

Tool analysis document

Pynguin: Automated Unit Test Generation for Python, <https://doi.org/10.1145/3510454.3516829>

Team members: Crisan Dragos, Matei Daria, Mereu Bianca

Installation

The application is easy to install. The user needs to open a terminal and write `pip install pynguin`. After a successful installation, the only step left is to add an environment variable `PYNGUIN_DANGER_AWARE` and set it to a random value.

Usage

To generate unit tests for a specific module, the user needs to write inside the terminal the following command `pynguin --project-path C:\Users\Daria\PycharmProjects\Calculator\src -output-path C:\Users\Daria\PycharmProjects\Calculator\pynguin-results --module-name Calculator` and replace the project path, output path and module name with their own information. If there are no issues, Pynguin successfully generates a Python file with the unit tests in the output path. It also generates in the directory in which the terminal is opened a directory `pynguin-report` with files `pynguin-config.txt` and `statistics.csv`. The first one contains information about the project path, output path, module name, the algorithms and strategies used, while `statistics.csv` contains, for each execution of the previously mentioned command, the test coverage achieved by the generated tests for the target module.

Results

The tool generates valid and useful test cases, especially if parameters and return types are annotated. The reports are useful as well, especially the code coverage statistics. However, some test cases are either useless, too complicated for a simple function they test (in our case, simple methods like arithmetic add, subtract) or test for too many things inside one unit test, which defeats the purpose of a unit test. In conclusion, this tool is generally useful and, even when its performance is not as great, is a good starting point for writing unit tests manually.

```
Arguments: (<_RefreshThread(Thread-1, started daemon 10560)>,)
(.venv) PS C:\Users\Daria\PycharmProjects\Calculator> pynguin --project-path C:\Users\Daria\PycharmProjects\Calculator\src --output-path C:\Users\Daria\PycharmProjects\Calculator\pynguin-results --module-name Calculator
! Running Pynguin...
```

Pynguin running

Project

- Calculator C:\Users\Daria\PycharmProjects\Calculator
- .venv library root
- pynguin-report
 - pynguin-config.txt
 - statistics.csv
- pynguin-results
 - test_Calculator.py
- src
 - Calculator.py

```
@pytest.mark.xfail(strict=True)
def test_case_1():
    bool_0 = False
    calculator_0 = module_0.Calculator()
    float_0 = calculator_0.divide(bool_0, bool_0)
    assert float_0 == "Cannot divide by zero."
    calculator_1 = module_0.Calculator()
    float_1 = calculator_1.subtract(bool_0, bool_0)
    bool_1 = True
    float_2 = -1547.9
    float_3 = calculator_1.add(bool_1, float_2)
    assert float_3 == pytest.approx(-1546.9, abs=0.01, rel=0.01)
    bool_2 = True
    float_4 = calculator_1.add(bool_0, bool_2)
    assert float_4 == 1
    calculator_2 = module_0.Calculator()
    float_5 = calculator_2.add(bool_0, bool_0)
    int_0 = 435
    int_1 = 2141
    float_6 = calculator_0.divide(int_0, int_1)
    assert float_6 == pytest.approx( expected: 0.203176085941149, abs=0.01, rel=0.01)
    module_0.Calculator(**bool_0)
```

Test which is too complicated

```
def test_case_2():
    int_0 = 3027
    calculator_0 = module_0.Calculator()
    float_0 = calculator_0.subtract(int_0, int_0)
    assert float_0 == 0

def test_case_3():
    int_0 = -1375
    calculator_0 = module_0.Calculator()
    float_0 = calculator_0.add(int_0, int_0)
    assert float_0 == -2750
```

Useful tests

Run Python tests in test_Calculator.py

Test Results

Test	Time	Status
test_Calculator	1 ms	Passed
test_case_2	0 ms	Passed
test_case_3	0 ms	Passed

Tests passed: 2, ignored: 3 of 5 tests - 1 ms

Testing started at 22:24

Launching pytest with arg