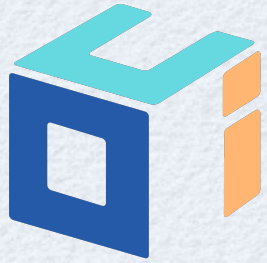
Modern JavaScript Tools

slides at https://github.com/mvolkmann/talks

R. Mark Volkmann
Object Computing, Inc.
http://objectcomputing.com
Email: mark@objectcompuing.com
Twitter: @mark_volkmann
GitHub: mvolkmann



OCI | TRAINING

Table of Contents

The Plan

For each topic we will cover ...

1) purpose

2) alternatives

3) how to configure and use

Demo App

- "Hello World" web app
  - very simple to avoid distracting from focus on tools
  - doesn't use any framework to avoid bias toward one
  - available at https://github.com/mvolkmann/modern-js-tools
- Notable files
  - `index.html`
  - `demo.css`
  - `src/index.js`
  - `src/demo.js`
  - `src/demo.test.js`
  - `package.json`
  - plus several tool configuration files discussed later

npm Overview

https://www.npmjs.com/

- Purpose
  - installs Node packages
  - manages dependencies
  - scripts common tasks
- Alternatives
  - for installing and managing dependencies - **yarn**
  - for scripting tasks - **gulp** and **Grunt**

npm Details

- Automatically installed when Node.js is installed
  - can install separately, but why?
- Initially an acronym for Node Package Manager
  - they are trying hard to convince us that it's not an acronym now ... not sure why
- Common commands
  - **`npm init`** - asks questions and creates **`package.json`**
  - **`npm install`** ***`name`*** - installs a specified package as a runtime dependency
  - **`npm install -D`** ***`name`*** - installs a specified package as a development dependency
  - **`npm install`** - installs all dependencies listed in **`package.json`**
  - **`npm run`** ***`script-name`*** and **`npm`** ***`script-name`*** - runs an npm script

for "special" scripts

npm Scripts

- Defined in **`package.json`**
- Can write in a way that works on Windows and *nix platforms
  - **shx** - "Portable Shell Commands for Node"
    - https://github.com/shelljs/shx
  - **cross-env** - "Run scripts that set and use environment variables across platforms"
    - https://github.com/kentcdodds/cross-env
- Examples

```
"build": "npm-run-all verify bundle",
"bundle": "webpack",
"clean": "rm -rf build coverage",
"cover": "CI=true jest src -- --coverage",
"cover-open": "open coverage/lcov-report/index.html",
"flow": "flow",
"format": "prettier-eslint --write src/**/*.js",
"lint": "eslint --quiet src --ext .js",
"sync": "browser-sync start --server --files 'index.html, build/bundle.js'",
"test": "jest --watch src",
"verify": "npm-run-all lint flow cover"
```

ESLint Overview

http://eslint.org/

- Purpose
  - "The pluggable linting utility for JavaScript and JSX"
  - can report many syntax errors and potential run-time errors
  - can report deviations from specified coding guidelines
- Alternatives
  - **JSLint** - from Douglas Crockford
  - **JSHint** - a more configurable, less opinionated version of JSLint

ESLint Details

- Error messages identify violated rules,
  making it easy to adjust them if you disagree
- Has **`--fix`** mode that can fix violations of many rules
  - modifies source files
- To install, **`npm install -D eslint`**
- To use from an npm script, add following to **`package.json`**

```
"lint": "eslint --quiet src --ext .js",
```

`--quiet` only reports errors

- Editor/IDE integrations available
  - Atom, Eclipse, emacs, Intellij IDEA, Sublime, Visual Studio Code, Vim, WebStorm

may also want **`eslint-plugin-flowtype`**,
`eslint-plugin-html`, and **`eslint-plugin-react`**

ESLint Rules

- No rules are enforced by default
- Desired rules must be configured
- See list of current rules at http://eslint.org/docs/rules/
- Configuration file formats supported
  - JSON - **`.eslintrc.json`**; can include JavaScript comments; most popular
  - JavaScript - **`.eslintrc.js`**
  - YAML - **`.eslintrc.yaml`**
  - inside **`package.json`** using **`eslintConfig`** property
  - use of **`.eslintrc`** containing JSON or YAML is deprecated

see mine at https://github.com/mvolkmann/MyUnixEnv/blob/master/**.eslintrc.json**

- Searches upward from current directory for these files
  - combines settings in all configuration files found with settings in closest taking precedence
  - configuration file in home directory is only used if no other configuration files are found

ESLint Demo

- See **`lint`** script in **`package.json`**
- Modify **`src/emo.js`**
  - remove semicolon from **`if`** statement in **`getGreeting`** function
  - remove an "**`e`**" from name of **`handleGreet`** function
- **`npm run lint`**

Prettier Overview

https://github.com/prettier/prettier

- Purpose
  - "An opinionated JavaScript formatter ...
    with advanced support for language features
    from ES2017, JSX, Flow, TypeScript, CSS, LESS, and SCSS"
  - "Parses your JavaScript into an AST and pretty-prints the AST,
    completely ignoring any of the original formatting"
  - can also format JSX, Flow, TypeScript, JSON, CSS, Less, Sass, Markdown, and more
- Alternatives
  - **Standard** - https://standardjs.com/
    - no semicolons
    - space after function names before left paren

Prettier Details

- To install, **`npm install -D prettier`**
- To use from an npm script, add following to **`package.json`**

```
"format": "prettier --no-bracket-spacing --single-quote --write src/**/*.{css,js}",
```

  - to format all **`.js`** files under **`src`** directory, enter **`npm run format`**
  - **`--write`** option overwrites existing files with formatted versions
- Can also configure in a **`.prettierrc`** file

```
{
  "bracketSpacing": false,
  "singleQuote": true
}
```

- Doesn't run on files under **`node_modules`** by default
- Editor/IDE integrations available
  - Atom, Emacs, JetBrains, Sublime, Vim, Visual Studio Code

Prettier Options

- **`--jsx-bracket-same-line`**
  - puts closing > of JSX start tags on last line instead of on new line

```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1"
>
  content
</something>
```

VS.

```
<something
  prop1="value1"
  prop2="value1"
  prop3="value1"
  prop4="value1">
  content
</something>
```

- **`--no-bracket-spacing`**
  - omits spaces between brackets in object literals

```
{ foo='1' bar=true }
```

VS.

```
{foo='1' bar=true}
```

- **`--no-semi`** - omits semicolons
- **`--print-width n`** - defaults to 80
- **`--single-quote`**
  - uses single quotes instead of double quotes for string delimiters
- **`--tab-width n`** - defaults to 2
- **`--trailing-comma`**
  - adds trailing commas wherever possible; defaults to none
- **`--use-tabs`** - uses tabs instead of spaces for indentation
- and more lesser used options

prettier-eslint-cli

https://github.com/prettier/prettier-eslint-cli

- Command-line interface to prettier-eslint
- "Formats your JavaScript using **`prettier`** followed by **`eslint --fix`**"
- "Get the benefits of Prettier's superior formatting capabilities, but also benefit from the configuration capabilities of ESLint"
- **`npm install -D prettier-eslint-cli`**
- To use from an npm script, change **`prettier`** to **`prettier-eslint`**

Prettier and CSS

- While Prettier can process CSS files, ESLint cannot
- So it doesn't make sense to run prettier-eslint-cli on CSS files
- Consider adding a separate npm script like

```
"format-css": "prettier --write src/**/*.css",
```

- If using Sass
  - no need to format generated CSS files

Prettier Demo

- See **`format`** script in **`package.json`**
- Modify **`src/demo.js`**
  - remove several semicolons
  - mess up lots of indentation
  - put each **`handleGreet`** parameter on a separate line
- **`npm run format`** and reload file in editor to see changes
  or trigger from editor/IDE plugin in my Vim, press **`,f`**

Babel Overview

- Purpose
  - transpiles JavaScript code to different JavaScript code
  - to use newer JS features in environments that don't support them yet
    - reads modern JS code and generates new JS code that runs in older environments; called "transpiling"
    - ex. **ES modules**
  - to use JS features not yet finalized by ECMAScript (via plugins)
    - ex. **object rest and spread operators**
  - to use features that may never be part of ECMAScript
    - ex. **Flow**
- Alternatives
  - **TypeScript** - also adds types and support for custom syntax

Babel Details

- To install, **`npm install -D babel-cli babel-preset-env`**
- To use from an npm script, add following to **`package.json`**

```
"babel": "babel src -d build"
```

  - not needed if using **`webpack`** and **`babel-loader`**

Babel Plugins

- Recommended plugins
  - **babel-preset-env**
    - "automatically determines the Babel plugins you need based on your supported environments"
    - can target specific browser versions and versions of Node.js
    - https://babeljs.io/docs/plugins/preset-env/
  - **babel-plugin-transform-flow-strip-types**
    - removes Flow type declarations from `.js` files
    - https://babeljs.io/docs/plugins/transform-flow-strip-types/
  - **babel-plugin-transform-object-rest-spread**
    - transforms code that uses object spread and object rest into equivalent code that doesn't
    - https://babeljs.io/docs/plugins/transform-object-rest-spread/
- To use a plugin
  - install with npm as a dev dependency
  - configure in **`.babelrc`** (see next slide)

Babel Configuration

- In `.babelrc` file
- Example

```
{
  "presets": [
    ["env", {
      "targets": {
        "browsers": ["last 2 versions"],
        "node": 8.0
      }
    }]
  ],
  "plugins": [
    "transform-flow-strip-types",
    ["transform-object-rest-spread", {"useBuiltIns": true}]
  ]
}
```

will use `Object.assign` instead of a polyfill

Babel Demo

- See **`babel`** script in **`package.json`**
- **`rm -rf build`**
- **`npm run babel`**
- Note files in **`build`** directory

Why Use Types?

- Can find type errors before runtime
  - more convenient than waiting until runtime
- Types document expectations about code
  - types of variables, object properties, function parameters, and function return types
  - comments can be used instead, but those
    - are more verbose
    - tend to be applied inconsistently
    - easily go out of date when code is updated
- Increases refactoring confidence
  - don't have to wonder what assumptions callers made about supported types
- Removes need to write ...
  - error checking code for type violations
  - type-related unit tests
- Editor/IDE plugins can use types to highlight issues and provide code completion

Why Avoid Types?

- Takes time to ...
  - learn type syntax
  - master applying them
- Makes code more verbose
- Can hamper prototyping and rapid development
  - developers can lose focus when distracted by having to satisfy a compiler or type checker

When to Use Types

- Use types when
  - application is large, complex, or critical
  - expected lifetime of code is long and refactoring is likely
  - code will be written and maintained by a team of developers
- Avoid types when
  - the conditions above are not present

Flow Overview

https://flow.org/

- Purpose
  - "A static type checker, designed to find type errors in JavaScript programs"
  - catches many errors without types
    - using **type inference** and **flow analysis**
    - "precisely tracks the types of variables as they flow through the program"
  - can gradually add types
- Alternatives
  - **TypeScript** - https://www.typescriptlang.org/

Flow Details

- Open source tool from Facebook
- Most ES6+ features are supported
  - for a list, see https://github.com/facebook/flow/issues/560
- Supports React and JSX
- To install, **`npm install -D`** ***`name`***
  - where ***`name`*** is **`babel-cli`**, **`babel-eslint`**, **`babel-plugin-transform-flow-strip-tpyes`**, **`eslint-plugin-flow-type`**, and **`flow-bin`**
- Editor/IDE integrations available
  - Atom, emacs, Sublime, Visual Studio Code, Vim, WebStorm
- Too much to say about this
  - see slides at https://github.com/mvolkmann/talks/blob/master/flow.key.pdf and talk video at https://www.youtube.com/watch?v=5kt3urZOg4g

To install type definitions for all dependencies in **`package.json`**,
`npm install -g flow-typed`
`flow-typed install`
`flow init` (creates **`.flowconfig`**)
edit **`.flowconfig`** and
add **`flow-typed`** after **`[libs]`**

Reasons to Prefer Flow Over TS

- Catches more errors without adding types
  - via better flow analysis
- Strict null checking is the default
  - also true for new TS projects that use "`tsc --init`"
- Uses nominal rather than structural type checking for classes
  - the right thing to do
- Just does type checking, not transpiling,
  so Babel can be used for transpiling
  - can tell TS to target ES6 and then run that output through Babel,
    but that feels awkward
- Just adds types
  - TS extends the language with features that may not be added to JavaScript

Flow Demo

- See **`flow`** script in **`package.json`**
- Modify **`src/demo.js`**
  - change type of **`getGreeting`** **`name`** parameter to **`number`**
  - change return type of **`getGreeting`** to **`number`**
  - change type of **`handleGreet`** **`messageDiv`** parameter to **`HTMLInputElement`**
- **`npm run flow`**
  or see errors provided by editor/IDE plugin

Jest Overview

https://facebook.github.io/jest/

- Purpose
  - a JavaScript test framework "built on top of Jasmine"
  - "runs your tests with a fake DOM implementation (via jsdom) so that your tests can run on the command line"
  - can watch source and test files and automatically reruns tests when they change
    - can run all tests or only those that failed in last run
- Alternatives
  - **Mocha** - https://mochajs.org/
  - **Jasmine** - https://jasmine.github.io/
  - **AVA** - https://github.com/avajs/ava
  - **Tape** - https://github.com/substack/tape

Jest Details

- To install, **`npm install -D jest`**
- To use from an npm script, add following to **`package.json`**

```
"test": "jest",
```

- Has good support for mocking and spies
  - create a mock function with **`jest.fn()`**
  - add a mock implementation with **`jest.fn.mockImplementation(someFn)`**
  - test that a mock function was called with
    `expect(mockFn).toBeCalled()` or
    `expect(mockFn).toBeCalledWith(arg1, arg2, ...)`
  - for more see **`https://facebook.github.io/jest/docs/en/mock-function-api.html`**
- Can use to test React components
  - but isn't specific to React
  - support "snapshot tests" for React components (more on next slide)
  - default test framework of apps created with create-react-app

Jest Snapshot Tests

- Snapshot tests assert that ...
  - a component will render same content as last successful test
- The first time snapshot tests are run ...
  - `toMatchSnapshot` matchers save a representation of the rendered output in a subdirectory of the test file named `__snapshots__`
- In subsequent runs ...
  - the same representation is generated again and compared to what was saved in last successful run
- When snapshot tests fail ...
  - scroll back to review differences in rendered output
  - if changes are correct, press "u" to accept them
    - overwrites previous snapshot files with new ones
  - if changes are incorrect, fix code and run tests again
- Requires react-test-renderer
  - `npm install -D react-test-renderer`

`__snapshot__` directories should be checked into version control

Jest Watch Mode

- To enable, add **`--watch`** option to **`jest`** command
- Can iteratively change code being tested and tests and have tests rerun automatically on save from any editor/IDE
- Can filter tests to run on filenames
  - to filter on file name, press "p" and enter a regex pattern
  - to filter on test name, press "t" and enter a regex pattern
  - to return to running all tests, press "a"
  - to quit watch mode, press "q"
  - to show usage help for additional options, press "w"
- Add **`.only`** and/or **`.skip`** to **`describe`** and **`test`** function names to focus testing

Enzyme Overview

http://airbnb.io/enzyme/

- Purpose
  - tests interactions with React components
    by finding elements and simulating events on them
- Alternatives
  - React **test-utils** - https://reactjs.org/docs/test-utils.html

Enzyme Details

- To install, **`npm install -D enzyme`**
- Steps
  - render a component with **`mount`**, **`render`**, or **`shallow`**
    - these return a wrapper object representing what was rendered
  - **`find`** an input element whose interaction will be tested
    - by calling **`find`** on wrapper object
    - supports a subset of CSS selectors
  - **`simulate`** an event on it
    - by calling **`simulate`** on wrapper returned by **`find`**
  - make assertions about changes that should occur
    - can use **`expect`** from Jest

`render` performs static rendering.
This generates static HTML.
Assertions can only test
what is rendered.

`shallow` performs shallow rendering.
The component and its
top-level children are rendered,
but not their descendants.
Assertions can test what
the parent renders and can
simulate events on those elements.

`mount` performs full rendering.
The top component and
all its ancestors are rendered.
Assertions can test everything
that is rendered and
simulate events on everything.

Jest Demo

- See **`test`** script in **`package.json`**
- **`npm t`**
  - runs tests in watch mode
  - initially all tests pass
- Modify **`src/demo.js`**
  - remove comma from string returned by **`getGreeting`**
  - change `===` to `!==` in **`handleNameChange`**
  - note errors when tests run automatically
  - fix errors one at a time
  - press "**`w`**" to see options
  - press "**`q`**" to quit

Istanbul Overview

https://istanbul.js.org/

- Purpose
  - collects and reports code coverage statistics
- Alternatives
  - nothing notable

Istanbul Details

- Ships with Jest
  - Jest reports on code coverage of tests using Istanbul
- If not using Jest, **`npm install -D istanbul`**
- To use from an npm script, add following to **`package.json`**

```
"cover": "CI=true npm run test-nowatch -- --coverage",
"cover-open": "open coverage/lcov-report/index.html",
```

- To exclude code from coverage statistics, use special comment
  `// istanbul ignore` ***`word`***
  where ***`word`*** is **`next`**, **`if`**, or **`else`**
  - good for code that should never be executed
    or is very difficult to execute from a test
  - is using this considered cheating?

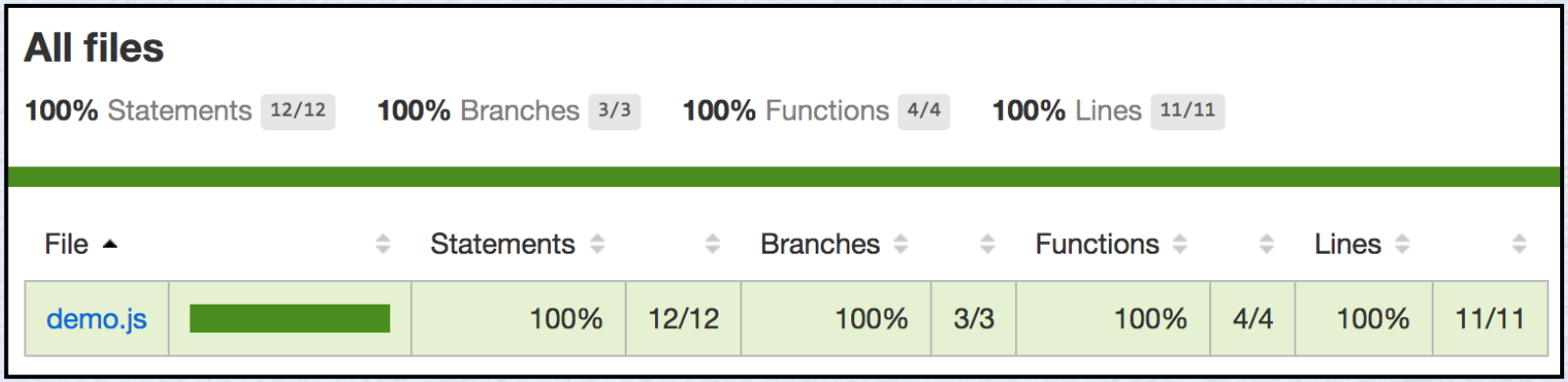
Istanbul Configuration

- Can configure to fail if coverage is below specified thresholds
- **`package.json`** changes

```
"jest": {
  "collectCoverageFrom": [
    "src/**/*.js",
    "!src/index.js"
  ],
  "coverageThreshold": {
    "global": {
      "branches": 100,
      "functions": 100,
      "lines": 100,
      "statements": 100
    }
  }
},
```

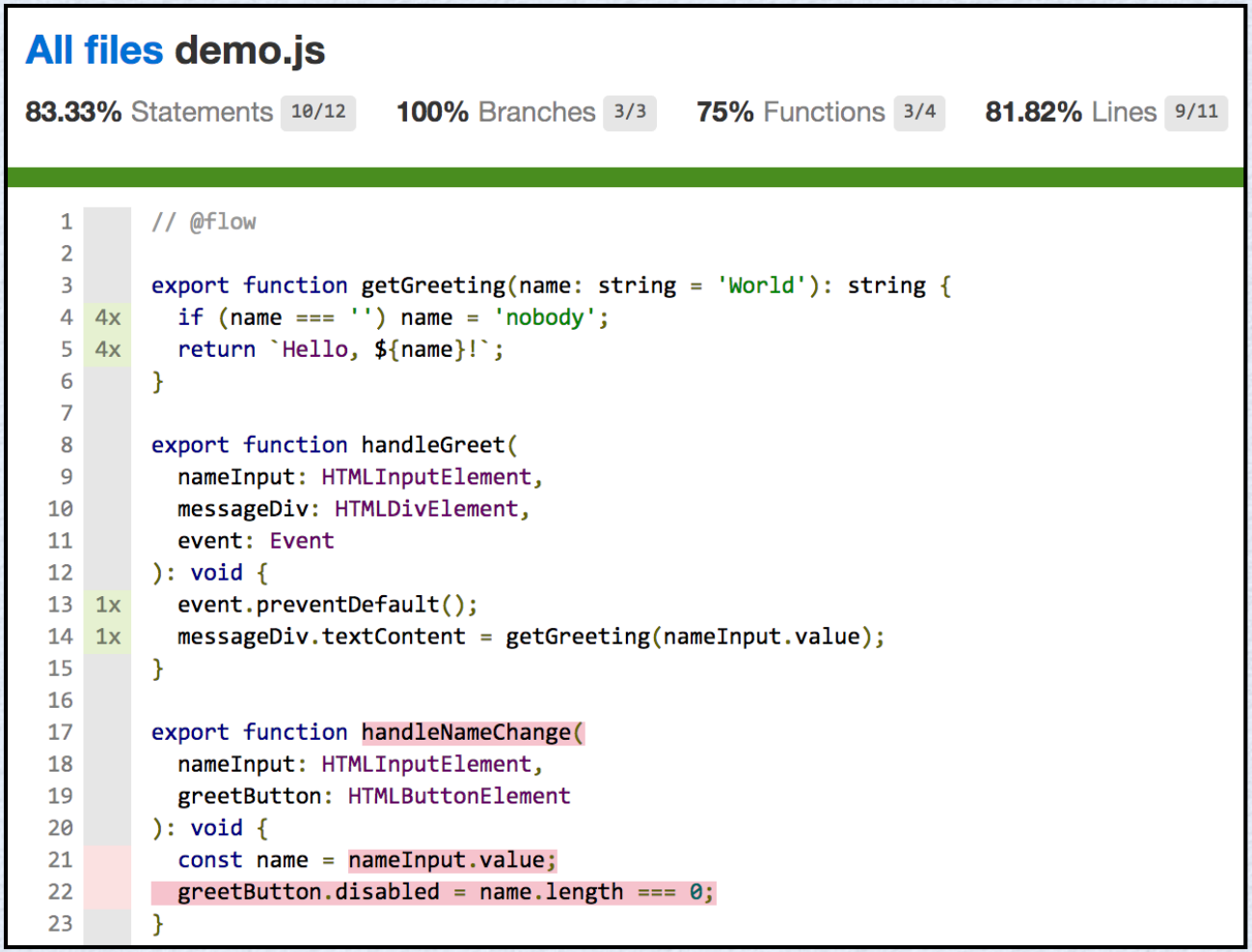
Istanbul Demo ...

- See **`cover`**, **`cover-open`**, **`verify`**, and **`build`** scripts in **`package.json`**
- **`npm run cover`**
- **`npm run cover-open`**



… Istanbul Demo

- Change **`handleNameChange test`** in **`demo.test.js`** to **`test.skip`**
- **`npm run cover`**
- Refresh browser
- Click **`demo.js`** to see detail
- Note uncovered code paths



webpack Overview

https://webpack.js.org/

- Purpose
  - bundle JavaScript files and other resources into a single JavaScript file to load into browsers faster
  - enable use of ES6 `import`/`export` syntax rather than adding a **script** tag for each `.js` file in main HTML
- Alternatives
  - **Rollup** - https://rollupjs.org/
  - **Browserify** - http://browserify.org/
  - **Parcel** - https://parceljs.org/

webpack Loaders

- Optionally uses "loaders" to convert non-JS files into JS files so they can be bundled
- Examples of loaders see https://webpack.js.org/loaders/
  - **babel-loader** - "allows transpiling JavaScript files using Babel"
  - **css-loader** - "interprets `@import` and `url()` like `import`/`require()` and will resolve them"
  - **sass-loader** - "loads a SASS/SCSS file and compiles it to CSS"
  - **style-loader** - "adds CSS to the DOM by injecting a `<style>` tag"

"**Loaders** enable webpack to process more than just JavaScript files (webpack itself only understands JavaScript). They give you the ability to leverage webpack's bundling capabilities for all kinds of files by converting them to valid modules that webpack can process."

webpack Details

- To install, **`npm install -D webpack`**
- To install loaders used here
  - **`npm install -D babel-loader css-loader style-loader`**
- To use from an npm script, add following to **`package.json`**

```
"bundle": "webpack",
```

- Has watch mode to automatically create a new bundle when files change
- Also consider webpack-dev-server

webpack Configuration

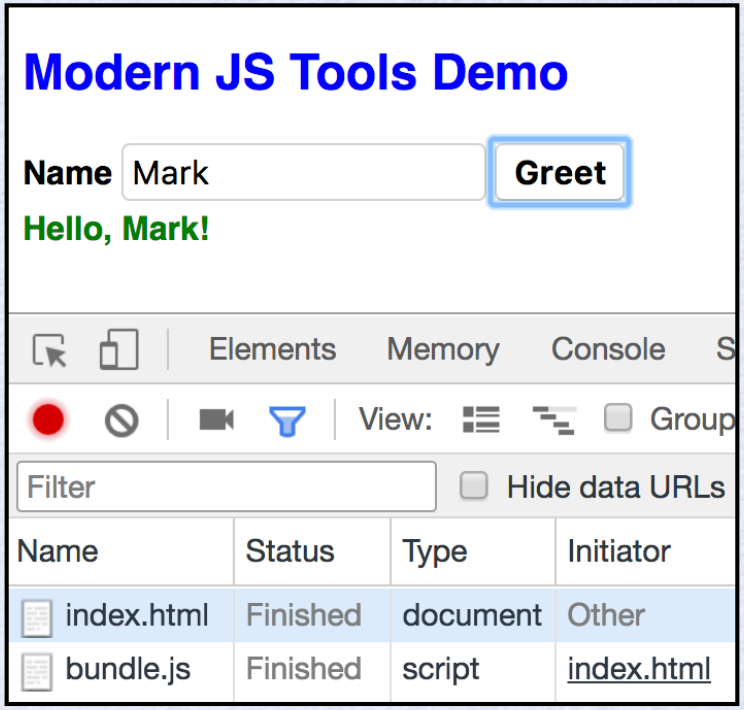
- A common complaint about webpack is that it is difficult to configure
- It can be, but here is a simple **`webpack.config.js`**

```
module.exports = {
  entry: './src/index.js',
  output: {
    path: __dirname,
    filename: 'build/bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      },
      {
        test: /\.css$/,
        exclude: /node_modules/,
        loader: 'style-loader!css-loader'
      },
    ]
  },
  watch: true
};
```

current directory

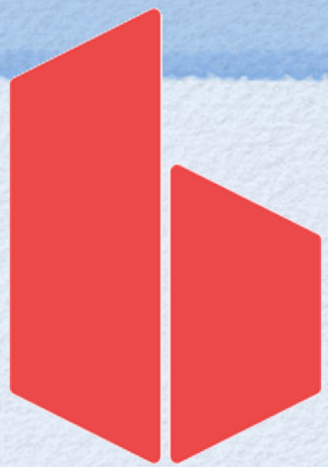
webpack Demo

- `rm -rf build`
- `npm bundle`
- Note contents of `build` directory
  - only `bundle.js`
- Open `index.html` in browser
- Note files loaded in devtools Network tab



Browsersync

https://www.browsersync.io/



- Provides live reload of browser for development testing
- To install
  - `npm install -D browser-sync`
- To start from an npm script, add following to **`package.json`**

```
"sync": "browser-sync start --server --files 'index.html, build/bundle.js'",
```

- Many more options!

Wrap Up

- Configuring tools requires a bit of work,
  but the automation they provide
  is well worth the effort
- Tools reduce time spent performing tedious tasks
  - like finding bugs, formatting code, and running tests
- Go forth and automate!