

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение*  *высшего образования*  ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных

технологий (МОСИТ)

ПРАКТИЧЕСКАЯ РАБОТА №5

по дисциплине

**«Тестирование и верификация программного обеспечения»**

Тема: **«Статический и динамический анализ исходного кода»**

**Студенты группы** ИКБО-20-21

Хитров Н.С.

Емельянов И.В.

Квашнин Ю.В.

Мухаметшин А.Р.

(подпись студента)

**Принял руководитель работы** Овчинникова М.А.

(подпись руководителя)

Практические работы выполнены « » 2023 г.

Зачтено « » 2023 г.

**ОГЛАВЛЕНИЕ**

[1. Цель работы 3](#_Toc150202935)

[2. Ход выполнения работы 3](#_Toc150202936)

[2.1. Установка выбранных анализаторов 3](#_Toc150202937)

[2.2. Статический анализ выбранного приложения 4](#_Toc150202938)

[2.3. Динамический анализ выбранного приложения 6](#_Toc150202939)

[3 Выводы 7](#_Toc150202940)

[4. Список использованной литературы 8](#_Toc150202941)

1. Цель работы

Получить практические навыки по работе с инструментами для статического и динамического анализа кода.

2. Ход выполнения работы

Состав команды, выполнившей работу: Емельянов И. В., Хитров Н.С., Квашнин Ю. В., Мухаметшин А. Р.

Для выполнения работы было решено использовать следующие анализаторы:

* Flake8 - Инструмент для анализа Python-кода, который проверяет стиль кода, наличие потенциальных ошибок и соответствие PEP 8 (стандарту кодирования Python);
* mypy, определяющий правильность указанной и фактической типизации в Python-коде;
* ЕSLint – Инструмент для статического анализа JavaScript-кода, который проверяет стиль кода, наличие потенциальных ошибок и соответствие кода с конвенциями;
* JSHint - Другой инструмент для анализа JavaScript, который проверяет стиль кода, выявляет потенциальные ошибки и предостерегает от использования устаревших конструкций.

2.1. Установка и настройка выбранных анализаторов

Данные инструменты было решено установить в систему, чтобы их можно было использовать вне IDE. Этот процесс показан на Рисунках 1-3.

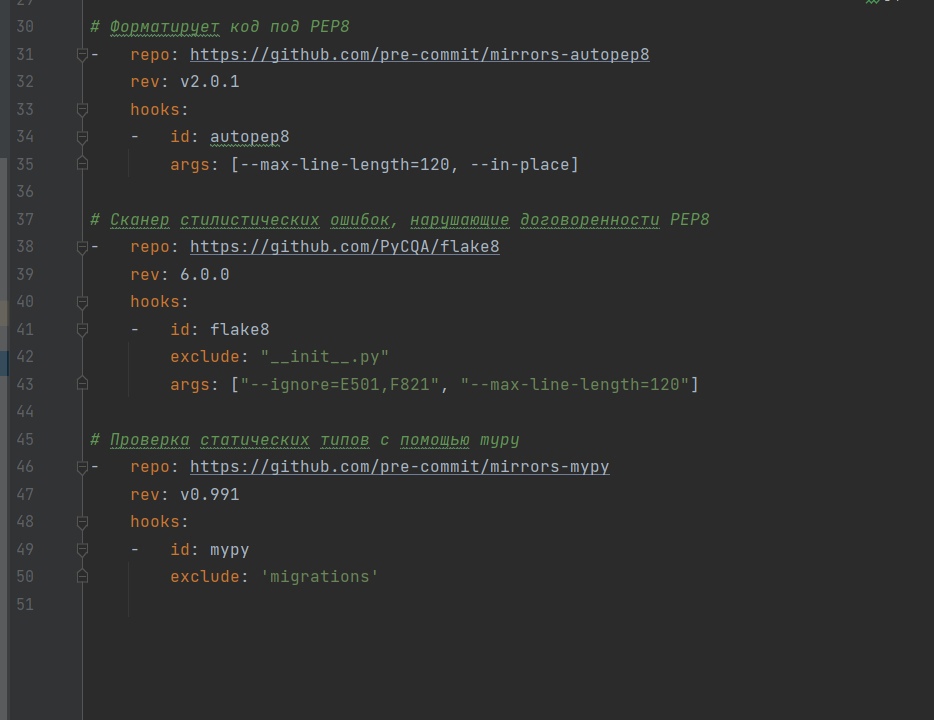


Рисунок 1 – Установка инструментов

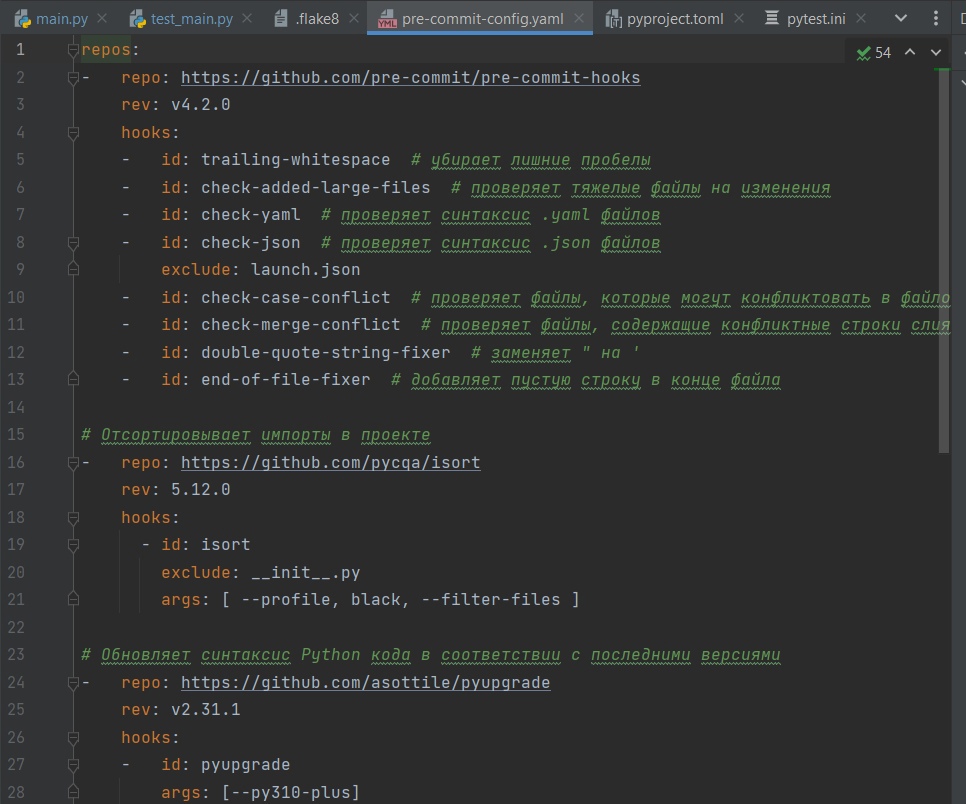


Рисунок 2 – Настройка

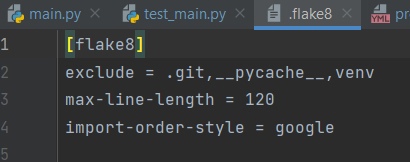


Рисунок 3 – конфигурация flake8

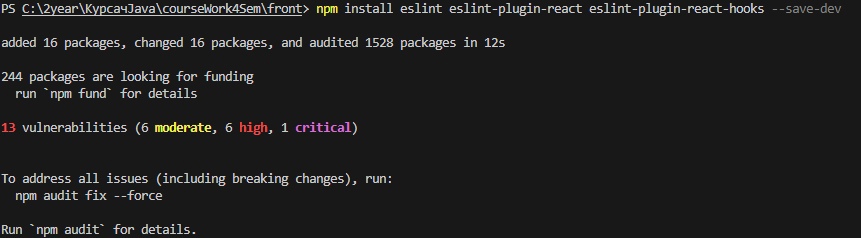


Рисунок 4 – установка ESLint

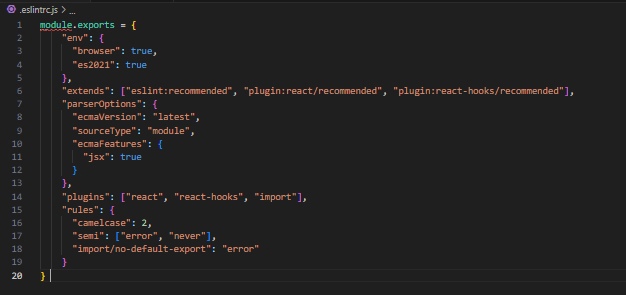


Рисунок 5 – конфигурация eslintrc.js

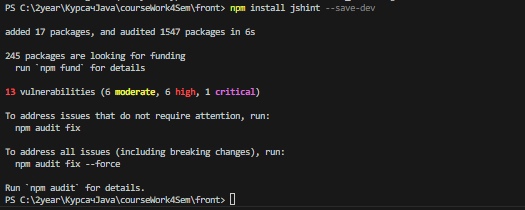


Рисунок 6 – установка jshint



Рисунок 7 – конфигурация jshintrc.js

2.2. Статический анализ выбранного приложения

2.2.1 Анализ веб-приложения ресторана

Для анализа было выбрано веб-приложение ресторана.

Проверим приложение. Результаты проверки приведены на Рисунках 8-10.



Рисунок 8 – Автоисправление ошибок

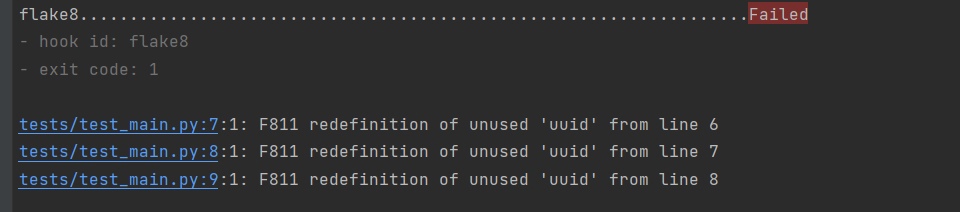


Рисунок 9 – Проверка файлов

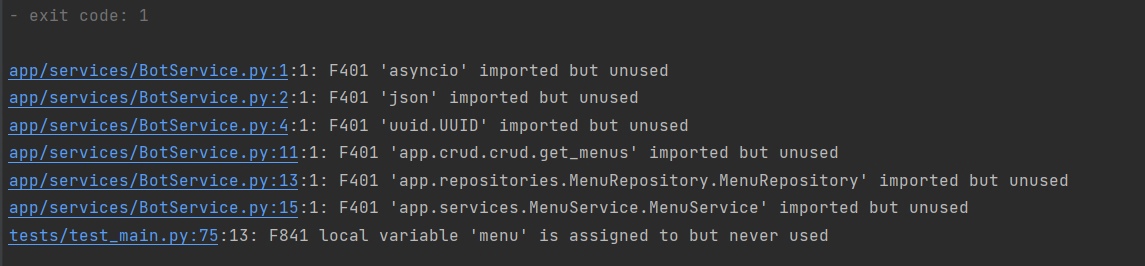


Рисунок 10 - упорядочивание импортов

Согласно заданию, нужно сделать вывод об адекватности найденных ошибок. Инструменты тестирования указали на неиспользованные импорты и переменные.

**2.2.2 Анализ клиентской части сервиса подбора путешествий**

Результаты проверки приведены на рисунках 11 - 13.

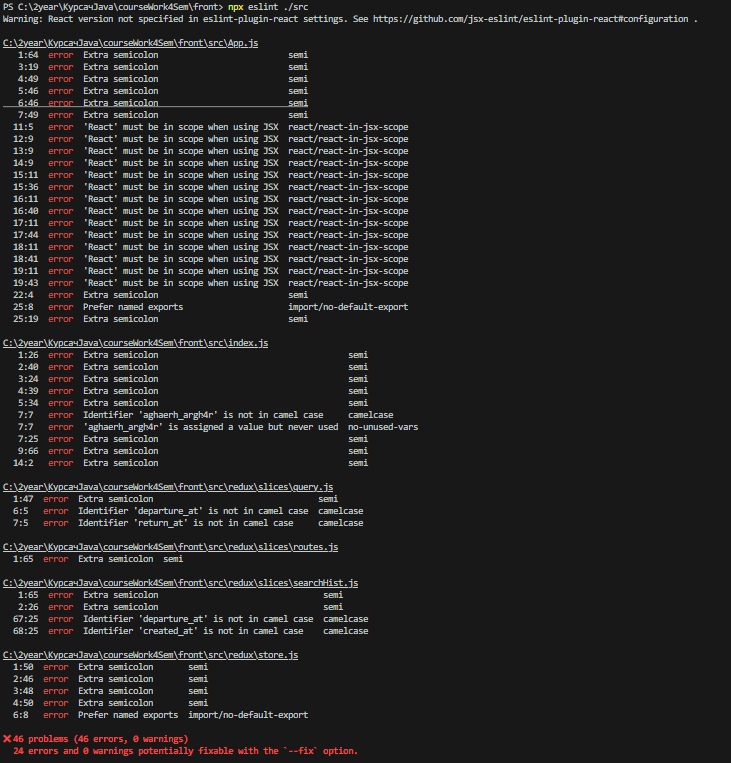


Рисунок 11 - проверка на ошибки согласно базовым встроенным и импортированным правилам

https://sun1-16.userapi.com/impg/U-glH1FHE86jPXvtF7OaFRfyRO8JuyTiaTEoWg/J0xs0oxh4Nk.jpg?size=713x44&quality=96&sign=e2e60c7425ef52b03b4a30768a65f8cc&type=album

Рисунок 12 - результат исправления ошибок

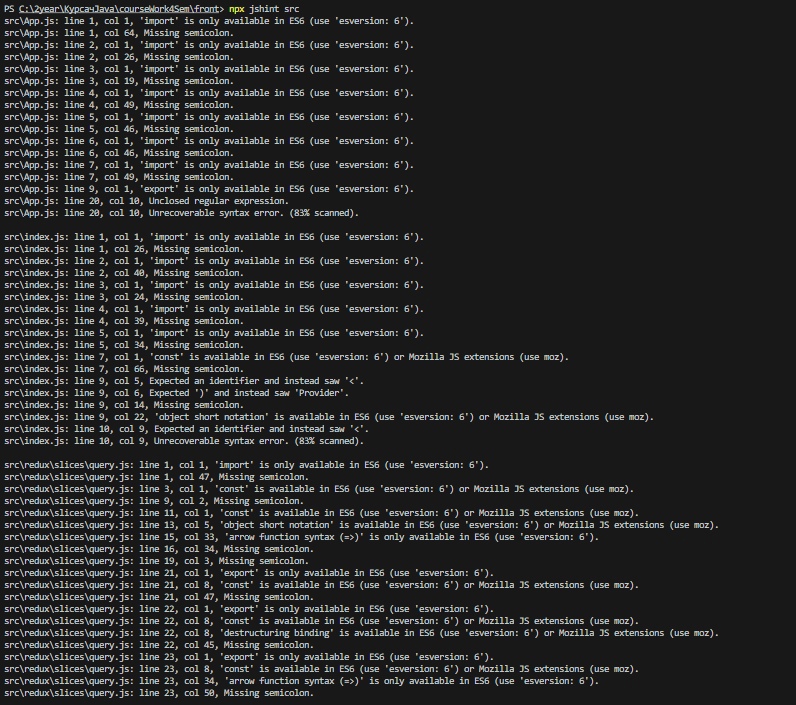


Рисунок 13 – проверка на ошибки с помощью jshint

По результату выполненных проверок с помощью инструментов ESLint и JSHint можно сделать вывод о том, что ошибки соответствуют ожидаемым результатам.

На основании полученных данных можно сделать вывод, что статический анализ безусловно полезен, однако при его использовании и анализе найденных ошибок программист должен также ориентироваться на контекст происходящего.

Самая часто встречающаяся ошибка при работе с Python – несоответствие ожидаемого типа данных, на которые указывает переменная и фактического. Иногда такие ошибки требуют много времени на обнаружение, что существенно удлиняет время работы над проектом. К счастью, писать свой собственный анализатор нет нужды, поскольку есть Mypy, хорошо справляющийся с этой задачей.

2.3. Динамический анализ выбранного приложения

Для динамического анализа воспользуемся плагином SonarLint для IDE PyCharm. Для начала просто протестируем текущий программный код на наличие каких-либо ошибок. Результаты проверки приведены на Рисунке 14.

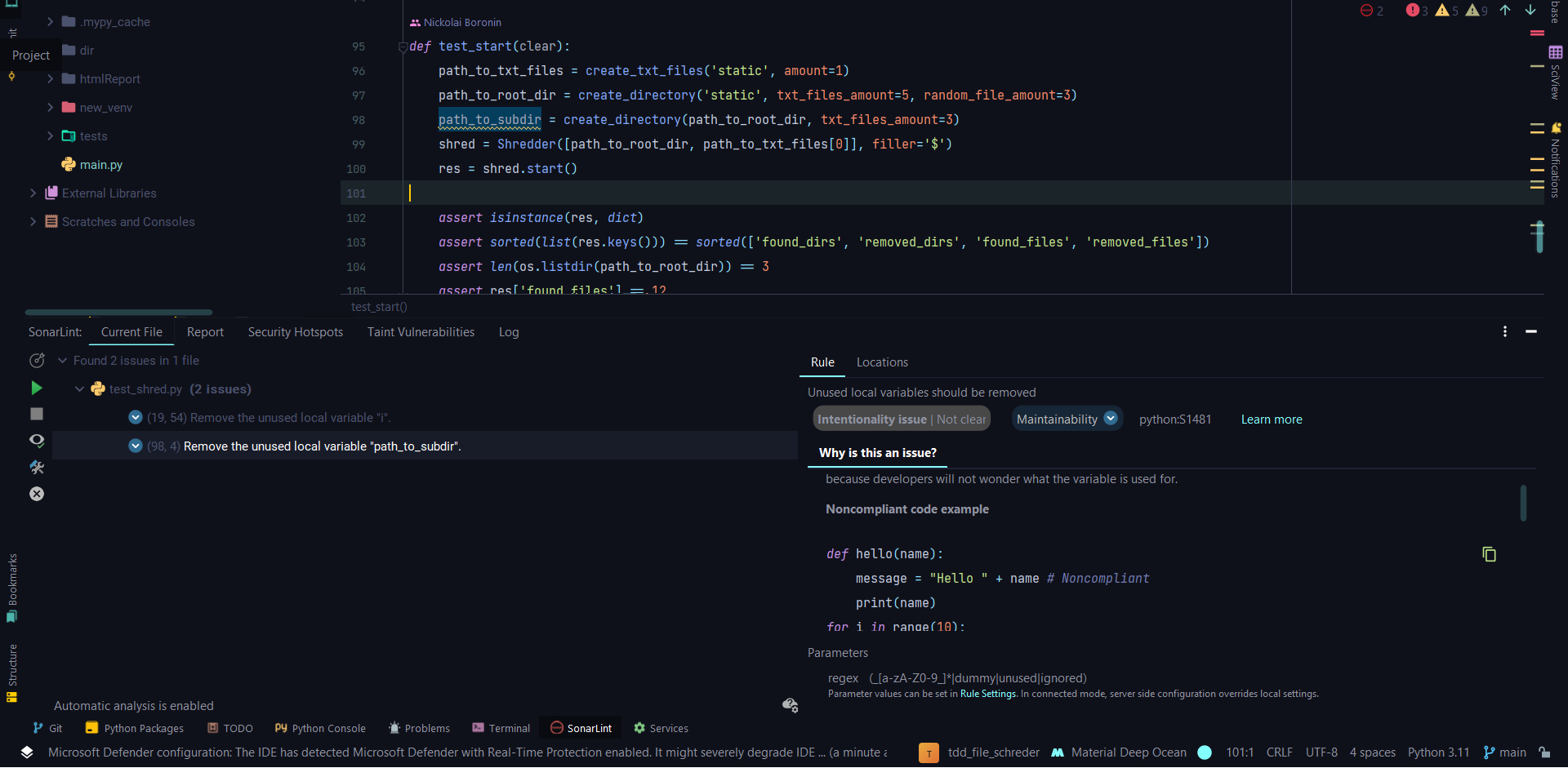


Рисунок 14 – Результат работы SonarLint

Заметим, что анализатор обратил внимание на две неиспользуемые переменные. Результаты исправления представлены на Рисунках 15 и 16.

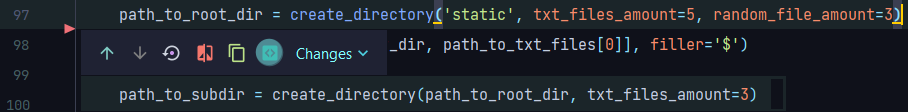


Рисунок 15 – Удаление неиспользуемой переменной path\_to\_root\_dir

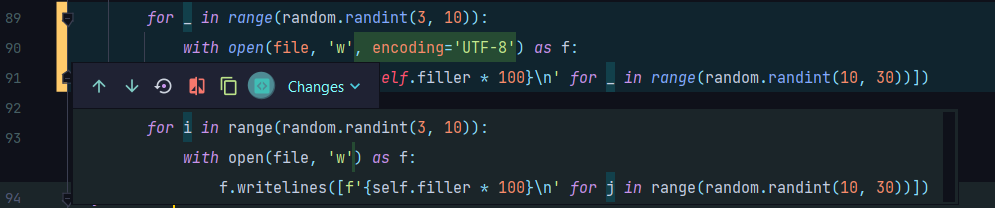


Рисунок 16 – Замена неиспользуемых переменных специальной переменной \_

Начнем писать код, чтобы проверить, сможет ли динамический анализатор в режиме реального времени найти критическую ошибку. Результат представлен на Рисунке 17.

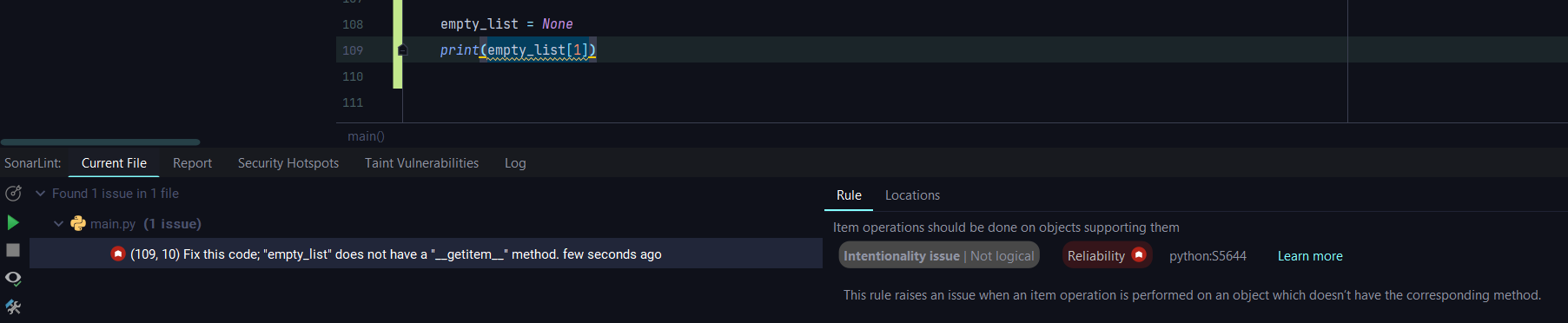


Рисунок 17 – Результат динамического анализа кода

Динамический анализатор работает адекватно и «на лету» находит допускаемые разработчиком ошибки.

3 Выводы

В ходе выполнения данной практической работы были получены практические навыки по работе со статическими и динамическими анализаторами: поиск, идентификация и исправление логических ошибок и редактирование стилистических недочетов так, чтобы код соответствовал принятым соглашениям.

4. Список использованной литературы

* 1. Лекции Петренко А.А. по дисциплине «Тестирование и верификация программного обеспечения». МИРЭА, 2023 (Дата обращения 03.11.2023);
  2. Python's documentation, tutorials, and guides are constantly evolving [Электронный ресурс]. – URL: <https://www.python.org/doc/> (Дата обращения 03.11.2023);
  3. Mypy Documentation [Электронный ресурс]. – URL: <https://mypy.readthedocs.io/en/stable/index.html> (Дата обращения: 03.11.2023);
  4. SonarLint Documentation [Электронный ресурс]. – URL: <https://docs.sonarsource.com/sonarlint/intellij> (Дата обращения: 03.11.2023).