Reading Code

Martin Kellogg

Reading code

Today’s agenda:

- Why does reading code matter?
- Strategies for reading code effectively
- Role of documentation
- Examples from Covey.Town

But first: reading quiz!

Q1: In IP1, you'll implement which of these classic games?

A. Connect4
B. Tic-Tac-Toe
C. Snakes and Ladders
D. None of these

Q2: **TRUE** or **FALSE**: one of the articles includes a surprising analogy to the classic carnival game of guessing how many jellybeans are in a jar without actually counting them

But first: reading quiz!

Q1: In IP1, you’ll implement which of these classic games?

A. Connect4

[✓ **B.** Tic-Tac-Toe]

C. Snakes and Ladders

D. None of these

Q2: **TRUE** or **FALSE**: one of the articles includes a surprising analogy to the classic carnival game of guessing how many jellybeans are in a jar without actually counting them

But first: reading quiz!

Q1: In IP1, you'll implement which of these classic games?

A. Connect4
B. Tic-Tac-Toe
C. Snakes and Ladders
D. None of these

Q2: **TRUE** or **FALSE**: one of the articles includes a surprising analogy to the classic carnival game of guessing how many jellybeans are in a jar without actually counting them

Reading code

Today’s agenda:

- **Why does reading code matter?**
- Strategies for reading code effectively
- Role of documentation
- Examples from Covey.Town

Why does reading code matter?

- Research suggests that developers spend **up to 70%** [1, 2] of their time trying to understand what code does

[1] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In Intl. Conf. on Prog. Compr. (ICPC

[2] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. Trans. on Soft. Eng. (TSE) 44, 10 (2018), 951–976

Why does reading code matter?

- Research suggests that developers spend **up to 70%** [1, 2] of their time trying to understand what code does
  - This is a **huge** amount of time? Why?

[1] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In Intl. Conf. on Prog. Compr. (ICPC

[2] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. Trans. on Soft. Eng. (TSE) 44, 10 (2018), 951–976

Why does reading code matter?

- Research suggests that developers spend **up to 70%** [1, 2] of their time trying to understand what code does
  - This is a **huge** amount of time? Why?
- Code understanding is a **necessary precursor** for most other development activities:
  - debugging, testing, using an API, adding new features, etc.

[1] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In Intl. Conf. on Prog. Compr. (ICPC

[2] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. Trans. on Soft. Eng. (TSE) 44, 10 (2018), 951–976

Why does reading code matter?

- Research suggests that developers spend **up to 70%** [1, 2] of their time trying to understand what code does
  - This is a **huge** amount of time? Why?
- Code understanding is a **necessary precursor** for most other development activities:
  - debugging, testing, using an API, adding new features, etc.
- Most "code reading" is done **in service to some other goal**
  - i.e., a developer reads code because they want to add a new feature, fix a bug, etc.; not for its own sake

[1] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In Intl. Conf. on Prog. Compr. (ICPC

[2] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. Trans. on Soft. Eng. (TSE) 44, 10 (2018), 951–976

Why does reading code matter?

- Research suggests that developers spend **up to 70%** [1, 2] of their time trying to understand what code does
  - This is a **huge** amount of time? Why?
- Code understanding is a **necessary precursor** for most other development activities:
  - debugging, testin c.
- Most “code reading” i
  - i.e., a developer re ew feature, fix a bug, etc.; not for its own sake

My advice: Keep the goal in mind whenever you’re reading code. It’s easy to spend a long time looking at an irrelevant part of the system!

[1] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In Intl. Conf. on Prog. Compr. (ICPC

[2] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. Trans. on Soft. Eng. (TSE) 44, 10 (2018), 951–976

Auxiliary benefits of reading code

Auxiliary benefits of reading code

- Reading code can help **build your intuition** for a system
  - makes it easier to diagnose problems, find what you’re looking for in the codebase, explain it to others…

Auxiliary benefits of reading code

- Reading code can help **build your intuition** for a system
  - makes it easier to diagnose problems, find what you’re looking for in the codebase, explain it to others…
- Reading code can help you **reduce duplication**
  - “Oh, we don’t need to write this—there’s already something similar/the same in another file.”

Auxiliary benefits of reading code

- Reading code can help **build your intuition** for a system
  - makes it easier to diagnose problems, find what you’re looking for in the codebase, explain it to others…
- Reading code can help you **reduce duplication**
  - “Oh, we don’t need to write this—there’s already something similar/the same in another file.”
- Reading code can **make you a better developer**, especially if you’re reading good code
  - when writing code, try to emulate the best code you’ve read!

Auxiliary benefits of reading code

- Reading code can help **build your intuition** for a system
  - makes it at you're looking s…
- Reading coc
  - "Oh, we dy something similar/the same in another file."
- Reading code can **make you a better developer**, especially if you're reading good code
  - when writing code, try to emulate the best code you've read!

Foreshadowing: the benefits of reading code are also one of the main advantages of "modern" code review, which we'll discuss later

Why do we need code reading strategies?

Why do we need code reading strategies?

- Real software systems are **too large** for you to read the whole codebase (especially when we include dependencies)
  - e.g., many real codebases are 100k+ lines

Why do we need code reading strategies?

- Real software systems are **too large** for you to read the whole codebase (especially when we include dependencies)
    - e.g., many real codebases are 100k+ lines
    - it's not realistic for you to hold all of the behaviors of the code in your head at the same time

Why do we need code reading strategies?

- Real software systems are **too large** for you to read the whole codebase (especially when we include dependencies)
  - e.g., many real codebases are 100k+ lines
  - it's not realistic for you to hold all of the behaviors of the code in your head at the same time
- To be productive in such a codebase, you need to be capable of making changes without having read all of the code

Why do we need code reading strategies?

- Real software systems are **too large** for you to read the whole codebase (especially when we include dependencies)
  - e.g., many real codebases are 100k+ lines
  - it’s not realistic for you to hold all of the behaviors of the code in your head at the same time
- To be productive in such a codebase, you need to be capable of making changes without having read all of the code
  - implication: you need strategies for figuring out which parts of the code are actually **important** to read for the task at hand!

Strategies for reading code

Strategies for reading code

- “***Bottom-up***” code comprehension:

Strategies for reading code

- “***Bottom-up***” code comprehension:
    - start with low-level syntax that you understand

Strategies for reading code

- “***Bottom-up***” code comprehension:
  - start with low-level syntax that you understand
  - reason through what the code **actually does**

Strategies for reading code

- "*Bottom-up*" code comprehension:
  - start with low-level syntax that you understand
  - reason through what the code **actually does**
  - build a higher-level abstraction of the code's **purpose** that you can use to reason about how it's used

Strategies for reading code

- "***Bottom-up***" code comprehension:
  - start with low-level syntax that you understand
  - reason through what the code **actually does**
  - build a higher-level abstraction of the code's **purpose** that you can use to reason about how it's used
- This technique is best when you have **no preconceived notions** about what the code is for
  - cf. reading a code example on an exam

Strategies for reading code

- "*Bottom-up*" code comprehension:
  - start with low-level syntax that you understand
  - reason through what the code **actually does**
  - build a higher-level abstraction of the code's **purpose** that you can use to reason about how it's used
- This technique is best when you have **no preconceived notions** about what the code is for
  - cf. reading a code example on an exam
- Useful when you're unfamiliar with the code's application domain

Strategies for reading code

- When you’re already familiar with an application domain, you can instead use a “**top-down**” comprehension approach:

Strategies for reading code

- When you’re already familiar with an application domain, you can instead use a “**top-down**” comprehension approach:
  - look for familiar structures from the application domain, and scaffold your understanding around them

Strategies for reading code

- When you’re already familiar with an application domain, you can instead use a “**top-down**” comprehension approach:
  - look for familiar structures from the application domain, and scaffold your understanding around them
    - e.g., if you know there must be a database write, you could go looking for that

Strategies for reading code

- When you’re already familiar with an application domain, you can instead use a “**top-down**” comprehension approach:
  - look for familiar structures from the application domain, and scaffold your understanding around them
    - e.g., if you know there must be a database write, you could go looking for that
  - this technique requires you to have some idea of what you’re looking for, though

Strategies for reading code

- In practice, you’ll often want something in-between the two extremes of bottom-up and top-down comprehension

Strategies for reading code

- In practice, you'll often want something in-between the two extremes of bottom-up and top-down comprehension
- Here is one helpful strategy:

Strategies for reading code

- In practice, you'll often want something in-between the two extremes of bottom-up and top-down comprehension
- Here is one helpful strategy:
    - study what the code's purpose is long enough to **identify one important thing** that it does that you'll recognize
        - e.g., producing output

Strategies for reading code

- In practice, you’ll often want something in-between the two extremes of bottom-up and top-down comprehension
- Here is one helpful strategy:
  - study what the code’s purpose is long enough to **identify one important thing** that it does that you’ll recognize
    - e.g., producing output
  - then, search the code for that thing and use it as an **anchor**

Strategies for reading code

- In practice, you'll often want something in-between the two extremes of bottom-up and top-down comprehension
- Here is one helpful strategy:
    - study what the code's purpose is long enough to **identify one important thing** that it does that you'll recognize
        - e.g., producing output
    - then, search the code for that thing and use it as an **anchor**
    - trace the code backwards from there using a bottom-up strategy

What about documentation?

What about documentation?

- Most software systems come with some ***documentation***: written material describing the functionality and/or purpose of the system
  - e.g., user manuals, design documents, READMEs

What about documentation?

- Most software systems come with some ***documentation***: written material describing the functionality and/or purpose of the system
  - e.g., user manuals, design documents, READMEs
- Documentation is useful for building up a **high-level model** of what a system is supposed to do
  - in particular, you can use it find **anchor** behaviors!

What about documentation?

- Most software systems come with some ***documentation***: written material describing the functionality and/or purpose of the system
  - e.g., user manuals, design documents, READMEs
- Documentation is useful for building up a **high-level model** of what a system is supposed to do
  - in particular, you can use it find **anchor** behaviors!
- However, documentation is limited: it is often **not up to date** vis-a-vis the code (usually because someone forget to update docs)
  - code is the source of truth about what the system actually does

What about documentation?

- Most software systems come with some ***documentation***: written material describing the functionality and/or purpose of the system
  - e.g., user manuals, design documents, READMEs
- Documentation is u … at a system is suppos
  - in particular, y
- However, docume … vis-a-vis the code (…)
  - code is the source of truth about what the system actually does

My advice: Trust documentation until you see evidence that it's wrong. But, always be willing to dive into the code if there is an inconsistency between docs and the behavior that you observe. Think critically!

“Self-documenting” code

- Some engineers advocate for “***self-documenting***” code—that is, code that follows **naming conventions** and **standard structures** like those we discussed in the last lecture to such an extent that no external documentation is necessary

“Self-documenting” code

- Some engineers advocate for “***self-documenting***” code—that is, code that follows **naming conventions** and **standard structures** like those we discussed in the last lecture to such an extent that no external documentation is necessary
- Whether this is possible for real systems is still an **open question**

“Self-documenting” code

- Some engineers advocate for “***self-documenting***” code—that is, code that follows **naming conventions** and **standard structures** like those we discussed in the last lecture to such an extent that no external documentation is necessary
- Whether this is possible for real systems is still an **open question**
- One major criticism of this approach is that self-documenting code might explain *what* it does, but does not explain *why*

“Self-documenting” code

- Some engineers advocate for “*self-documenting*” code—that is, code that follows **naming conventions** and **standard structures** like those we discussed in the last lecture to such an extent that no external documentation is necessary
- Whether this is possible for real systems is still an **open question**
- One major criticism of this approach is that self-documenting code might explain *what* it does, but does not explain *why*
    - i.e., documentation is necessary to explain the **rationale** for design decisions, what the intended use-case is, etc.

“Self-documenting” code

- Some engineers advocate for “***self-documenting***” code—that is, code that follows **naming conventions** and **standard structures** like those we discussed in the last lecture to such an extent that no external docume
- Whether this is p
- One major critici might explain *wh*
  - i.e., documen design decisions, what the intended use-case is, etc.

My advice: Following the best practices we talked about in the code-level design lecture gives most of the benefits of self-documenting code anyway. Use documentation to explain **rationale/why**, not what the code does (assume other devs know how to read code, too).

Example: how do tile maps work in covey.town?

- Suppose that for a course project, we’re interested in making some kind of modification to the “main map” of covey.town
    - this could be modifying the layout, adding a new area, etc.
- Let’s figure out how we would do something like this together!

Example: how does async/await work?

- Suppose that we want to modify how Town.ts (in the backend) adds a player to the town

Example: how does async/await work?

- Suppose that we want to modify how Town.ts (in the backend) adds a player to the town
  - it’s an “async” function
    - what does that mean?

async and promises

- Typescript maintains a pool of processes, called ***promises***

async and promises

- Typescript maintains a pool of processes, called ***promises***
- A promise always executes until it is completed
    - This is called "***run-to-completion semantics***"

async and promises

- Typescript maintains a pool of processes, called ***promises***
- A promise always executes until it is completed
  - This is called "***run-to-completion semantics***"
- A promise can create other promises to be added to the pool

async and promises

- Typescript maintains a pool of processes, called ***promises***
- A promise always executes until it is completed
  - This is called "***run-to-completion semantics***"
- A promise can create other promises to be added to the pool
- Promises interact mostly by passing values to one another

async and promises

- Typescript maintains a pool of processes, called ***promises***
- A promise always executes until it is completed
    - This is called "***run-to-completion semantics***"
- A promise can create other promises to be added to the pool
- Promises interact mostly by passing values to one another
    - minimizes ***data races*** (a data race occurs when two instructions from different processes access the same memory location, and at least one of them is a write)

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency
    - Consider: a 1Ghz CPU executes an instruction every 1 ns

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency
  - Consider: a 1Ghz CPU executes an instruction every 1 ns
  - Almost anything else takes forever (approximately)

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency
    - Consider: a 1Ghz CPU executes an instruction every 1 ns
    - Almost anything else takes forever (approximately)
        - e.g., 150,000 ns to read from the SSD, at least 100 million ns just to ping a nearby server over the internet

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency
    - Consider: a 1Ghz CPU executes an instruction every 1 ns
    - Almost anything else takes forever (approximately)
        - e.g., 150,000 ns to read from the SSD, at least 100 million ns just to ping a nearby server over the internet
- Utilize this “wasted” time by **doing something else**

async and promises

- The reason for the promise system is to mask ***latency*** (i.e., slow operations) with concurrency
    - Consider: a 1Ghz CPU executes an instruction every 1 ns
    - Almost anything else takes forever (approximately)
        - e.g., 150,000 ns to read from the SSD, at least 100 million ns just to ping a nearby server over the internet
- Utilize this "wasted" time by **doing something else**
    - e.g., processing data, communicating with remote hosts, timers that countdown while our app is running, waiting for users to provide input, etc., by **running a promise**

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise
- The **await** keyword means that the current process is ***blocked*** on some “slow” activity

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise
- The **await** keyword means that the current process is ***blocked*** on some “slow” activity
  - allows the runtime (node.js) to move on to some other promise

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise
- The **await** keyword means that the current process is ***blocked*** on some “slow” activity
    - allows the runtime (node.js) to move on to some other promise
    - a new promise to return to whatever we were doing is created
        - eligible to run after the “slow” activity finishes

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise
- The **await** keyword means that the current process is ***blocked*** on some “slow” activity
  - allows the runtime (node.js) to move on to some other promise
  - a new promise to return to whatever we were doing is created
    - eligible to run after the “slow” activity finishes
- Whenever you do any kind of I/O (or other “slow” activity), you should use the promise system!

async and promises

- The **async** keyword on a function indicates that it creates and returns a promise
- The await keyword means that the current process is ***blocked*** on some "slow" activity
  - allows the runti
  - a new promise
    - eligible to
- Whenever you do
  should use the promise system!

Aside: a software engineer can be "blocked" if they're waiting for something from a coworker. This is a direct analogy to the I/O sense of "blocked" on this slide.

Example: starting a concurrent computation

Example: starting a concurrent computation

```
async function makeRequest(requestNumber : number) {
  // some code (to be executed now)
  const response =
      await axios.get('https://rest-example.covey.town')
      // more code (to be executed after the .get() returns).
}
```

Example: starting a concurrent computation

```
async function makeRequest(requestNumber : number) {
  // some code (to be executed now)
  const response =
      await axios.get('https://rest-example.covey.town')
      // more code (to be executed after the .get() returns).
}
```

- The http request is sent immediately.

Example: starting a concurrent computation

```
async function makeRequest(requestNumber : number) {
  // some code (to be executed now)
  const response =
      await axios.get('https://rest-example.covey.town')
      // more code (to be executed after the .get() returns).
}
```

- The http request is sent immediately.
- A promise is created to run the more code after the http call returns
  - (i.e., the code after “await” is blocked)

Example: starting a concurrent computation

```
async function makeRequest(requestNumber : number) {
  // some code (to be executed now)
  const response =
      await axios.get('https://rest-example.covey.town')
      // more code (to be executed after the .get() returns).
}
```

- The http request is sent immediately.
- A promise is created to run the more code after the http call returns
  - (i.e., the code after “await” is blocked)
- The caller of `makeRequest` resumes immediately.

General Rules for Writing Asynchronous Code

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
    - You must send the value to another promise that is awaiting it.

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
    - You must send the value to another promise that is awaiting it.
- Call async procedures only from other async functions or from the top level.

General Rules for Writing Asynchronous Code

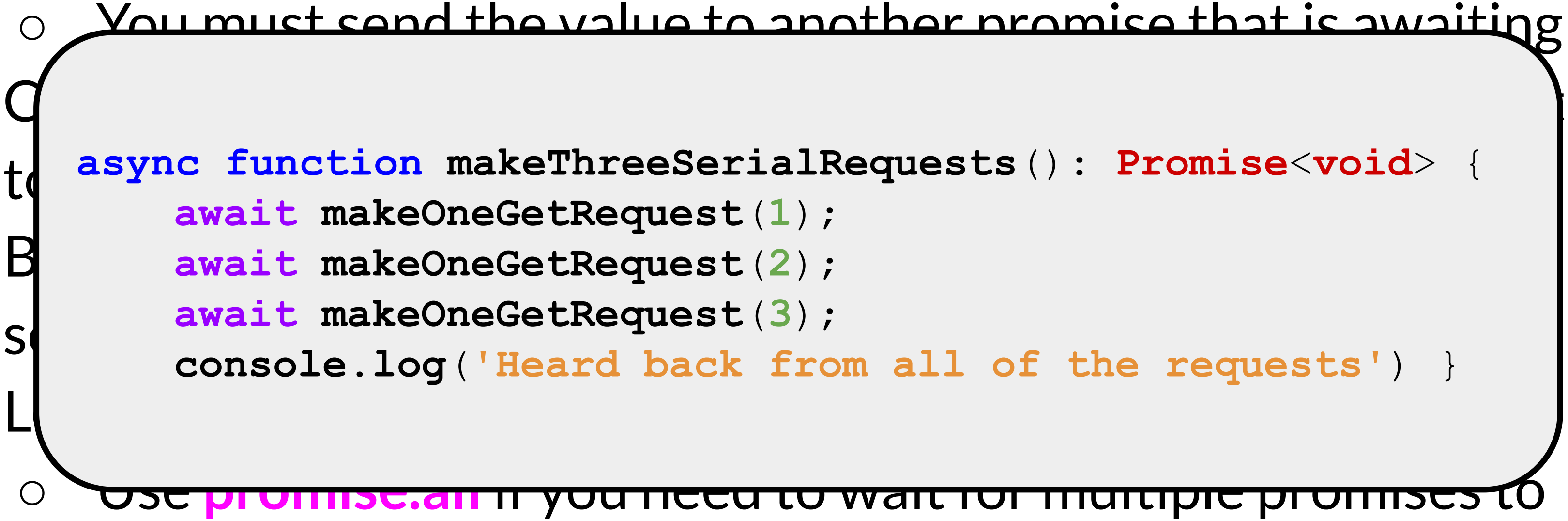
- You can’t return a value from a promise to an ordinary procedure.
    - You must send the value to another promise that is awaiting it.
- Call async procedures only from other async functions or from the top level.
- Break up any long-running computation into **async**/**await** segments so other processes will have a chance to run.

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
  - You must send the value to another promise that is awaiting it.
- Call async procedures only from other async functions or from the top level.
- Break up any long-running computation into **async**/**await** segments so other processes will have a chance to run.
- Leverage concurrency when possible
  - Use **promise.all** if you need to wait for multiple promises to return.

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
  - You must send the value to another promise that is awaiting it.
- C the to
- B s
- L
  - Use promise.all if you need to wait for multiple promises to return.



```
async function makeThreeSerialRequests(): Promise<void> {
    await makeOneGetRequest(1);
    await makeOneGetRequest(2);
    await makeOneGetRequest(3);
    console.log('Heard back from all of the requests') }
```

General Rules for Writing Asynchronous Code

- You can't return a valu ure.
  - You must send the ng it.
- C the
  to
- B
  s
- L
  - Use promise.all if you need to wait for multiple promises to return.

"Don't make another request until you got the last response back"

```
async function makeThreeSerialRequests(): Promise<void> {
    await makeOneGetRequest(1);
    await makeOneGetRequest(2);
    await makeOneGetRequest(3);
    console.log('Heard back from all of the requests') }
```

General Rules for Writing Asynchronous Code

- You can't return a value from a promise to an ordinary procedure.
  - You must send the value to another promise that is awaiting it.
- C the to
- B S
- L
  - Use promise.all if you need to wait for multiple promises to return.

```
async function makeThreeSerialRequests(): Promise<void> {
    await Promise.all([
        makeOneGetRequest(1),
        makeOneGetRequest(2),
        makeOneGetRequest(3)]);
    console.log('Heard back from all of the requests') }
```

General Rules for Writing Asynchronous Code

- You can't return a valu ure.
  - You must send the ng it.
- C he
  t
- B
  s
- L
  - Use promise.all if you need to wait for multiple promises to return.

"Make all of the requests now, then wait for all of the responses"

```
async function makeThreeSerialRequests(): Promise<void> {
    await Promise.all([
        makeOneGetRequest(1),
        makeOneGetRequest(2),
        makeOneGetRequest(3)]);
    console.log('Heard back from all of the requests') }
```

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
    - You must send the value to another promise that is awaiting it.
- Call async procedures only from other async functions or from the top level.
- Break up any long-running computation into **async**/**await** segments so other processes will have a chance to run.
- Leverage concurrency when possible
    - Use **promise.all** if you need to wait for multiple promises to return.

General Rules for Writing Asynchronous Code

- You can’t return a value from a promise to an ordinary procedure.
  - You must send the value to another promise that is awaiting it.
- Call async procedures only from other async functions or from the top level.
- Break up any long-running computation into **async**/**await** segments so other processes will have a chance to run.
- Leverage concurrency when possible
  - Use **promise.all** if you need to wait for multiple promises to return.
- Check for errors with **try**/**catch**

Takeaways

- Reading code is an important software engineering skill
  - like any skill, it requires **practice**!
- It’s usually infeasible to read all of the code, so you should focus on the parts that matter for whatever you’re trying to do
- Documentation is often useful, but also often wrong
  - important for context, but for details read the source code
- async/await are useful concurrency tools in TypeScript
  - you’ll need them for the course project

Advertising

- I’m coaching the ICPC team this year, and I’d love to have any/all of you participate
  - info session **TODAY** at 4pm, GITC 2121
  - ICPC is a team programming contest
    - excellent prep for LeetCode-style technical interviews!
  - we’ll run weekly practices until the real contest in November (?)
  - NJIT was most-improved team in our region last year
    - but we still finished 23rd(!), so plenty of room to improve
    - who doesn’t want to show up Rutgers/Columbia/NYU/etc?