Splat: Simple Python Lazy Automated Tester

tiny.cc/lwy08

Lee Wei Yeong
lwy08@doc.ic.ac.uk

□ Introduction

Motivation

- writing unit tests
  - crucial
  - costly but invisible
  - can be automated

python™

- dynamically typed
- clean, concise & elegant syntax
- builtin introspection & reflection facility

Why bytecode?

- "*Growing numbers of...* ***closed source*** *applications... using...Python...*" (Smith, 2010)
- simple
- fast

Background

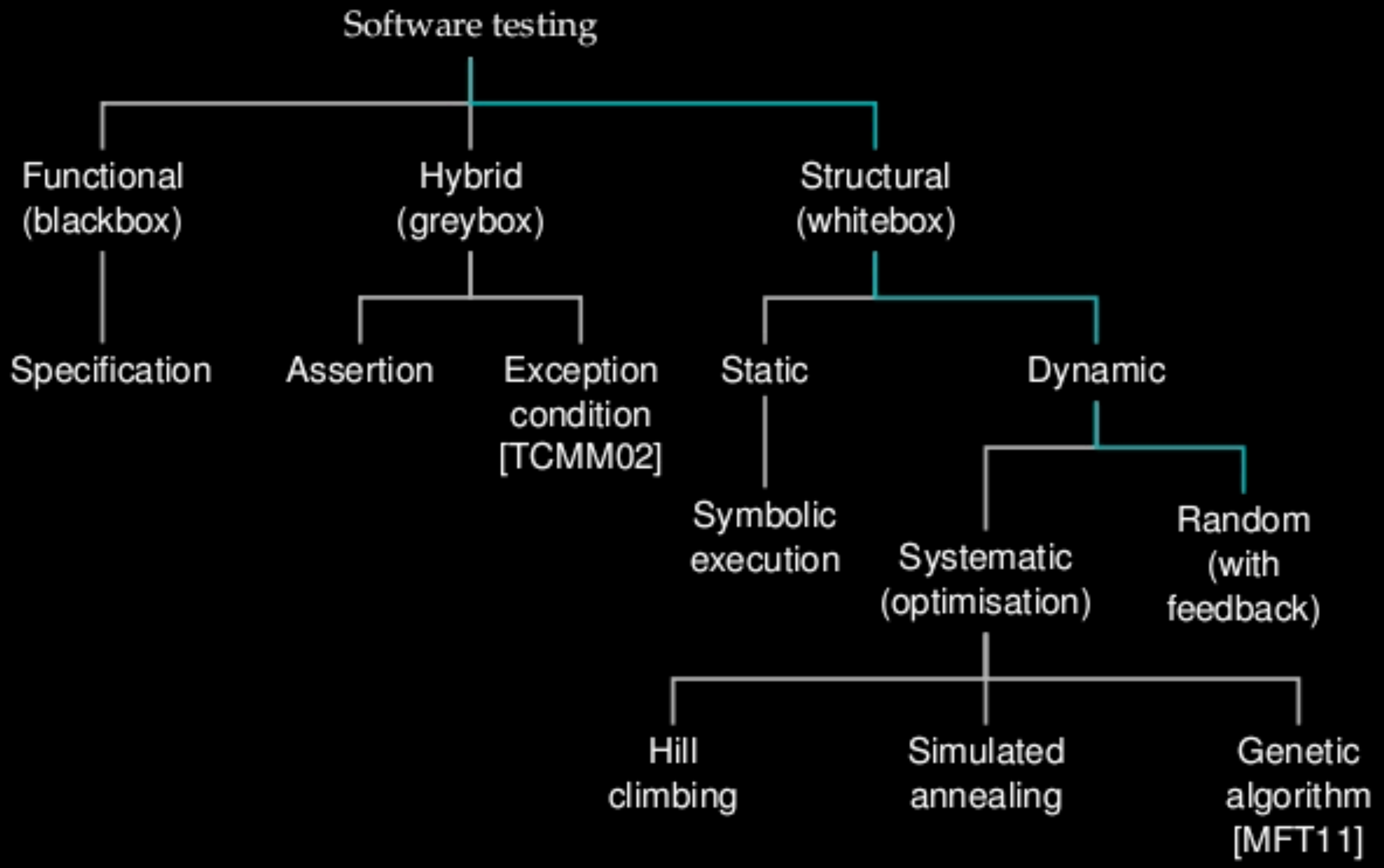
Existing tools

- Pythoscope
- Pytestsgenerator
- ...but neither works!

Existing tools

- Ruby Test case Generator (RuTeG)
  - structural testing
  - evolutionary search
  - tests object-oriented programs

Software testing



Software testing

Functional (blackbox)

Specification

Hybrid (greybox)

Assertion

Exception condition [TCMM02]

Structural (whitebox)

Static

Symbolic execution

Dynamic

Systematic (optimisation)

Hill climbing

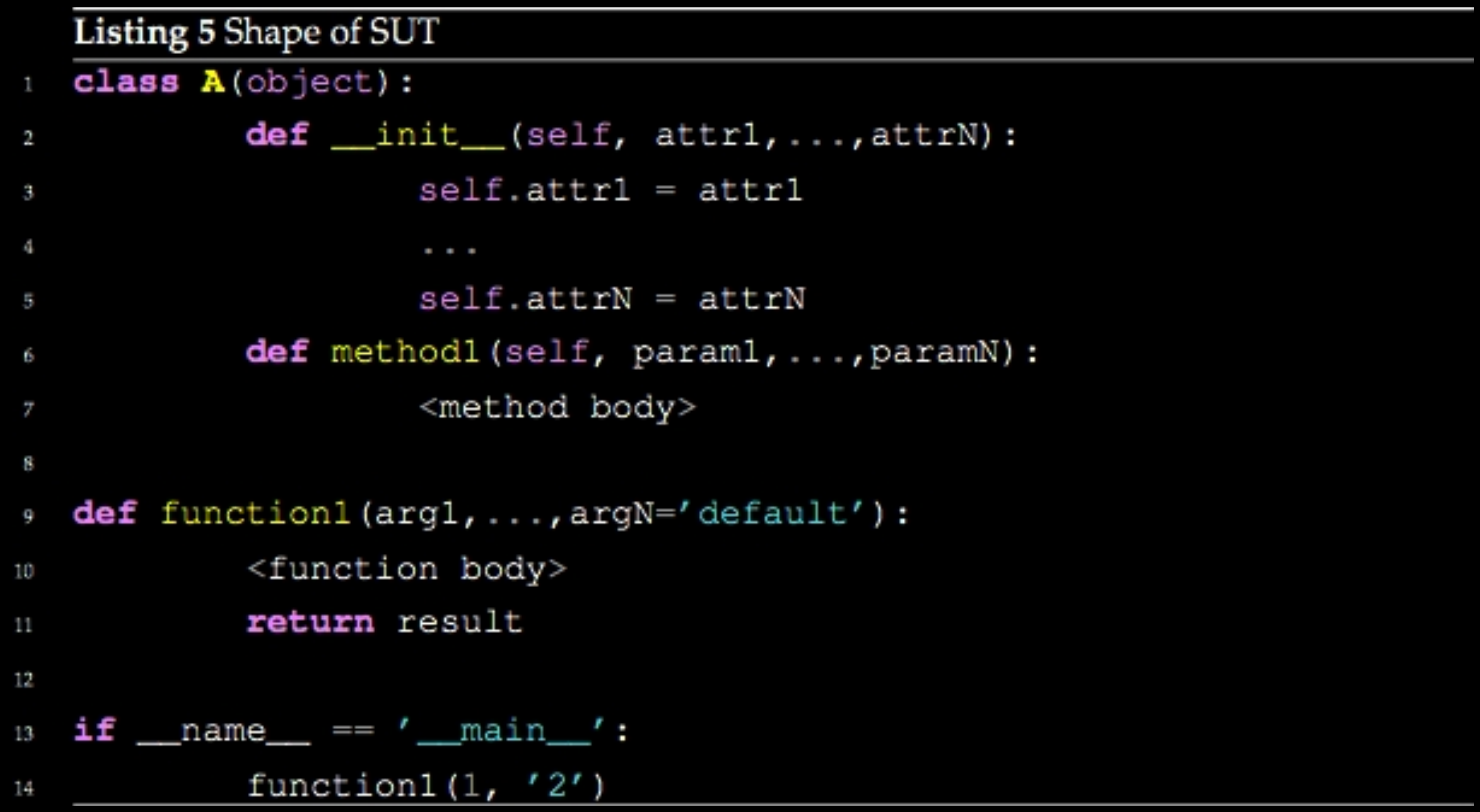
Simulated annealing

Genetic algorithm [MFT11]

Random (with feedback)

Figure 2.1.: High level overview of software testing

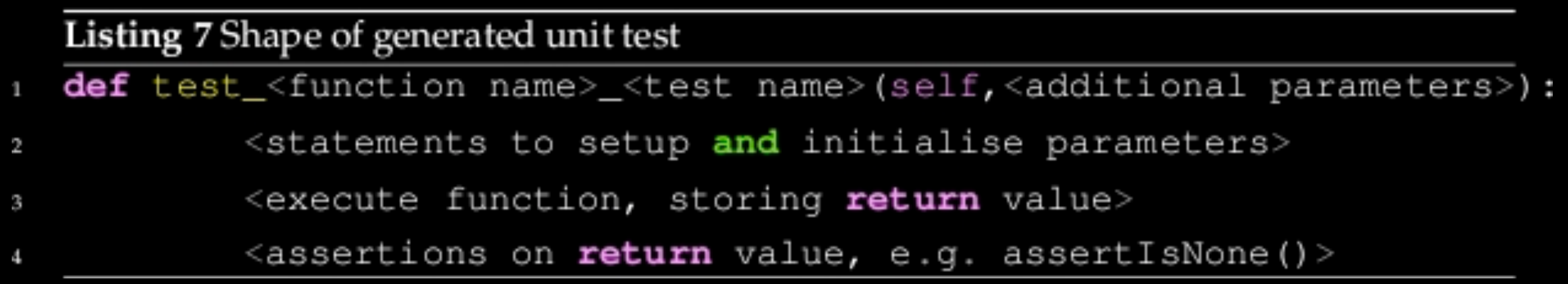
Sample test target



Listing 5 Shape of SUT

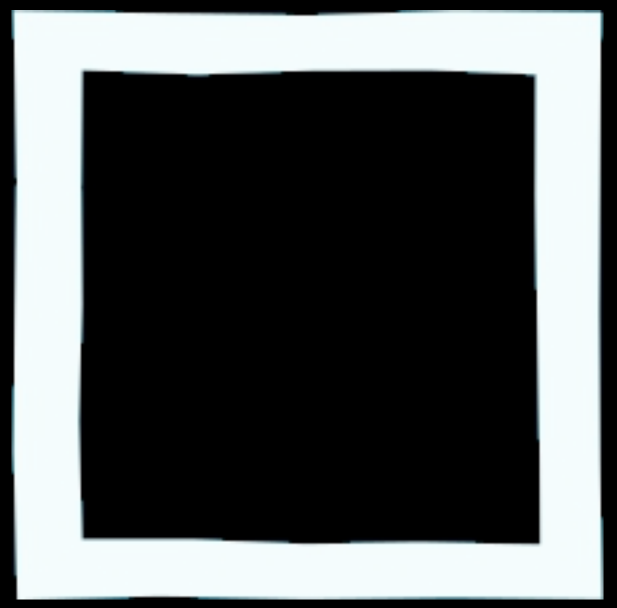
```
class A(object):
        def __init__(self, attr1,...,attrN):
                self.attr1 = attr1
                ...
                self.attrN = attrN
        def method1(self, param1,...,paramN):
                <method body>

def function1(arg1,...,argN='default'):
        <function body>
        return result

if __name__ == '__main__':
        function1(1, '2')
```

Sample unit test



Listing 7 Shape of generated unit test

```
def test_<function name>_<test name>(self,<additional parameters>):
        <statements to setup and initialise parameters>
        <execute function, storing return value>
        <assertions on return value, e.g. assertIsNone()>
```

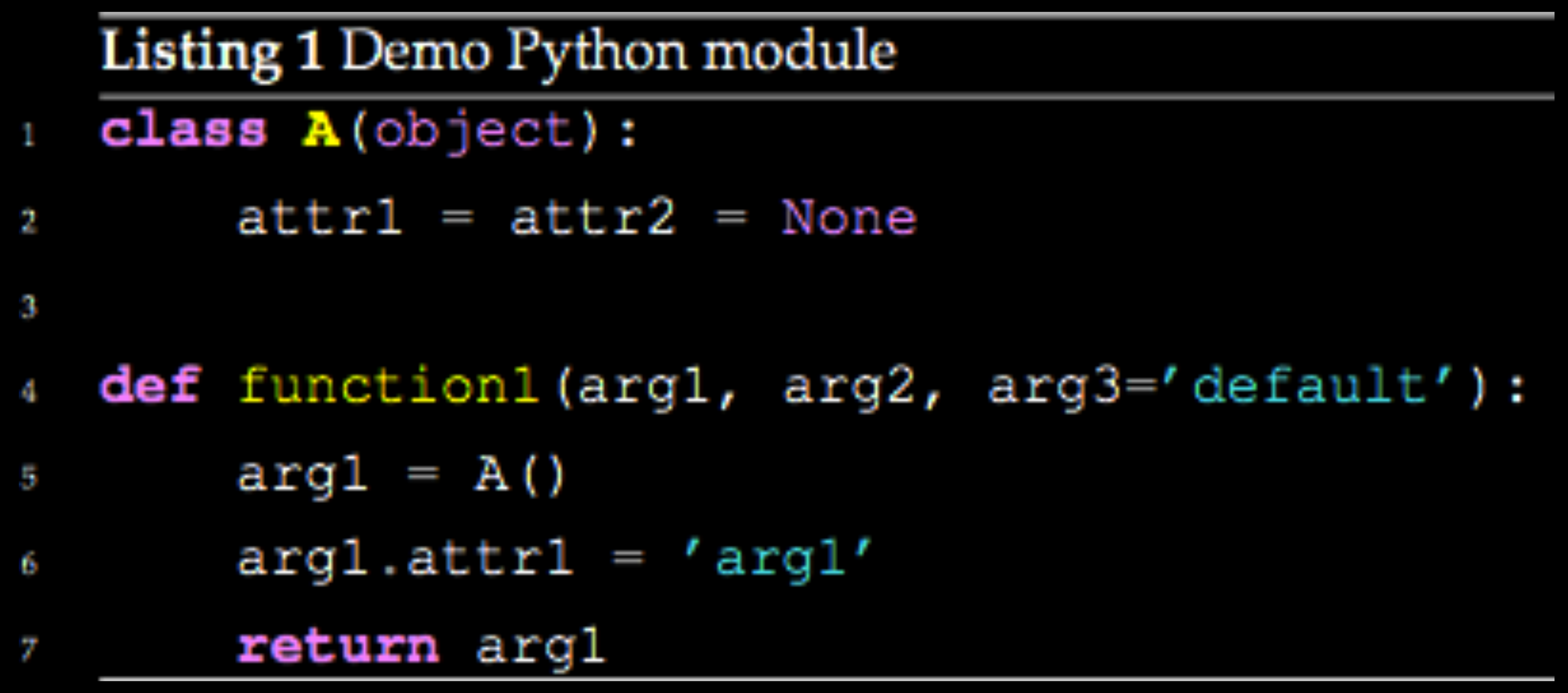


Splat

Lazy instantiation

- technique inspired by IRULAN
- initially: None proxy metaclass wrapper
- test data only generated for active variables

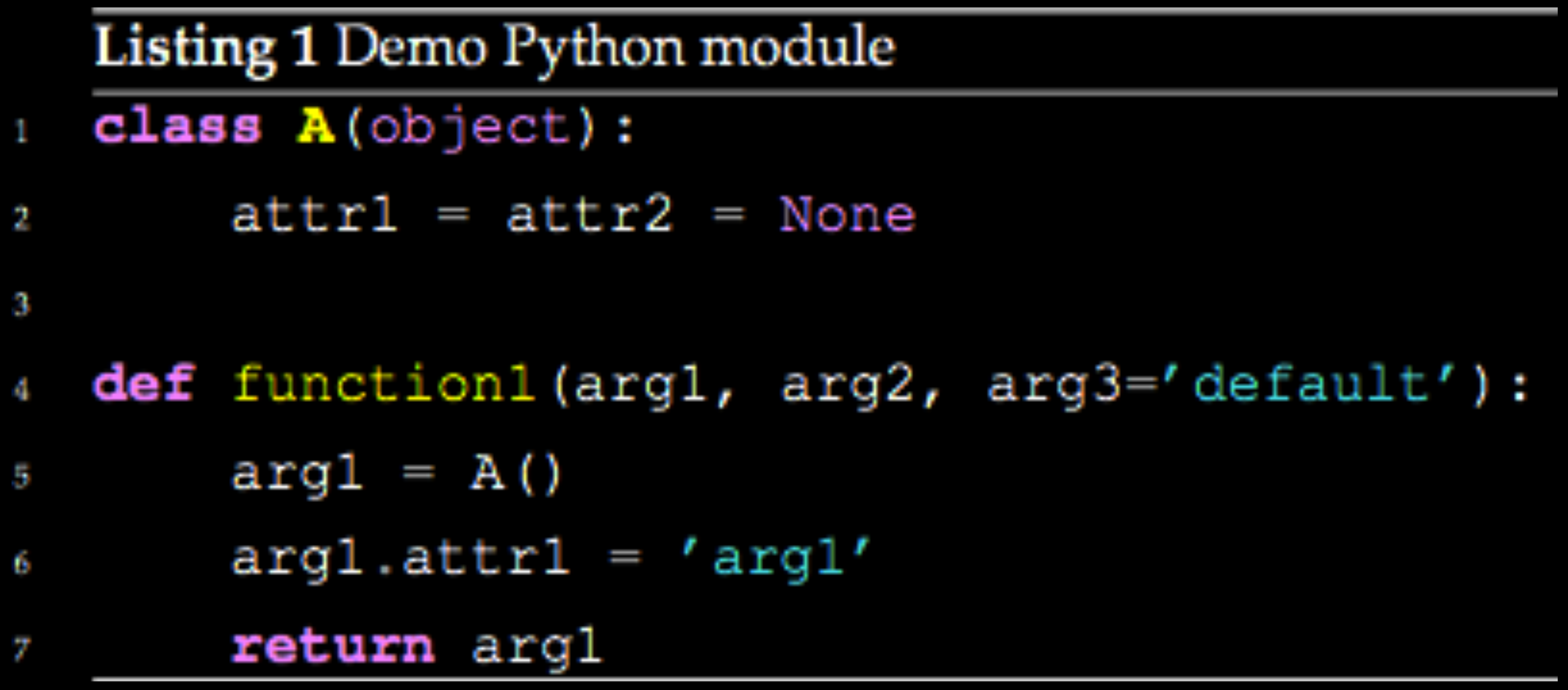
Lazy instantiation



Listing 1 Demo Python module

```
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

Lazy instantiation

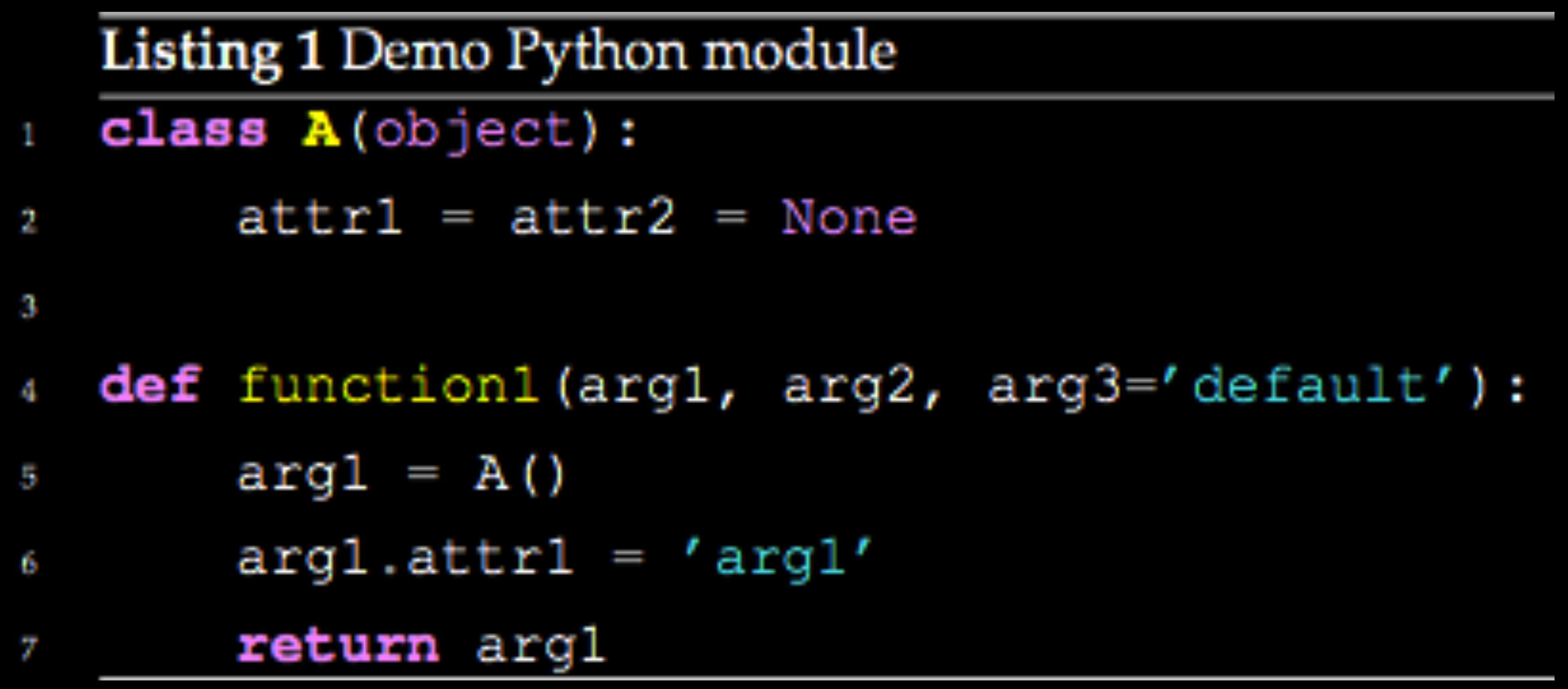


Listing 1 Demo Python module

```python
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	('arg1')
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

Lazy instantiation



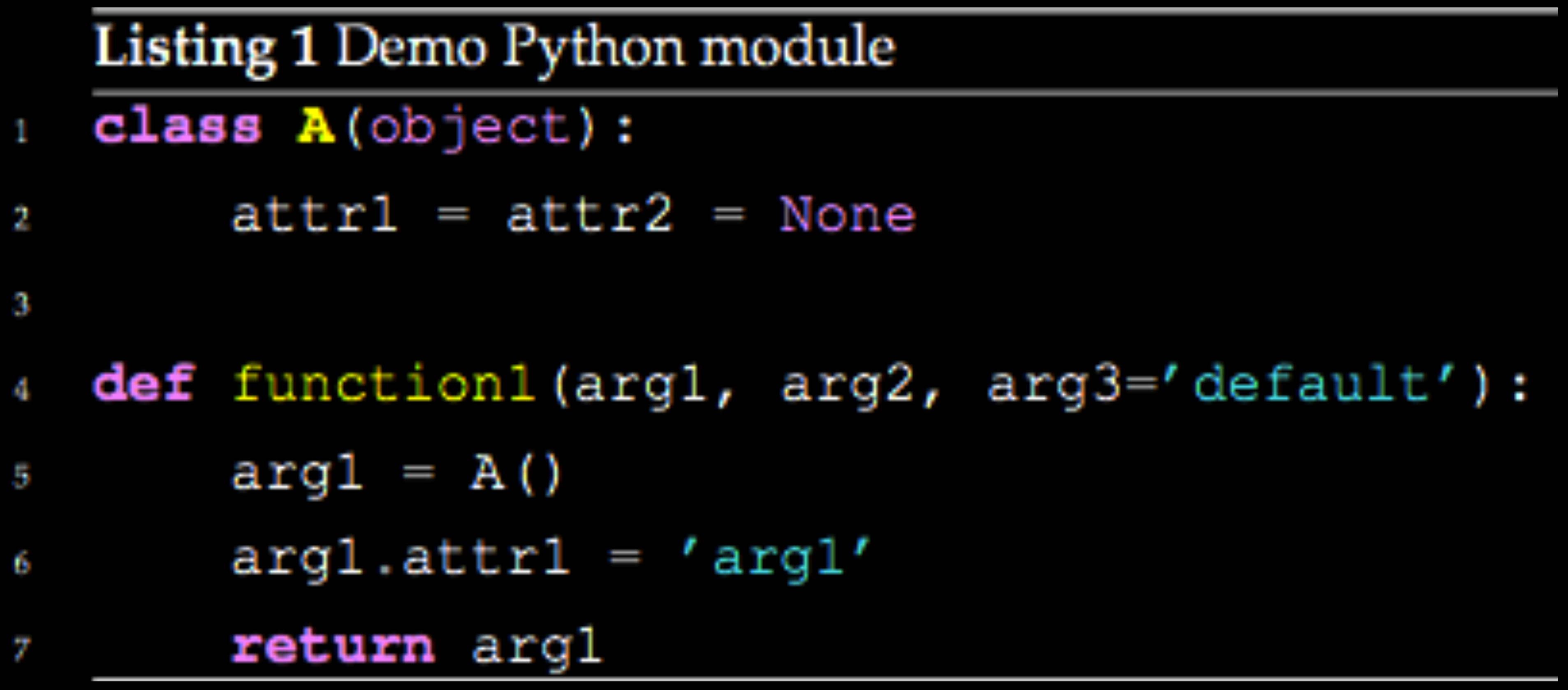
Listing 1 Demo Python module

```python
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3=’default’):
    arg1 = A()
    arg1.attr1 = ’arg1’
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	(‘arg1’)
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

```
function1
(
 Param1(None),
 Param2(None),
 Param3(None)
 )
```

Lazy instantiation



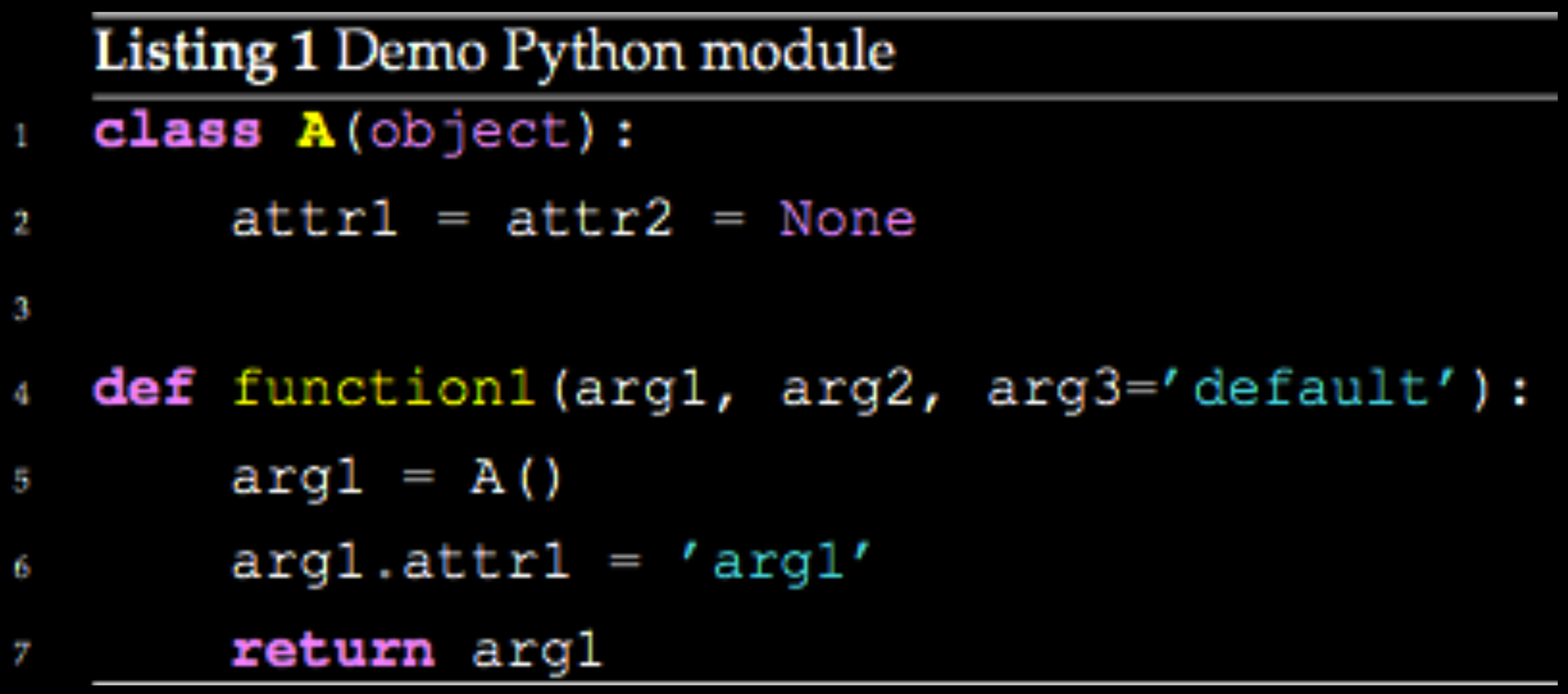
Listing 1 Demo Python module

```python
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	('arg1')
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

```
function1
(
 Param1(None),
 Param2(None),
 'default'
 )
```

Lazy instantiation



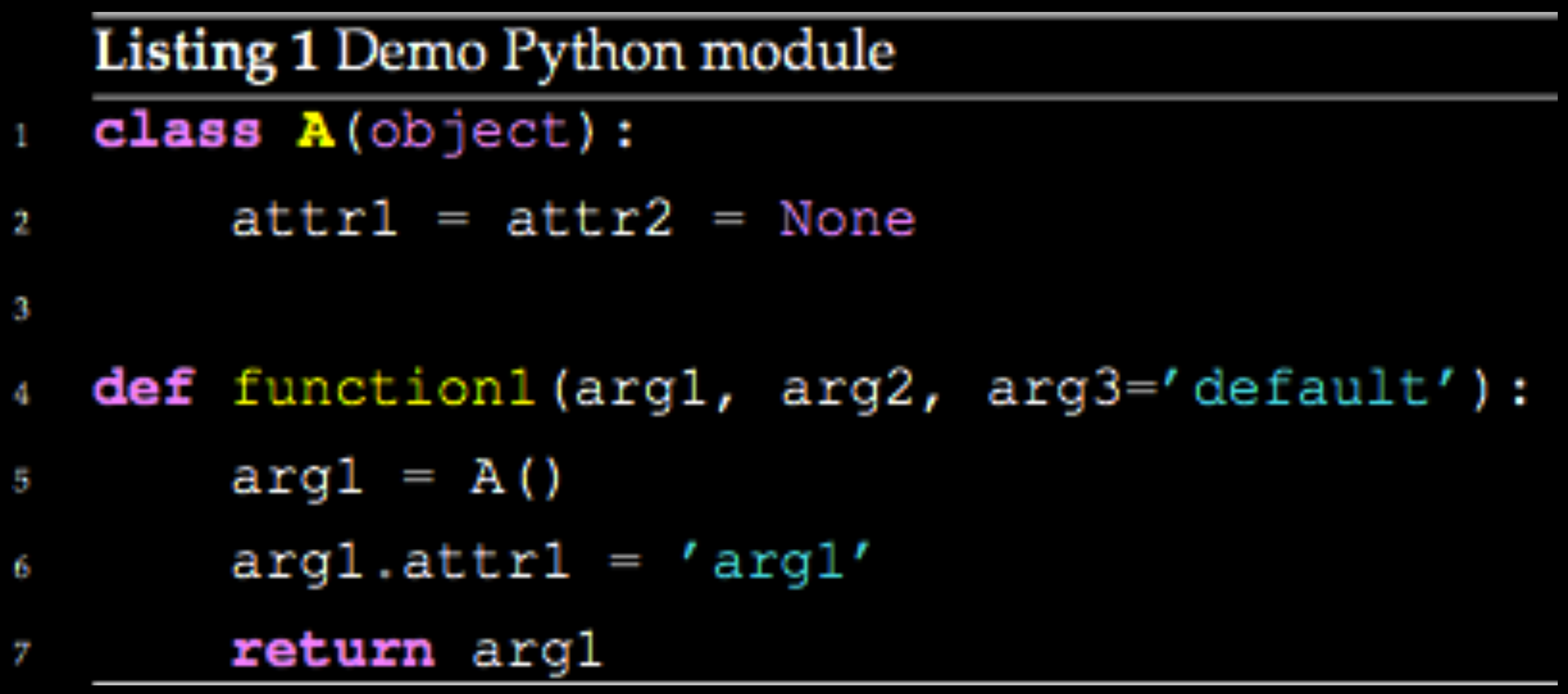
Listing 1 Demo Python module

```
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	('arg1')
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

```
function1
(
 Param1(None),
 Param2(None),
 'default'
 )
```

Lazy instantiation



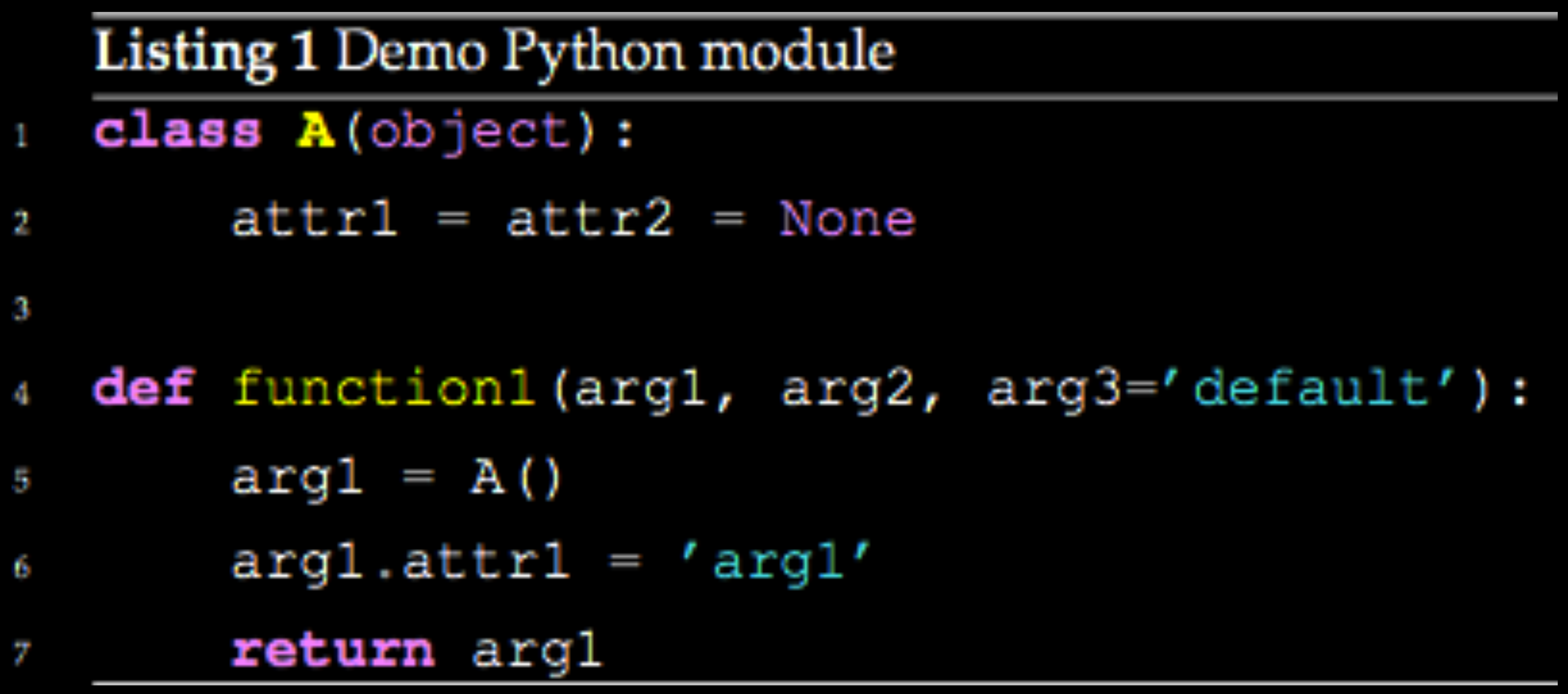
Listing 1 Demo Python module

```
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3=’default’):
    arg1 = A()
    arg1.attr1 = ’arg1’
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	(‘arg1’)
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

TypeError!

Lazy instantiation



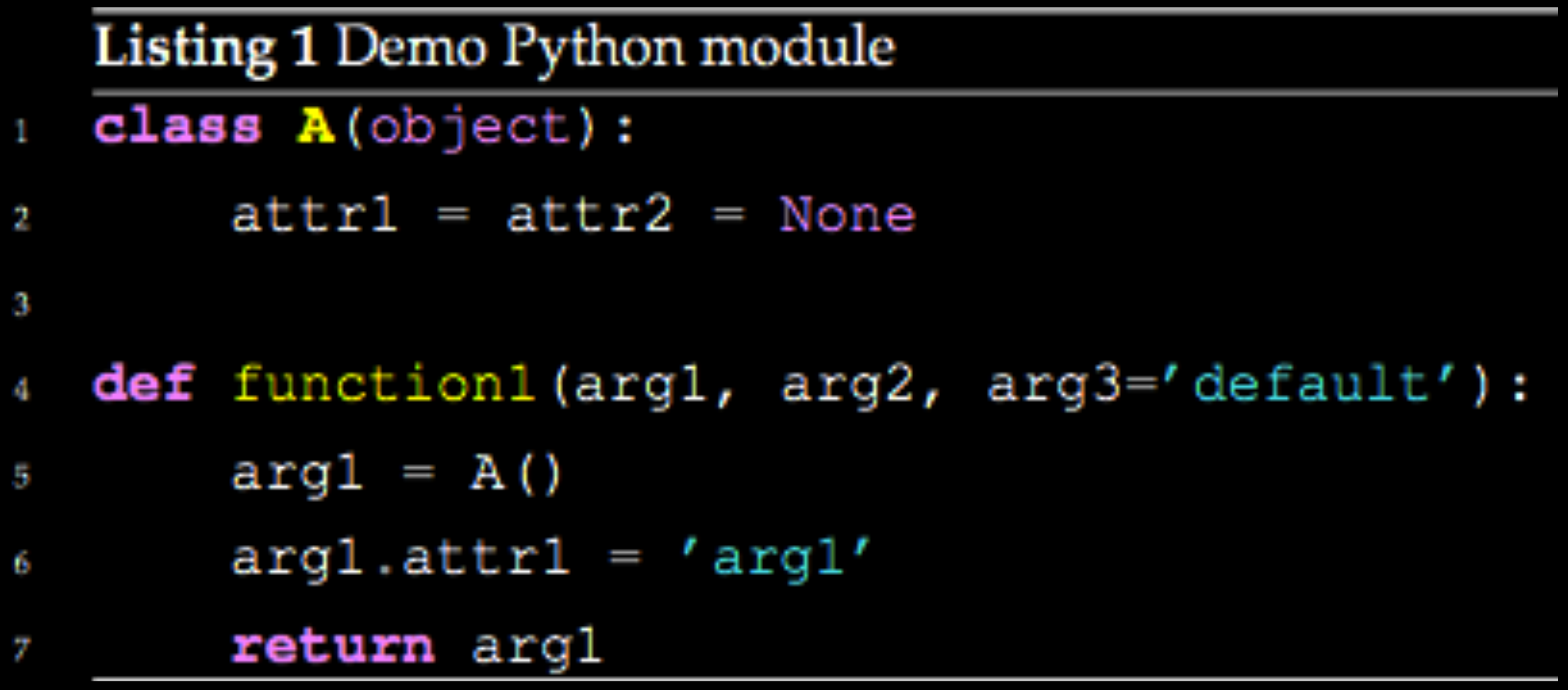
Listing 1 Demo Python module

```python
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	('arg1')
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

```
function1
(
 {
  'attr1':
   Param1_attr1(None)
 },
 Param2(None),
 'default'
 )
```

Lazy instantiation



Listing 1 Demo Python module

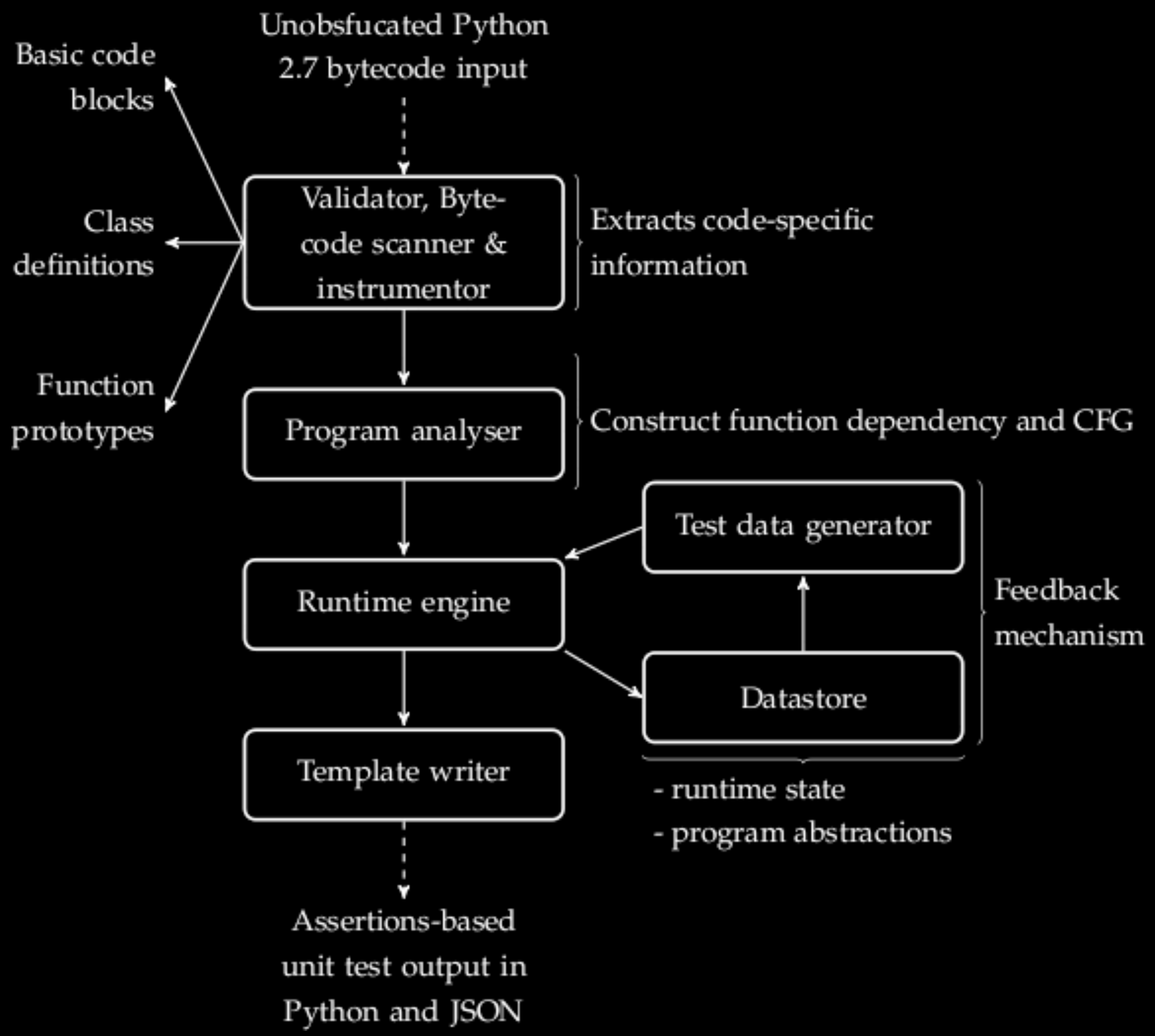
```python
class A(object):
    attr1 = attr2 = None

def function1(arg1, arg2, arg3='default'):
    arg1 = A()
    arg1.attr1 = 'arg1'
    return arg1
```

LOAD_GLOBAL	(A)
CALL_FUNCTION	0
STORE_FAST	(arg1)
LOAD_CONST	(‘arg1’)
LOAD_FAST	(arg1)
STORE_ATTR	(attr1)
LOAD_FAST	(arg1)
RETURN_VALUE	

```
function1
(
 {
  ‘attr1’:
   Param1_attr1(None)
 },
 None,
 ‘default’
 )
```

□ Demonstration

Architecture



Challenges

- Vague error messages
- Lack of existing tool support
- Test program contains imports

Challenges

- Relationship between input arguments
- Various programming constructs
- Range of test data values

Limitations

- Custom Exceptions, e.g. `InvalidGraphType`
- Random functions
- Generators

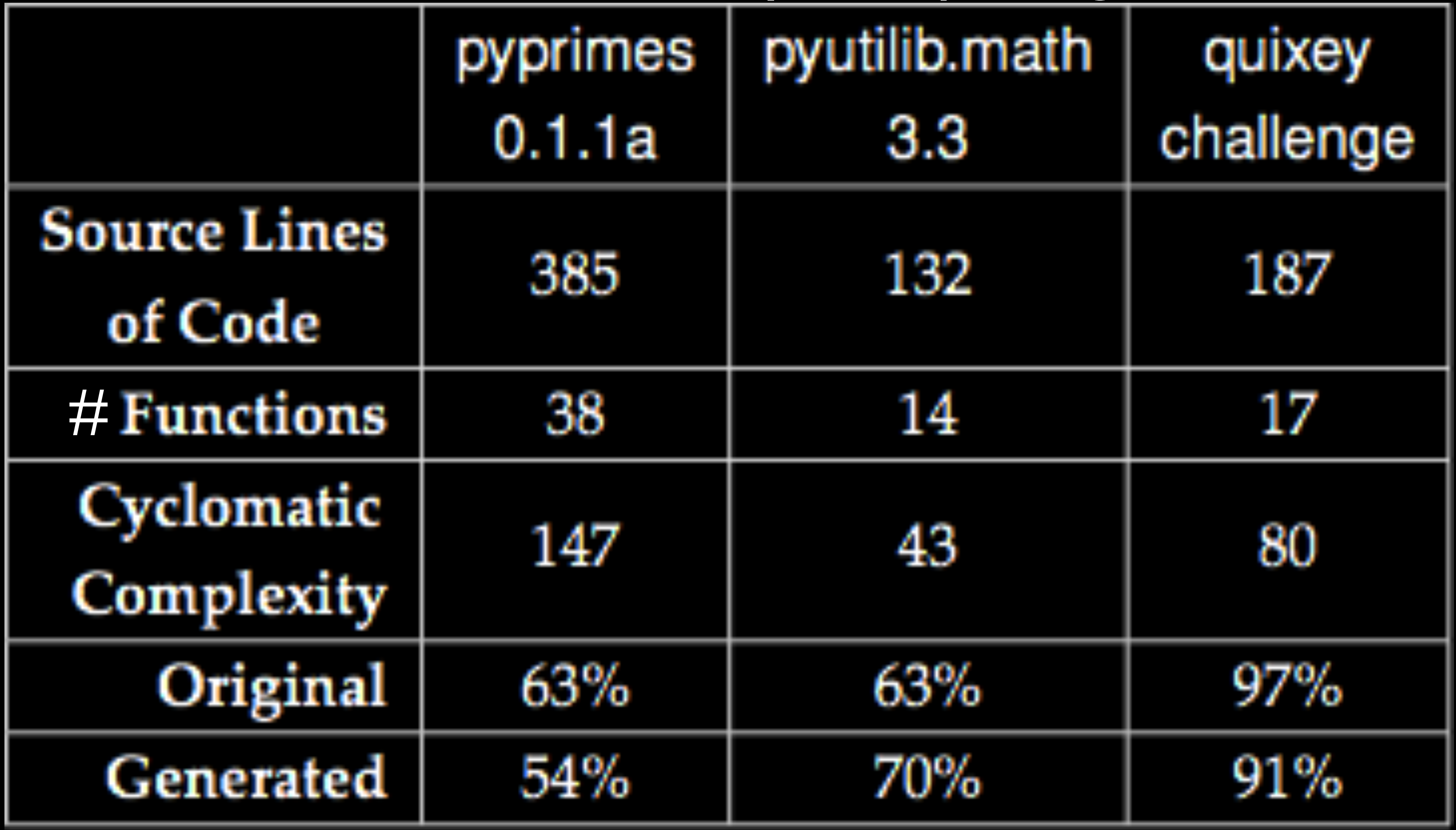
□ Evaluation

Criteria

- Quality
- Performance
- Generality

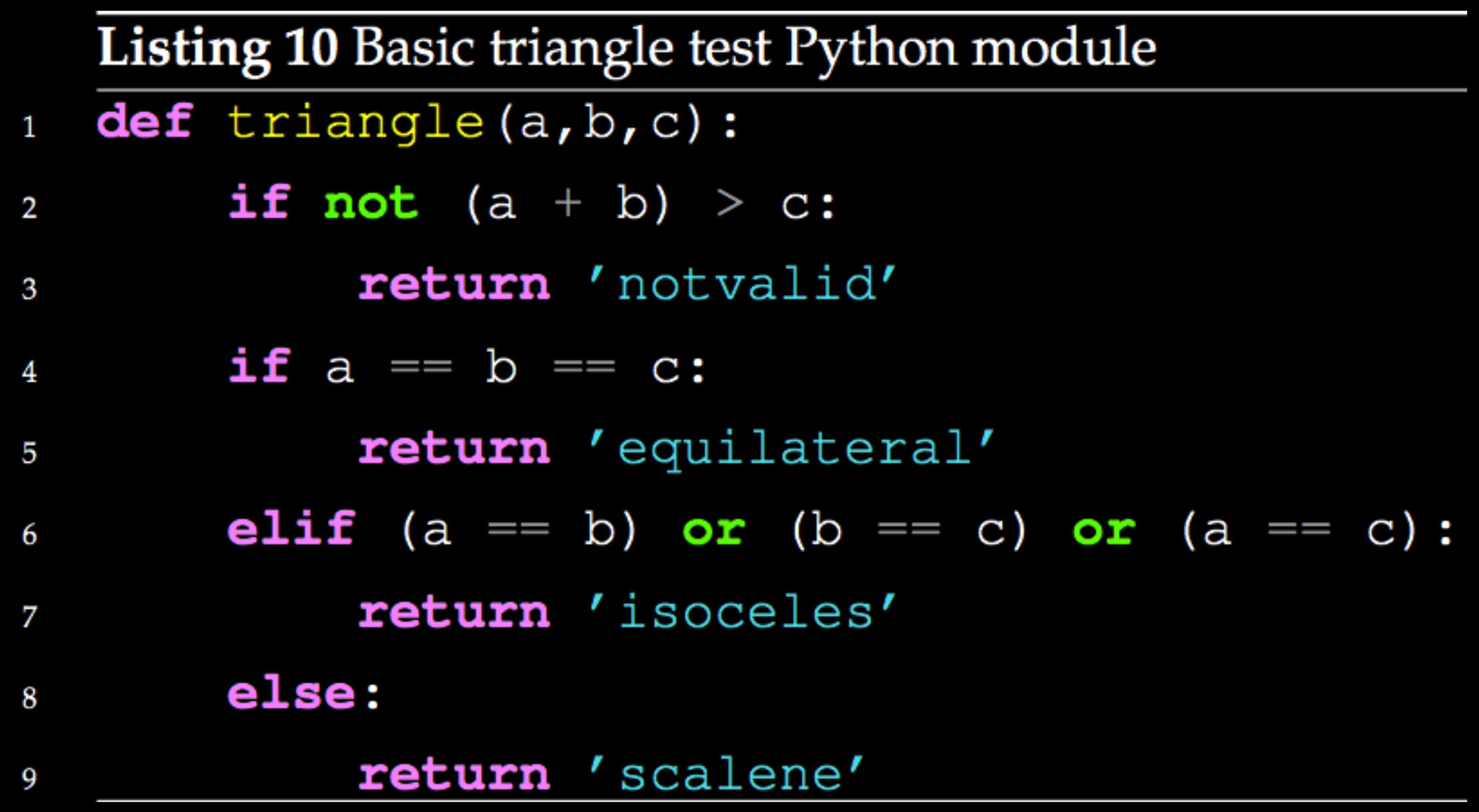
Results

Python package



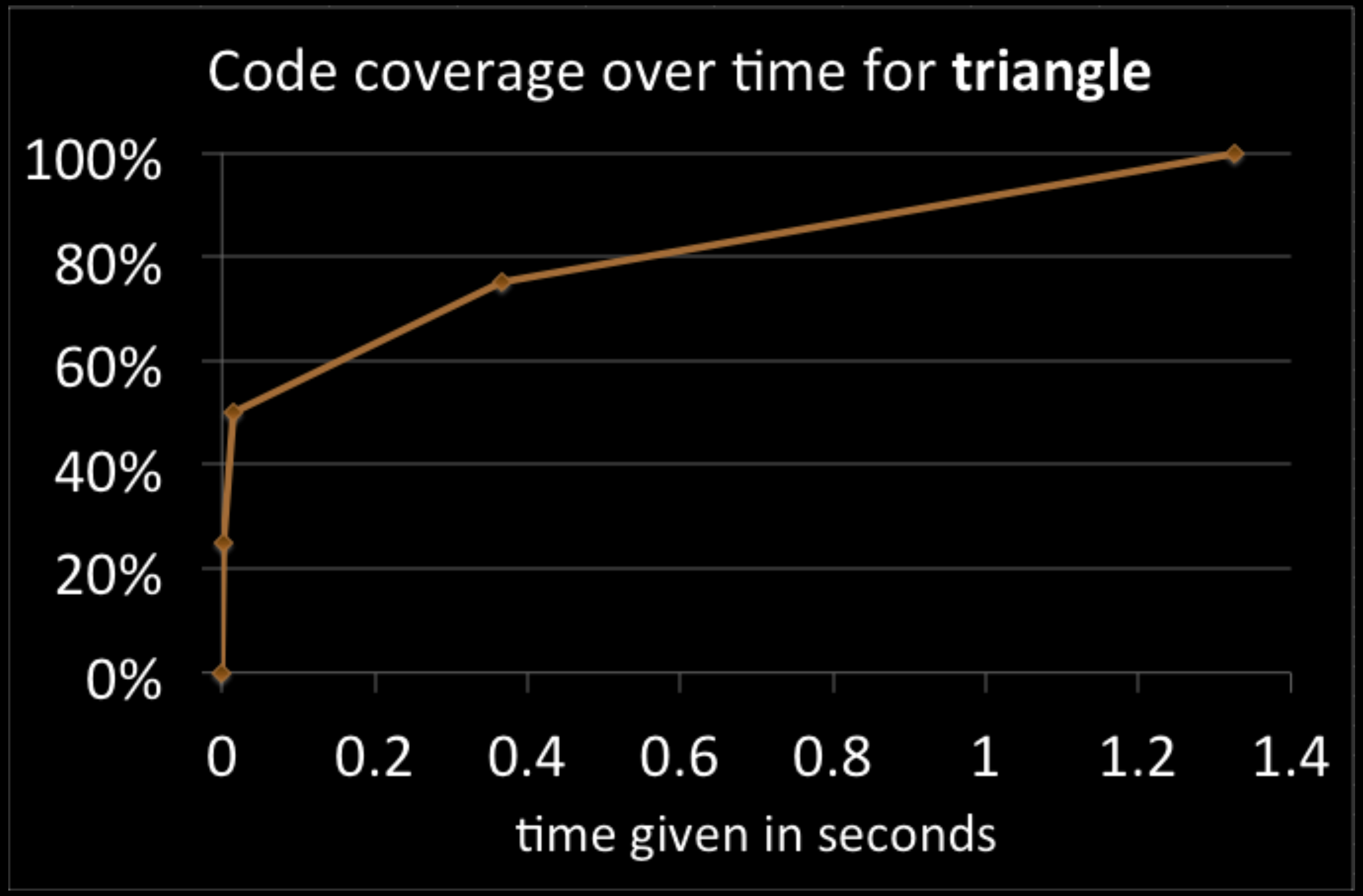
	pyprimes 0.1.1a	pyutilib.math 3.3	quixey challenge
Source Lines of Code	385	132	187
# Functions	38	14	17
Cyclomatic Complexity	147	43	80
Original	63%	63%	97%
Generated	54%	70%	91%

Results



Listing 10 Basic triangle test Python module

```
def triangle(a,b,c):
    if not (a + b) > c:
        return 'notvalid'
    if a == b == c:
        return 'equilateral'
    elif (a == b) or (b == c) or (a == c):
        return 'isoceles'
    else:
        return 'scalene'
```



Mean number of iterations to achieve 100% coverage = ***497***

□ Conclusion

Future work

- improve tool sophistication
- optimise using PyPy
- more comprehensive benchmarks

evandrix.github.com/Splat