KINGSLAND
UNIVERSITY

Unit Testing and Modules

Error and Exception Handling, Modules, Unit Testing, Assertions

Table of Contents

- ✓ Error handling
- ✓ Exception Handling
- ✓ Modules
- ✓ Unit Testing – Concepts
- ✓ Mocha and Chai for Unit Testing

Concepts, Examples, Exceptions

Error Handling

Why Error Handling is Important?

- ✓ Error handling empowers the developer
  - ✓ Differentiates the **type** and **reason** of the error
  - ✓ **Logs** of the errors are **hopeful while bug fixing**
  - ✓ Exceptions are the **object-oriented way** for errors

Types of Errors

There are **three types** of errors in programming:

- **Syntax Errors** - occur at compile time
  - Not applicable for JS
- **Runtime Errors** - occur during execution
  - After compilation, when the application is running
- **Logical Errors** - occur when a mistake has been made in the logic of the script and the expected result is incorrect
  - Also known as bugs

Error Handling

A function failed to do what its name suggests should:

- ✓ Return a special value (e.g. **undefined** / **false** / **-1**)
- ✓ Throw an **exception** / **error**

```
let str = "Hello, Kingsland";
console.log(str.indexOf(“USA")); // -1
// Special case returns a special value to indicate "not found"
```

Error Handling

- The fundamental **principle** of error handling says that a function (method) should either:
  - Do what its **name** suggests
  - Indicate a **problem**
  - Any other behavior is **incorrect**

Error Handling – Exceptions (Errors)

- **Exception** - a function is unable to do its work (**fatal error**)

```
let arr = new Array(-1);    // Uncaught RangeError
```

```
let bigArr = new Array(9999999999); // RangeError
```

```
let index = undefined.indexOf("hi"); // TypeError
```

```
console.log(George);    // Uncaught ReferenceError
```

```
console.print('hi');    // Uncaught TypeError
```

Error Handling – Special Values

```
let sqrt = Math.sqrt(-1); // NaN (special value)
```

```
let sub = "hello".substring(2, 1000); // llo
let sub = "hello".substring(-100, 100); // hello
// Soft error - substring still does its job: takes
all available chars
```

```
let invalid = new Date("Christmas"); // Invalid Date
let date = invalid.getDate(); // NaN
```

Unexpected Behavior

- In JavaScript, the **first** month (January) is month number **0**, so December **returns** month **number 11**

```
let date = new Date(2016, 1, 20);      // Feb 20 2016
```

```
let date1 = new Date(1, 1, 1);         // Feb 01 1901
```

```
let dateMinus1 = new Date(-1, -1, -1); // Nov 29 -2
```

```
let dateNext = new Date(2016, 1, 30)   // Mar 01 2016 (next month)
```

```
let datePrev = new Date(2016, -1, 30); // Dec 30 2015 (prev month)
```

Throwing / Catching Errors

Exception Handling

Throwing Errors (Exceptions)

- The **throw** statement lets you create custom errors
  - **General Error** - throw new Error("**Invalid state**")
  - **Range Error** - throw new RangeError("**Invalid index**")
  - **Type Error** - throw new TypeError("**String expected**")
  - **Reference Error** - throw new ReferenceError("**Missing age**")
- Good practices say that you should use **Error** when throwing

Try – Catch

- The **try** statement tests a block of code for **errors**
- The **catch** statement **handles** the error
- **Try** and **catch** come in pairs

```
try {
  // Code that can throw an exception
  // Some other code - not executed in case of error!
} catch (ex) {
  // This code is executed in case of exception
  // Ex holds the info about the exception
}
```

Exception Properties

- An **Error object** with properties is be created

```
try {
    throw new RangeError("Invalid range.");
    console.log("This will not be executed.");
  } catch (ex) {
    console.log("Exception object: " + ex);
    console.log("Type: " + ex.name);
    console.log("Message: " + ex.message);
    console.log("Stack: " + ex.stack);
}
```

Definition, Import, Export

Modules

Modules

- A **set of functions** to be included in applications
- Group related behavior
- Resolve naming collisions
  - **`http.get`**`(url)` and **`students.get()`**
- Expose only public behavior
  - They do not populate the global scope with unnecessary objects

a module for loading indicator

```
const loading = {

  show() { },
  hide() { },

};
```

Approaches for Modules

- Since, modules were not native in JS, there are different approaches to create modules:
  - **Using IIFE**
  - **Using Nodejs require/export**
  - **Using ES2015 import/export**

IIFE Modules

- **IIFE modules** are essential for front-end JS
- They hide the unnecessary and expose only needed behavior/objects to the global scope

```
(function(scope) {
  const selector = 'loading';
  const loadingElement = document.querySelector(selector)
  const show = () => loadingElement.style.display = '';
  const hide = () => ladingElement.style.display = 'none'
  // Only this is visible to the global scope
  scope.loading = { show, hide };
  }(window));
```

Node.js Modules

- **require()** is used to **import** modules

```
const http = require('http');
// For NPM packages
```

```
const myModule = require('./myModule.js');
// For internal modules
```

- **Internal** modules need to be **exported before** being required
- In **Node.js** each file has its own scope

Node.js Modules

- Whatever value has **module.exports** will be the value when using **require**

```
const myModule = () => {...};

module.exports = myModule;
```

- To **export more than one** function, the value of **module.exports** will be an **object**

```
module.exports = {
  toCamelCase: convertToCamelCase,
  toLowerCase: convertToLowerCase
};
```

ES6 Modules

- **Always** import and export an **object**
- Only a **specific** function can be **imported**

```
import toLowerCase from './toLowerCase.js';
```

- To import the **whole** object

```
import * as myModules from './myModules.js';
```

- To **change the name** after importing

```
import { toLowerCase as convertToLowerCase }
from './myModules.js';
```

Definition, Structure, Examples, Frameworks

Unit Testing

Unit Testing

- A **unit test** is a piece of code that checks whether certain functionality **works as expected**
- Allows developers to see **where** & **why errors occur**

```
function sortNums(arr) {
    arr.sort((a,b) => a - b);
}
```

```
let nums = [2, 15, -2, 4];
sortNums(nums);
if (JSON.stringify(nums) === "[-2,2,4,15]") {
    console.error("They are equal!");
}
```

Unit Testing

- Testing enables the following:
  - **Easier maintenance** of the code base
    - Bugs are found ASAP
  - **Faster development**
    - The so called "Test-driven development"
    - Tests before code
  - **Automated way to find code wrongness**
    - If most of the features have tests, running them shows their correctness

Unit Tests Structure

- The **AAA** Pattern: **Arrange**, **Act**, **Assert**

```
// Arrange all necessary preconditions and inputs
let nums = [2, 15, -2, 4];
// Act on the object or method under test
sortNums(nums);
// Assert that the obtained results are what we expect
if (JSON.stringify(nums) === "[-2,2,4,15]") {
    console.error("They are equal!");
}
```

Unit Testing Frameworks

- JS Unit Testing:
  - **Mocha**, **QUnit**, **Unit.js**, **Jasmine**
- Assertion frameworks (perform checks):
  - **Chai**, **Assert.js**, **Should.js**
- Mocking frameworks (mocks and stubs):
  - **Sinon**, **JMock**, **Mockito**, **Moq**

Unit Testing with Mocha and Chai

Mocha and Chai

What is Mocha?

- Feature-rich JS test framework
- Provides common testing functions including **it**, **describe** and the **main function** that runs tests

```
describe("title", function () {
    it("title", function () { … });
});
```

- Usually used together with **Chai**

What is Chai?

- A library with many assertions
- Allows the usage of a lot of different assertions such as **assert.equal**

```
let assert = require("chai").assert;
describe("pow", function() {
   it("2 raised to power 3 is 8", function() {
    assert.equal(pow(2, 3), 8);
  });
});
```

Global Installation

Mocha and Chai

Global Installation

- To install **frameworks** and **libraries globally**, use the CMD
  - Installing **Mocha** and **Chai** through **npm**

```
npm install –g mocha
```

```
npm install –g chai
```

- Check if Mocha is installed

```
mocha --version
```

NODE_PATH Configuration

- By default Node.js does not find its globally installed modules
- You need to set the **NODE_PATH** environment variable

```
rem for any future sessions
setx NODE_PATH %AppData%\npm\node_modules
rem for current session
set NODE_PATH=%AppData%\npm\node_modules
```

- You may need to restart your IDE after changing **NODE_PATH**

Usage and Examples

- To load a library, we need to **require** it

```
const expect = require("chai").expect;
```

```
describe("Test group #1", function () {
    it("should… when…", function () {
        expect(actual).to.be.equal(expected);
    });
    it("should… when…", function () { … });
});
describe("Test group #2", function () {
    it("should… when…", function () {
        expect(actual).to.be.equal(expected);
    });
});
```

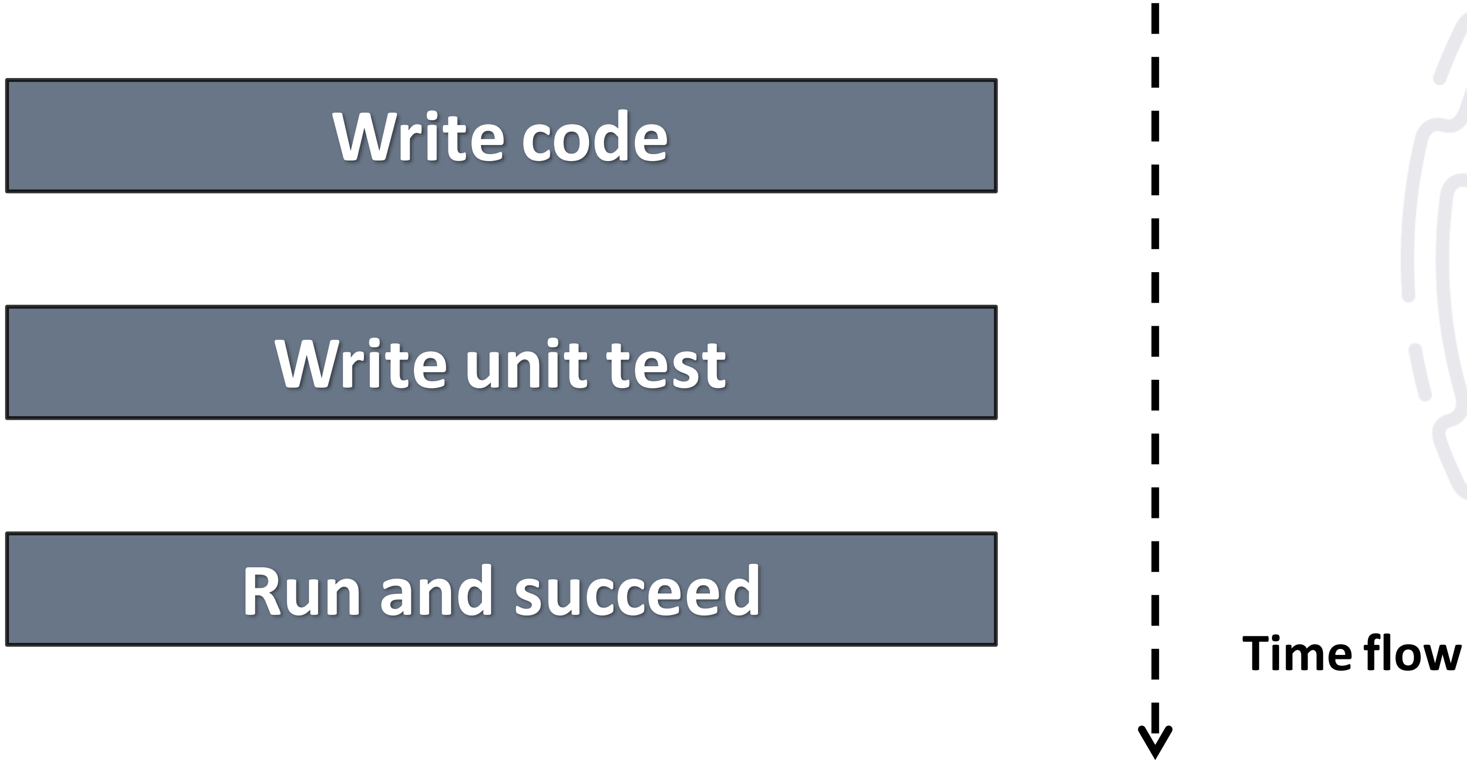
Learn the "Test First" Approach to Coding

Test Driven Development

Unit Testing Approaches

- "**Code First**" (code and test) approach
  - Classical approach
- "**Test First**" approach
  - **T**est-**d**riven **d**evelopment (**TDD**)

The Code and Test Approach



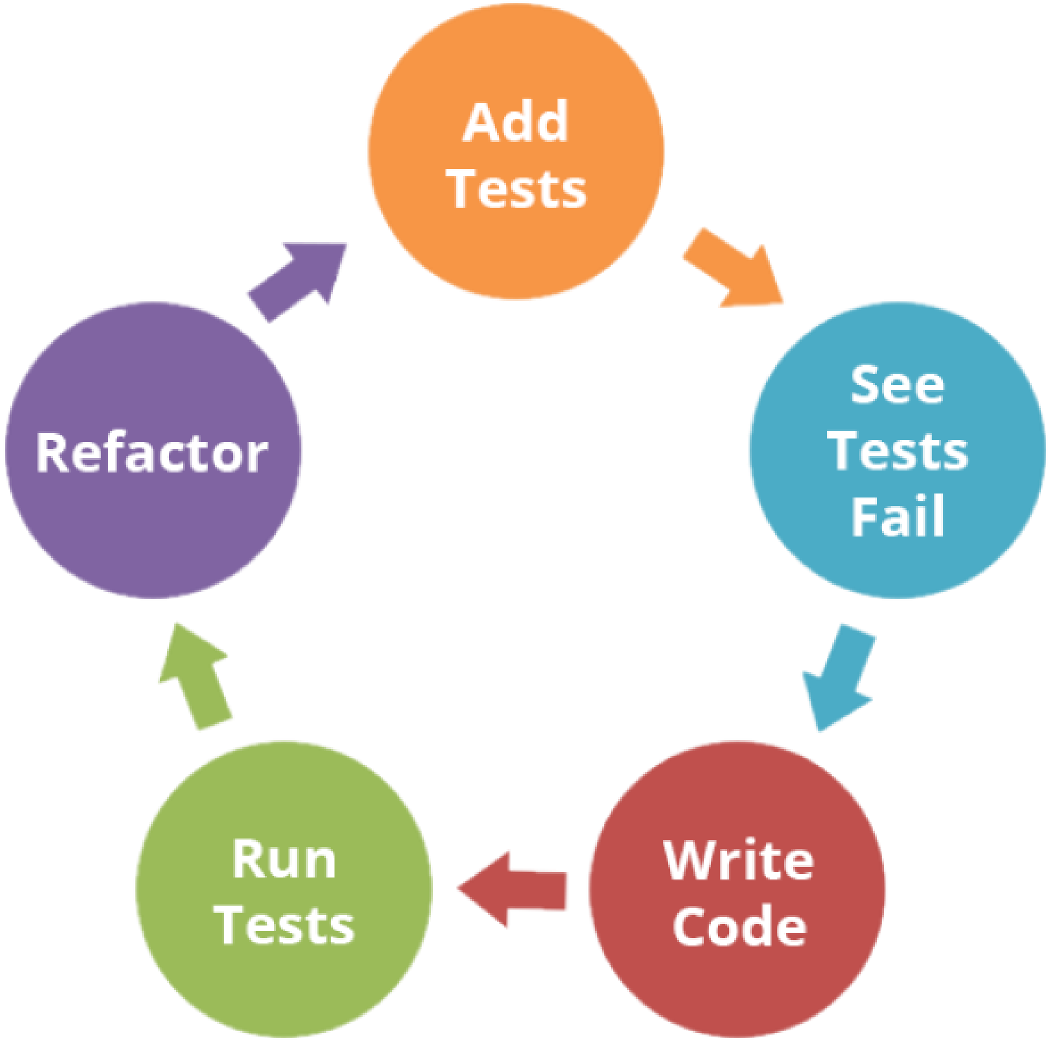
Write code

Write unit test

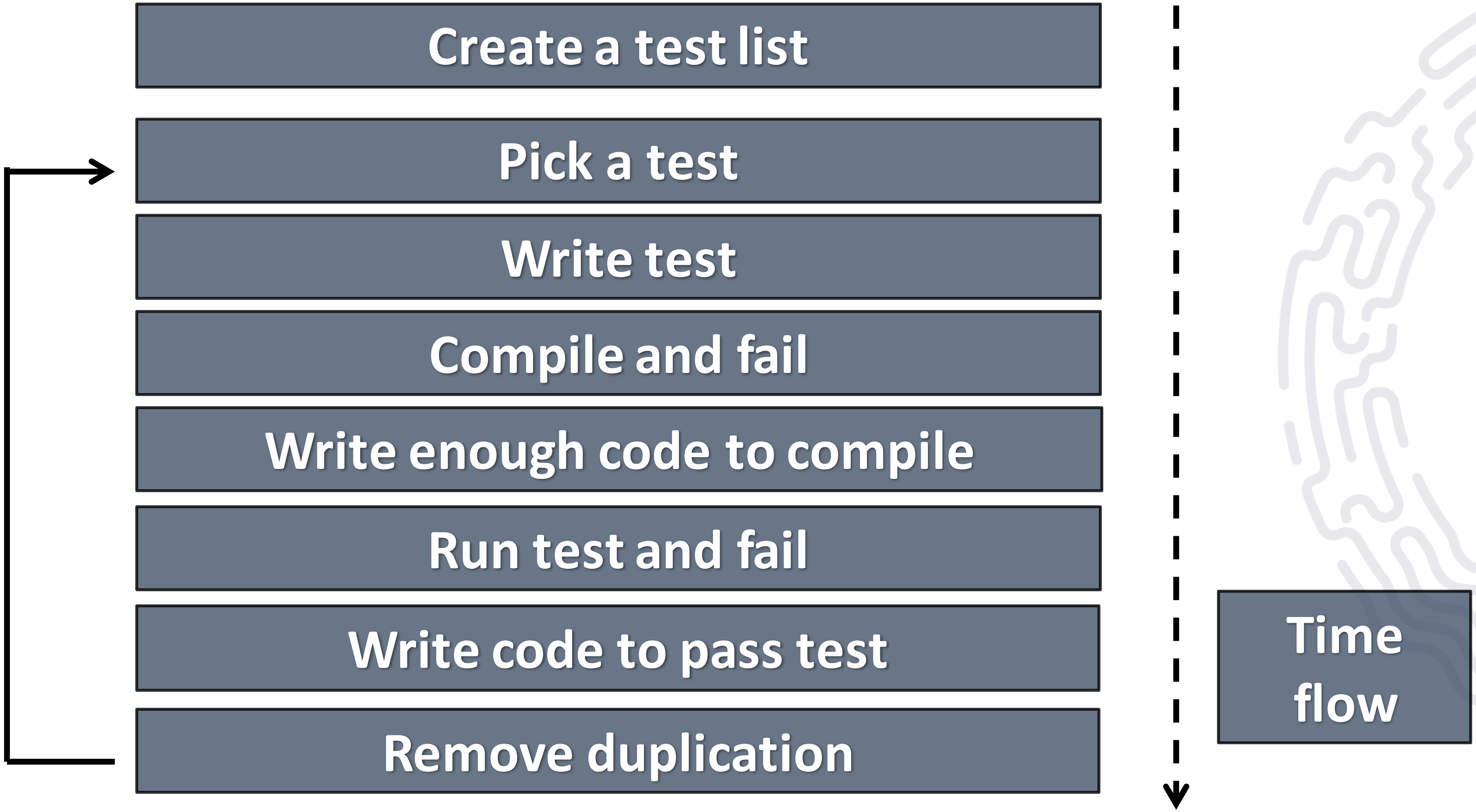
Run and succeed

Time flow

The Test-Driven Development Approach



Test-Driven Development (TDD)



Create a test list

Pick a test

Write test

Compile and fail

Write enough code to compile

Run test and fail

Write code to pass test

Remove duplication

Time flow

Why TDD?

- ✓ TDD helps find design issues early
  - ✓ Avoids reworking
- ✓ Writing code to satisfy a test is a focused activity
  - ✓ Less chance of error
- ✓ Tests will be more comprehensive than if they are written after the code

Unit Testing

Live Exercises

Summary

- A **function** should do what its **name suggests**
- The **throw** statement lets you create **custom errors**
- Modules are a **set of functions** to be included in applications
- Unit tests **check** if certain functionality **works as expected**

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com

KINGSLAND
UNIVERSITY

THANK YOU