Class 15: Promises

6.102 — Software Construction
Spring 2024

Warmup

ex:  yellkey.com/**term**

nq:  yellkey.com/**staff**

open `bank.ts`

`npm run bank` should print `you have 200 dollars in your account`

then follow the TODO to refactor it to use promises

- first change `fs.readFileSync` to `fs.promises.readFile`
- then fix the code to use `Promise`, `async`, `await`

`npm run bank` should still print `you have 200 dollars in your account`

We've refactored `getBalance()`, and the program compiles with no static errors
...but we haven't refactored `main()` yet:

```
async function getBalance():Promise<number> {
  const data: string = await fs.promises.readFile('savings',
                                                  { encoding: 'utf-8' });
  return parseInt(data);
}

function main():void {
  console.log('you have', getBalance(), 'dollars in your savings account');
}

main();
```

If we run the program now, what does `main()` print?

A. 200
B. Promise < 200 >
C. Promise < pending >
D. an error is thrown

Fetching web pages

open `web.ts`

`npm run web` to see it in action

refactor the code so that the web page downloads interleave

- **don't** change `download()`, just the code that calls it
- hint: call `download` on every url before doing any `await`

Fetching web pages

open `web.ts`

`npm run web` to see it in action

refactor the code so that the web page downloads interleave

- **don't** change `download()`, just the code that calls it
- hint: call `download` on every url before doing any `await`
- hint: use `map`

Fetching web pages

open `web.ts`

`npm run web` to see it in action

refactor the code so that the web page downloads interleave

- **don't** change `download()`, just the code that calls it
- hint: call `download` on every url before doing any `await`
- hint: use `map`
- hint: use `Promise.all`

Interleaving

open `fact.ts`

`npm run fact` to see it in action

- do 99! and 100! interleave?
  - or do they look completely synchronous, even though `factorial()` is `async`?
- when can an `async` function give up control to other concurrent computations?

Interleaving

open `fact.ts`

`npm run fact` to see it in action

- do 99! and 100! interleave?
  - or do they look completely synchronous, even though `factorial()` is `async`?
- when can an `async` function give up control to other concurrent computations?

try to change `factorial` so that 99! and 100! interleave

- hint: use `timeout(0)`