. `button.on('click', function handler (data) {...});`

middleware

e.g. `app.use(function middleware (... , next) {...});`



NON-BLOCKING CALLBACKS

think: control flow

event handlers

e.g. `button.on('click', function handler (data) {...});`

middleware

e.g. `app.use(function middleware (... , next) {...});`

vanilla async callback

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



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**

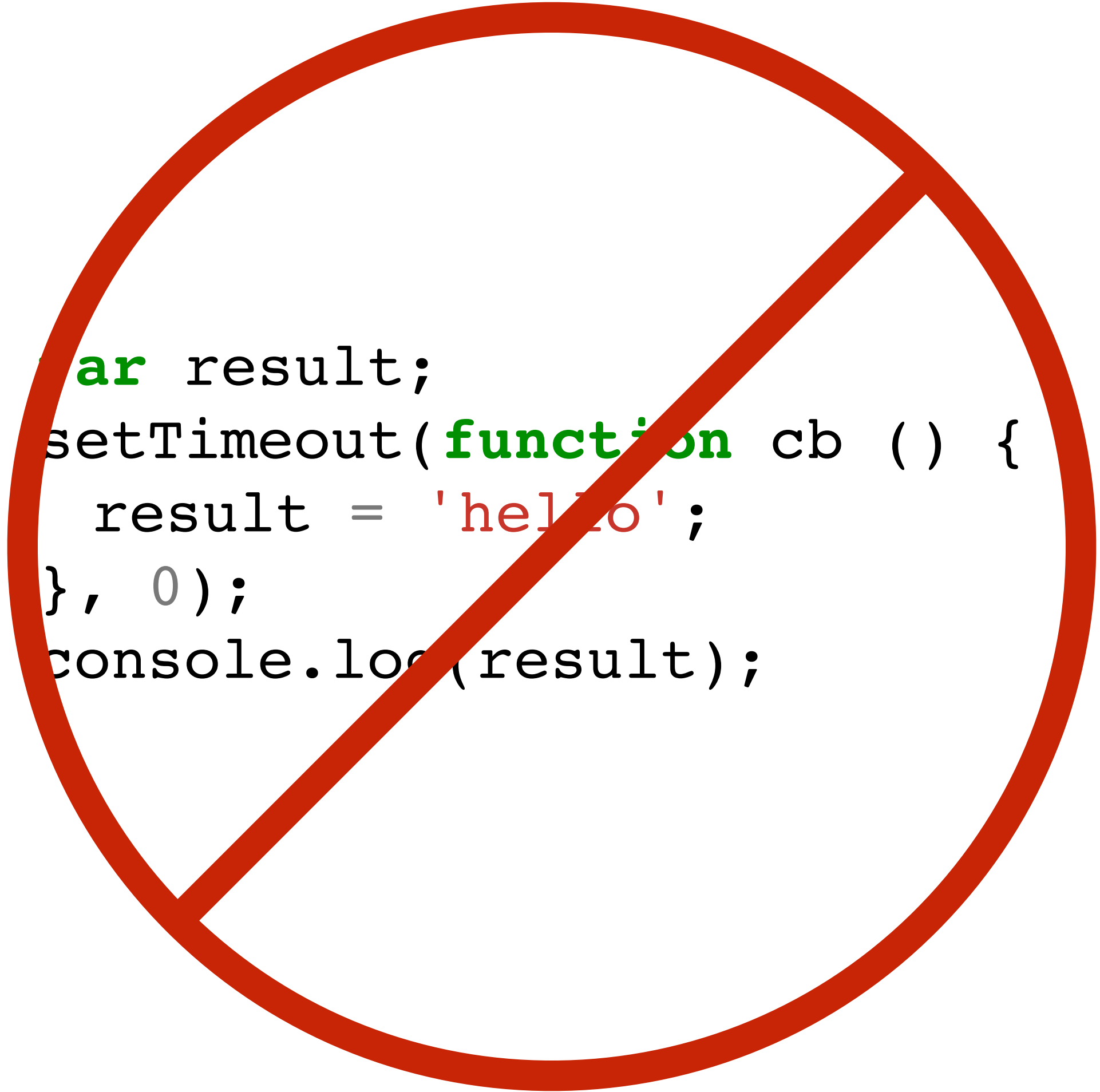


LIKE THIS?

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

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



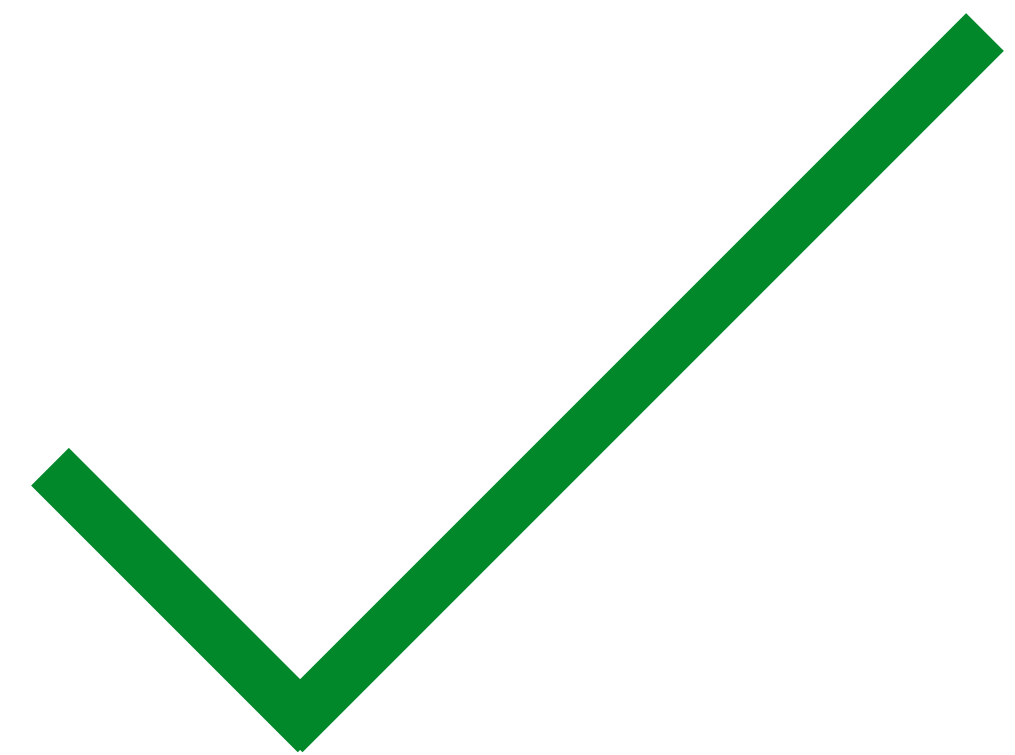
LIKE THIS?

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



LIKE THIS?

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





PROMISE



PROMISE

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

— THE PROMISES/A+ SPEC



CALLBACK V PROMISE

vanilla async **callback**

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

async **promise**

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

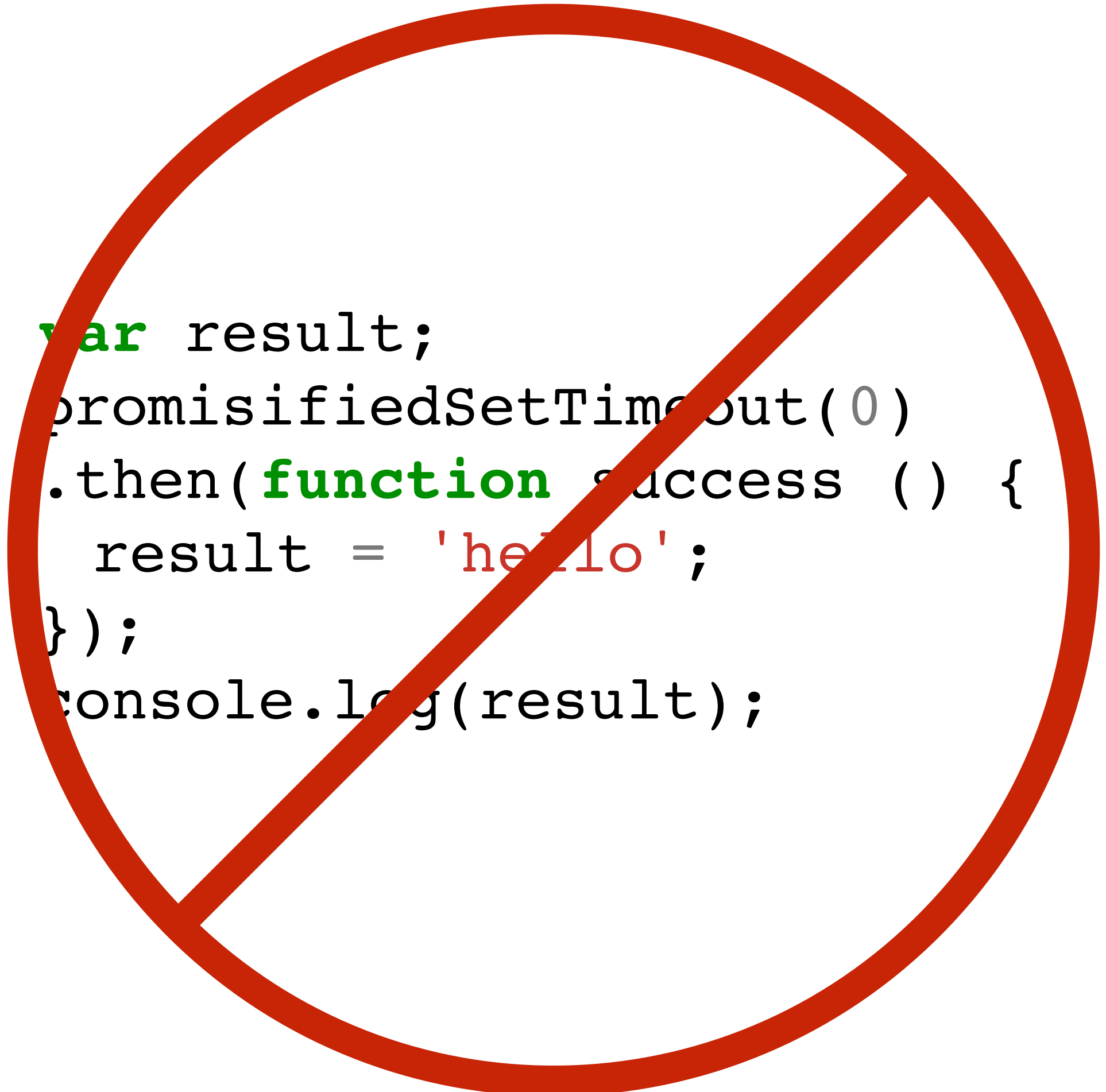


LIKE THIS?

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



LIKE THIS?



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




LIKE THIS?

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



LIKE THIS?

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





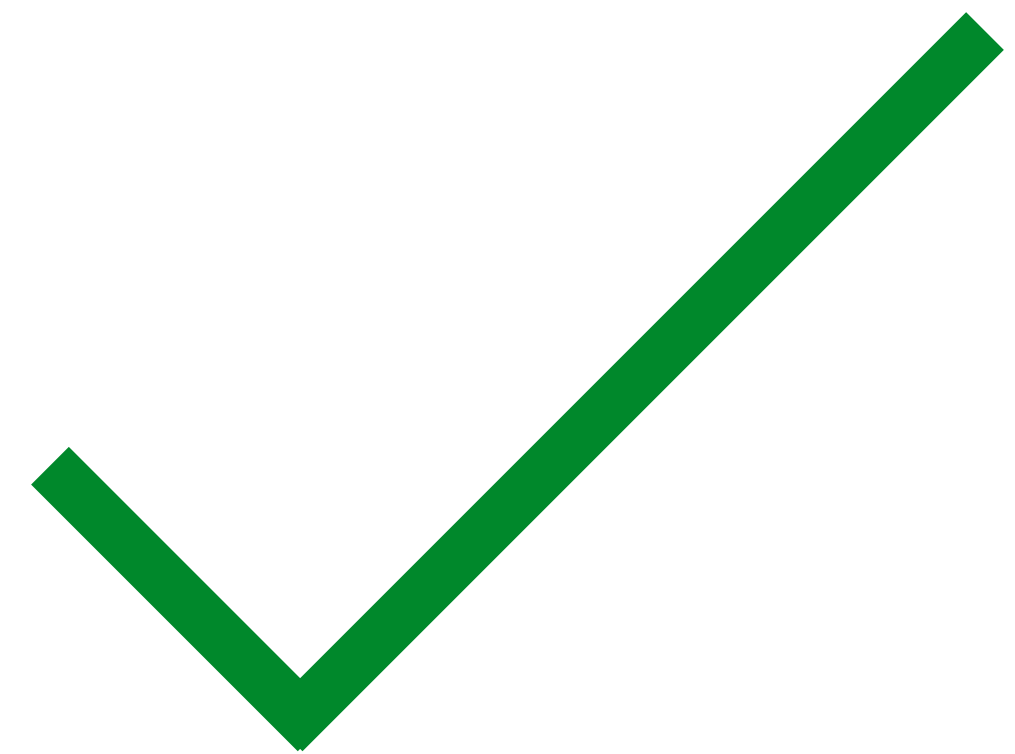
LIKE THIS?

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




LIKE THIS?

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



READING A FILE

SYNCHRONOUS

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

READING A FILE

SYNCHRONOUS

```
var path = 'demo-poem.txt';
console.log('- I am first -');
try {
  var buff = fs.readFileSync(path);
  console.log(buff.toString());
} catch (err) {
  console.error(err);
}
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 -');
```

READING A FILE

SYNCHRONOUS

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

PROMISE ADVANTAGES

- Portable
- Multiple handlers
- Unified error handling
- “Linear”



IMPLEMENTATIONS

- ◉ Adehun
- ◉ 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
- ◉ promeso
- ◉ promiscuous
- ◉ Promis
- ◉ Promix
- ◉ Promiz
- ◉ Q
- ◉ rsvp
- ◉ Shvua
- ◉ Ten.Promise
- ◉ then
- ◉ ThenFail
- ◉ typescript-deferred
- ◉ vow
- ◉ when
- ◉ yapa
- ◉ yapi
- ◉ Zousan



IMPLEMENTATIONS

- Adehun
- 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
- promeso
- promiscuous
- Promis
- Promix
- Promiz
- Q
- rsvp
- Shvua
- Ten.Promise
- then
- ThenFail
- typescript-deferred
- vow
- when
- yapa
- yapi
- Zousan