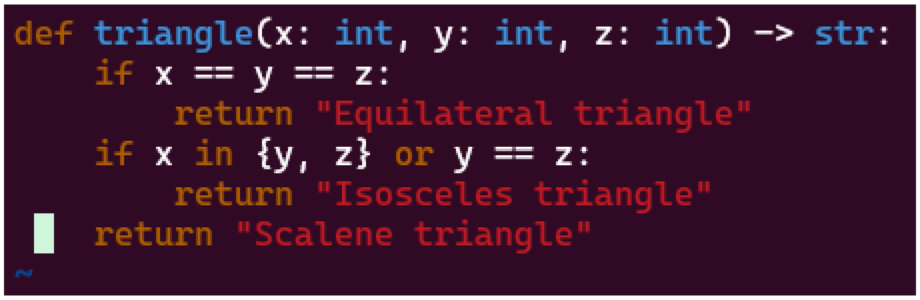
Sali Arnold – Reitler Alex – Pop David Alexandru - 936

Pynguin: Automated Unit Test Generation for Python

doi: 10.1145/3510454.3516829

Stephan Lukasczyk - Gordon Fraser

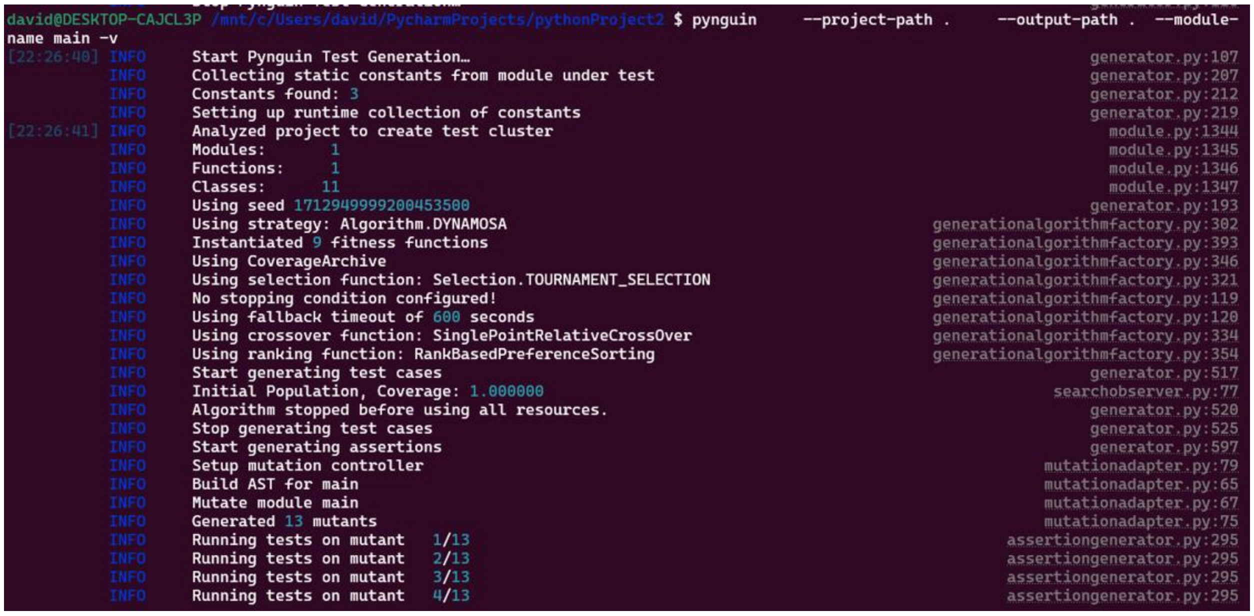
Pynguin is a tool for Python to generate unit tests for a method. Before starting to use Pynguin we must have a CLI such as WSL or PowerShell, a python interpreter installed and a python package manager, in this example we will use Pip. To start with Pynguin we go to the main website and get the install command using Pip, *pip install pynguin*. This is the easiest way to start with this package. After this, we will create a new Python project and define a simple method in this package. In this example we will create a method for triangle to check its sides and return the type of the triangle (isosceles, equilateral, scalene).



```python
def triangle(x: int, y: int, z: int) -> str:
    if x == y == z:
        return "Equilateral triangle"
    if x in {y, z} or y == z:
        return "Isosceles triangle"
    return "Scalene triangle"
~
```

Figure 1. Triangle method

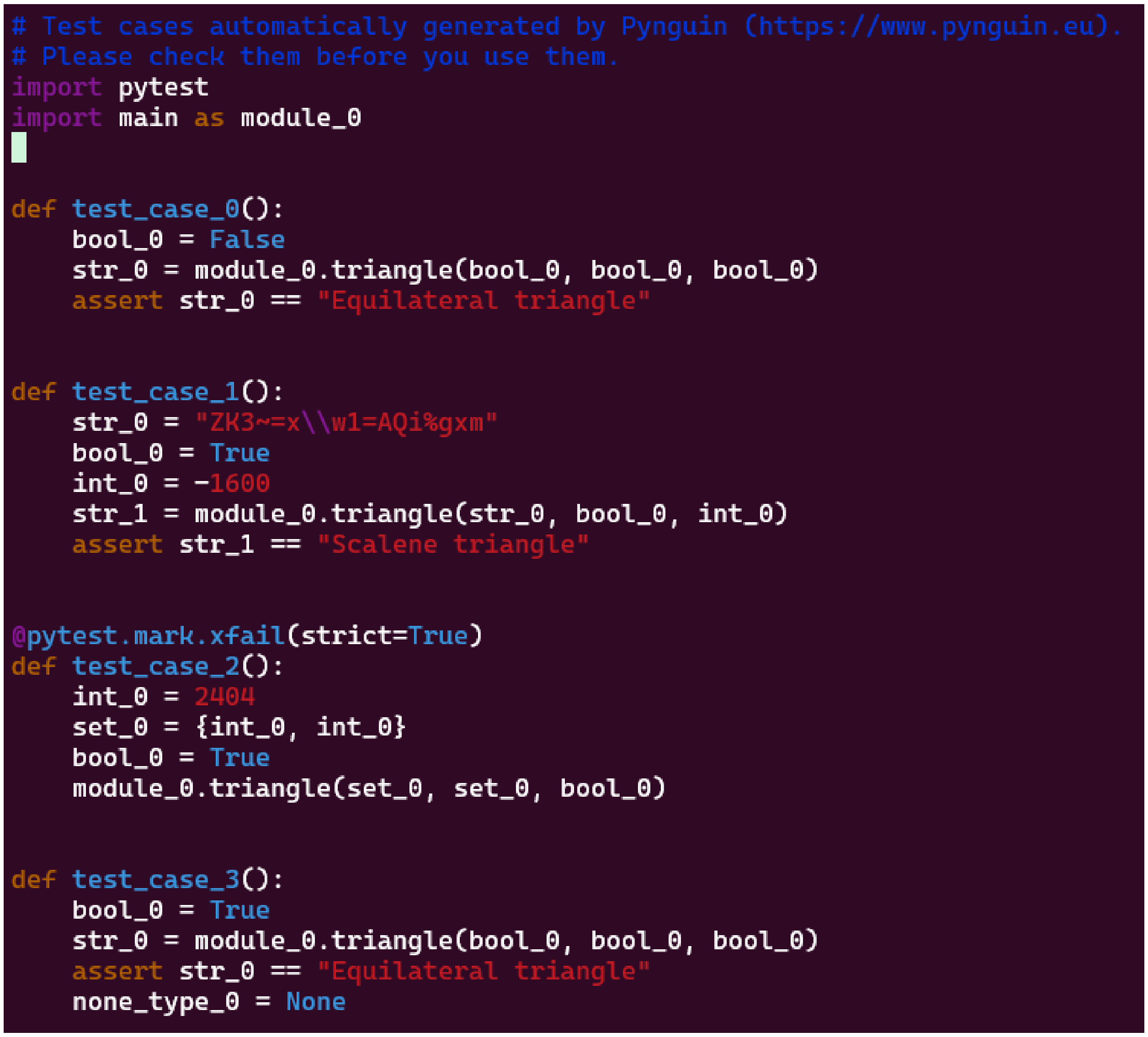
After we define the above method, we will open the terminal, change the path to our project and type the following command *pynguin –-project-path* . *--output-path* . *--module-name main -v*. This command will take the method from the main program and create the tests for this method. The new file created is call *test_main.py*, here are the tests created.



```
david@DESKTOP-CAJCL3P /mnt/c/Users/david/PycharmProjects/pythonProject2 $ pynguin     --project-path .     --output-path .  --module-
name main -v
[22:26:40] INFO     Start Pynguin Test Generation…                                                               generator.py:107
           INFO     Collecting static constants from module under test                                           generator.py:207
           INFO     Constants found: 3                                                                           generator.py:212
           INFO     Setting up runtime collection of constants                                                   generator.py:219
[22:26:41] INFO     Analyzed project to create test cluster                                                       module.py:1344
           INFO     Modules:       1                                                                              module.py:1345
           INFO     Functions:     1                                                                              module.py:1346
           INFO     Classes:      11                                                                              module.py:1347
           INFO     Using seed 1712949999200453500                                                               generator.py:193
           INFO     Using strategy: Algorithm.DYNAMOSA                                         generationalgorithmfactory.py:302
           INFO     Instantiated 9 fitness functions                                           generationalgorithmfactory.py:393
           INFO     Using CoverageArchive                                                      generationalgorithmfactory.py:346
           INFO     Using selection function: Selection.TOURNAMENT_SELECTION                   generationalgorithmfactory.py:321
           INFO     No stopping condition configured!                                          generationalgorithmfactory.py:119
           INFO     Using fallback timeout of 600 seconds                                      generationalgorithmfactory.py:120
           INFO     Using crossover function: SinglePointRelativeCrossOver                     generationalgorithmfactory.py:334
           INFO     Using ranking function: RankBasedPreferenceSorting                         generationalgorithmfactory.py:354
           INFO     Start generating test cases                                                                  generator.py:517
           INFO     Initial Population, Coverage: 1.000000                                                  searchobserver.py:77
           INFO     Algorithm stopped before using all resources.                                                generator.py:520
           INFO     Stop generating test cases                                                                   generator.py:525
           INFO     Start generating assertions                                                                  generator.py:597
           INFO     Setup mutation controller                                                               mutationadapter.py:79
           INFO     Build AST for main                                                                      mutationadapter.py:65
           INFO     Mutate module main                                                                      mutationadapter.py:67
           INFO     Generated 13 mutants                                                                    mutationadapter.py:75
           INFO     Running tests on mutant   1/13                                                     assertiongenerator.py:295
           INFO     Running tests on mutant   2/13                                                     assertiongenerator.py:295
           INFO     Running tests on mutant   3/13                                                     assertiongenerator.py:295
           INFO     Running tests on mutant   4/13                                                     assertiongenerator.py:295
```

Figure 2. The output of pynguin command

To check the results of tests we can check and run the file *test_main.py*.



```python
# Test cases automatically generated by Pynguin (https://www.pynguin.eu).
# Please check them before you use them.
import pytest
import main as module_0


def test_case_0():
    bool_0 = False
    str_0 = module_0.triangle(bool_0, bool_0, bool_0)
    assert str_0 == "Equilateral triangle"


def test_case_1():
    str_0 = "ZK3~=x\\w1=AQi%gxm"
    bool_0 = True
    int_0 = -1600
    str_1 = module_0.triangle(str_0, bool_0, int_0)
    assert str_1 == "Scalene triangle"


@pytest.mark.xfail(strict=True)
def test_case_2():
    int_0 = 2404
    set_0 = {int_0, int_0}
    bool_0 = True
    module_0.triangle(set_0, set_0, bool_0)


def test_case_3():
    bool_0 = True
    str_0 = module_0.triangle(bool_0, bool_0, bool_0)
    assert str_0 == "Equilateral triangle"
    none_type_0 = None
```

Figure 3. Tests generated