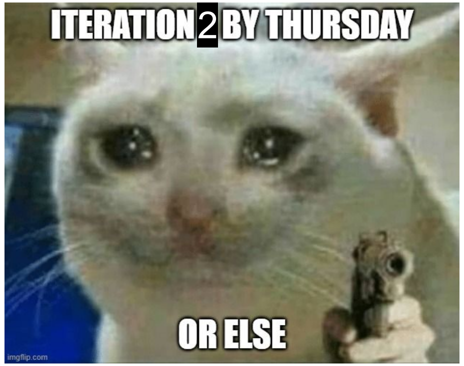
COMP1531 | T09B / H17B

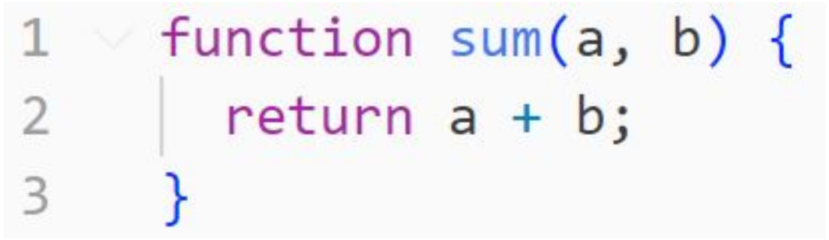
Week 4

william.huynh3@unsw.edu.au



Learning objectives

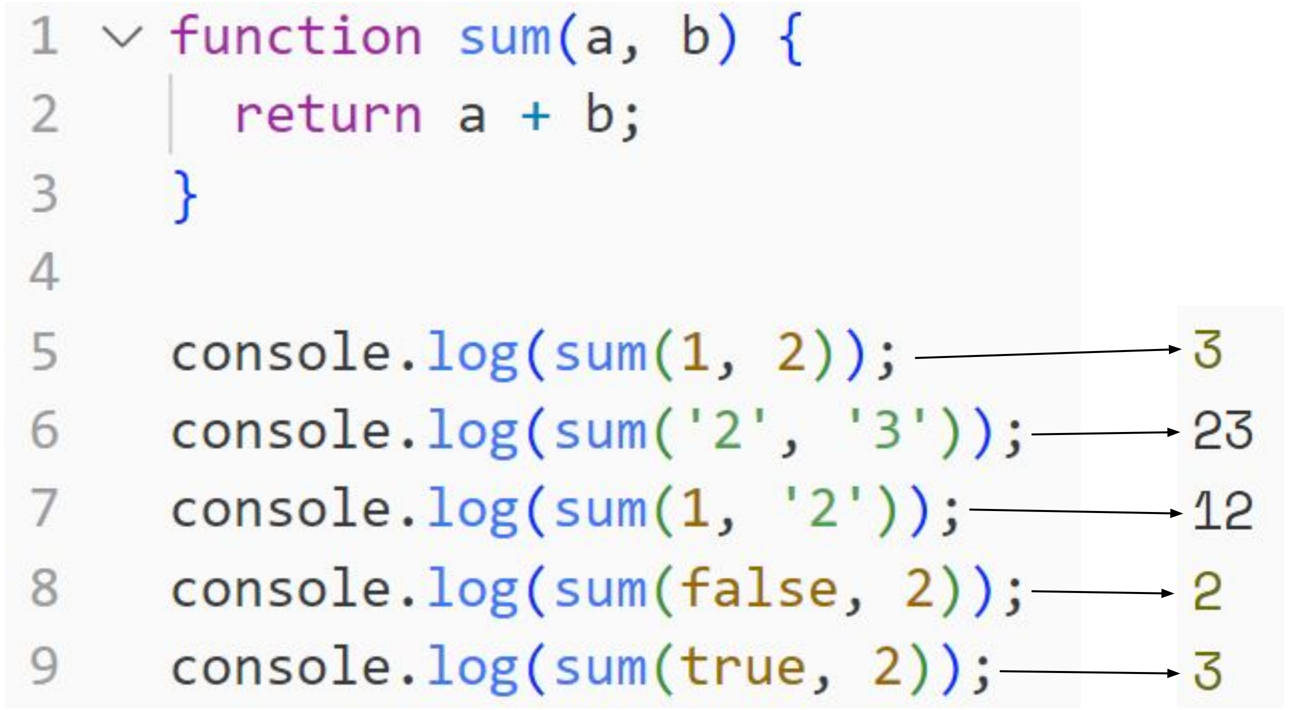
- *secret... for now...* 🤫
- *secret... for now...* 🤫
- **Linting** code to meet style requirements
- **Iteration 1 Q & A**



```
function sum(a, b) {
  return a + b;
}
```

What’s wrong with this code?

It’s not type safe!



```javascript
function sum(a, b) {
    return a + b;
}

console.log(sum(1, 2));
console.log(sum('2', '3'));
console.log(sum(1, '2'));
console.log(sum(false, 2));
console.log(sum(true, 2));
```

```
3
23
12
2
3
```

Learning objectives

- Using TypeScript for type safety
- Making our Iteration 1 functions Type Safe
- Linting and code review
- Iteration 1 Q & A

TypeScript: Type safety in JS

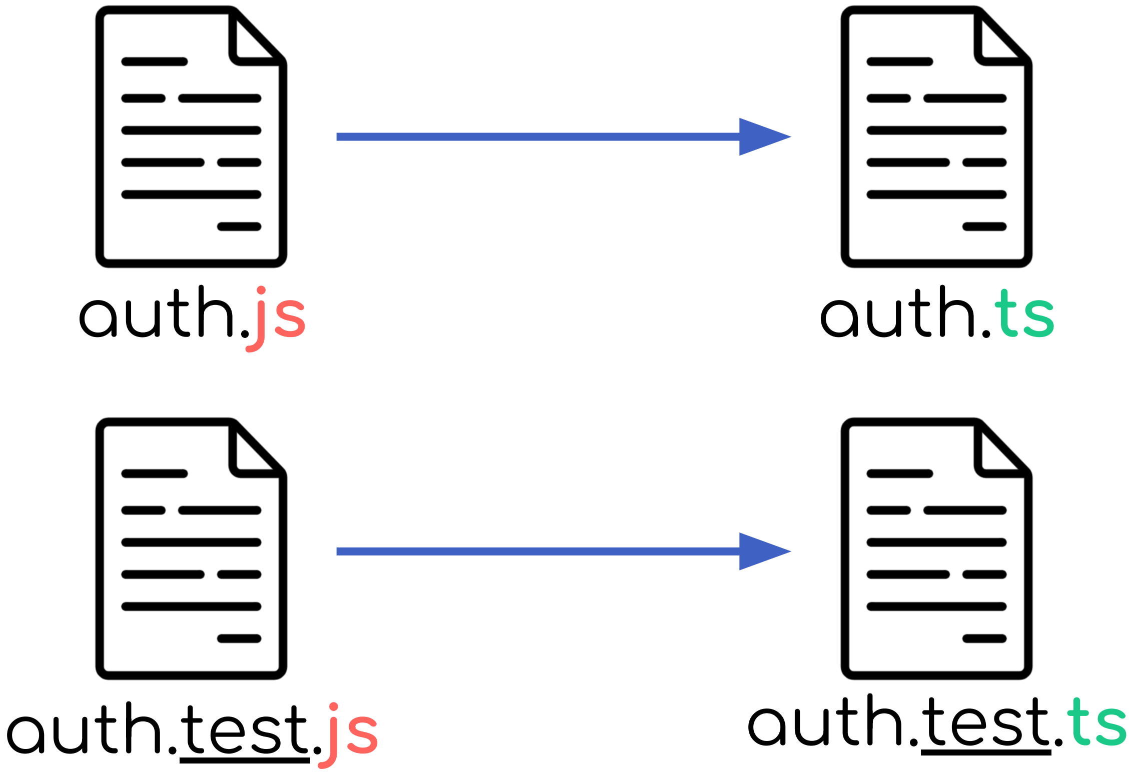
Typescript is a language built on top of Javascript. It is installed using npm.

It's job is to statically verify the types in your program and outputs Javascript that you can then run normally using `node`

```
npm install --save-dev typescript
```

Type Safety contributes towards 40% of Iteration 2 but is NOT assessed in Iteration 1

TypeScript: How to use?



TypeScript: The supported types

TypeScript supports the following data types

Boolean

Number

String

Array

Objects *(more on this soon)*

And lots more (but may not be relevant) ...

TypeScript: Where is it applied?

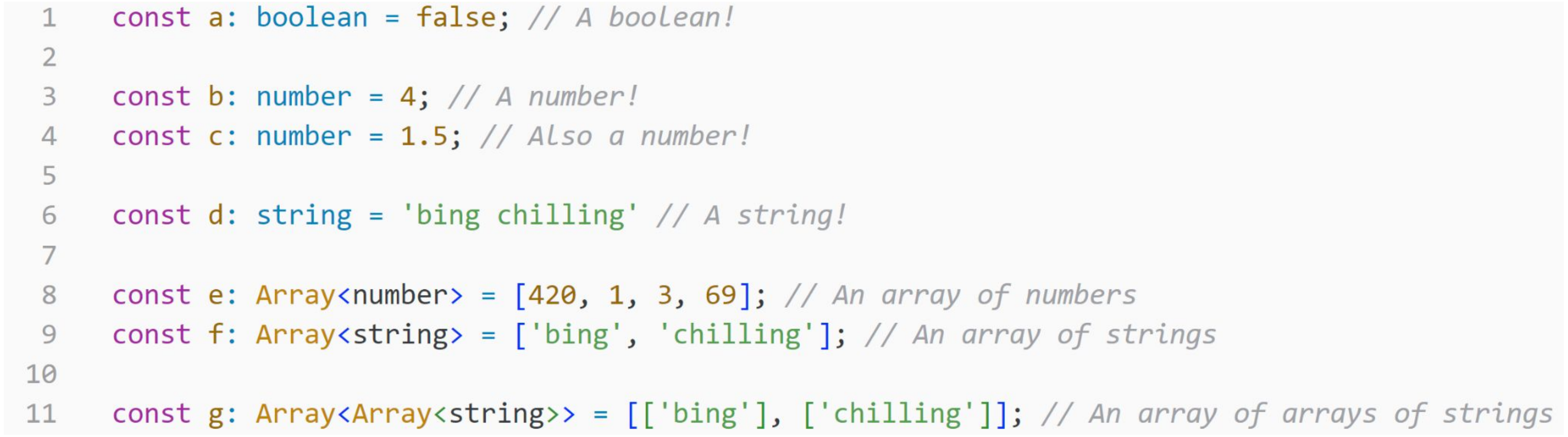
TypeScript is applied in:

Variable Declarations

Function Parameters

Function Returns

TypeScript Application: Variables



```typescript
const a: boolean = false; // A boolean!

const b: number = 4; // A number!
const c: number = 1.5; // Also a number!

const d: string = 'bing chilling' // A string!

const e: Array<number> = [420, 1, 3, 69]; // An array of numbers
const f: Array<string> = ['bing', 'chilling']; // An array of strings

const g: Array<Array<string>> = [['bing'], ['chilling']]; // An array of arrays of strings
```

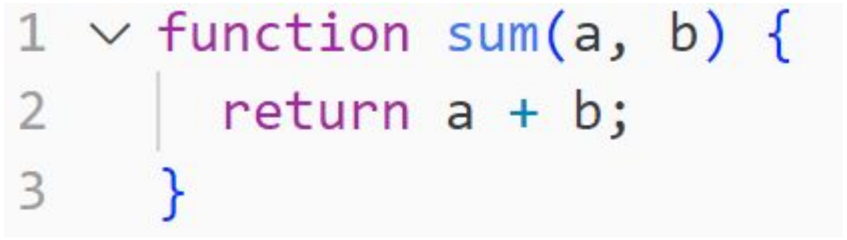
✗

```
const a = false; // A boolean!

const b = 4; // A number!
const c = 1.5; // Also a number!

const d = 'bing chilling' // A string!

const e = [420, 1, 3, 69]; // An array of numbers
const f = ['bing', 'chilling']; // An array of strings

const g = [['bing'], ['chilling']]; // An array of arrays of strings
```

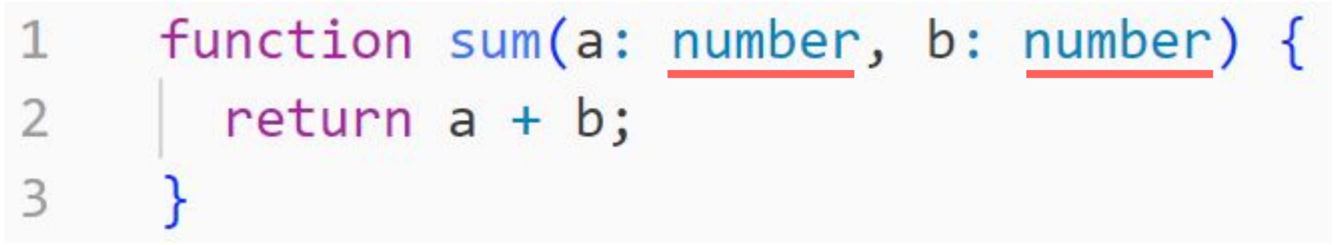
✓

```
const a: boolean = false; // A boolean!

const b: number = 4; // A number!
const c: number = 1.5; // Also a number!

const d: string = 'bing chilling' // A string!

const e: Array<number> = [420, 1, 3, 69]; // An array of numbers
const f: Array<string> = ['bing', 'chilling']; // An array of strings

const g: Array<Array<string>> = [['bing'], ['chilling']]; // An array of arrays of strings
```

TypeScript Application: Function Parameters

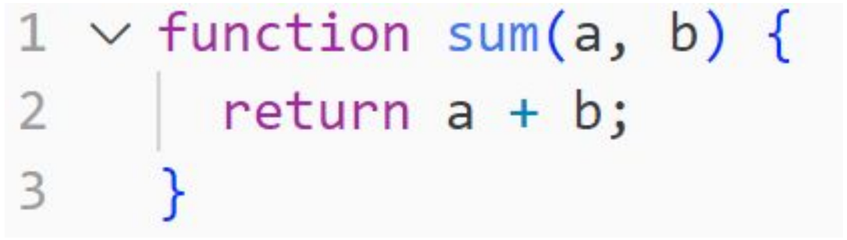


```javascript
function sum(a, b) {
    return a + b;
}
```

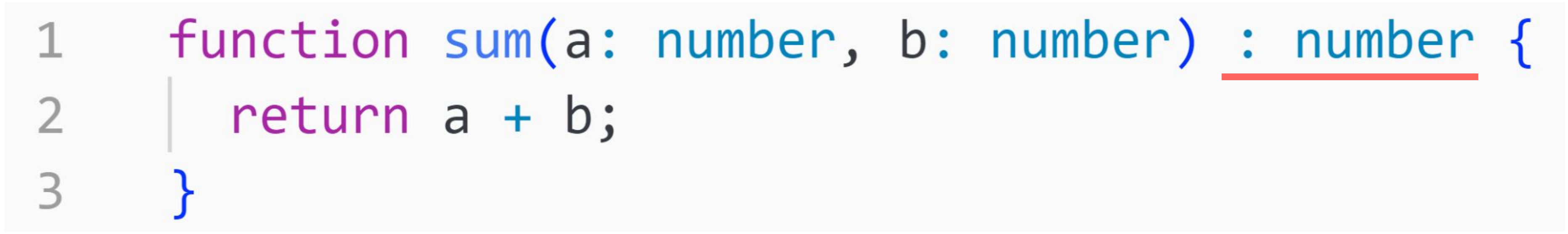


```typescript
function sum(a: number, b: number) {
    return a + b;
}
```

TypeScript Application: Function Returns



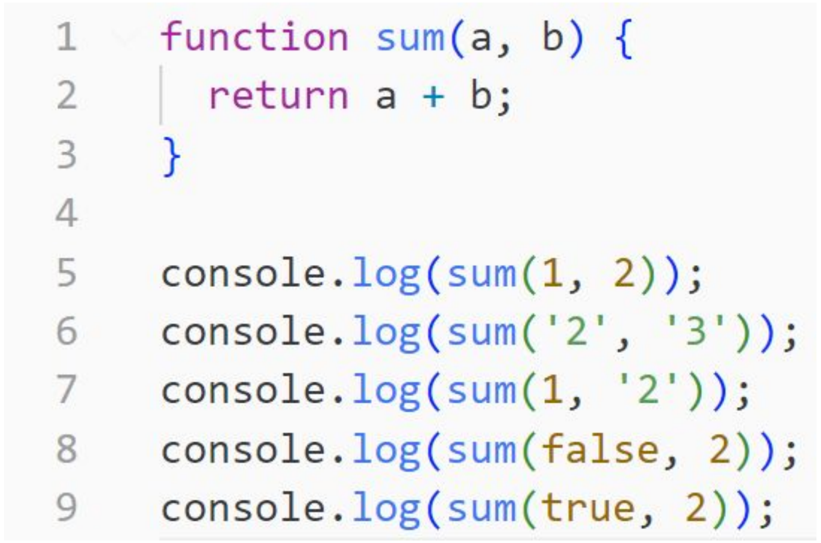
```
function sum(a, b) {
    return a + b;
}
```



```
function sum(a: number, b: number) : number {
    return a + b;
}
```

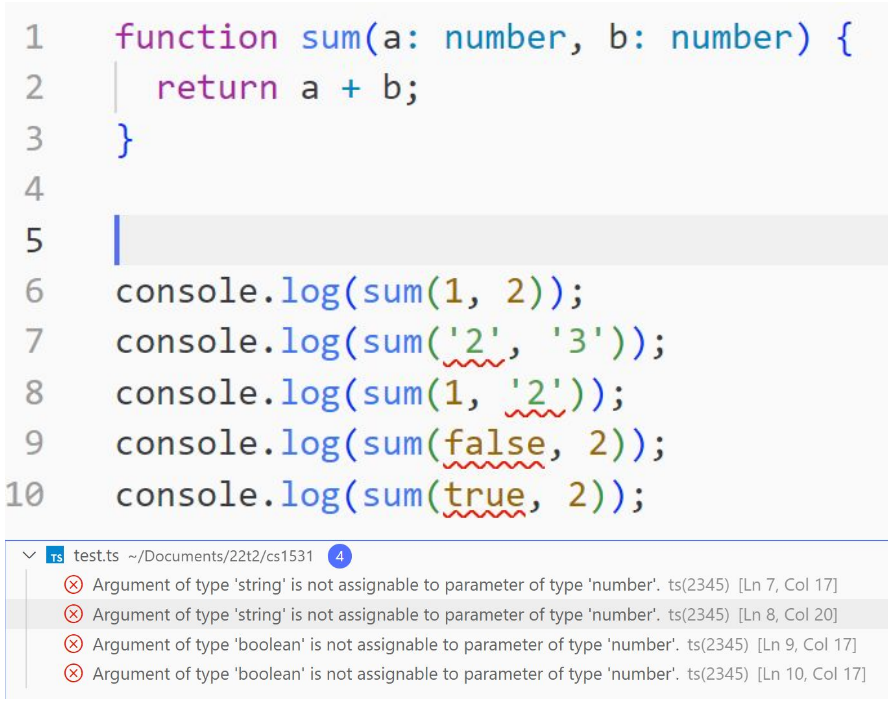
TypeScript: The differences

TypeScript not used



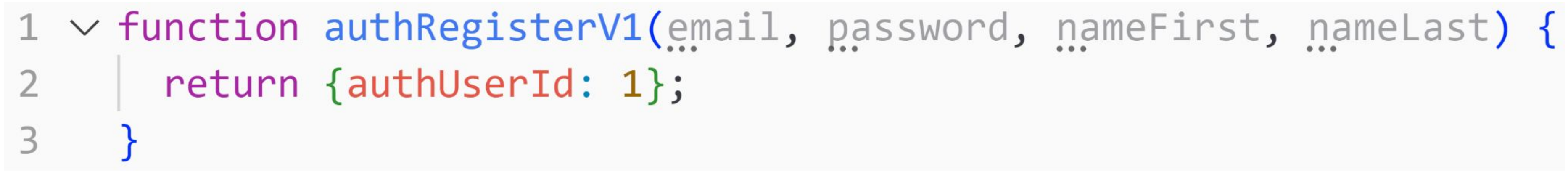
```
function sum(a, b) {
  return a + b;
}

console.log(sum(1, 2));
console.log(sum('2', '3'));
console.log(sum(1, '2'));
console.log(sum(false, 2));
console.log(sum(true, 2));
```

TypeScript is used

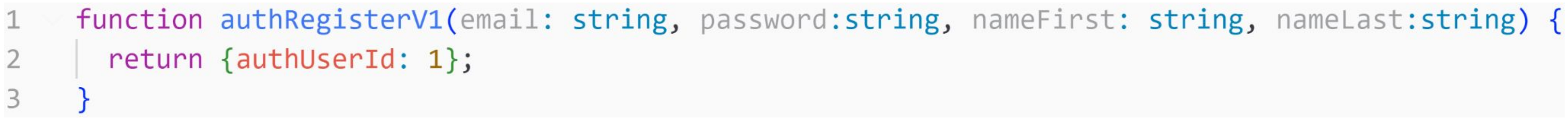


```
function sum(a: number, b: number) {
  return a + b;
}


console.log(sum(1, 2));
console.log(sum('2', '3'));
console.log(sum(1, '2'));
console.log(sum(false, 2));
console.log(sum(true, 2));
```

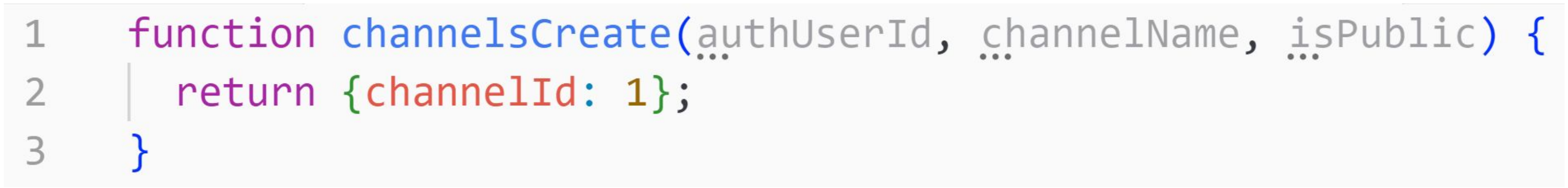
```
test.ts ~/Documents/22t2/cs1531 4
  Argument of type 'string' is not assignable to parameter of type 'number'. ts(2345) [Ln 7, Col 17]
  Argument of type 'string' is not assignable to parameter of type 'number'. ts(2345) [Ln 8, Col 20]
  Argument of type 'boolean' is not assignable to parameter of type 'number'. ts(2345) [Ln 9, Col 17]
  Argument of type 'boolean' is not assignable to parameter of type 'number'. ts(2345) [Ln 10, Col 17]
```



```
function authRegisterV1(email, password, nameFirst, nameLast) {
  return {authUserId: 1};
}
```

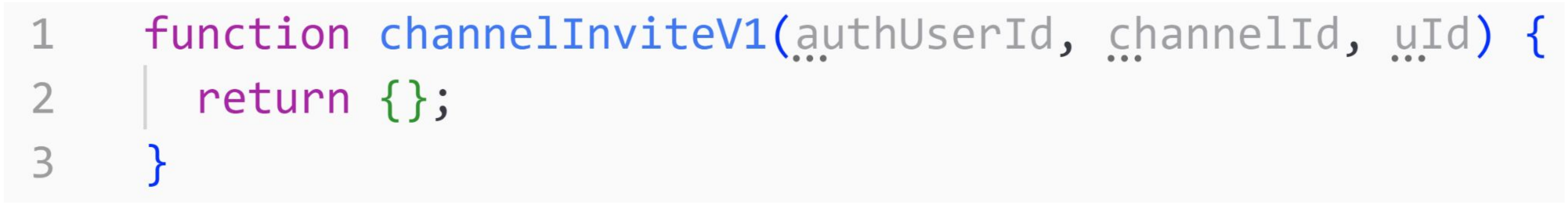


```
function authRegisterV1(email: string, password:string, nameFirst: string, nameLast:string) {
  return {authUserId: 1};
}
```

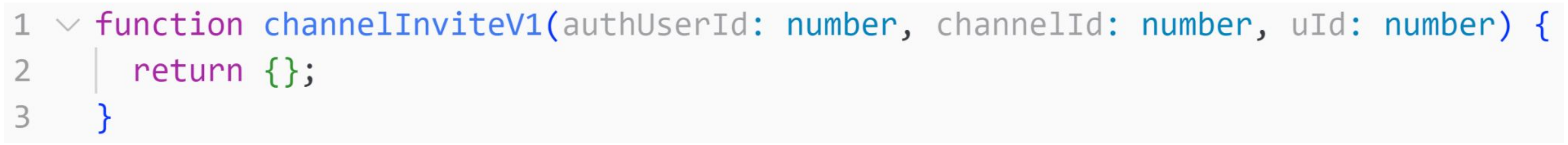


```javascript
function channelsCreate(authUserId, channelName, isPublic) {
  return {channelId: 1};
}
```

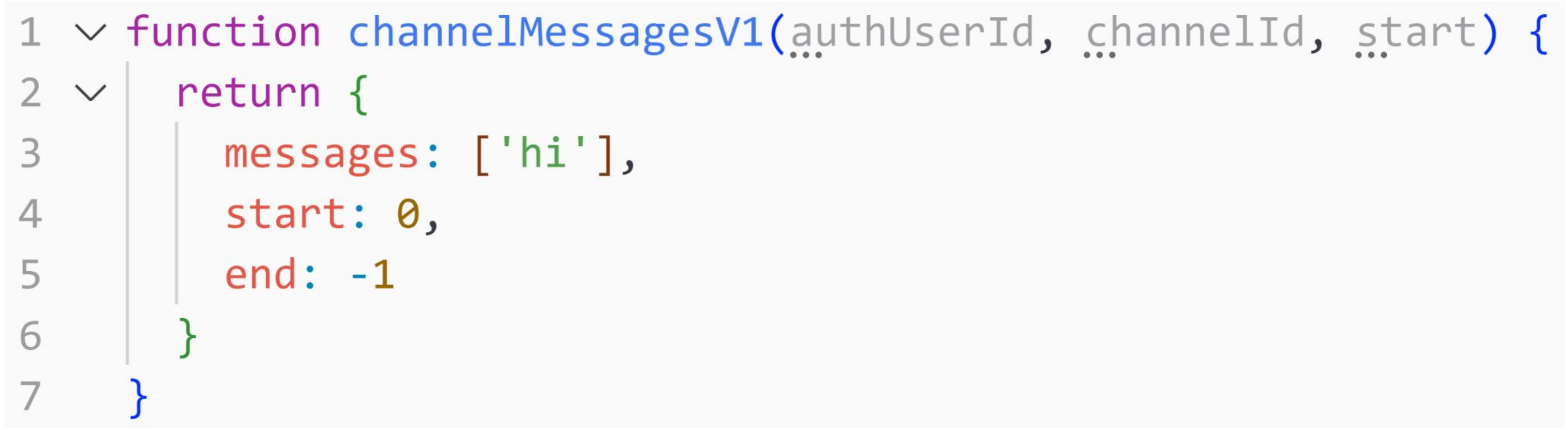
```
function channelsCreate(authUserId: number, channelName: string, isPublic: boolean) {
  return {channelId: 1};
}
```



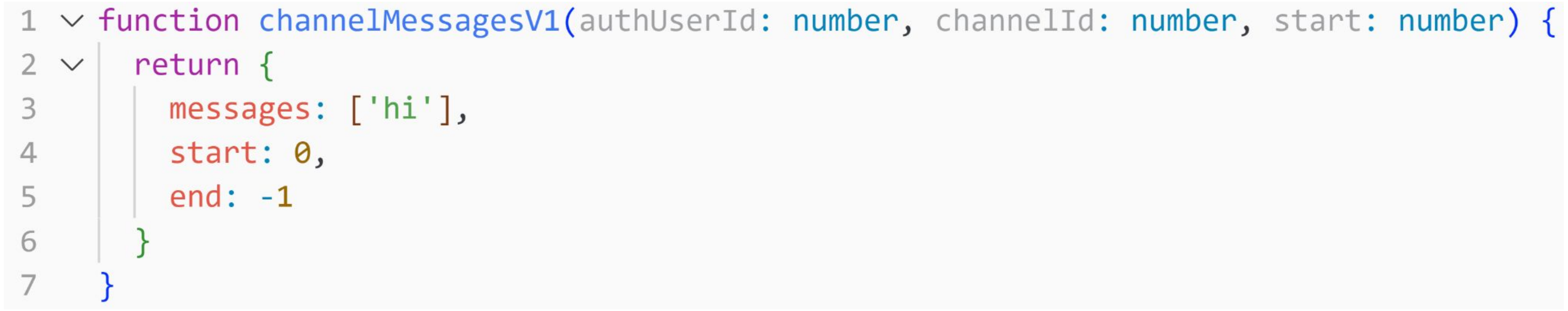
```
function channelInviteV1(authUserId, channelId, uId) {
  return {};
}
```



```
function channelInviteV1(authUserId: number, channelId: number, uId: number) {
  return {};
}
```



```javascript
function channelMessagesV1(authUserId, channelId, start) {
  return {
    messages: ['hi'],
    start: 0,
    end: -1
  }
}
```

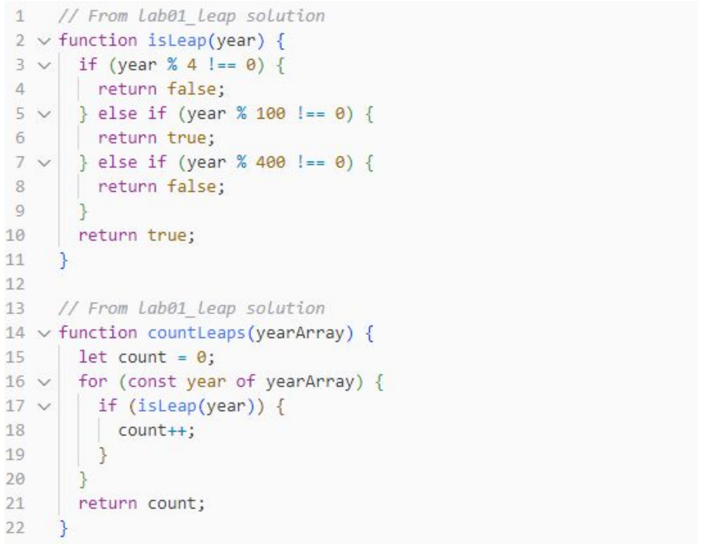


```
function channelMessagesV1(authUserId: number, channelId: number, start: number) {
  return {
    messages: ['hi'],
    start: 0,
    end: -1
  }
}
```

Do these functions only:

`function isLeap()`

`function countLeaps()`



```
// From lab01_leap solution
function isLeap(year) {
  if (year % 4 !== 0) {
    return false;
  } else if (year % 100 !== 0) {
    return true;
  } else if (year % 400 !== 0) {
    return false;
  }
  return true;
}

// From lab01_leap solution
function countLeaps(yearArray) {
  let count = 0;
  for (const year of yearArray) {
    if (isLeap(year)) {
      count++;
    }
  }
  return count;
}
```

Activity

Iteration 2 is nearly due and your team has realised that their functions are still in regular JavaScript and are not type safe.

Feeling worried of losing the 40% type safety marks, they choose their best programmer (you) to be tasked with type annotating the old JavaScript functions into newer and better TypeScript functions.

Can you do so in 25 minutes before It2 is due???

Go to the tute04 GitLab (found on webcms) and download the repo as a zip file. Extract the zip file into a folder and open that folder in vscode.

The file to annotate is `b.typing/rescript.js`

TypeScript: The supported types

TypeScript supports the following data types

Boolean

Number

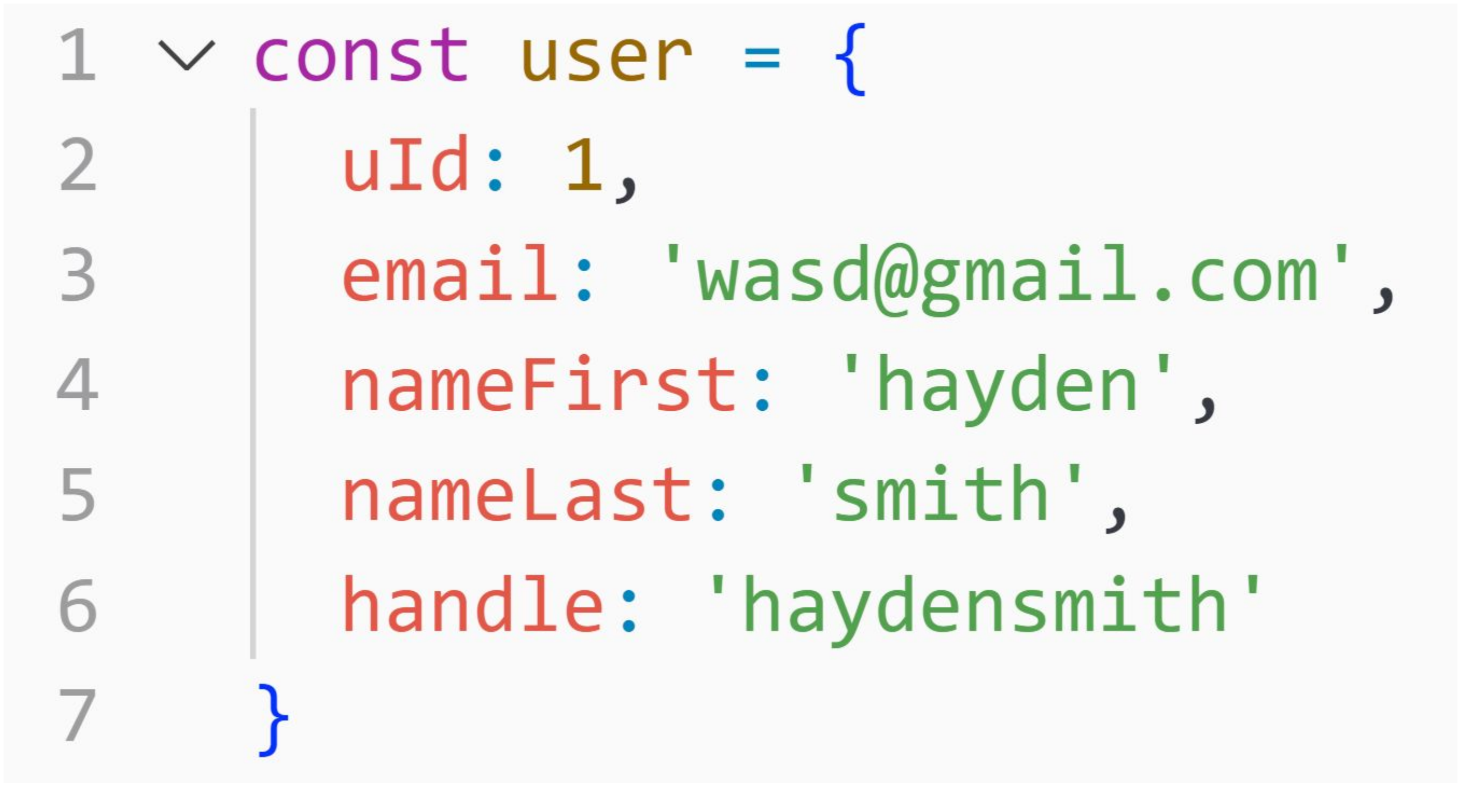
String

Array

Objects *(more on this soon)*

And lots more (but may not be relevant) ...

TypeScript



```typescript
const user = {
    uId: 1,
    email: 'wasd@gmail.com',
    nameFirst: 'hayden',
    nameLast: 'smith',
    handle: 'haydensmith'
}
```

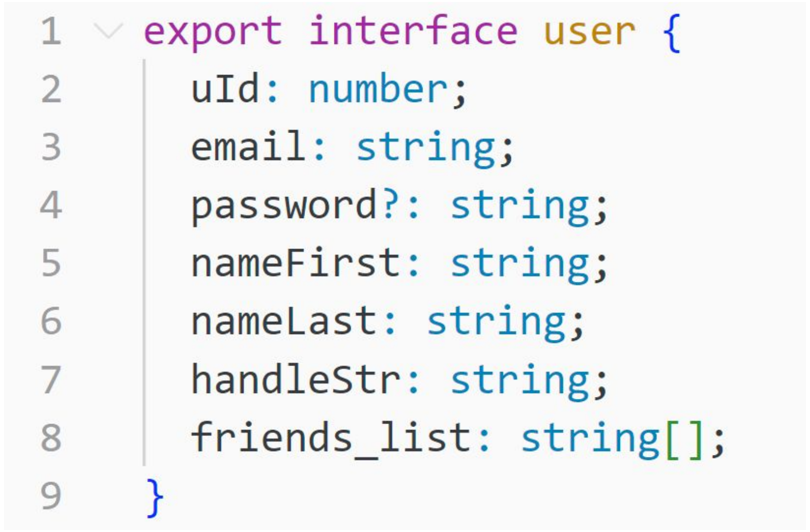
TypeScript

```
function userProfileV1(authUserId: number, uId: number) {
  return user; // where user is an object
}
```

```
function processUser(user){
  const uId: number = user.uId;
  const nameFirst: string = user.nameFirst
  // other stuff below
}
```

TypeScript: Interfaces

An interface allows us to declare the structure and types of an object before we use it !



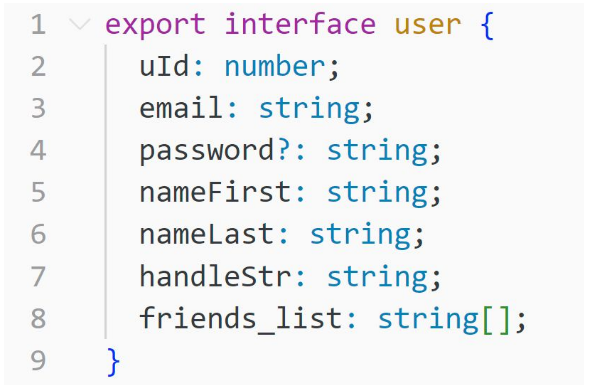
```typescript
export interface user {
  uId: number;
  email: string;
  password?: string;
  nameFirst: string;
  nameLast: string;
  handleStr: string;
  friends_list: string[];
}
```

```typescript
export interface user {
  uId: number;
  email: string;
  password?: string;
  nameFirst: string;
  nameLast: string;
  handleStr: string;
  friends_list: string[];
}
```

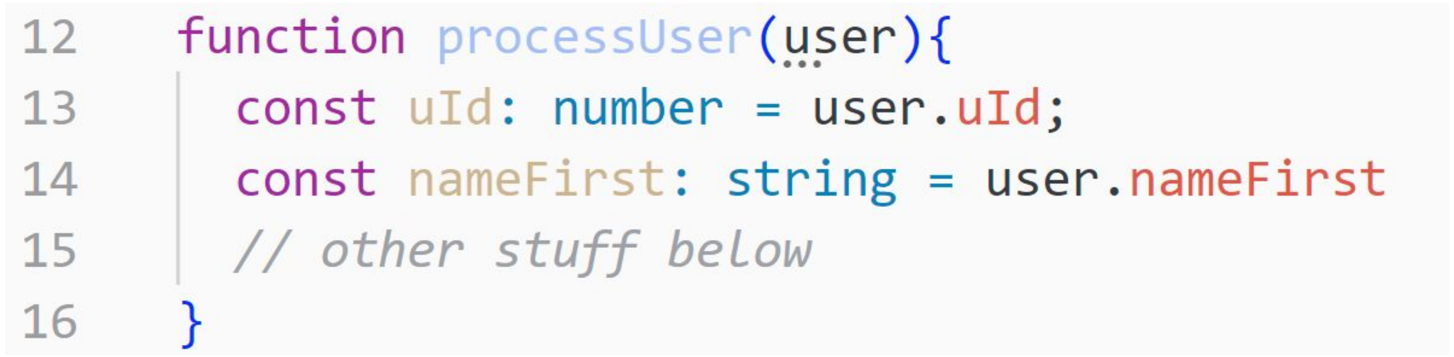
```typescript
function userProfileV1(authUserId: number, uId: number) {
  return user; // where user is an object
}
```

```
export interface user {
  uId: number;
  email: string;
  password?: string;
  nameFirst: string;
  nameLast: string;
  handleStr: string;
  friends_list: string[];
}
```

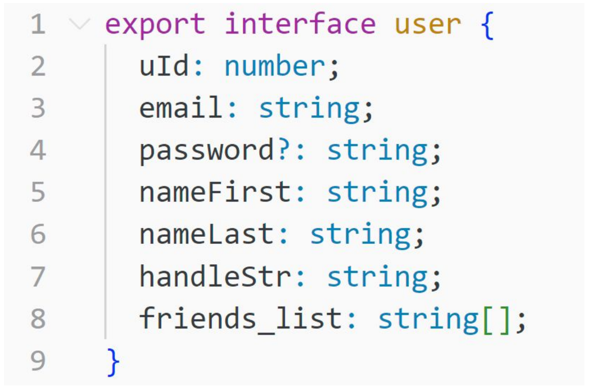
```
function userProfileV1(authUserId: number, uId: number): user {
  return user; // where user is an object
}
```



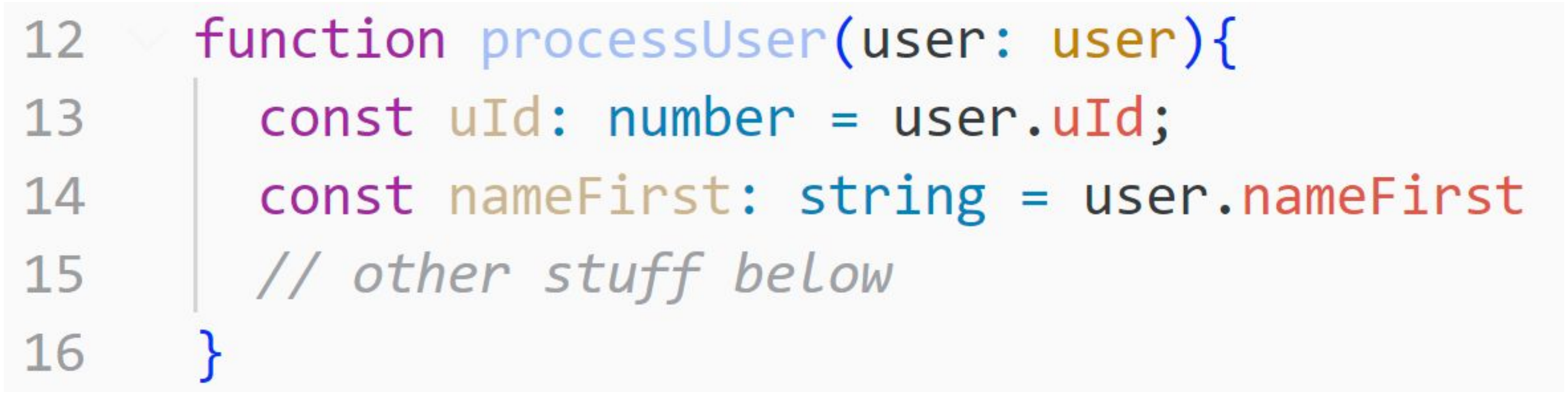
```
export interface user {
  uId: number;
  email: string;
  password?: string;
  nameFirst: string;
  nameLast: string;
  handleStr: string;
  friends_list: string[];
}
```



```
function processUser(user){
  const uId: number = user.uId;
  const nameFirst: string = user.nameFirst
  // other stuff below
}
```



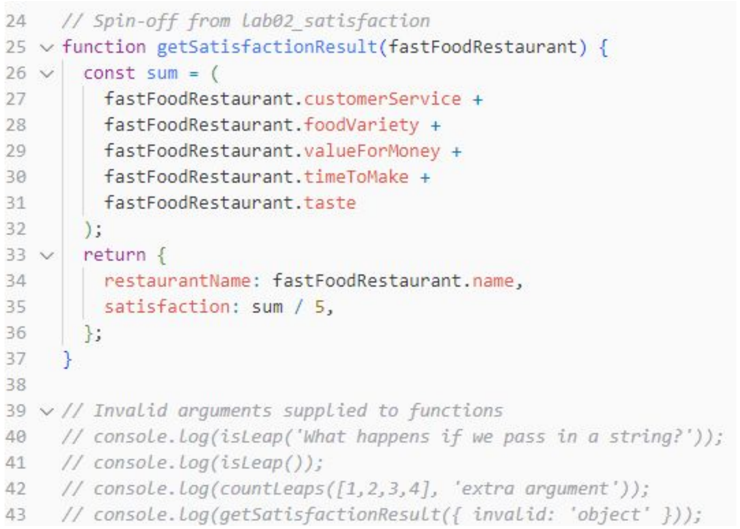
```
export interface user {
  uId: number;
  email: string;
  password?: string;
  nameFirst: string;
  nameLast: string;
  handleStr: string;
  friends_list: string[];
}
```



```
function processUser(user: user){
  const uId: number = user.uId;
  const nameFirst: string = user.nameFirst
  // other stuff below
}
```

Do these functions only:

```
function getSatisfactionResult()
```



```
// Spin-off from lab02_satisfaction
function getSatisfactionResult(fastFoodRestaurant) {
  const sum = (
    fastFoodRestaurant.customerService +
    fastFoodRestaurant.foodVariety +
    fastFoodRestaurant.valueForMoney +
    fastFoodRestaurant.timeToMake +
    fastFoodRestaurant.taste
  );
  return {
    restaurantName: fastFoodRestaurant.name,
    satisfaction: sum / 5,
  };
}

// Invalid arguments supplied to functions
// console.log(isLeap('What happens if we pass in a string?'));
// console.log(isLeap());
// console.log(countLeaps([1,2,3,4], 'extra argument'));
// console.log(getSatisfactionResult({ invalid: 'object' }));
```

Activity

Iteration 2 is nearly due and your team has realised that their functions are still in regular JavaScript and are not type safe.

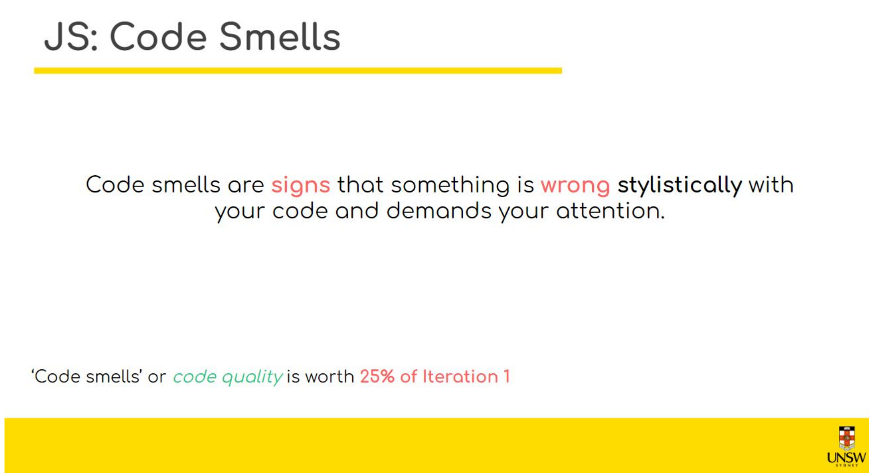
Feeling worried of losing the 40% type safety marks, they choose their best programmer (you) to be tasked with type annotating the old JavaScript functions into newer and better TypeScript functions.

Can you do so in 25 minutes before It2 is due???

Go to the tute04 GitLab (found on webcms) and download the repo as a zip file. Extract the zip file into a folder and open that folder in vscode.

The file to annotate is `b.typing/rescript.js`

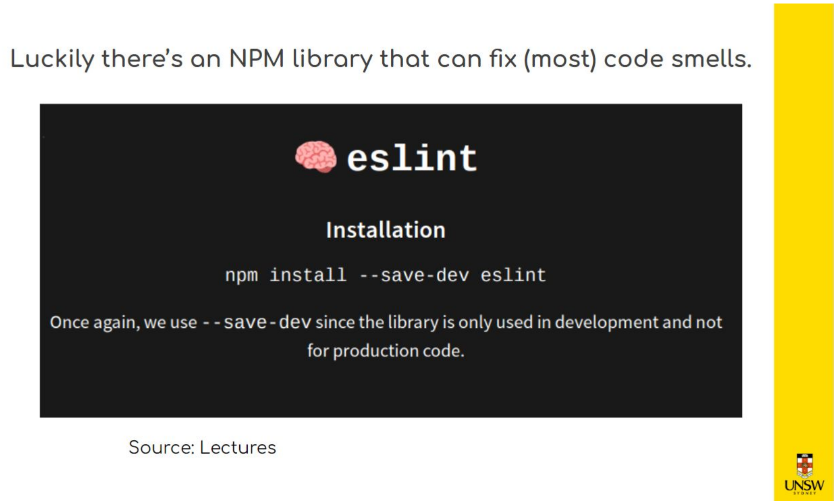
Last week we discussed...



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



Luckily there's an NPM library that can fix (most) code smells.

eslint

Installation

```
npm install --save-dev eslint
```

Once again, we use `--save-dev` since the library is only used in development and not for production code.

Source: Lectures

eslint: Linting our code

eslint is a node package that automatically styles our code for us!

It will fix most common style errors like indentation, line overflow or unaesthetic control flow

But it won't fix bad variable names, excessive function parameters, or overly-complex code.

A mixture of both automatic and manual checks is ideal !!!

Linting/code quality is worth 25% of Iteration 1

eslint: How to make our code use eslint?

Add a `lint` script to `package.json` along with any other scripts that may be useful.

```
"scripts": {
    "test": "jest src",
    "tsc": "tsc --noEmit",
    "ts-node": "ts-node",
    "lint": "eslint",
    "lint-fix": "eslint --fix",
}
```

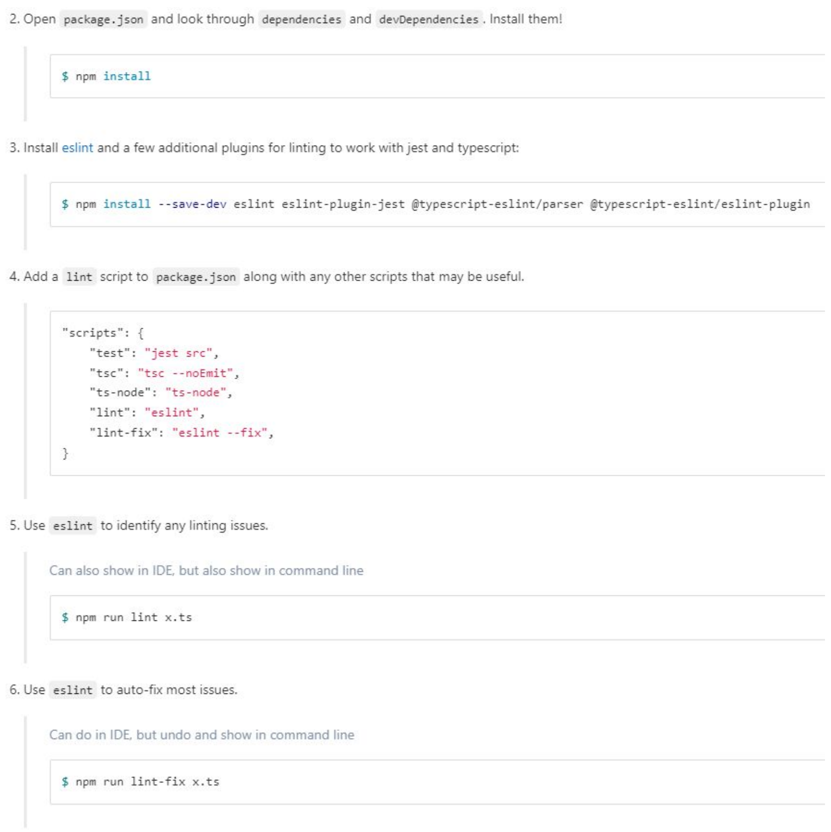
eslint: How to run eslint?

This command will identify the style-errors in your file

```
npm run lint myfile.ts
```

This command will automatically fix the style-errors in your file

```
npm run lint-fix myfile.ts
```



2. Open `package.json` and look through `dependencies` and `devDependencies`. Install them!

```
$ npm install
```

3. Install eslint and a few additional plugins for linting to work with jest and typescript:

```
$ npm install --save-dev eslint eslint-plugin-jest @typescript-eslint/parser @typescript-eslint/eslint-plugin
```

4. Add a `lint` script to `package.json` along with any other scripts that may be useful.

```
"scripts": {
    "test": "jest src",
    "tsc": "tsc --noEmit",
    "ts-node": "ts-node",
    "lint": "eslint",
    "lint-fix": "eslint --fix",
}
```

5. Use `eslint` to identify any linting issues.

Can also show in IDE, but also show in command line

```
$ npm run lint x.ts
```

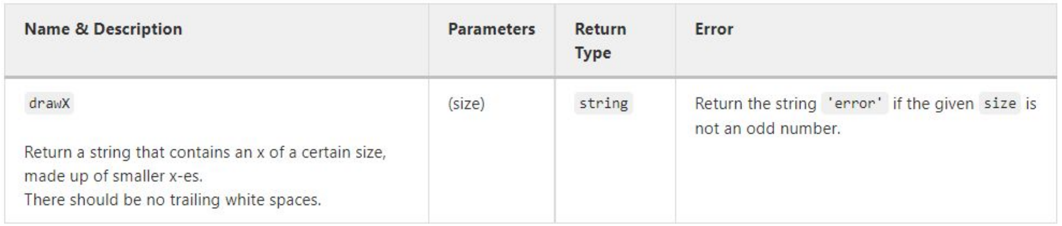
6. Use `eslint` to auto-fix most issues.

Can do in IDE, but undo and show in command line

```
$ npm run lint-fix x.ts
```

Activity

Below is a piece of software written by a COMP1531 student back when they were still a newbie programmer in COMP1511. This was the interface that they followed:



Name & Description	Parameters	Return Type	Error
`drawX` Return a string that contains an x of a certain size, made up of smaller x-es. There should be no trailing white spaces.	(size)	`string`	Return the string `'error'` if the given `size` is not an odd number.

Take a look at `c.linting/x.ts`.

Discuss in your groups:

- What, if any, are some good points about the implementation?
- What are some styling/design issues?
- (add comments in the code!)

Afterwards, fix the design issues with eslint !!

eslint: How to install (with TypeScript and Jest support) ??

```
npm install --save-dev eslint eslint-plugin-jest @typescript-eslint/parser @typescript-eslint/eslint-plugin
```

The main eslint package

Allows eslint to work with jest

Allows eslint to work with typescript

`--save-dev` will save the **package** in packages.json