

PYNOSE: A Test Smell Detector For Python

<https://arxiv.org/pdf/2108.04639.pdf>

Tongjie Wang - University of California, Irvine

Yaroslav Golubev - JetBrains Research

Oleg Smirnov - JetBrains Research

Jiawei Li - University of California, Irvine

Timofey Bryksin - JetBrains Research

Iftekhhar Ahmed - University of California, Irvine

What are code smells?

- Smells are certain structures in the **code** that indicate **violation** of fundamental design principles.
- **Subjective**, and varies by language, developer, and development methodology.
- They are certainly **not bugs**, but increase the risk of bugs or failures in the future.

Unit testing

- The goal is to isolate each part of the program and show that the individual parts are **correct**.
- It provides a strict, **written contract** that the piece of code must satisfy mostly using **validations**.
- **Safety net** to refactor code with confidence.
- Tests are your **documentation** which talks about the importance in maintainability of software.

Statically typed languages

- It is very popular among development of **backend systems**.
- Main reason being **compile time** safety leading to less bugs and easier maintenance.
- Expressing **business domains using types** and building business logic around them.
- Majority of **test smells research** has been focused around these languages.

Python

- **Dynamically** typed language.
- Hugely popular and adapted in **data science and machine learning** projects.
- This paper is mostly focused on studying test smells in **Python projects**.

Goal of the paper

- Identifying **test smells** from well established open source **python projects**.
- Providing a tool (**PYNOSE**) for the test smell detection during development stages and also **offline** analysis.
- Also an **empirical** study of test smell **pervasiveness** by applying the PYNOSE tool on matured python projects.

Systematic mapping study

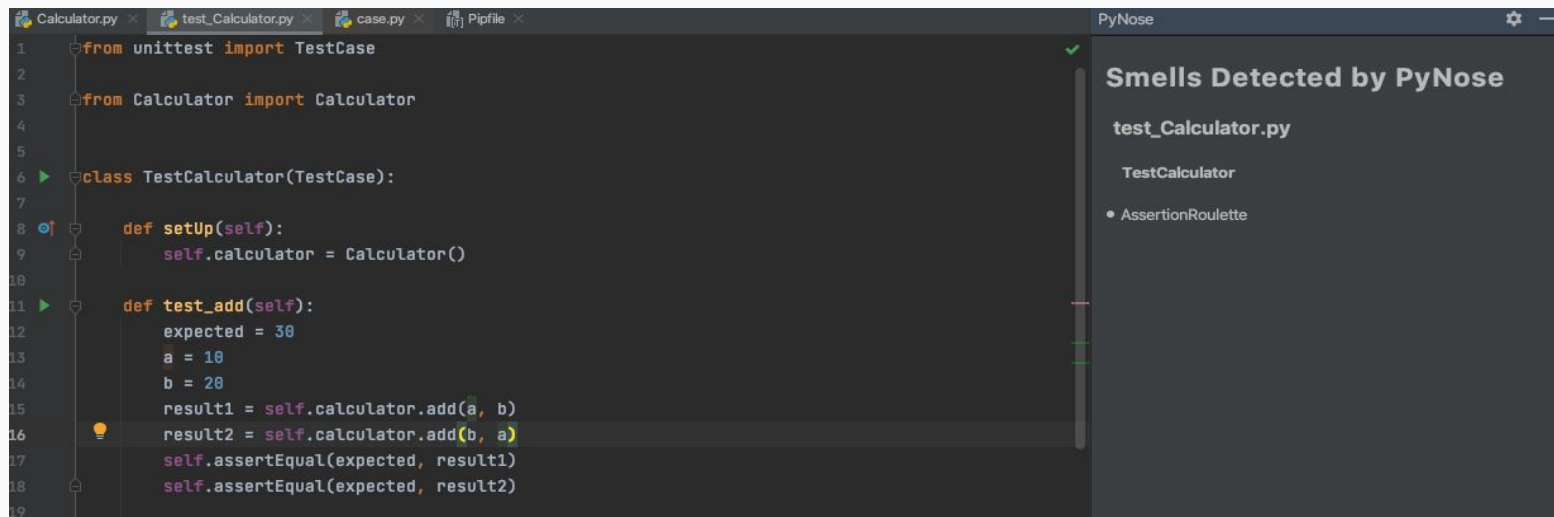
- Keyword search of "**test smell**" or "**test smells**" in digital libraries like ACM, IEEE and Scopus.
- Restrict to **digital** publications about **software** engineering methodologies, test **smell detection** and **refactoring**.
- There could be potential articles left out due to the above criteria and hence single iteration of **backward snowballing** is conducted on **left out papers**.
- Reliability of selected publications was **manually evaluated**.

Identifying test smells

- Focus was only on **Unit**test testing framework that is included in the Python Standard Library.
- Some of them are **only valid** for statically typed languages like **Java** or **Scala**.
- After the above process **17 smells** were **shortlisted**.
- Shortlisted test smells are **implemented** in Python.

Assertion Roulette

A test case that contains more than one assertion statement without an explanation/message.



```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         expected = 30
13         a = 10
14         b = 20
15         result1 = self.calculator.add(a, b)
16         result2 = self.calculator.add(b, a)
17         self.assertEqual(expected, result1)
18         self.assertEqual(expected, result2)
```

PyNose

Smells Detected by PyNose

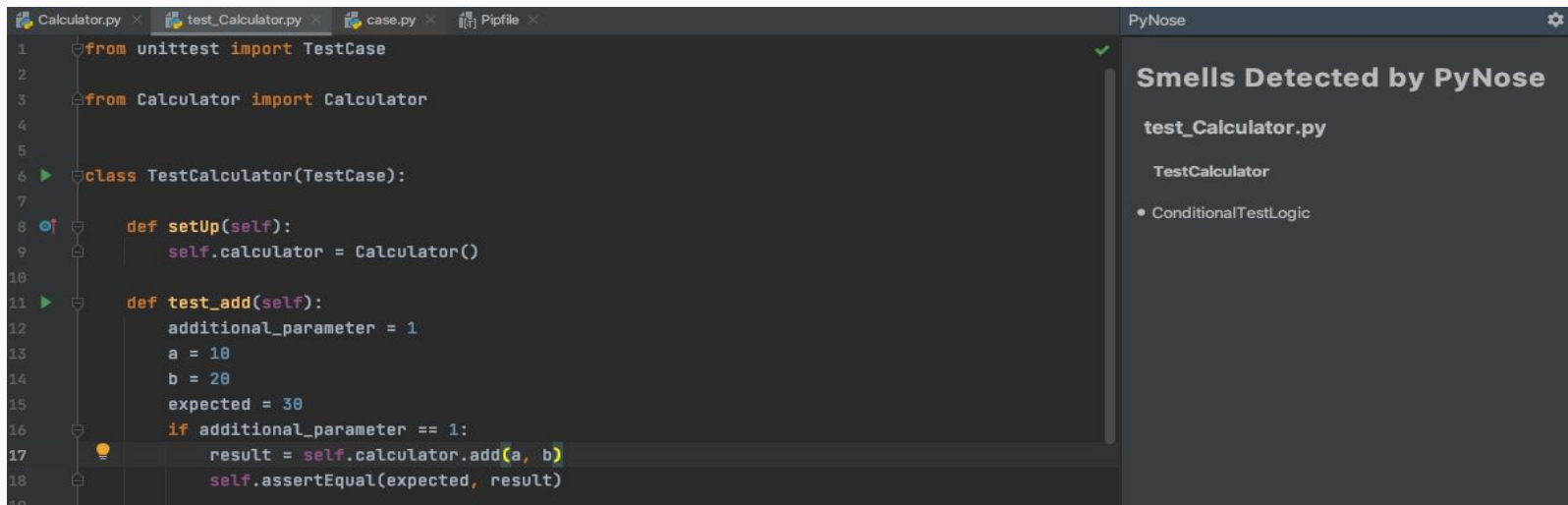
test_Calculator.py

TestCalculator

- AssertionRoulette

Conditional Test Logic

A test case that contains one or more control statements (i.e., if, for, while)



The screenshot displays a code editor with four tabs: Calculator.py, test_Calculator.py, case.py, and Pipfile. The active tab is test_Calculator.py, which contains the following Python code:

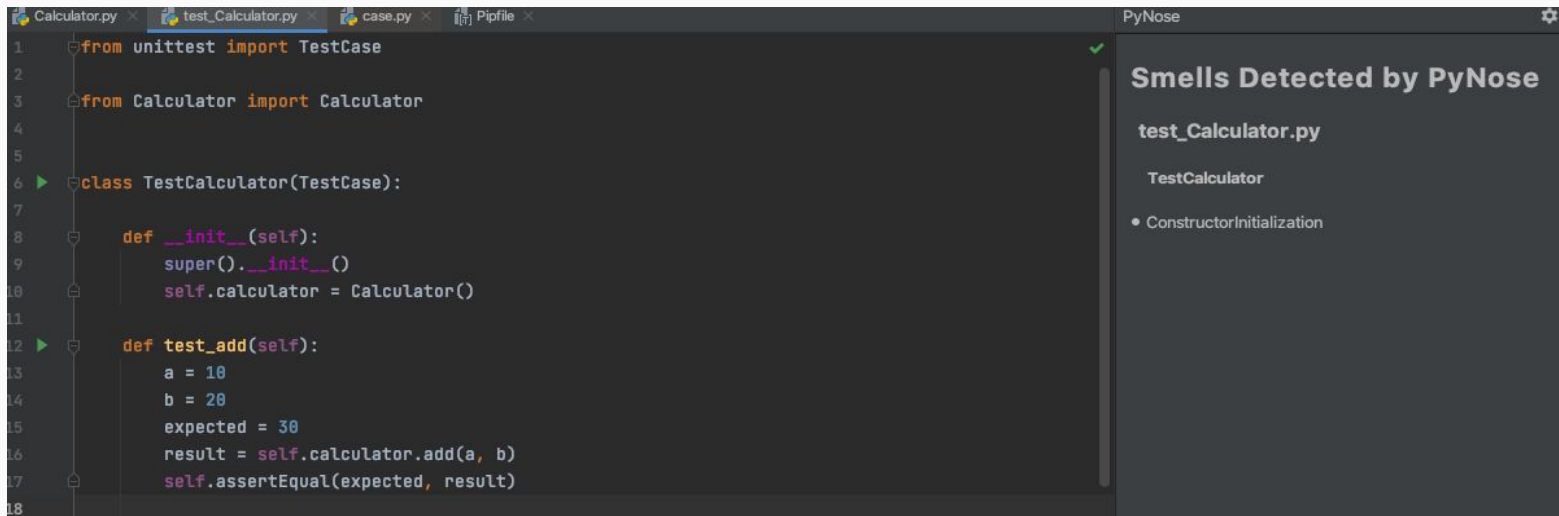
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         additional_parameter = 1
13         a = 10
14         b = 20
15         expected = 30
16         if additional_parameter == 1:
17             result = self.calculator.add(a, b)
18             self.assertEqual(expected, result)
```

On the right side of the editor, the PyNose panel is visible. It has a title bar "PyNose" with a settings icon. The main content area is titled "Smells Detected by PyNose" and lists the following:

- test_Calculator.py
 - TestCalculator
 - ConditionalTestLogic

Constructor Initialization

A test case that contains more than one assertion statement without an explanation/message.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

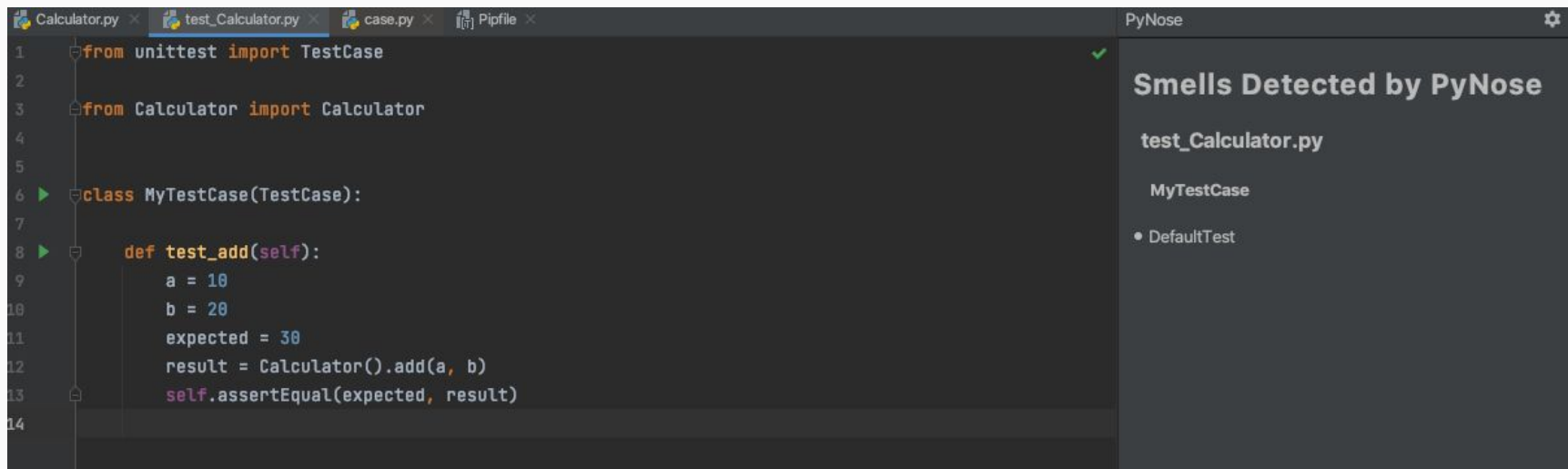
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def __init__(self):
9         super().__init__()
10        self.calculator = Calculator()
11
12    def test_add(self):
13        a = 10
14        b = 20
15        expected = 30
16        result = self.calculator.add(a, b)
17        self.assertEqual(expected, result)
18
```

On the right, the PyNose panel displays the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- TestCalculator**
- ConstructorInitialization

Default Test

A test suite is called MyTestCase which is a default name provided by IDE.



The screenshot shows an IDE with four tabs: Calculator.py, test_Calculator.py (active), case.py, and Pipfile. The active tab contains the following Python code:

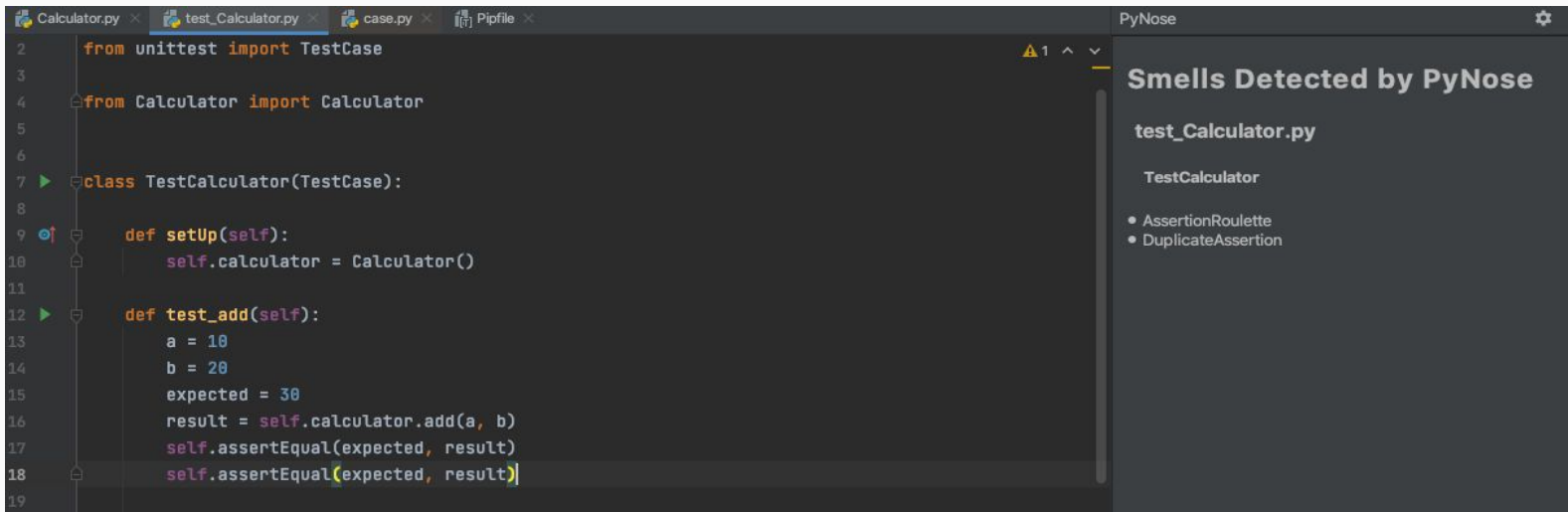
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class MyTestCase(TestCase):
7
8     def test_add(self):
9         a = 10
10        b = 20
11        expected = 30
12        result = Calculator().add(a, b)
13        self.assertEqual(expected, result)
14
```

On the right side, the PyNose panel displays the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- MyTestCase**
- DefaultTest

Duplicate Assert

A test case that contains more than one assertion statement with the same parameters.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

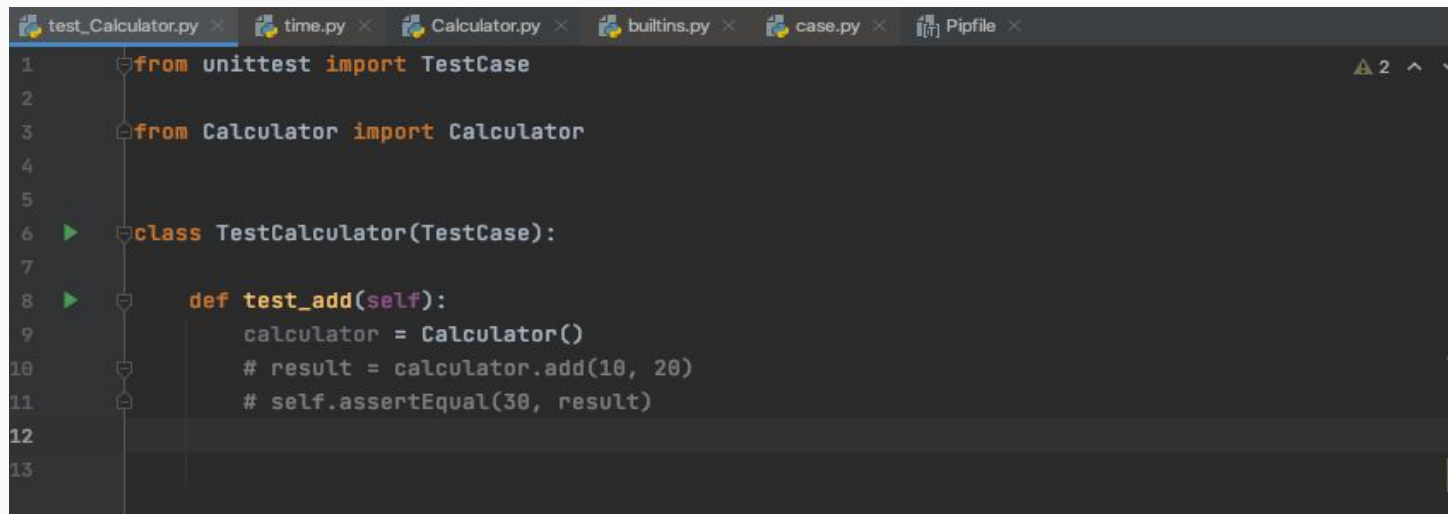
```
2 from unittest import TestCase
3
4 from Calculator import Calculator
5
6
7 class TestCalculator(TestCase):
8
9     def setUp(self):
10         self.calculator = Calculator()
11
12     def test_add(self):
13         a = 10
14         b = 20
15         expected = 30
16         result = self.calculator.add(a, b)
17         self.assertEqual(expected, result)
18         self.assertEqual(expected, result)
```

On the right, the PyNose panel displays the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- TestCalculator**
- AssertionRoulette
 - DuplicateAssertion

Empty Test

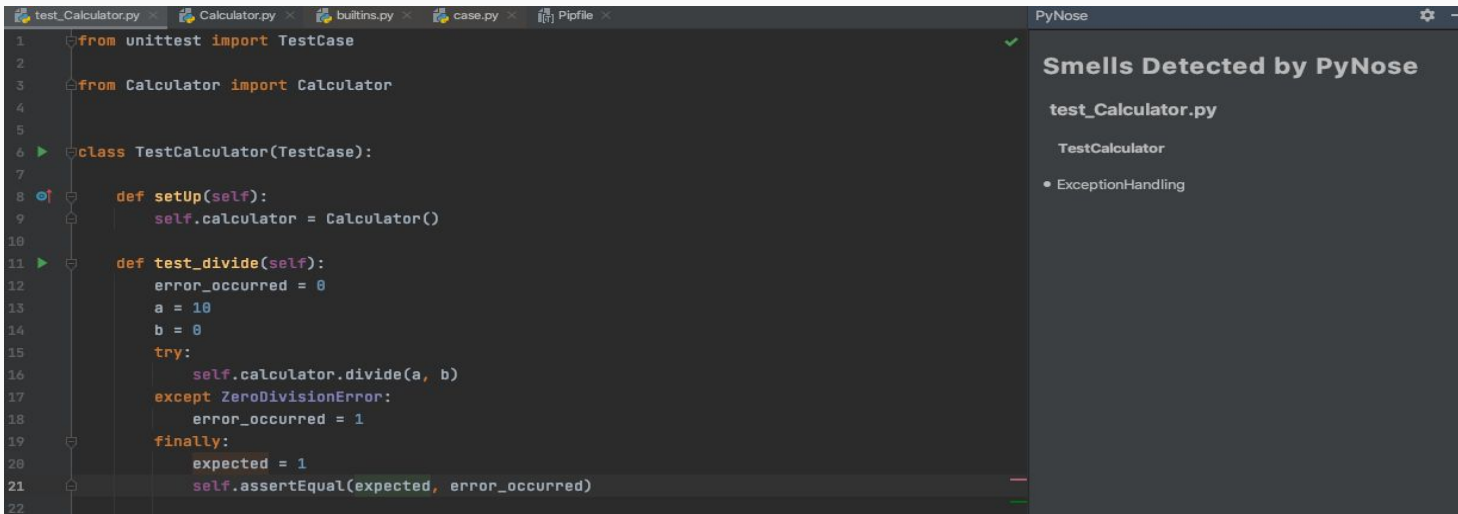
A test case where everything is commented out.



```
test_Calculator.py x time.py x Calculator.py x builtins.py x case.py x Pipfile x
1  from unittest import TestCase
2
3  from Calculator import Calculator
4
5
6  class TestCalculator(TestCase):
7
8      def test_add(self):
9          calculator = Calculator()
10         # result = calculator.add(10, 20)
11         # self.assertEqual(30, result)
12
13
```

Exception Handling

A test case that contains either the try/except statement or the raise statement.



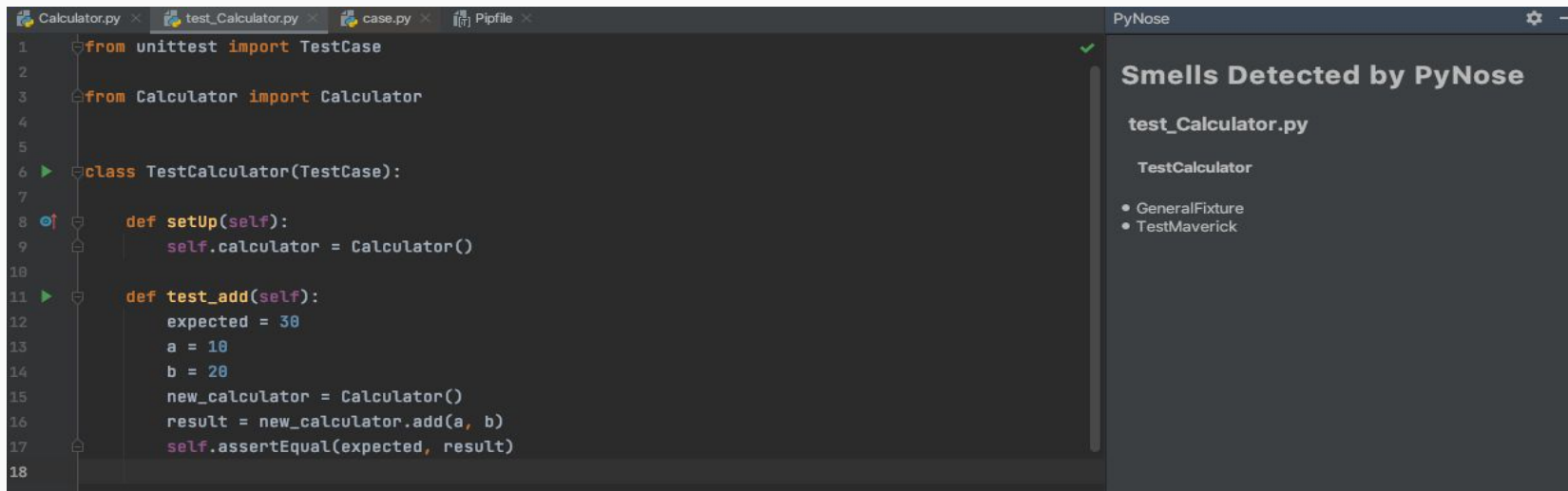
The screenshot shows a code editor with a Python test case and a PyNose sidebar. The code defines a `TestCalculator` class that inherits from `unittest.TestCase`. It includes a `setUp` method to initialize a `Calculator` instance and a `test_divide` method that tests division by zero. The `test_divide` method uses a try/except block to catch `ZeroDivisionError` and a `finally` block to assert the error occurred.

```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_divide(self):
12         error_occurred = 0
13         a = 10
14         b = 0
15         try:
16             self.calculator.divide(a, b)
17         except ZeroDivisionError:
18             error_occurred = 1
19         finally:
20             expected = 1
21         self.assertEqual(expected, error_occurred)
22
```

The PyNose sidebar on the right, titled "Smells Detected by PyNose", shows the file `test_Calculator.py` and the class `TestCalculator`. It lists a single smell: `ExceptionHandling`.

General Fixture

Not all fields instantiated within the setUp() method of a test suite are utilized by all test cases in this test suite.



The screenshot displays a code editor with a Python test suite and a PyNose extension window. The code editor shows the following code:

```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         expected = 30
13         a = 10
14         b = 20
15         new_calculator = Calculator()
16         result = new_calculator.add(a, b)
17         self.assertEqual(expected, result)
18
```

The PyNose extension window, titled "PyNose", displays the following information:

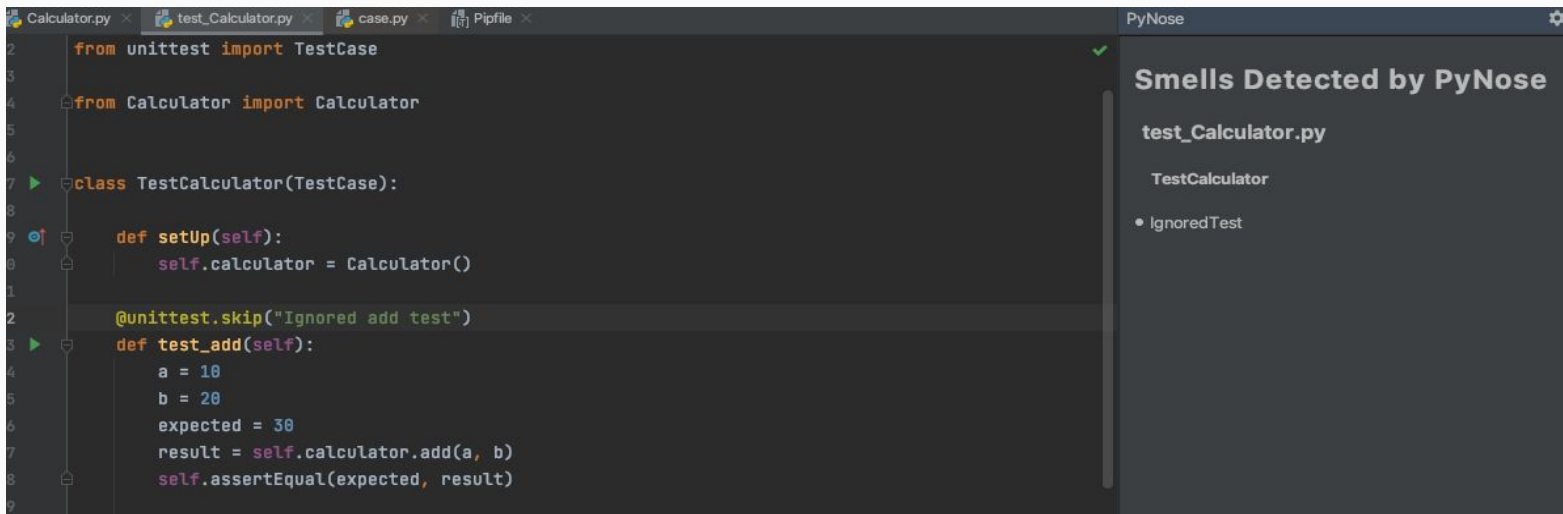
Smells Detected by PyNose

test_Calculator.py

- TestCalculator
 - GeneralFixture
 - TestMaverick

Ignored Test

A test case that contains the `@unittest.skip` decorator.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

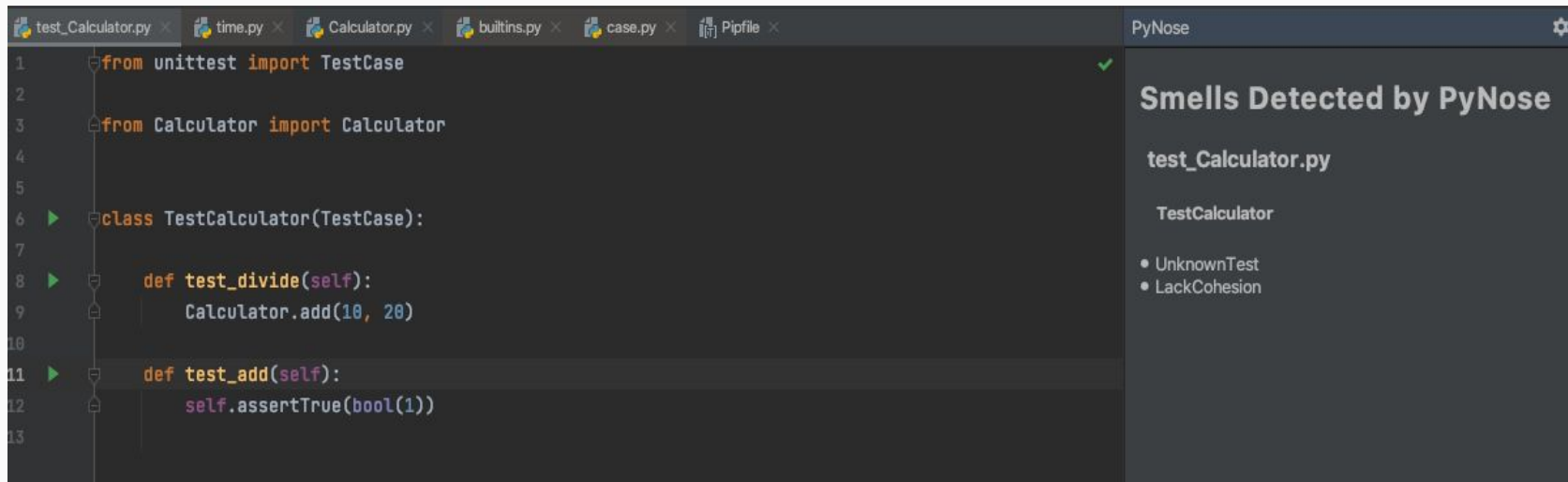
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     @unittest.skip("Ignored add test")
12     def test_add(self):
13         a = 10
14         b = 20
15         expected = 30
16         result = self.calculator.add(a, b)
17         self.assertEqual(expected, result)
```

On the right, the PyNose sidebar displays the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- TestCalculator**
- IgnoredTest

Lack of Cohesion of Test Cases

Test cases that are grouped together in one test suite but are not cohesive.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

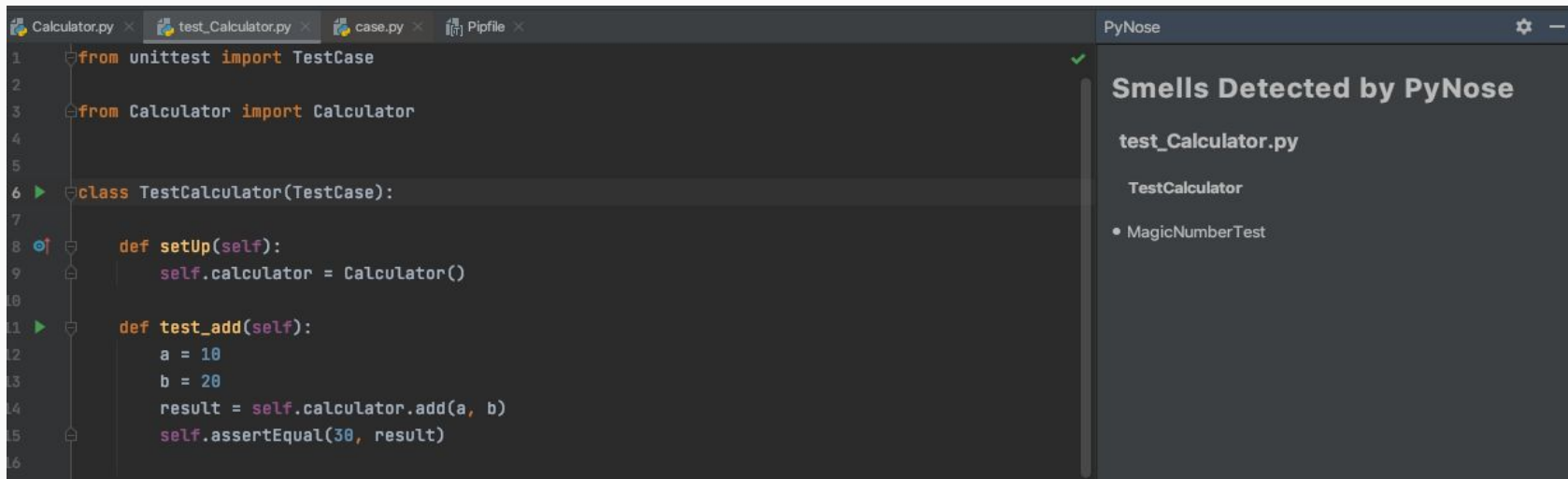
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def test_divide(self):
9         Calculator.add(10, 20)
10
11    def test_add(self):
12        self.assertTrue(bool(1))
13
```

The PyNose sidebar on the right, titled "Smells Detected by PyNose", shows the following information:

- test_Calculator.py
- TestCalculator
- UnknownTest
- LackCohesion

Magic Number Test

A test case that contains an assertion statement that contains a numeric literal as an argument.



The screenshot shows a code editor with a dark theme. The active file is `test_Calculator.py`. The code defines a `TestCalculator` class that inherits from `unittest.TestCase`. It includes a `setUp` method that initializes a `Calculator` instance and a `test_add` method that performs an addition and asserts the result is 30. The PyNose sidebar on the right, titled "Smells Detected by PyNose", lists the files `test_Calculator.py` and `TestCalculator`, and identifies a "MagicNumberTest" smell in the `test_add` method.

```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         a = 10
13         b = 20
14         result = self.calculator.add(a, b)
15         self.assertEqual(30, result)
16
```

PyNose

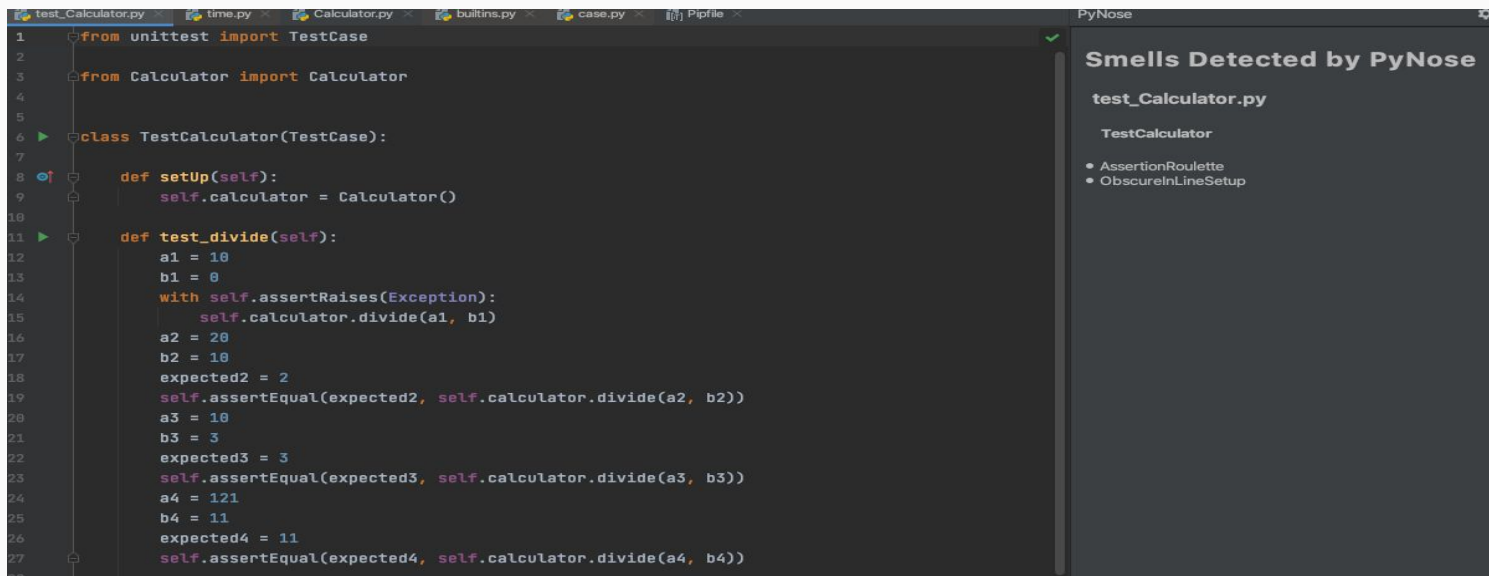
Smells Detected by PyNose

test_Calculator.py

- TestCalculator
 - MagicNumberTest

Obscure In-Line Setup

A test case that contains ten or more local variables declarations.



The screenshot displays a code editor with a Python test file named `test_Calculator.py`. The code defines a `TestCalculator` class that inherits from `unittest.TestCase`. The `setUp` method initializes a `calculator` attribute. The `test_divide` method contains ten local variable declarations (`a1`, `b1`, `a2`, `b2`, `expected2`, `a3`, `b3`, `expected3`, `a4`, `b4`) and uses `self.assertEqual` and `self.assertRaises` to test the `calculator` object's `divide` method.

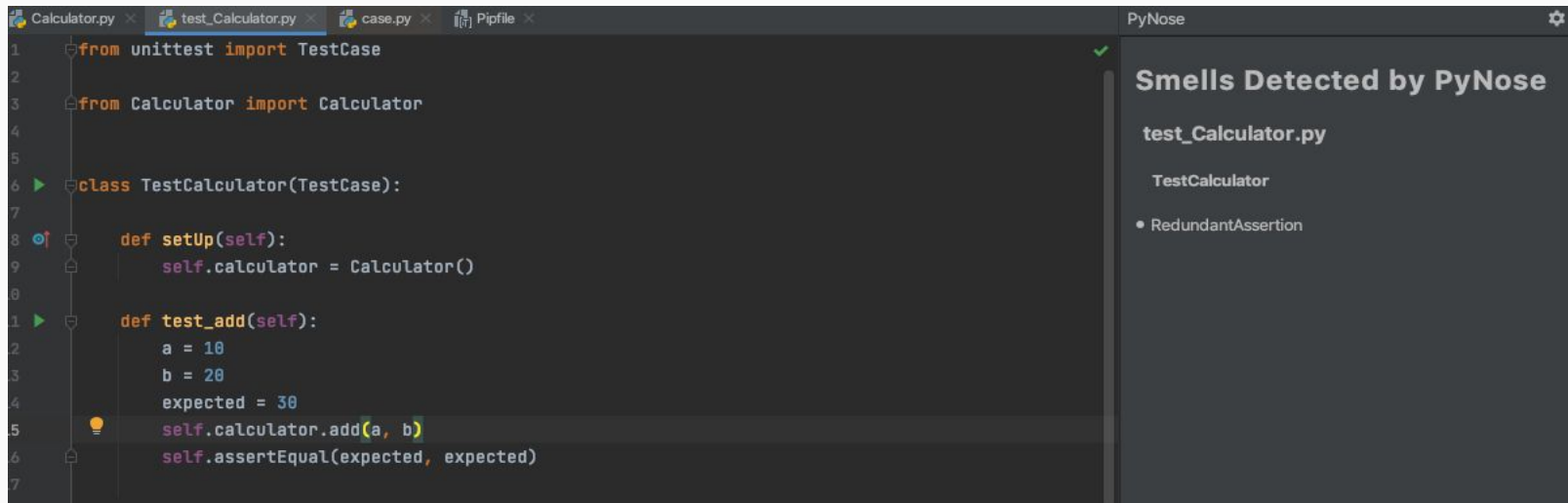
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_divide(self):
12         a1 = 10
13         b1 = 0
14         with self.assertRaises(Exception):
15             self.calculator.divide(a1, b1)
16         a2 = 20
17         b2 = 10
18         expected2 = 2
19         self.assertEqual(expected2, self.calculator.divide(a2, b2))
20         a3 = 10
21         b3 = 3
22         expected3 = 3
23         self.assertEqual(expected3, self.calculator.divide(a3, b3))
24         a4 = 121
25         b4 = 11
26         expected4 = 11
27         self.assertEqual(expected4, self.calculator.divide(a4, b4))
```

On the right side, the PyNose panel shows the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- TestCalculator**
- AssertionRoulette
 - ObscureInLineSetup

Redundant Assertion

A test case that contains an assertion statement in which the expected and actual parameters of equality are the same.



The screenshot displays a code editor with a Python test file named `test_Calculator.py`. The code defines a `TestCalculator` class that inherits from `unittest.TestCase`. It includes a `setUp` method to initialize a `Calculator` instance and a `test_add` method. In the `test_add` method, the values `a = 10` and `b = 20` are added to get an `expected = 30` result. The assertion `self.assertEqual(expected, expected)` is highlighted with a yellow lightbulb icon, indicating a redundant assertion where the expected and actual values are identical.

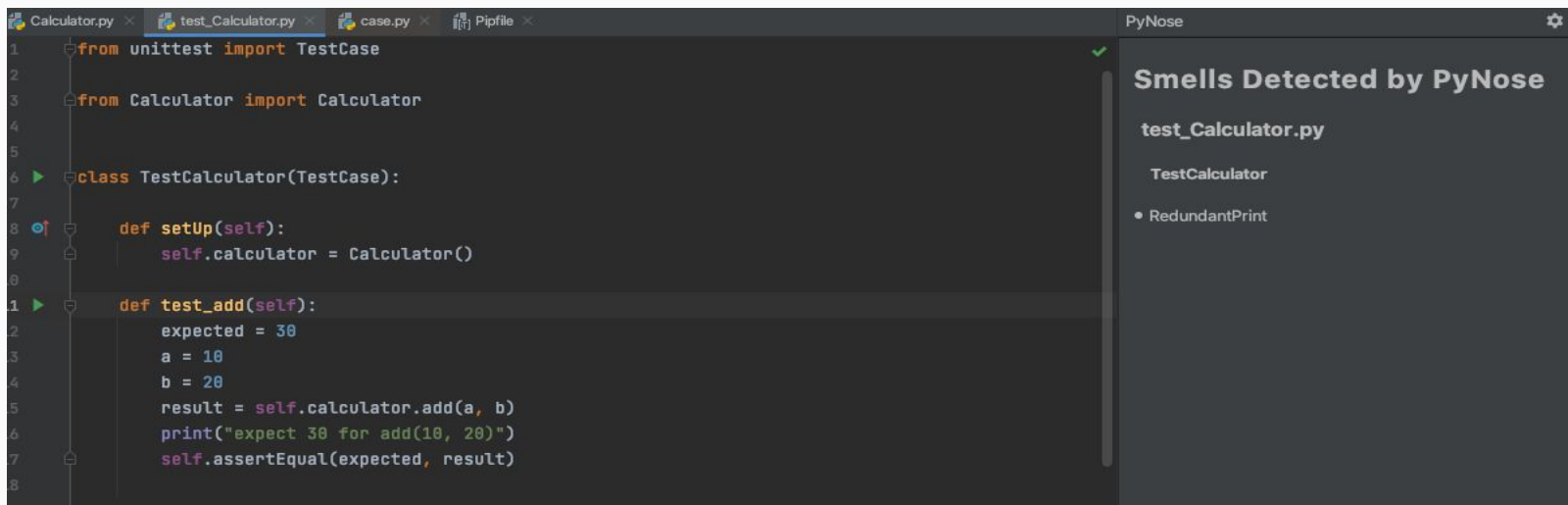
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         a = 10
13         b = 20
14         expected = 30
15         self.calculator.add(a, b)
16         self.assertEqual(expected, expected)
17
```

On the right side, the PyNose panel titled "Smells Detected by PyNose" shows the following structure:

- test_Calculator.py
 - TestCalculator
 - RedundantAssertion

Redundant Print

A test case that invokes the print() function.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

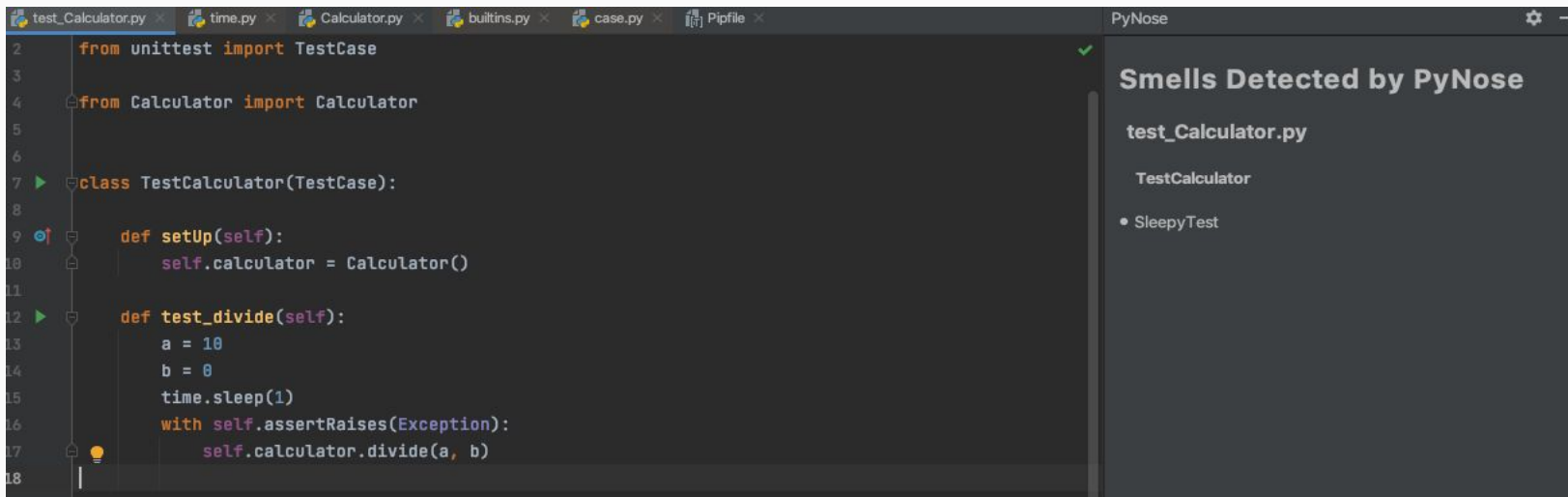
```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         expected = 30
13         a = 10
14         b = 20
15         result = self.calculator.add(a, b)
16         print("expect 30 for add(10, 20)")
17         self.assertEqual(expected, result)
```

On the right side, the PyNose panel displays the following information:

- Smells Detected by PyNose**
- test_Calculator.py**
- TestCalculator**
- RedundantPrint

Sleepy Test

A test case that invokes the `time.sleep()` function with no comment.



The screenshot shows a code editor with several tabs: `test_Calculator.py`, `time.py`, `Calculator.py`, `builtins.py`, `case.py`, and `Pipfile`. The `test_Calculator.py` tab is active, displaying the following code:

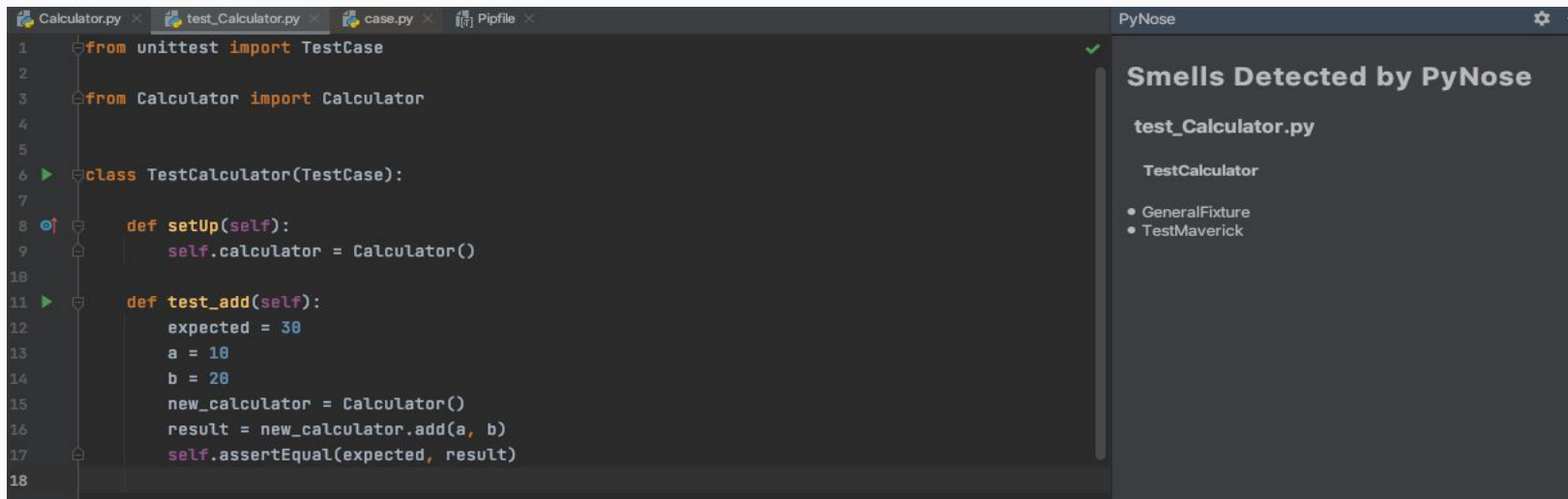
```
2 from unittest import TestCase
3
4 from Calculator import Calculator
5
6
7 class TestCalculator(TestCase):
8
9     def setUp(self):
10         self.calculator = Calculator()
11
12     def test_divide(self):
13         a = 10
14         b = 0
15         time.sleep(1)
16         with self.assertRaises(Exception):
17             self.calculator.divide(a, b)
18
```

On the right side of the editor, the PyNose interface is visible. It has a title bar "PyNose" and a settings icon. The main content area is titled "Smells Detected by PyNose" and lists the following:

- test_Calculator.py
 - TestCalculator
 - SleepyTest

Test Maverick

A test suite contains at least one test case that does not use a single field from the `SetUp()` method.



The screenshot shows a code editor with the following Python code in `test_Calculator.py`:

```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_add(self):
12         expected = 30
13         a = 10
14         b = 20
15         new_calculator = Calculator()
16         result = new_calculator.add(a, b)
17         self.assertEqual(expected, result)
```

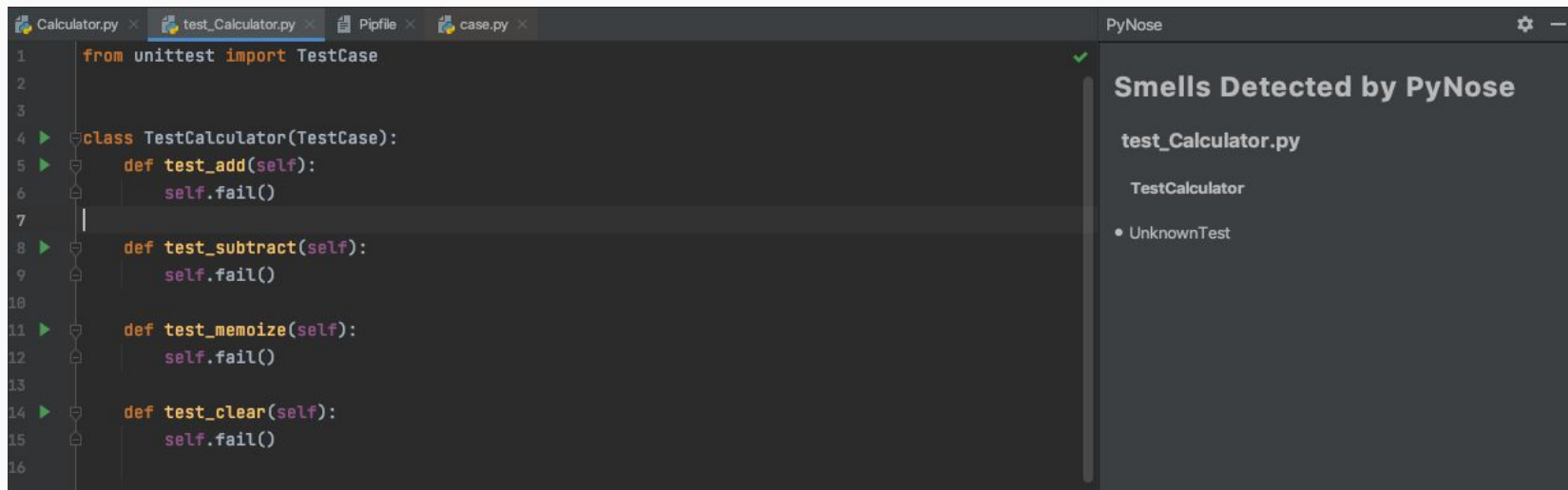
The `test_add` method is a TestMaverick because it creates a new `Calculator` instance instead of using `self.calculator` from the `setUp` method.

The PyNose sidebar on the right shows the following structure:

- Smells Detected by PyNose
 - test_Calculator.py
 - TestCalculator
 - GeneralFixture
 - TestMaverick

Unknown Test

A test case does not contain a single assertion statement.



The screenshot shows a code editor with a file named `test_Calculator.py`. The code defines a `TestCalculator` class that inherits from `unittest.TestCase`. It contains five test methods: `test_add`, `test_subtract`, `test_memoize`, and `test_clear`. Each of these methods simply calls `self.fail()` without containing any assertions. A green checkmark is visible in the top right corner of the editor area.

On the right side, the PyNose sidebar displays the title "Smells Detected by PyNose". Under the file `test_Calculator.py`, it lists the class `TestCalculator` and identifies a specific smell: `UnknownTest`.

Identifying Python specific test smells

- Selected a **primary dataset** of mature open source projects with atleast **1000 commits**, 10 contributors.
- Find all test files having "**test**" in its name. which is the **unittest** naming convention.
- Find patterns in the histories of the collected projects where changes made to test files that might be considered as **fixing the test or fixing the smell**.
- A Tool developed to mine change patterns in commit history is named **PythonChangeMiner**.

PythonChangeMiner

- It builds a **program dependence graph** with representing the code by showing its **data dependencies** and **control dependencies**.
- It builds a **change graph** for the fragment of code changes using the versions of code **before** and **after** the target change.
- It **mines** such change graphs from git repositories by traversing their **commit history**, and **discover patterns** in these change graphs.
- The pattern is defined by two thresholds: **minimum number of graph nodes** in the pattern, and **minimum number of repetitions** of the pattern in the corpus.

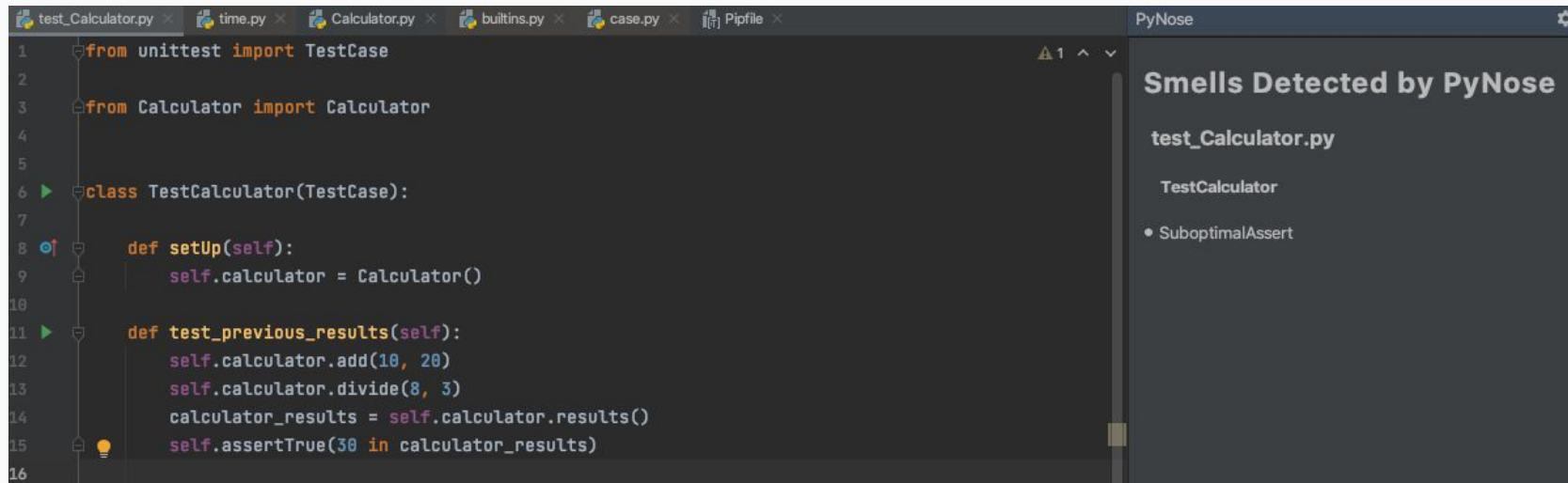
Suboptimal Assert

- After identifying common patterns more than 3 different projects, majority of patterns related to assertion functionality changes which are related to test smells.

1. Assertion changes that **alter the logic**. Ex. Changing from **assertEqual** to **assertRegex**.
2. Assertion changes that **do not alter the logic** and use more **appropriate** functions. Ex. **assertTrue(X in Y)** to **assertIn(X, Y)** or **assertEqual(X, False)** is changed to **assertFalse(X)**.
3. Assertion changes that **do not alter the logic** and **use less appropriate functions**. Ex. Moving from a more specific **assertIsNotNone(X)** to a more general **assertNotEqual(X, None)** which leads to test smell.

Suboptimal Assert

A test case that contains at least one of the suboptimal asserts.



The screenshot displays a code editor with several tabs: `test_Calculator.py`, `time.py`, `Calculator.py`, `builtins.py`, `case.py`, and `Pipfile`. The `test_Calculator.py` tab is active, showing the following code:

```
1 from unittest import TestCase
2
3 from Calculator import Calculator
4
5
6 class TestCalculator(TestCase):
7
8     def setUp(self):
9         self.calculator = Calculator()
10
11     def test_previous_results(self):
12         self.calculator.add(10, 20)
13         self.calculator.divide(8, 3)
14         calculator_results = self.calculator.results()
15         self.assertTrue(30 in calculator_results)
```

A yellow lightbulb icon is positioned next to line 15, indicating a detected issue. On the right side, the **PyNose** sidebar is open, titled "Smells Detected by PyNose". It lists the file `test_Calculator.py` and the class `TestCalculator`. Under the class, a single smell is listed: `SuboptimalAssert`.

PYNOSE

- **PYNOSE** is implemented as a **plugin** for **PyCharm** IDE.
- This tool identifies 18 test smells in actual Python code. It parses .py files to PSIFile objects and detects smells.
- Supports **GUI mode** in active code development or **CLI mode** for offline analysis.
- It can report the results as **JSON files** for further evaluation.

Evaluation of PyNose results

- Manually selected 8 projects that **did not make** it to **primary dataset** and marked as **validation set**.
- Then the validation are run against **PYNOSE** and results are compared against the manual set.
- The results like **precision, recall** and **F1 values** are compared with manual and looks very close. (94.0%, 95.8%, 94.9%)
- Also the overall results are compared with **TSDetect** (similar tool for Java). (96.0%, 97.1%, 96.5%)

Empirical study on test smell prevalence

- Apart from **primary dataset**, **secondary dataset** of projects (less commits) was selected for identifying test smells.
- The **purpose** of the **secondary dataset** is to make sure that the reported results are **unbiased**.
- Test smells **distribution** are similar when compared between **primary dataset** and **secondary dataset**.
- For better analysis, **co-occurrence of test smells** are identified across test suites. The test smell **distribution percentage** demonstrates that test smells have **relationship** with one another.

Future work

- Supporting more **PYTHON** test smells.
- Conducting a more thorough comparison of **PYNOSE** to other tools. For example, to **TSDetect** that works with Java/Scala.
- It would be of **great interest** to see how **test smells correlate** with **test coverage**.

Thanks!

