Test Driven Development

By Sascha Depold and Parinaz Roghany

Agenda

- Testing in theory (What, why, how)
- The Fibonacci sequence
- Implementation
    - Which test cases?
    - Add tests
    - Write fib function
- Homework

What is testing?

What is testing?

- Testing: Process of verifying expectations
- Manual testing: Verification through manual interaction
- Automated testing: Verification through automated processes
- Different kinds of testing:
  - Unit testing ← *Today's focus*
  - Integration testing
  - End-To-End testing
  - Acceptance testing
  - …

Homework?

Why would you care?

Why would you care?

- Ensures health of product after changes
  - Unhealthy products jeopardize customer satisfaction and ultimately revenue
- Automation severely decreases amount of required time
  - Right test strategy generates crucial insights almost instantly
  - Increased trust in changes and releases
- Important in the industry

Test Driven Development?

Test Driven Development?

- Expectations are usually already formed during brainstorming phase
- Set of expectations ⇒ Specification

Test Driven Development?

- Expectations are usually already formed during brainstorming phase
- Set of expectations ⇒ Specification
- TDD is about converting the specification into a format that is
    - Machine readable
    - Verifiable
    - Extendable
    - Easy to understand

Test Driven Development?

- Expectations are usually already formed during brainstorming phase
- Set of expectations ⇒ Specification
- TDD is about converting the specification into a format that is
  - Machine readable
  - Verifiable
  - Extendable
  - Easy to understand

Before implementation!

Which **tools** are available?

Which **tools** are available?

- Testing is a combination of various aspects
    - Structuring of tests
    - Execution of tests
    - Asserting of expectations
    - Temporary and reversible manipulation of dependencies
    - Remote controlling browsers
    - Generation of heavy load
    - …

$\left.\begin{array}{l}\text{Structuring of tests}\\ \text{Execution of tests}\\ \text{Asserting of expectations}\end{array}\right\}$ *Unit testing / today*

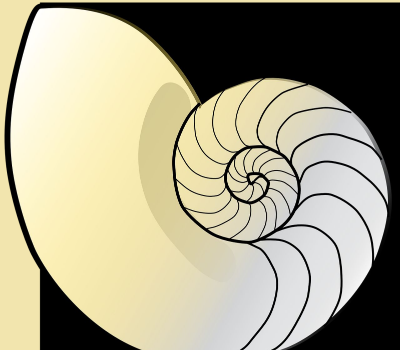
Which **tools** are available?

Today's focus

- Majority of JavaScript modules focus on separation of concerns
  - They do one and only one thing as best as possible
  - Combination of modules as solution for complex tasks
  - Today's de-facto standard: Mocha (test runner/structure) + Chai (Assertion library)
- BUT: React movement brought a new solution that combines all aspects
  - Jest

More info later

Meet **Fibonacci**



Meet **Fibonacci**

- Fibonacci numbers are a mathematical sequence of numbers
- Starting from 0 and 1
- Each number is the sum of the two preceding ones

Meet **Fibonacci**

- Fibonacci numbers are a mathematical sequence of numbers
- Starting from 0 and 1
- Each number is the sum of the two preceding ones

$fib(1) = 1$

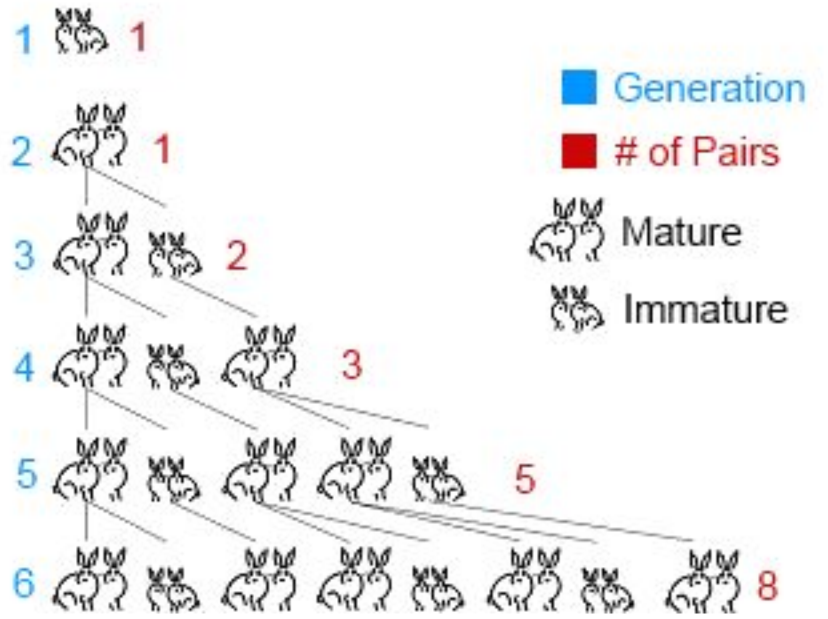
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

$fib(0) = 0$

$$\begin{aligned} fib(8) &= fib(7) + fib(6) \\ &= 13 + 8 \\ &= 21 \end{aligned}$$

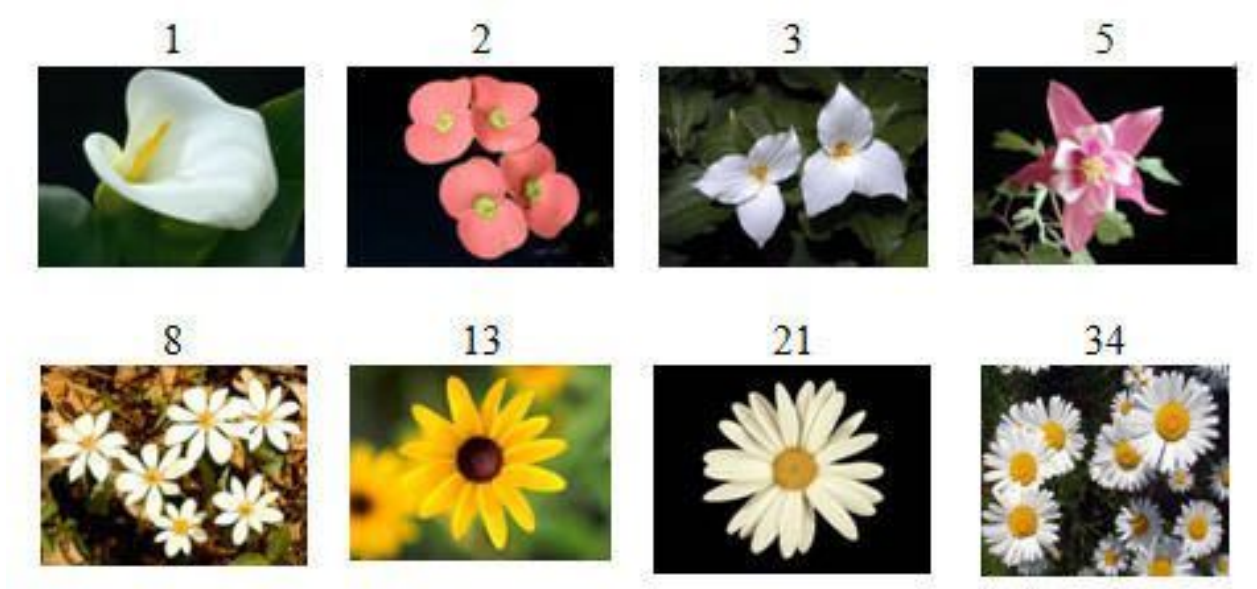
Meet **Fibonacci | Application**

- Nature
  - Growth of rabbit/honey bees/… population



Meet **Fibonacci | Application**

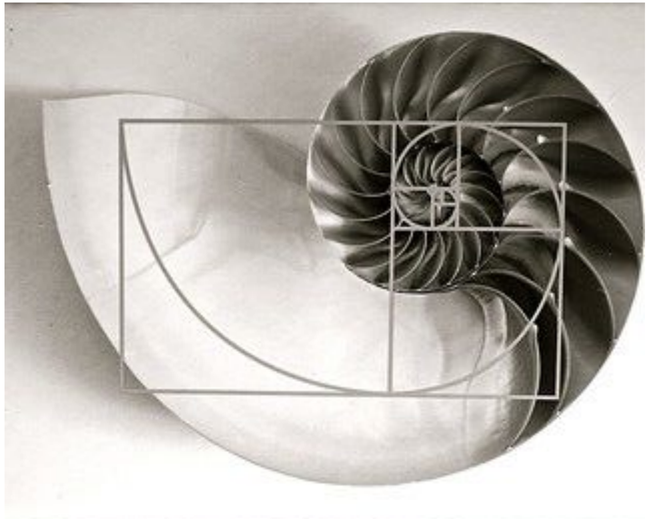
- Nature
  - Growth of rabbit/honey bees/… population
  - Tree branching / Flower petals



The Fibonacci Sequence and Golden Ratio in Nature!

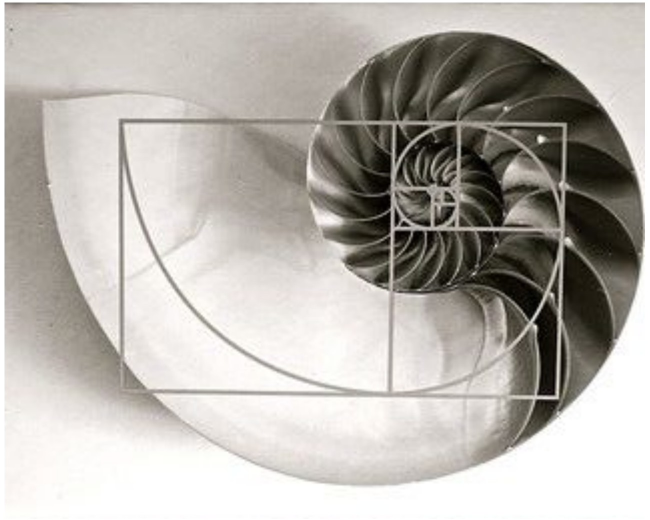
Meet **Fibonacci | Application**

- Nature
  - Growth of rabbit/honey bees/… population
  - Tree branching / Flower petals
  - Spiral shells



Meet **Fibonacci | Application**

- Nature
  - Growth of rabbit/honey bees/… population
  - Tree branching / Flower petals
  - Spiral shells
- Maths
  - Quotient of two consecutive Fibonacci numbers strives towards the golden ratio (~1.618…)
    - fib(5) / fib(4) = 5 / 3 = 1.666666667
    - fib(7) / fib(6) = 13 / 8 = 1.625



Test cases

Test cases | Plan

- We will implement the Fibonacci function
  - Takes a number (an index)
  - Returns the respective number of the sequence
- Strategy
  - We start with the tests
  - See them fail
  - Implement fib step by step (red/green refactoring)
  - → TDD!

function fib(n) {} → Number

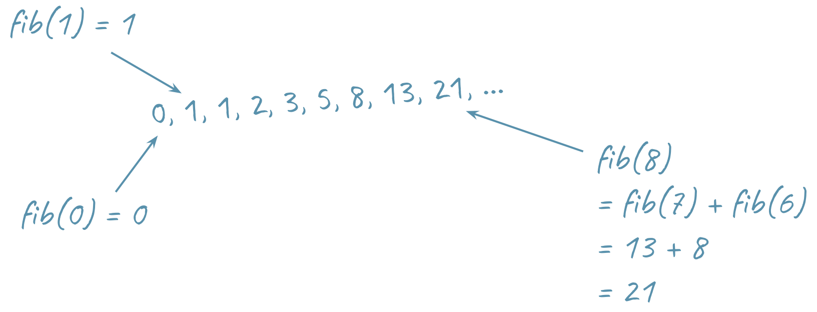
Test **cases | What?**

- Writing test cases is about verifying
  - The *normal* behavior superficially
  - The *edge cases* as much as possible

Test **cases** | **What?**

- Writing test cases is about verifying
    - The *normal* behavior superficially
    - The *edge cases* as much as possible

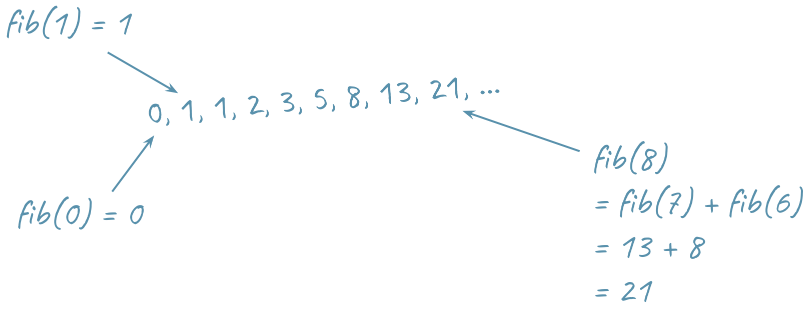
- Recap of Fibonacci
    - fib(0) = 0
    - fib(1) = 1
    - fib(n) = fib(n-1) + fib(n-2)



Test cases | What?

- Writing test cases is about verifying
  - The *normal* behavior superficially
  - The *edge cases* as much as possible

- Recap of Fibonacci
  - fib(0) = 0
  - fib(1) = 1
  - fib(n) = fib(n-1) + fib(n-2)



What do we need to test?

ebay

Let’s **do** it! | Preparation

- Download this repo: https://github.com/sdepold/js-basics-tdd
- Install dependencies: **npm install**

- index.js → Will contain our logic (the fibonacci function)

```
module.exports = function fib(n) {

};
```

Let’s **do** it! | Preparation

- Download this repo: https://github.com/sdepold/js-basics-tdd
- Install dependencies: **npm install**

- index.js → Will contain our logic (the fibonacci function)
- test.js → Will contain our automated test cases

```
const fib = require('./index');
const { expect } = require('chai');
```

Let’s **do** it! | Writing the tests

- Test structure

```
describe('Fibonacci', function () {
    it('should do something', function () {
        // assertions go here
    });
});
```

Let’s **do** it! | Writing the tests

- Test structure

```
describe('Fibonacci', function () {
    it('should do something', function () {
        // assertions go here
    });
});
```

- Assertions

```
expect(1).to.equal(1);
expect(fib(1)).to.equal(1);
expect(1).to.be.closeTo(1, 0.5)
expect(()=>{ fib(-1); }).to.throw('1')
```

Let's **do** it! | Writing the tests

- Test structure

```
describe('Fibonacci', function () {
    it('should do something', function () {
        // assertions go here
    });
});
```

- Assertions

```
expect(1).to.equal(1);
expect(fib(1)).to.equal(1);
expect(1).to.be.closeTo(1, 0.5)
expect(()=>{ fib(-1); }).to.throw('1')
expect(something).to.be.a( 'datatype here')
```

- Test cases → Chat

Go write the tests now!

Let’s **do** it! | Running the tests

- Tests can be triggered via: **npm test**
- You should see all your tests failing now (maybe one actually works)

Let’s **do** it! | Implementation

- Focus on one particular test case and make it green
- Run the tests again to see it covered
- Take the next and start over till everything is green

- Recap of Fibonacci
    - fib(0) = 0
    - fib(1) = 1
    - fib(n) = fib(n-1) + fib(n-2)

Fixing **fib(100)**

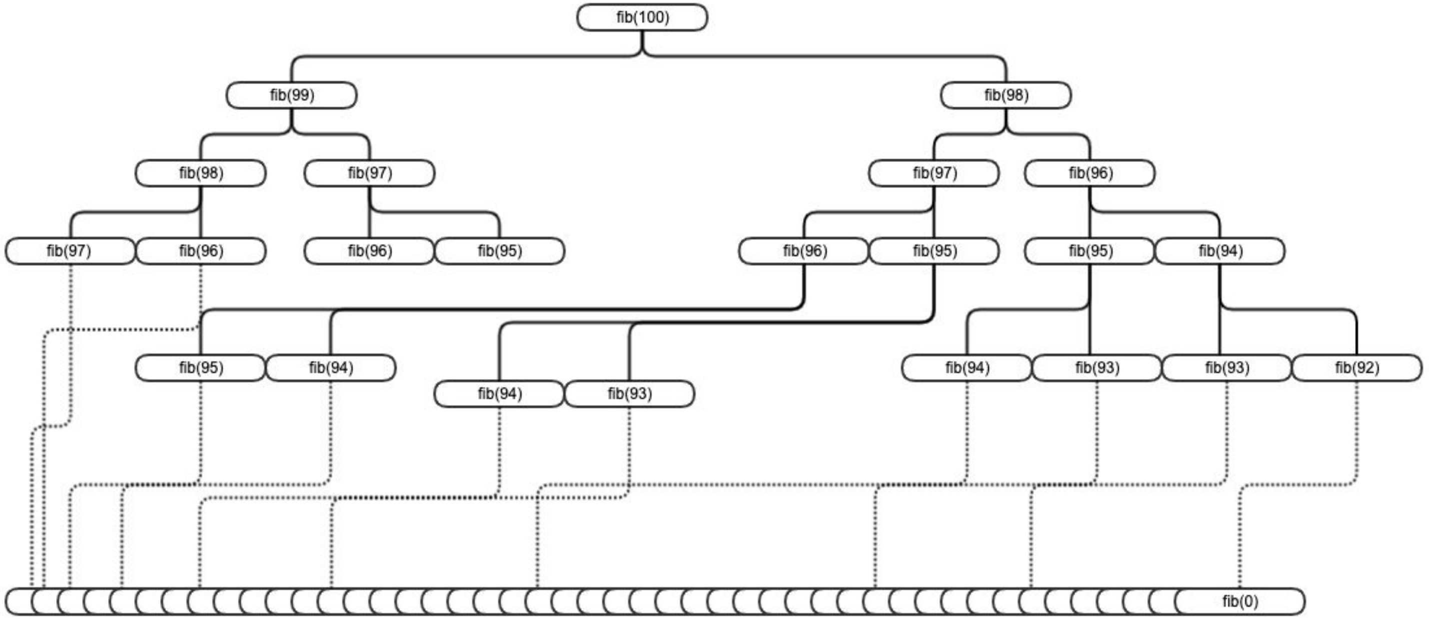
Fixing **fib(100) | Problem**

- Solution likely contains something like

```
function fib(n) => {
  if (n < 0) {
    return undefined;
  }

  if (n === 0) {
    return 0;
  }

  if (n === 1) {
    return 1;
  }

  return fib(n - 1) + fib(n - 2);
};
```

Fixing **fib(100)** | Problem

- Solution likely contains something like `return fib(n-1) + fib(n-2)`
- Causes JS runtime to nest deeply blocking CPU and memory



Fixing fib(100) | Approach

- Introduce a cache
- Store calculated values in the cache
- Lookup cache before calculating anything

Fixing fib(100) | Approach

- Introduce a cache
- Store calculated values in the cache
- Lookup cache before calculating anything

```
const cache = [];    // Our cache

cache[n] = 123       // store something in cache
cache[n]             // cache lookup --> 123
```

Fixing fib(100) | Possible solutions

```
let cache = [];

const fib = n => {
  const result = cache[n] || calc(n);
  return (cache[n] = result);
};

const calc = n => {
  if (n < 0) {
    return undefined;
  }
  if (n === 0) {
    return 0;
  }
  if (n === 1 || n === 2) {
    return 1;
  }
  return fib(n - 2) + fib(n - 1);
};

module.exports = fib;
```

or:

```
const cache = [];

module.exports = function fibonacci(n) {
  if (cache[n]) {
    return cache[n];
  } else if (n === 0) {
    return cache[n] = 0;
  } else if (n === 1) {
    return cache[n] = 1
  } else if (n < 0) {
    return;
  } else {
    return cache[n] = fibonacci(n - 1) +
fibonacci(n - 2);
  }
}
```

Recap

Recap

- Tests ensure health of application
- Automation allows instant feedback loops
- Test Driven Development is converting the specification into tests before coding
- Fibonacci is about rabbits 🐰, honeybees 🐝 and the golden ratio 📷

HOME
WORK

Homework | Resources

- Test Runner: Mocha
- Assertion Library: Chai
- All in one test solution: Jest
- Lengthy article about The Practical Test Pyramid
- Behavior-Driven Development (BDD)

Homework | Task

- fib(100) is currently returning: 35422484817926**2000000**
- It should actually return: 35422484817926**1915075**

- Find out about the why here and here (if you feel adventurous)
- Find out about and convert solution to BigInt

Homework | Q&A

- Any feedback, questions and solutions?
- Share them with your instructors and they’ll forward them to us!
- Better communication channel in the future