Learning Objectives

Learners will be able to...

- **Define style steps**
- **Automate styles fixes**
- **Explain types of code quality checks**

info

Make Sure You Know

Learners should be familiar with `nodejs` and its basic use.

Limitations

Assignment will require an active GitHub respository for activities.

Quality control

Quality control is an important step to ensure that code meets the standards of the organization and help keep projects running smoothly by:
- Improving code readability across teams
- Avoiding potentially unsafe operations
- Flagging outdated practices or depreciated methods

Linting tools provide a comprehensive way to evaluate code and alert developers about any potential issues before they become problems. They scan the code of your project and look for potential errors. This includes common mistakes such as forgetting to use semicolons or spacing mistakes.

Main tasks for quality checks:

Deprecated methods

Linting tools can detect usage of deprecated methods in 3rd party libraries. This helps preventing unexpected behavior changes on library versions updates.

Bad code practices

One of the most advanced feature of linting tools is their ability to check and warn about bad practices, that can cause unexpected errors ion specific scenarios. Usually those type of errors hard to catch due to their inconsistent nature.

To check out how this works, let's look at an example. We, first, need to make sure we have all of our dependencies for our project.

Click the button below to install necessary dependencies.

For example, iterating over a JavaScript `Object` with a `for` loop would generate either an error or warning in the most popular JavaScript linting tools:

```
const buz = new Object()
buf.fog = 'stack'

for (const name in buz) {
  console.log(name)
}
```

Click the button below to run the `yarn lint` command in the terminal.

In this case, most popular JavaScript linting tools would generate an error or warning when running a scan of this code. This means that if the code had already been committed to a repository, it would need to be fixed before further development could take place.

Checkpoint

Style tasks

Formatting

Consistent **formatting** improves readability of the code by any member of the team. By running linter rules on our code, we can quickly identify mistakes in formatting or style that may otherwise be hard to catch.

This helps maintain consistency across all parts of our project and keeps everyone on the same page when it comes to coding standards.

We can see where this can be beneficial, for example, when welcoming a new member of a software development team. It's helpful not to need to review the code for style errors that can be found automatically by linting tools so talented developers can focus their energy on more complex problems.

Let's take a look at some example code snippets highlighting common style errors that a linter would catch.

```
function addTwoNumbers(num1, num2) {

    return num1 + num2    //Missing semicolon

}
```

Let's run a formatting check on our project.

Click the button below to run a formatting check with the `yarn style` command.

In this example, the linter will look for the presence of a semicolon `;` at the end of each line or spacing errors. Since there is no semicolon at the end of the `return` line, it should be flagged as an error and you should add one to comply with style conventions.

Because our code has style errors, our command will return an error.

```
Checking formatting...
[warn] examples/example2.js
[warn] Code style issues found in the above file. Forgot to run Prettier?
error Command failed with exit code 1.
```

Prettier styling error message in the terminal

We can use the `yarn format` command to run an automated command defined in our `package.json` file to address any issue that our linter can address automatically.

Click the button below to run the `yarn format` command in the terminal.

If we run our `yarn style` again, we should find that our errors clear.

Checkpoint

Precommit task

We can set up our repository to perform style actions before we commit to avoid poorly styled code. Adding linting style actions to a repository before we commit our code can help us identify any potential problems before they become bigger headaches in the future.

Using **Husky**, we can set up our own script for commit that will check for style errors before each commit.

If we check our `package.json` file, we should now see a `"prepare"` task included in our `scripts` section that runs the `"husky install"` command.

Once Husky is installed, we can add a hook that will check for any style errors before we commit to our repository.

Run the commands below to add a `pre-commit` hook.

```
npx husky add .husky/pre-commit "npm style"
git add .husky/pre-commit
```

Now if we try to commit changes with bad style or failing tests the hook will fail.

Let's explore how this works.

Make a commit to your active repository.

```
git commit -m "Add styling"
```

If the code passes linting and any defined tests, the commit is successful and pushed to the repository. If a style error is found, we can run `npm style --write` to fix errors where it is possible.

This particular task will fail because we do not have any tests or test files defined.

Including pre-commit linting actions in your repository helps to ensure that high-quality software is created and that any errors are detected early in the development process. It only takes a few lines of code to get Husky up and running, but the benefits will go a long way toward increasing the quality of your software applications.

Checkpoint

Github action

Adding style checks to GitHub Actions for software development can help ensure that code follows style guidelines and reduce debugging time.

Similar to tests and compilation, we can automate style checks using GitHub Actions as well. We can combine this action with our existing `test` job by adding an additional job named `Run Lint Checks`

Let's look at an example of the code needed for implementing style checking with GitHub Actions.

```
test:
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Install dependencies
    run: yarn
  - uses: actions/download-artifact@v2
    with:
      name: assets
      path: dist
  - name: Run Lint Checks
    run: yarn style
```

If any style violations are found in the codebase during these style check runs, they will be displayed as errors and warnings.

Checkpoint