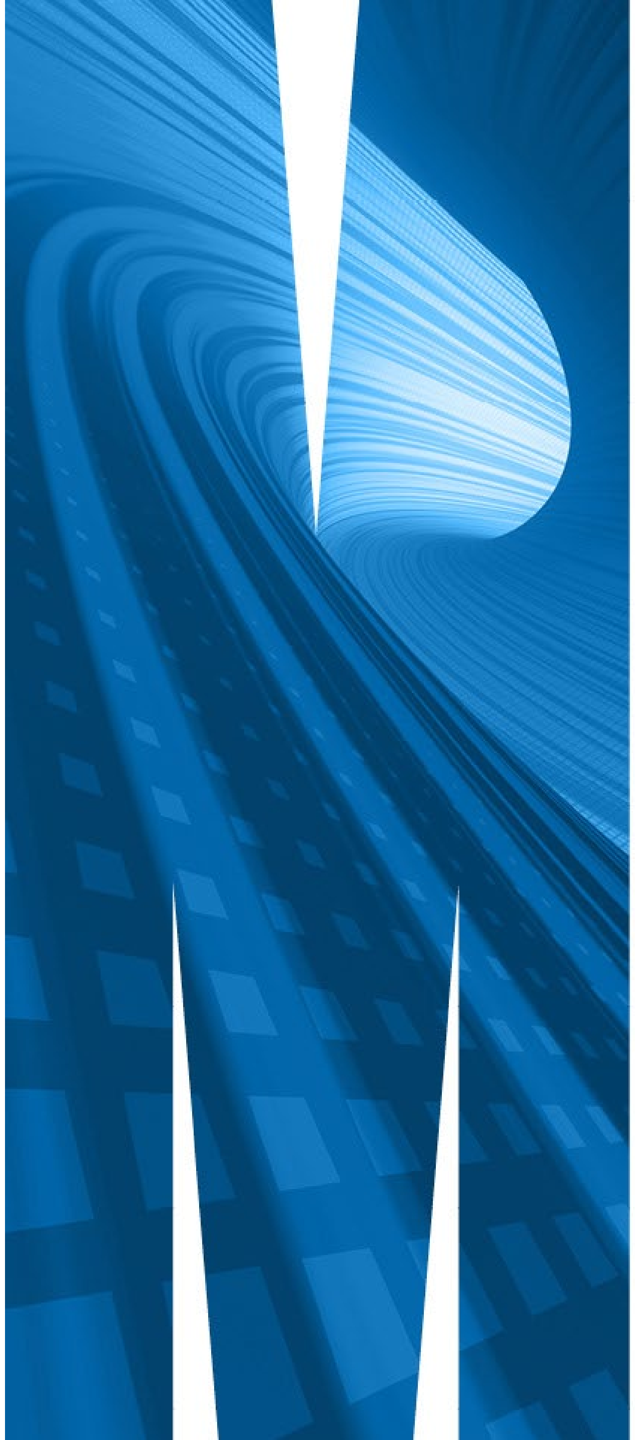
MONASH
University

FIT2107-Software Quality & Testing

Lecture 5 – Whitebox Testing



1st September, 2020

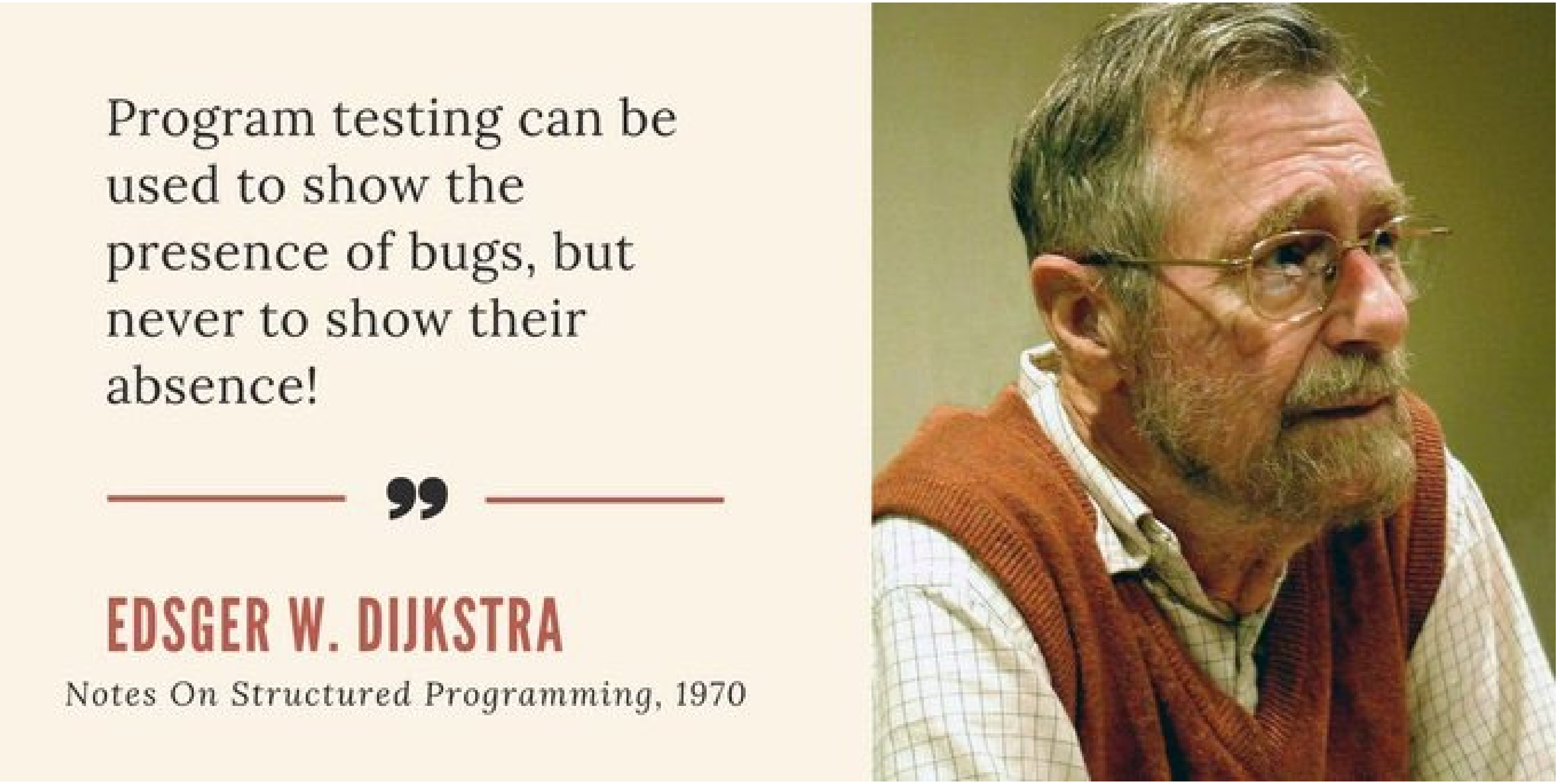
Dr Najam Nazar

Outline

- Testing Strategies
- Blackbox vs Whitebox
- Control Flow Graph
- What is Whitebox Testing?
- Line (Statement) Coverage
- Branch Coverage
- Condition Coverage

Announcements

- Assignment 1 due (4th September).
- Preliminary iSETU (Moodle).
- Quiz 3.
- Use workshop time to work on assignment.



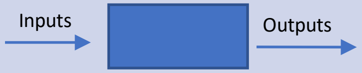
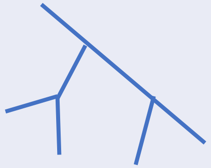
Program testing can be used to show the presence of bugs, but never to show their absence!

EDSGER W. DIJKSTRA

Notes On Structured Programming, 1970

Edsger Dijkstra (1930 – 2002) was a famous Dutch computer scientist and the inventor of Dijstra algorithm.

Basic Testing Strategies

Test Strategy	Tester’s View	Knowledge Source	Methods
Blackbox		Requirements	Equivalence Class
		Specifications	Boundary Value Analysis
		Domain Knowledge	Category Partitioning
Whitebox		Code Structure	Statement Coverage
		Code Graphs	Branch Coverage
		Cyclomatic Complexity	Condition Coverage

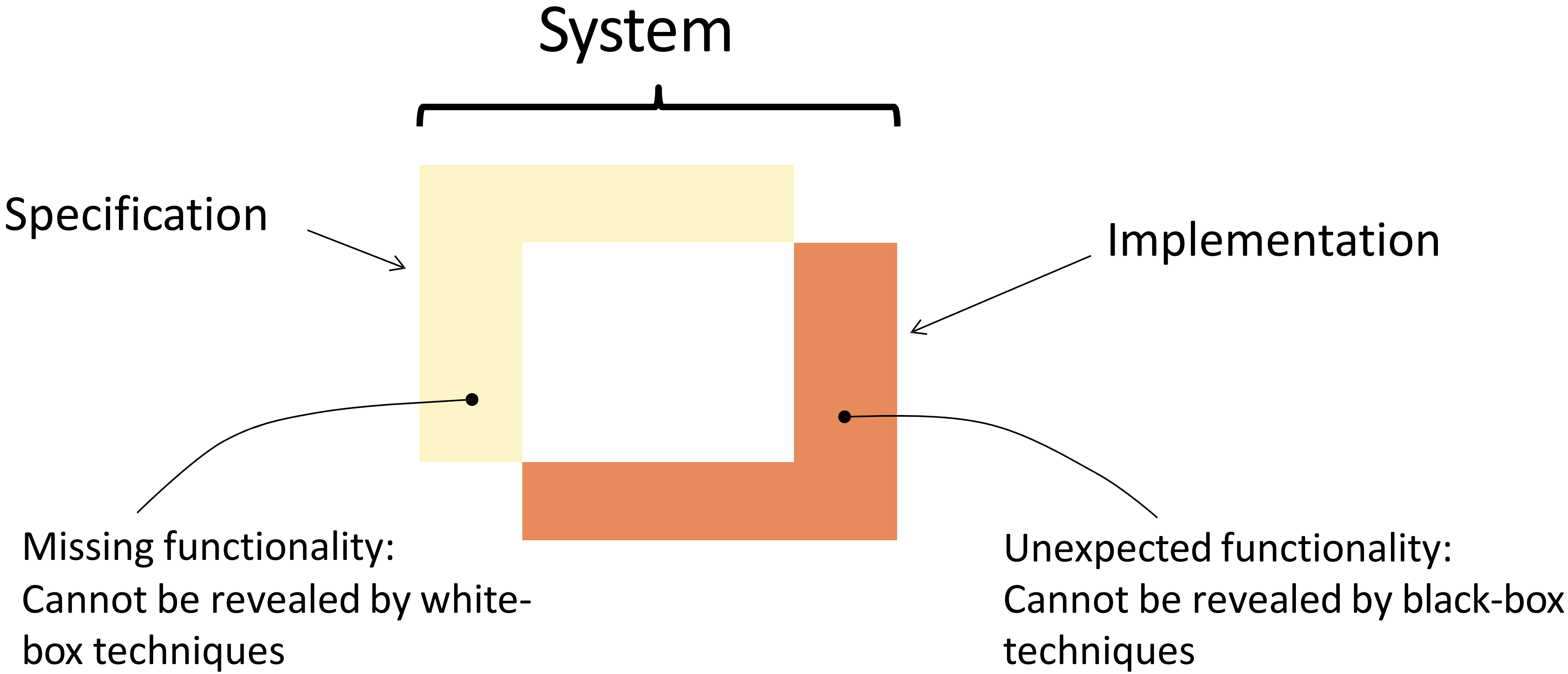
Blackbox vs Whitebox

- **External/user view:**
  - Check conformance with specification (=verification)
- **Abstraction from details:**
  - Source code not needed
- **Scales up:**
  - Different techniques at different levels of granularity

- Internal/developer view:
  - Allows tester to be confident about test coverage
- Based on control or data flow:
  - Easier debugging
- Does not scale up:
  - Mostly applicable at unit and integration testing levels

USE BOTH!

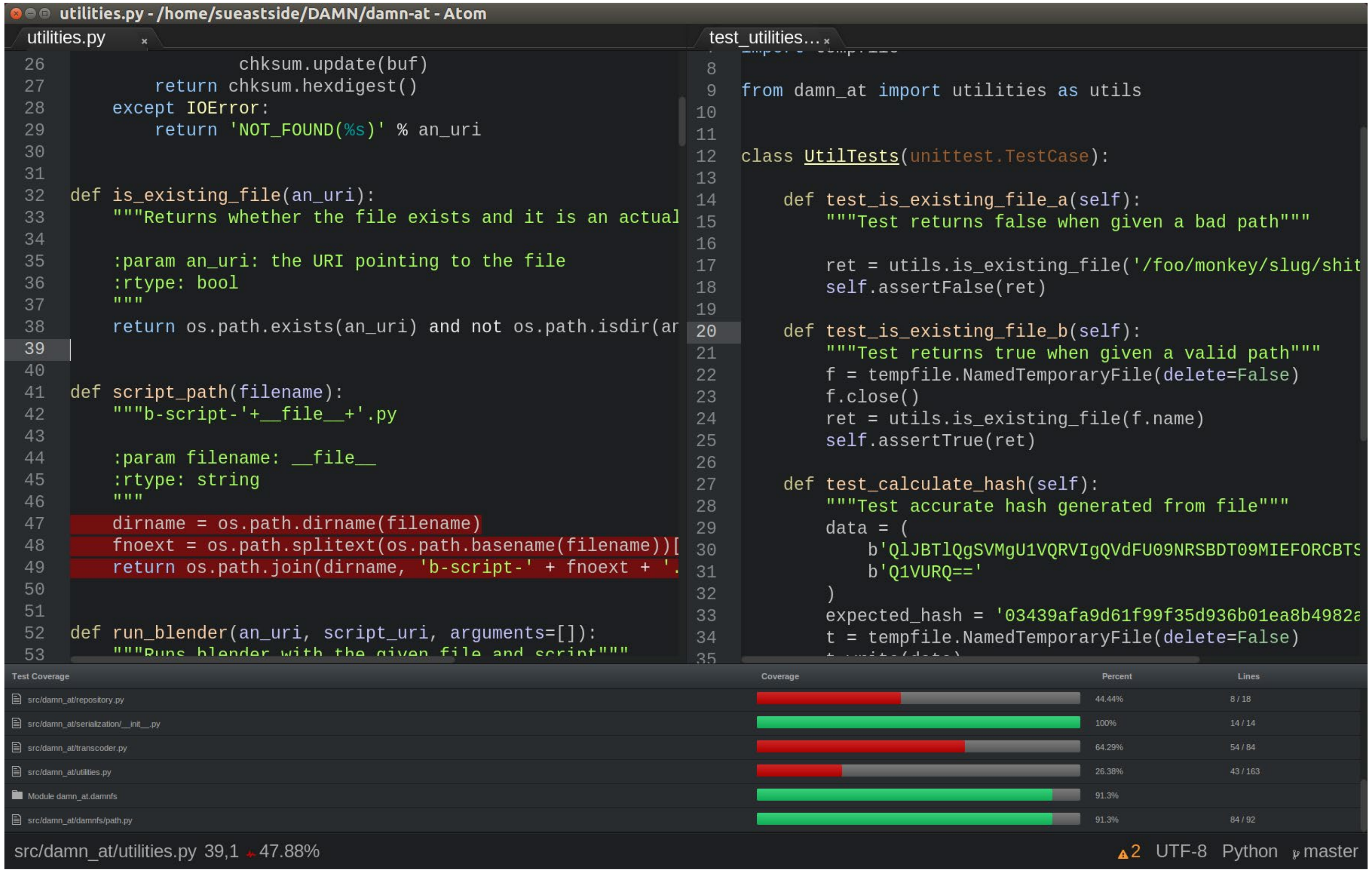
Blackbox vs Whitebox



Missing functionality: Cannot be revealed by white-box techniques

Unexpected functionality: Cannot be revealed by black-box techniques

In Practice – Software View



utilities.py - /home/sueastside/DAMN/damn-at - Atom

utilities.py

```
                chksum.update(buf)
            return chksum.hexdigest()
        except IOError:
            return 'NOT_FOUND(%s)' % an_uri


def is_existing_file(an_uri):
    """Returns whether the file exists and it is an actual

    :param an_uri: the URI pointing to the file
    :rtype: bool
    """
    return os.path.exists(an_uri) and not os.path.isdir(ar


def script_path(filename):
    """b-script-'+__file__+'.py

    :param filename: __file__
    :rtype: string
    """
    dirname = os.path.dirname(filename)
    fnoext = os.path.splitext(os.path.basename(filename))[
    return os.path.join(dirname, 'b-script-' + fnoext + '.


def run_blender(an_uri, script_uri, arguments=[]):
    """Runs blender with the given file and script"""
```

test_utilities...

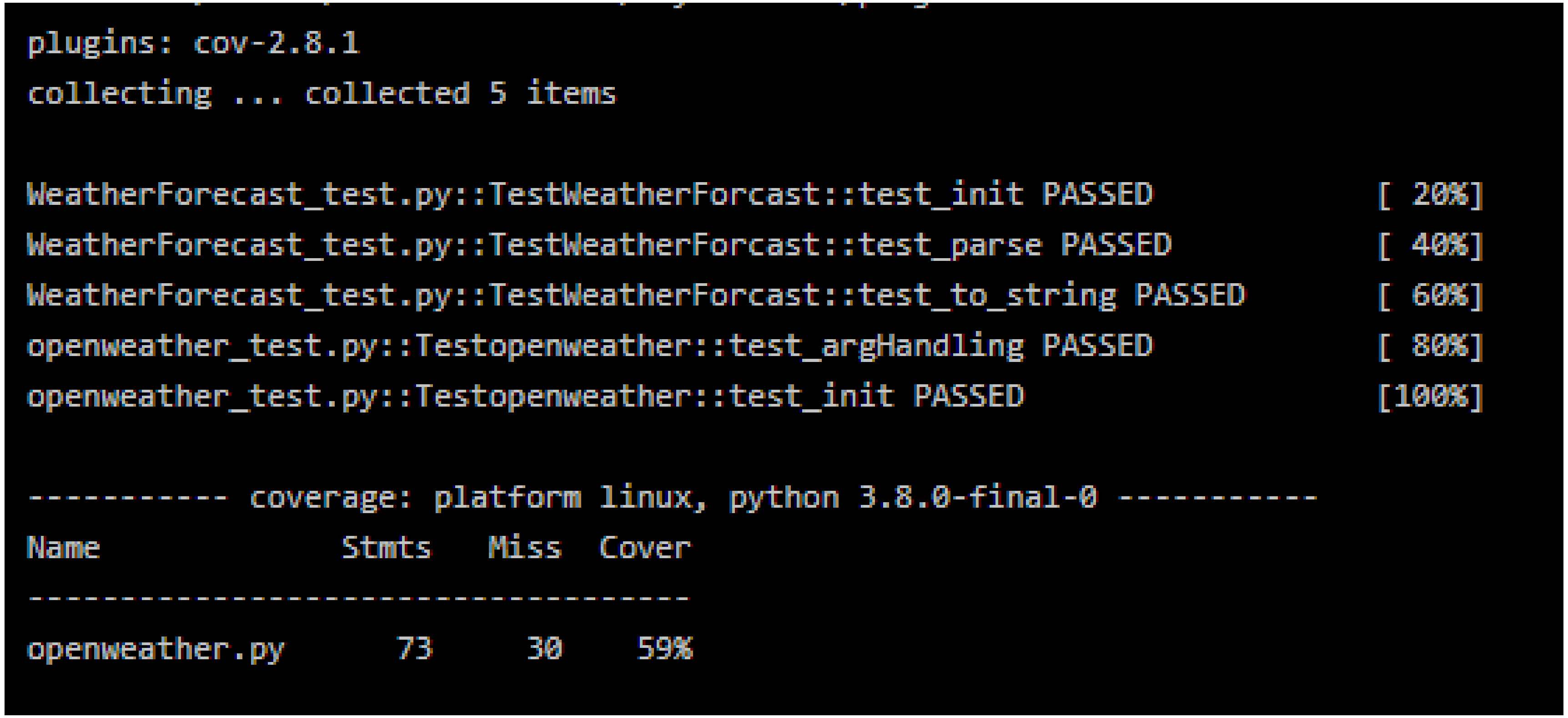
```
import tempfile

from damn_at import utilities as utils


class UtilTests(unittest.TestCase):

    def test_is_existing_file_a(self):
        """Test returns false when given a bad path"""

        ret = utils.is_existing_file('/foo/monkey/slug/shit
        self.assertFalse(ret)

    def test_is_existing_file_b(self):
        """Test returns true when given a valid path"""
        f = tempfile.NamedTemporaryFile(delete=False)
        f.close()
        ret = utils.is_existing_file(f.name)
        self.assertTrue(ret)

    def test_calculate_hash(self):
        """Test accurate hash generated from file"""
        data = (
            b'QlJBTlQgSVMgU1VQRVIgQVdFU09NRSBDT09MIEFORCBTS
            b'Q1VURQ=='
        )
        expected_hash = '03439afa9d61f99f35d936b01ea8b4982a
        t = tempfile.NamedTemporaryFile(delete=False)
        t.write(data)
```

Test Coverage	Coverage	Percent	Lines
src/damn_at/repository.py	🟥	44.44%	8 / 18
src/damn_at/serialization/__init__.py	🟩	100%	14 / 14
src/damn_at/transcoder.py	🟥	64.29%	54 / 84
src/damn_at/utilities.py	🟥	26.38%	43 / 163
Module damn_at.damnfs	🟩	91.3%	
src/damn_at/damnfs/path.py	🟩	91.3%	84 / 92

src/damn_at/utilities.py 39,1 47.88%   ▲2 UTF-8 Python master

https://atom.io/packages/python-coverage

In Practice – Git View

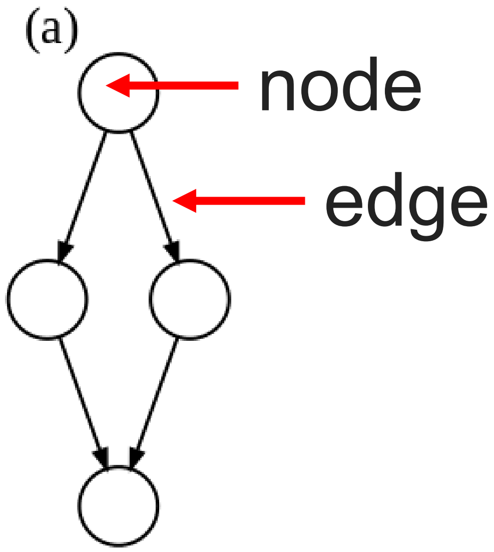


```
plugins: cov-2.8.1
collecting ... collected 5 items

WeatherForecast_test.py::TestWeatherForcast::test_init PASSED            [ 20%]
WeatherForecast_test.py::TestWeatherForcast::test_parse PASSED           [ 40%]
WeatherForecast_test.py::TestWeatherForcast::test_to_string PASSED       [ 60%]
openweather_test.py::Testopenweather::test_argHandling PASSED            [ 80%]
openweather_test.py::Testopenweather::test_init PASSED                   [100%]

----------- coverage: platform linux, python 3.8.0-final-0 -----------
Name              Stmts   Miss  Cover
------------------------------------
openweather.py       73     30    59%
```

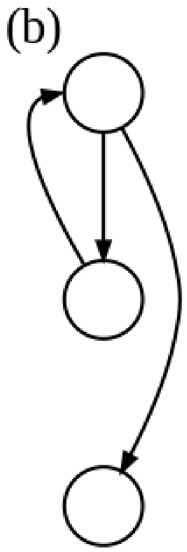
Theory: Control Flow Graphs (CFG)

- Structurally, a path is a sequence of statements in a program unit.
- Semantically, it is an execution instance of the unit.
- For a given set of input data, the program unit executes a certain path.
- CFG is a graph representation of all paths that might be traversed through a program during its execution (nodes and directed edges).
- Each node is a sequence of statements
- Each edge is a potential path between two statements.
    - Label edges with conditions for edge to be taken

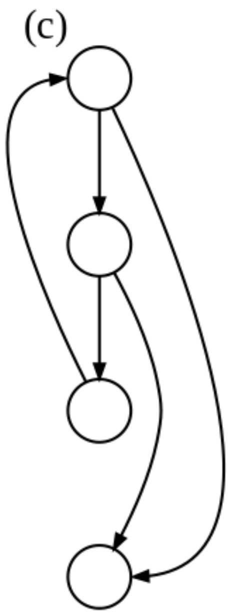
CFGs



if-then-else



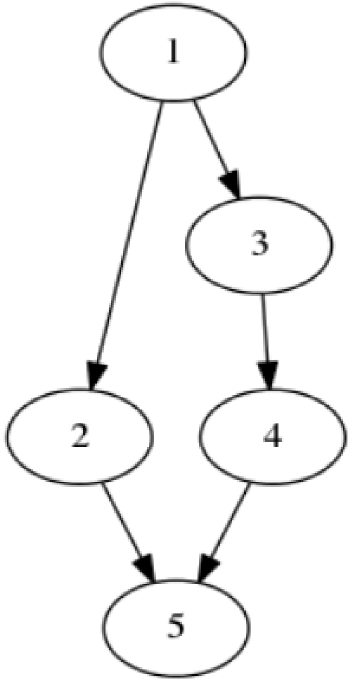
a while loop



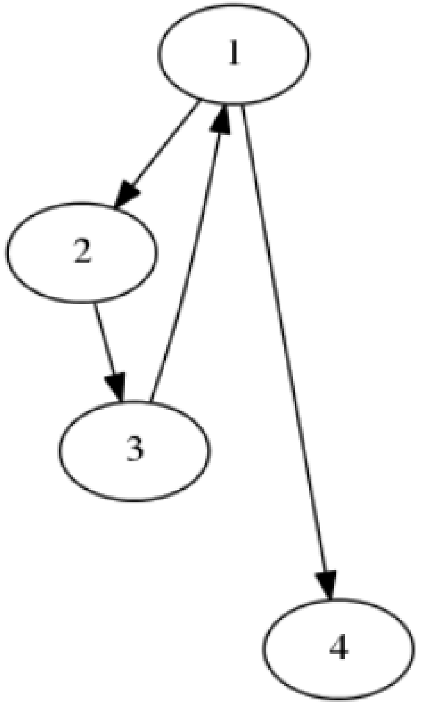
while loop + if…break

CFG – More Examples

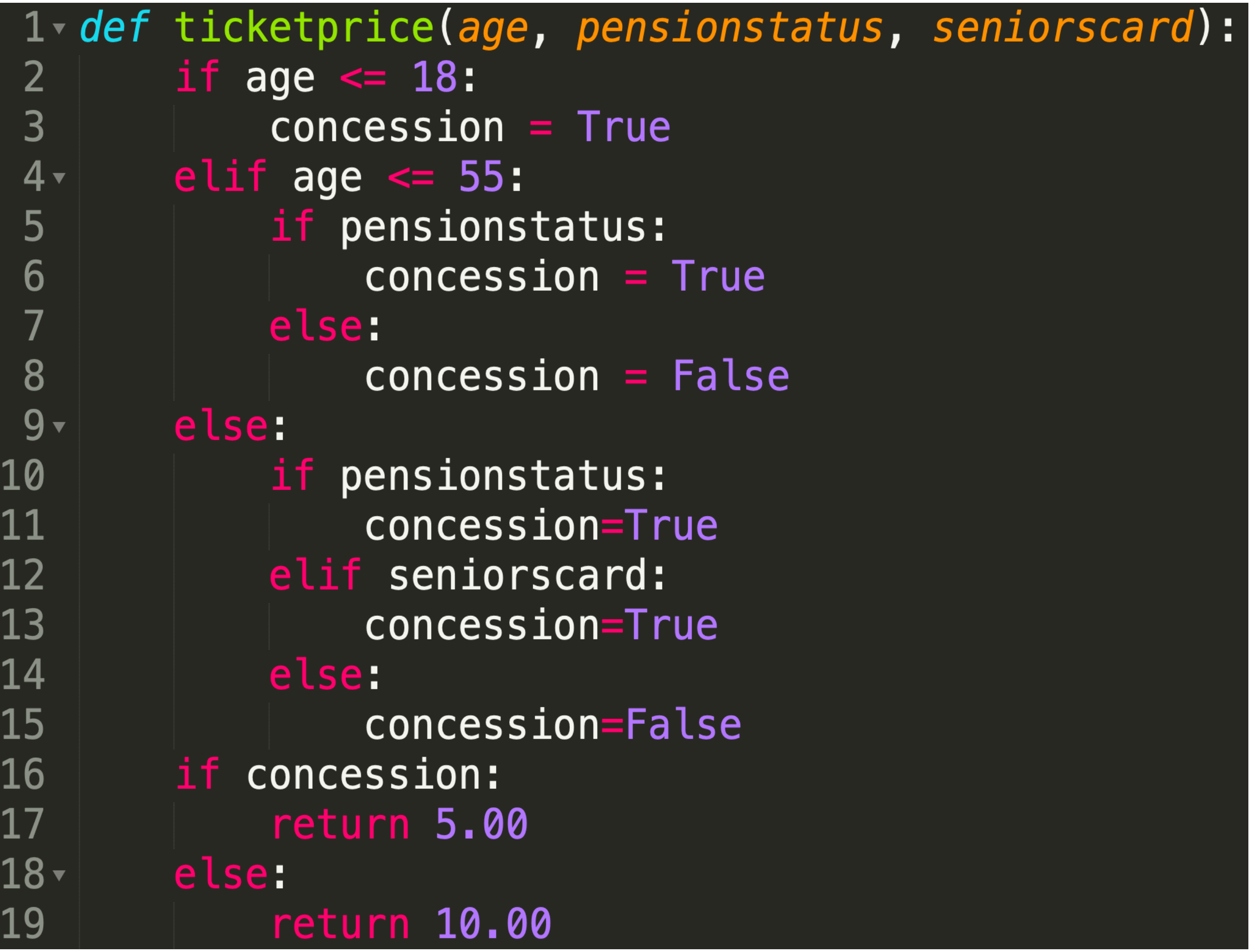
```
1      if CONDITION:
2          do_this()
3      else:
4          do_that()
5      continue_doing_other_things()
```



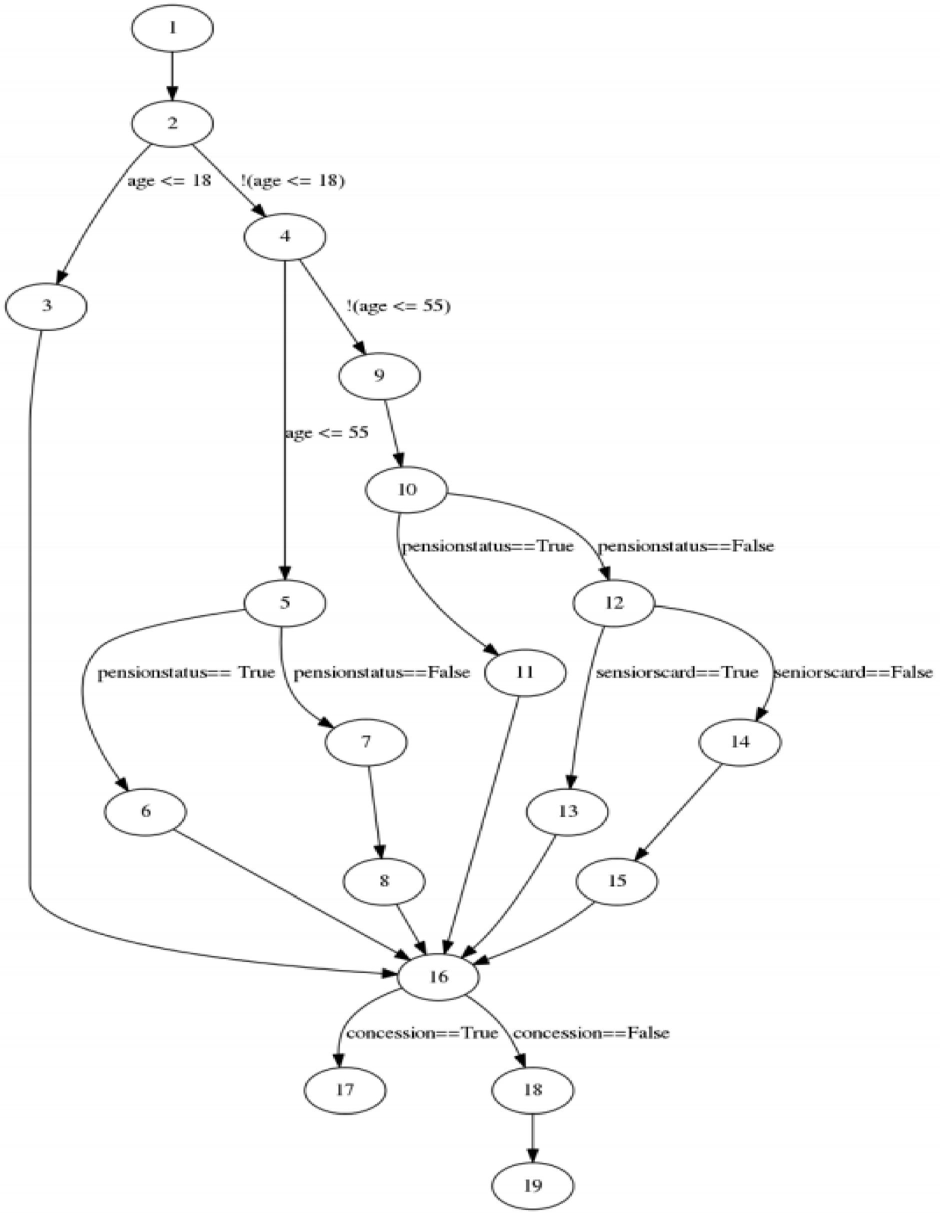
```
1      while x < 10:
2          print x
3          x+=1
4      print "done"
```



CFG – Bigger Example



```python
def ticketprice(age, pensionstatus, seniorscard):
    if age <= 18:
        concession = True
    elif age <= 55:
        if pensionstatus:
            concession = True
        else:
            concession = False
    else:
        if pensionstatus:
            concession=True
        elif seniorscard:
            concession=True
        else:
            concession=False
    if concession:
        return 5.00
    else:
        return 10.00
```

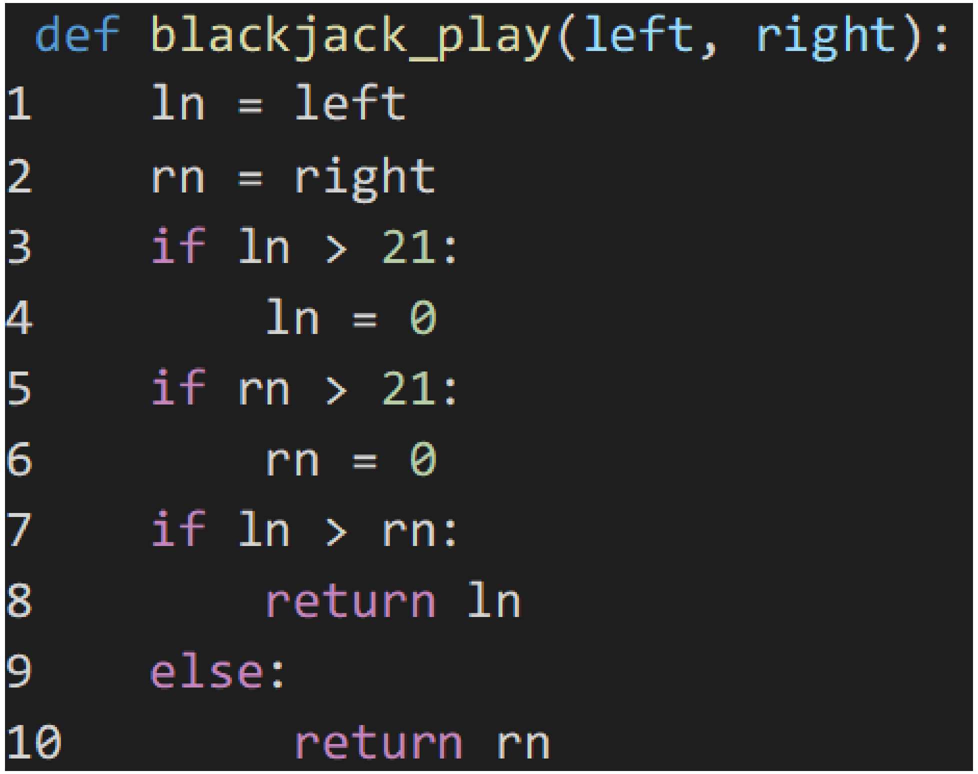


MONASH University

Whitebox Testing

- Whitebox testing checks the internals of the programme.
  - Also called structural testing
- Systematically cover all the behaviour of the software as it exists, and not solely on the specification.
- Coverage is the amount (or percentage) of code that is exercised by the tests .

Example: Blackjack Game



```
 def blackjack_play(left, right):
1    ln = left
2    rn = right
3    if ln > 21:
4        ln = 0
5    if rn > 21:
6        rn = 0
7    if ln > rn:
8        return ln
9    else:
10        return rn
```

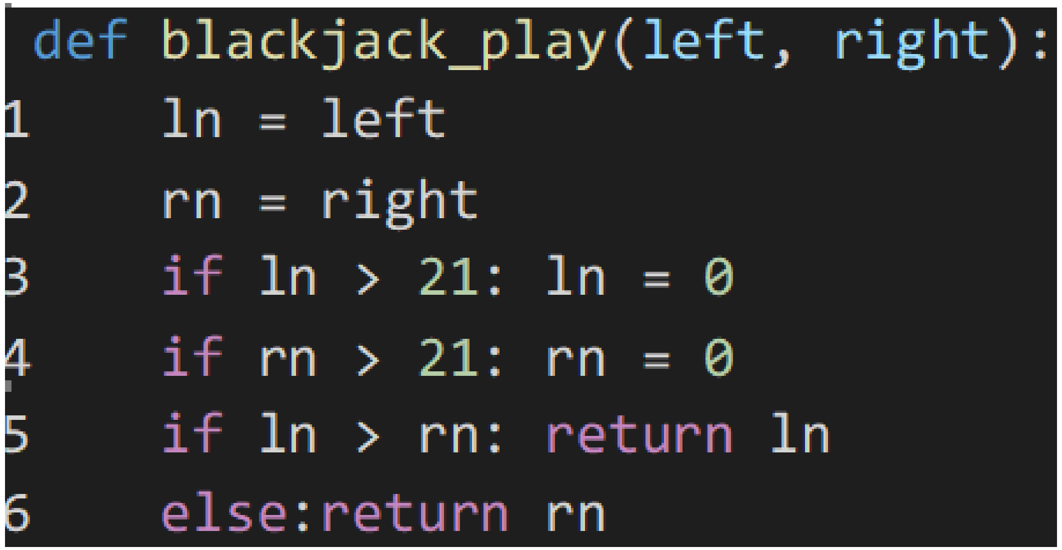
- Let’s say we pass blackjack_play(30,30)
- The programme will execute all lines except the line 8.
- coverage is 90% (9/10)
- Let’s say we pass blackjack_play(10,9)
- This will execute line 8 so the coverage is 100%.
- So we can say inputs {30,30} & {10,9} gives 100% coverage (2 test cases)
  - T1= {30, 30}
  - T2 = {10, 9}

Line Coverage

- Line (Statement) Coverage requires every possible statement in the code to be tested.
    - Minimum number of tests that can cover all lines.

$$linecoverage = \frac{\#\ of\ lines\ covered}{Total\ \#\ of\ lines} * 100$$

Problem with line coverage?



```python
def blackjack_play(left, right):
    ln = left
    rn = right
    if ln > 21: ln = 0
    if rn > 21: rn = 0
    if ln > rn: return ln
    else:return rn
```

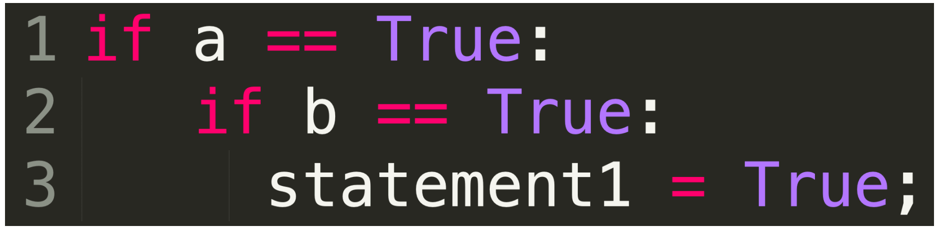
- Counting the covered lines is not always a good way of calculating the coverage.
- The amount of lines in a piece of code is heavily dependent on the programmer that writes the code.
- Using the same inputs {30,30} it actually covers all lines which is not the ideal representation of the coverage.
- Is having more lines better?

Branch (Decision) Coverage

- Branch Coverage requires every possible branch (i.e., if-else, other conditional loops) in the code to be tested.
- Whenever you have a decision block there are two possible outcomes, true and false.
- a = true , b = true will give 100% coverage.

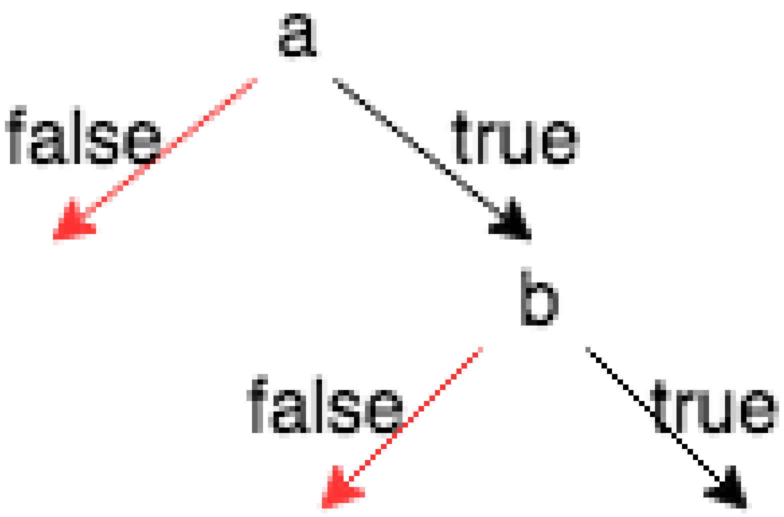
$$branchcoverage = \frac{\#\ of\ executed\ branches}{Total\ \#\ of\ branches} * 100$$

Branch (Decision) Coverage



```
if a == True:
   if b == True:
     statement1 = True;
```

- What are the missing branches?
- (a=true,b=false),
  (a=false,b=true),
  (a=false,b=false)
- 100% line coverage does not imply 100% branch coverage.
- 100% branch coverage implies 100% line coverage.



(Basic) Condition Coverage

- Branch coverage gives two branches for each decision.
- When branches become more complicated it contains more decisions
  - a > 10 && b < 20 && c < 10
- So branch coverage is not enough to test all possible cases
  - T1 (a=20, b=10, c=5)
  - T2 (a=5, b=10, c=5)
  - T3 (a=20, b=30, c=5)
- Rule: Conditions are tested separately and not the “big decision block”

$$conditioncoverage = \frac{conditions\ outcome\ covered}{Total\ \#\ of\ codition\ outcomes} * 100$$

Summary

- Blackbox testing uses requirements as a basis to devise tests.
- Whitebox testing needs criteria to figure out tests.
  - Criteria is coverage that is the code is covered or not?
- Line coverage is minimum number of tests that cover all lines.
- Branch coverage is that all branches or decisions should be satisfied.
- Branch gets complex when there are many conditions thus, we need a condition coverage.
- And
  - CFG is a graphical representation of a code using nodes and edges.

QUESTIONS???