COMP1531 | T09B / H17B

Week 3

william.huynh3@unsw.edu.au

Iteration 0

Iteration 1
(If you start early)

Iteration 1
(If you start late)

UNSW SYDNEY

Learning objectives

- Tips for Iteration 1
- **Refactoring** code using JS principles
- Installing and using **NPM packages**
- Using **JEST** to test our code

Iteration 1 Tasks

Major Tasks (40% weighting)

Write tests for the 11 functions

Implement those 11 functions

```
authLoginV1
authRegisterV1
channelsCreateV1
channelsListV1
channelsListAllV1
channelDetailsV1
channelJoinV1
channelInviteV1
channelMessagesV1
userProfileV1
clearV1
```

Minor Tasks (60%)

Code quality (discussed later in tute - 25%)

Git Practices (15%)

Project Team Work (15%)

Assumptions File (5%)

UNSW SYDNEY

JS: Code Smells

Code smells are signs that something is wrong **stylistically** with your code and demands your attention.

'Code smells' or *code quality* is worth 25% of Iteration 1

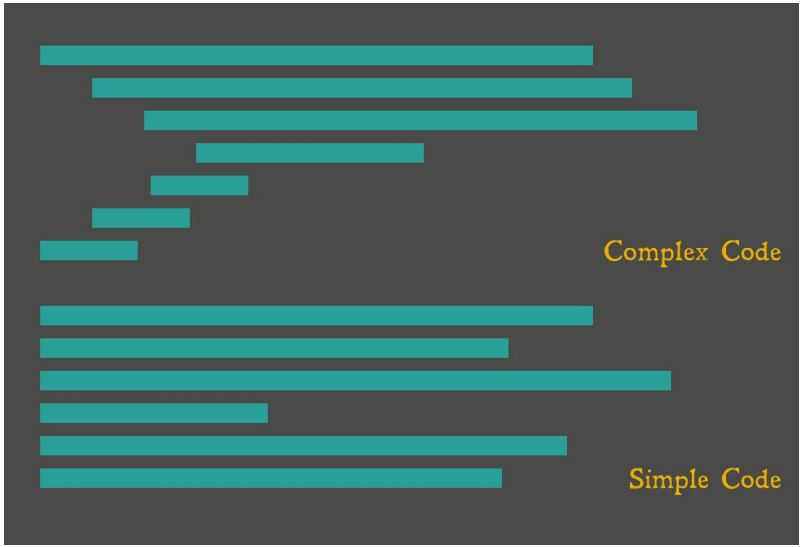
```
function add(numbers) {
  var result = 0;
  var parts = numbers.split(',');
  for (var i = 0; i < parts.length; i++) {
    var integer = parseInt(parts[i]);
    if (!isNaN(integer)) {
      if (integer >= 0) {
        if (integer <= 1000) {
          result += integer;
        }
      }
    }
  }

  return result;
}
```

What looks immediately wrong with this code?

What could be a potential fix?

Code Smell: Excessive Indentation




```
1  function add(numbers) {
2    var result = 0;
3    var parts = numbers.split(',');
4    for (var i = 0; i < parts.length; i++) {
5      var integer = parseInt(parts[i]);
6      if (!isNaN(integer)) {
7        if (integer >= 0) {
8          if (integer <= 1000) {
9            result += integer;
10         }
11       }
12     }
13   }
14
15   return result;
16 }
```

- Turn complex logic into functions
- Avoid nested loops or nested conditions (again use functions!)
- See if there are existing NPM libraries that can do complex things for you

UNSW SYDNEY

```
function do_stuff() {
  let fail_flag = false;

  if (problem == true) {
    fail_flag = true;
  }

  if (another_problem == true) {
    fail_flag = true;
  }

  if (even_another_problem == true) {
    fail_flag = true;
  }

  if (fail_flag === true) {
    return {"status": "somethings wrong! "};
  }

  else if (fail_flag === false) {
    return {"status": "everythings ok :) "}
  }
}
```

What looks immediately wrong with this code?

What could be a potential fix?

UNSW SYDNEY

Code Smell: Unnecessary logic

```
function do_stuff() {
  let fail_flag = false;

  if (problem == true) {
    fail_flag = true;
  }

  if (another_problem == true) {
    fail_flag = true;
  }

  if (even_another_problem == true) {
    fail_flag = true;
  }

  if (fail_flag === true) {
    return {"status": "somethings wrong! "};
  }

  else if (fail_flag === false) {
    return {"status": "everything ok :) "}
  }
}
```

```
function do_stuff() {

  if (problem == true || another_problem == true ||
      even_another_problem == true) {
    return {"status": "somethings wrong! "};
  }
  return {"status": "everything ok :) "}
}
```

UNSW SYDNEY

```
function process_vehicle(vehicle_type, num_wheels, num_carriages, num_sails, ticket_cost ) {

  if (vehicle_type == 'train') {
    return {"carriages": num_carriages, "fare": ticket_cost};
  }

  if (vehicle_type == 'car') {
    return {"wheels": num_wheels};
  }

  if (vehicle_type == 'boat') {
    return {"sails": num_sails};
  }

}
```

What looks immediately wrong with this code?

What could be a potential fix?

Code Smell: Too Many Parameters

```
function process_vehicle(vehicle_type, num_wheels, num_carriages, num_sails, ticket_cost ) {

  if (vehicle_type == 'train') {
    return {"carriages": num_carriages, "fare": ticket_cost};
  }

  if (vehicle_type == 'car') {
    return {"wheels": num_wheels};
  }

  if (vehicle_type == 'boat') {
    return {"sails": num_sails};
  }

}
```

```
function process_train (num_carriages, ticket_cost) {
    return {"carriages": num_carriages, "fare": ticket_cost};
  }

function process_car (num_wheels) {
    return {"wheels": num_wheels};
  }

function process_boat (num_sails) {
    return {"sails": num_sails};
  }
```

```
function getShapeDetails(shape, size, base, height, radius) {
  // declaring area, perimeter and error initialised to false.
  let area;
  let perimeter;
  let error = false;

  if (shape === 'square') {
    if (size < 0) {
      // size negative, set error to true
      error = true;
    }
    // The shape is a square, area is size times size
    area = size * size;
    // The shape is a square, perimeter is 4 * size
    perimeter = 4 * size;
  } else if (shape === 'rectangle') {
    if (base < 0 || height < 0) {
      // base or height is negative, error true!
      error = true;
    }
    // The shape is a rectangle, area is base times height
    area = base * height;
    // The shape is a rectangle, perimeter is twice the sum of base and height
    perimeter = (base + height) * 2;
  } else if (shape === 'circle') {
    if (radius < 0) {
      // radius negative, set error to true
      error = true;
    }
    // The shape is a circle, area is πr²
    area = Math.PI * radius * radius;
    // The shape is a circle, return 2πr
    perimeter = 2 * Math.PI * radius;
  } else {
    // user's at fault, not an error with numbers.
    // just set to 0. They should read the Notes!
    area = 0;
    perimeter = 0;
  }

  if (error === true) {
    // there is an error somewhere, likely negative numbers.
    return { error: 'error' };
  }

  // success, return area and perimeter
  return { area: area, perimeter: perimeter };
}

```

Activity

One of your team members has merged one of their functions without anyone's approval !!!

Upon inspection, you realise their function is full of code smells, and your team feels at risk of losing the 25% style marks.

There are 20 minutes before the assignment is due.

Can you figure out all the code smells and refactor them before gitlab auto-submits your repo for marking ????

Note: The code can be found in the Tute03 Gitlab Repo (Webcms)

UNSW SYDNEY

Luckily there's an NPM library that can fix (most) code smells. But this is not expected for iteration 1.

Source: Lectures

npm: Packages

Unlike C/C++, Javascript supports the importing of official and unofficial packages using npm.

This allows us to use existing, user-made functions rather than writing our own!

Whilst the use of packages is not assessed, it will save you LOTS of time.

npm: using packages vs no packages

no packages

```
const bfs = (node) => {
  visited[node] = true;
  queue.equeue(node);
  while (!queue.isEmpty()) {
    let visiting = queue.dequeue();
    console.log(`we visited ${visiting}`)
    for (let j = 0; j < graphAdj[visiting].length; j++) {
      if ((graphAdj[visiting][j] === 1) && (visited[j] === false)) {
      visited[j] = true;
      queue.equeue(j);
      }
    }
  }
}
```

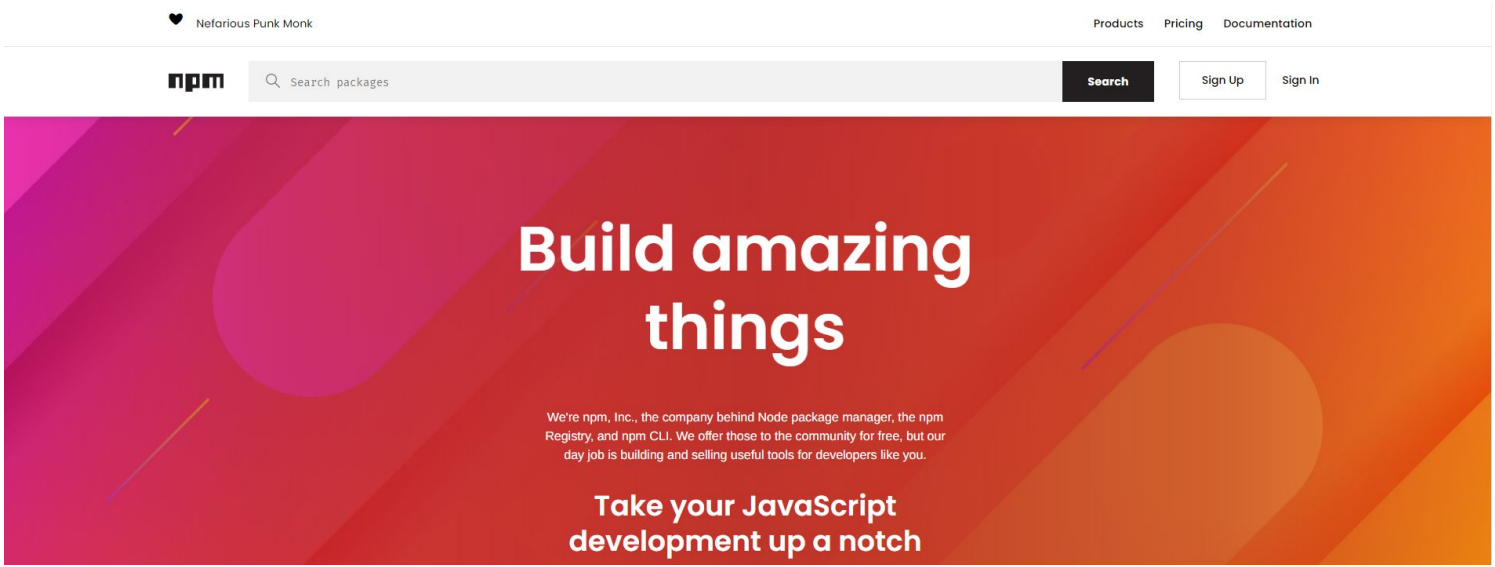
using packages

```
import bfs from 'bfs-package'

bfs(source, destination);
```

npm: Where to find packages?

npmjs.com



npm: How to install packages?

npmjs.com

```
npm install <package-name>
```

Interface: Functions

Name & Description	Parameters	Return Object	Errors
`stamp` Given an email, stamp it with a unique `identifier` and a `timeString`. The `identifier` should be universally unique (e.g. someone from across the world has an improbable chance of generating the same string). It should not rely on the characters in the email.	(email)	`{identifier, timeString}`	Return `{error}` when the email is invalid

Interface: Data Types

If the variable name	It is of type
is **error**	`string` with value exactly `'error'`
is **email**	`string`
is **identifier**	`string` that is globally unique
is **timeString**	`string` in the format `'WEEKDAY - hh:mm:ss [am/pm]"`. e.g. `'Saturday - 06:03:54 pm'`

For example:

```
stamp('valid@email.com');
```

may return something like:

```
return {
  identifier: 'some unique string that no one can guess',
  timeString: 'Saturday - 06:03:23 pm'
}
```

Discuss with your group the kinds of functions you'd likely need to solve this problem.

Try find some npm packages that can do it for you!

1. Email Validator
2. Current Date Time
3. Generate a UNIQUE string

```
1  import validator from 'validator';
2  // Formats: https://date-fns.org/docs/format
3  import { format } from 'date-fns';
4  import { v4 } from 'uuid';
5
6  function stamp(email) {
7    if (!validator.isEmail(email)) {
8      return { error: 'error' };
9    }
10   return {
11     identifier: v4(),
12     timeString: format(new Date(), 'EEEE - hh:mm:ss a'),
13   };
14 }
15
16 console.log(stamp('invalid@@email'));
17 console.log(stamp('valid@email.com'));
```

`validator` package will validate an email

`date-fns` package will create a Date() object given a date format.

`uuid` package will always generate a unique integer (id)

tests: JEST

When writing multi-file code for ‘big’ projects, it is crucial to test your code.

This ensures that your code is behaving as expected, and serves as a way to continually check your code is correct as you implement new functions.

In JS we use the jest library to test our code.

Writing tests using jest is worth 40% of Iteration 1

Setting up jest: Installing

```
npm install --save-dev jest @babel/preset-env
```

Setting up jest: Make our code use jest

Modify `package.json` to use `jest` as our test script

```
"scripts": {
  "test": "jest"
}
```

Setting up jest: How to setup files with jest?

auth.js

auth.test.js

Setting up jest: Running our tests

```
npm run test
```

```
 PASS  __test__\reducers.test.js
  >>> R E D U C E R S <<<
    √ +++ INIT STORE (4ms)
    √ +++ CREATE_ORDER (1ms)
    √ +++ COUNTER_UP: 0 -> 1 (1ms)
    √ +++ COUNTER_UP: 49 -> 50
    √ +++ FILTER: fo -> foo (1ms)
    √ +++ ORDER_NEXT_STEP: move to next step (1ms)
    √ +++ ORDER_PREVIOUS_STEP: move between steps (1ms)

 PASS  __test__\actions.test.js
  >>> A C T I O N S <<<
    √ +++ actionCreator: count (2ms)
    √ +++ actionCreator: filterTable (1ms)

Test Suites: 2 passed, 2 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        0.873s, estimated 2s
Ran all test suites.
-------------|----------|----------|----------|----------|-----------------|
File         |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Lines |
-------------|----------|----------|----------|----------|-----------------|
All files    |       94 |       88 |      100 |       94 |                 |
 actions     |      100 |      100 |      100 |      100 |                 |
  index.js   |      100 |      100 |      100 |      100 |                 |
  types.js   |      100 |      100 |      100 |      100 |                 |
 reducers    |    92.31 |       88 |      100 |    92.31 |                 |
  counter.js |      100 |      100 |      100 |      100 |                 |
  filter.js  |      100 |      100 |      100 |      100 |                 |
  index.js   |      100 |      100 |      100 |      100 |                 |
  order.js   |    90.63 |    84.21 |      100 |    90.63 |       26,35,37  |
-------------|----------|----------|----------|----------|-----------------|
Done in 2.17s.
```

UNSW SYDNEY

using jest: Creating a basic test

auth.test.js

```
1  test('a good test name', () => {
2    expect(yourOutput).toEqual(expectedOutput);
3  })
```

test -> Initialises a test block for JEST to run

expect -> JEST function that MUST be true for the test to pass

toEqual -> compares your output with expected output

Teamwork Principles

Brainstorm a list of good and poor attributes for team members working on a project.

Suppose there is a member in a group who has gone silent and has yet to complete much of their assigned work. The reason is unknown.

- How should the remaining members approach the situation?
- What can groups do beforehand such that if this occurs, they can manage or minimise the impact?

🏡 Be a good team member !!!

📢 Communication

⏰ Time Management

💕 Honesty

📋 Responsibility

✨ Enthusiasm *(for your group + project!)*

🏡 But what if...

A team member suddenly disappears without doing their work!

What should you do?

- Immediately notify tutor AND allocate the work to backup group member(s)
- Maintain contact with the group member (maybe a message a day)
  - Please don't be aggressive
- Document it (in meeting minutes)

using jest: Let's give it a spin

1. Setup jest in our project folder
2. Create a new file called sum.js
3. Create a new file called sum.test.js
4. Write some tests for a basic sum function
5. Implement the sum function
6. Run the tests