

# Software Engineering (Part 3)

Copyright © 2026 by  
Robert M. Dondero, Ph.D.  
Princeton University

# Objectives

- We will cover these software engineering topics:

Stages of SW dev

How to order the stages

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- Process models

You're reasonably sure that your code is bug-free. What's next?

# Agenda

- Requirements analysis
- Design
- Implementation
- Debugging
- **Testing**
- Evaluation
- Maintenance
- Process models

# Testing

- **Debugging:** How can I **fix** the system?
- *Testing*: How can I **break** the system?

# Testing: Heuristic 1

- Do *internal testing*
  - Design your code to test itself
  - Done by *programmers*

# Testing: Heuristic 1

- Internal testing techniques
  - Check for function/method failures
  - Validate parameters
  - Check invariants
  - Leave testing code intact!!!

# Aside: Asserts

## C: `assert` macro

```
assert(count >= 0);
```

Essentially same as:

```
if (count < 0)
{
    fprintf(stderr,
            "assertion failed: (count >= 0),");
    fprintf(stderr,
            "function XXX, file YY, line ZZZ.");
    exit(134);
}
```

Asserts are **enabled** by default; to **disable** asserts:

```
gcc -D NDEBUG somefile.c
```

# Aside: Asserts

## Python: `assert` statement

```
assert count >= 0, 'count must be >= 0'
```

Essentially same as:

```
if count < 0:  
    raise AssertionError('count must be >= 0')
```

Asserts are **enabled** by default; to **disable** asserts:

```
python -O somefile.py
```

# Aside: Asserts

**Java:** `assert` statement (since JDK 1.4)

```
assert count >= 0 : "count must be >= 0";
```

Essentially same as:

```
if (count < 0)
    throw new AssertionError("count must be >= 0");
```

Asserts are **disabled** by default; to **enable** asserts:

```
java -ea SomeFile.java
```

# Aside: Asserts

## JavaScript (browsers):

`console.assert` function

```
console.assert(count >= 0, 'count must be >= 0');
```

Essentially same as:

```
if (count < 0)
  console.error('count must be >= 0');
```

Doesn't affect control flow!

Can be disabled only by build tools (Webpack, Vite, Babel, ...)

# Aside: Asserts

## JavaScript (Node.js): `assert` function

```
const assert = require('assert-plus');  
...  
assert(count >= 0, 'count must be >= 0');
```

Essentially same as:

```
if (count < 0)  
  throw new AssertionError('count must be >= 0');
```

Asserts are **enabled** by default; to **disable** asserts:

```
export NODE_DEBUG=1
```

# Aside: Asserts

- Assert controversy:
  - Enable or disable asserts in production code?

# Testing: Heuristic 1

- **Example: Assignment 2**
  - regoverviews.py:

```
def response_is_valid(response):  
  
    if not isinstance(response, list):  
        print_error('The response must be a list')  
        return False  
    if len(response) != 2:  
        print_error('The list must have 2 elements')  
        return False  
    successful = response[0]  
    data = response[1]  
    if not isinstance(successful, bool):  
        print_error('The 1st element must be a bool')  
        return False
```

Continued on next slide

# Testing: Heuristic 1

Continued from previous slide

```
if successful:
    if not isinstance(data, list):
        print_error('The 2nd element must be a list')
        return False
    for row in data:
        if not isinstance(row, dict):
            print_error('Each overview must be a dict')
            return False
        if len(row) != 5:
            print_error('Each dict must have have 5 bindings')
            return False
        if 'classid' not in row:
            print_error('Each dict msut contain key classid')
            return False
        if not isinstance(row['classid'], int):
            print_error('Each classid must be an int')
            return False
        if 'dept' not in row:
            print_error('Each dict must contain key dept')
            return False
        if not isinstance(row['dept'], str):
            print_error('Each dept must be a str')
            return False
```

Continued on next slide

# Testing: Heuristic 1

Continued from previous slide

```
    if 'coursenum' not in row:
        print_error('Each dict must contain key coursenum')
        return False
    if not isinstance(row['coursenum'], str):
        print_error('Each coursenum must be a str')
        return False
    if 'area' not in row:
        print_error('Each dict must contain key area')
        return False
    if not isinstance(row['area'], str):
        print_error('Each area must be a str')
        return False
    if 'title' not in row:
        print_error('Each dict must contain key title')
        return False
    if not isinstance(row['title'], str):
        print_error('Each title must be a str')
        return False
else:
    if not isinstance(data, str):
        print_error('The 2nd element must be a str')
        return False

return True
```

# Testing: Heuristic 1

Then, at a higher level:

```
...  
assert response_is_valid(response), \  
    'Response must be valid'  
...
```

# Testing: Heuristic 2

- Do *external testing*
  - Design external code or data to test your code

# Testing: Heuristic 2.1

- **Do *white box external testing***
  - Maybe better term: **clear box** external testing
  - External testing with knowledge of structure of tested code
  - Done by **programmers**

# Testing: Heuristic 2.1

- White box external testing techniques
  - ***Boundary (corner case) testing***
    - Testing with **input** values at, just below, and just above limits of **input** domain
    - Testing with input values causing **output** values to be at, just below, and just above the limits of the **output** domain

*Glossary of Computerized System and Software Development Terminology*

# Testing: Heuristic 2.1

- White box external testing techniques
  - *Path testing*
    - Testing to make sure each **logical path** is followed at least once
  - *Statement (coverage) testing*
    - Testing to make sure each **statement** is executed at least once

# Aside: Statement Testing Tools

Tools for statement/coverage testing:

Language	Statement Testing Tool
Python	<i>coverage</i> *
Java	<i>JaCoCo</i>
C	<i>gcov</i>
JavaScript (Node.js)	<i>nyc/Istanbul</i> ** <i>Jest/Istanbul</i> **
JavaScript (browser)	<i>Chrome Coverage</i>

\* Described in the following slides

\*\* See me if you want an example

# Aside: Statement Testing Tools

- Python **module** testing
  - Recall **Fraction/**
    - **euclid.py**
    - **fraction.py**
    - **fractionclient.py**

# Aside: Statement Testing Tools

```
$ python -m coverage run -p fractionclient.py
Numerator 1: 1
Denominator 1: 2
Numerator 2: 3
Denominator 2: 4
frac1: 1/2
frac2: 3/4
frac1 does not equal frac2
frac1 is less than frac2
frac1 is less than or equal to frac2
-frac1: -1/2
frac1 + frac2: 5/4
frac1 - frac2: -1/4
frac1 * frac2: 3/8
frac1 / frac2: 2/3
$
```

# Aside: Statement Testing Tools

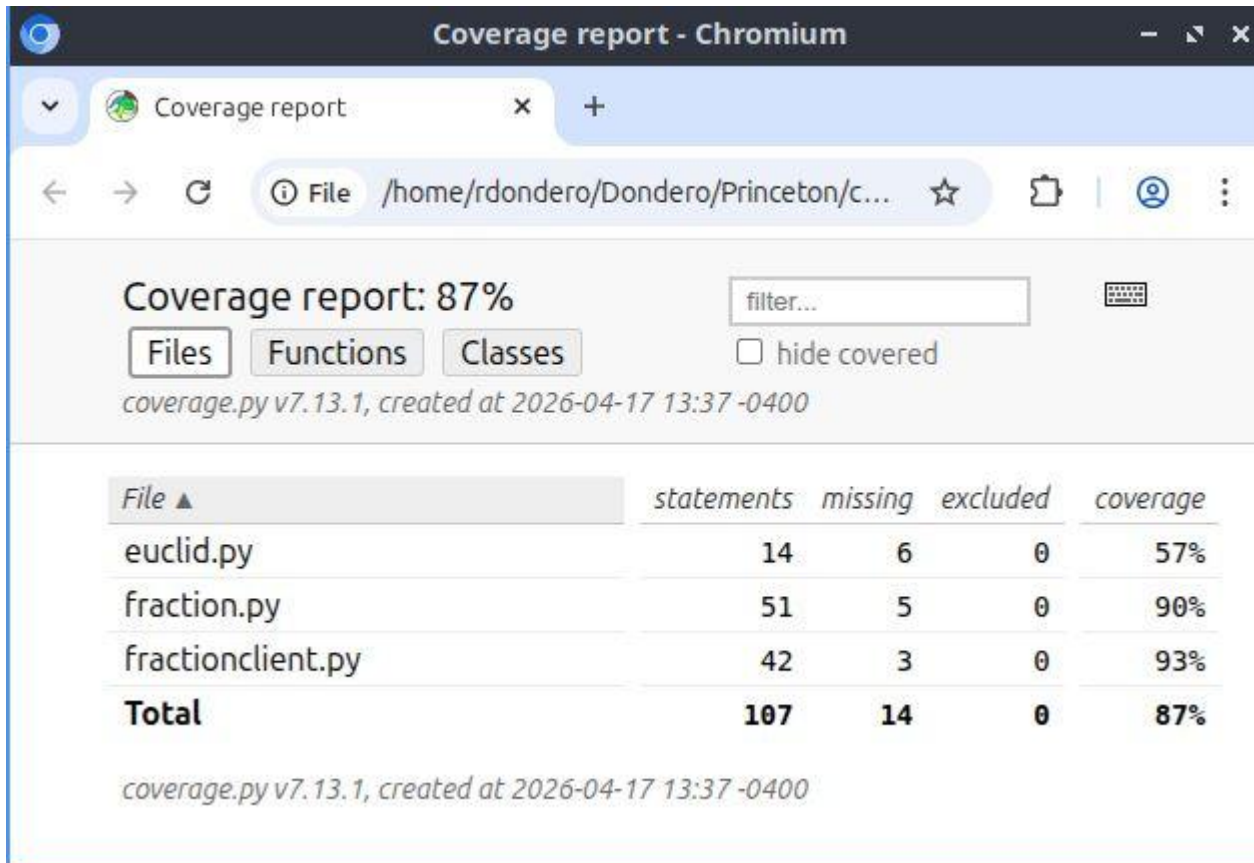
```
$ python -m coverage run -p fractionclient.py
Numerator 1: 3
Denominator 1: 4
Numerator 2: 1
Denominator 2: 2
frac1: 3/4
frac2: 1/2
frac1 does not equal frac2
frac1 is greater than frac2
frac1 is greater than or equal to frac2
-frac1: -3/4
frac1 + frac2: 5/4
frac1 - frac2: 1/4
frac1 * frac2: 3/8
frac1 / frac2: 3/2
$
```

# Aside: Statement Testing Tools

```
$ python -m coverage combine
Combined data file
    .coverage.rdondero-latitudee6430s.pid4800.XmWXSufx.HRcim92nc10h
Combined data file
    .coverage.rdondero-latitudee6430s.pid4805.X1f0Epbx.H0qhG6VTeC1h
$ python -m coverage html
Wrote HTML report to htmlcov/index.html
$
```

Then browse to [htmlcov/index.html](http://htmlcov/index.html)...

# Aside: Statement Testing Tools



Coverage report: 87%

filter...

hide covered

Files Functions Classes

*coverage.py v7.13.1, created at 2026-04-17 13:37 -0400*

File ▲	statements	missing	excluded	coverage
euclid.py	14	6	0	57%
fraction.py	51	5	0	90%
fractionclient.py	42	3	0	93%
<b>Total</b>	<b>107</b>	<b>14</b>	<b>0</b>	<b>87%</b>

*coverage.py v7.13.1, created at 2026-04-17 13:37 -0400*

# Aside: Statement Testing Tools

Coverage for euclid.py: 57% - Chromium

Coverage for euclid.py: 57%

14 statements 8 run 6 missing 0 excluded

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 13:37 -0400

```
1 #!/usr/bin/env python
2
3 #-----
4 # euclid.py
5 # Author: Bob Dondero
6 #-----
7
8 def gcd(i, j):
9
10     if (i == 0) and (j == 0):
11         raise ZeroDivisionError(
12             'gcd(i,j) is undefined if i and j are 0')
13     i = abs(i)
14     j = abs(j)
15     while j != 0: # Euclid's algorithm
16         i, j = j, i%j
17     return i
18
19 #-----
20
21 def lcm(i, j):
22
23     if (i == 0) or (j == 0):
24         raise ZeroDivisionError(
25             'lcm(i,j) is undefined if i or j is 0')
26     i = abs(i)
27     j = abs(j)
28     return (i // gcd(i, j)) * j
```

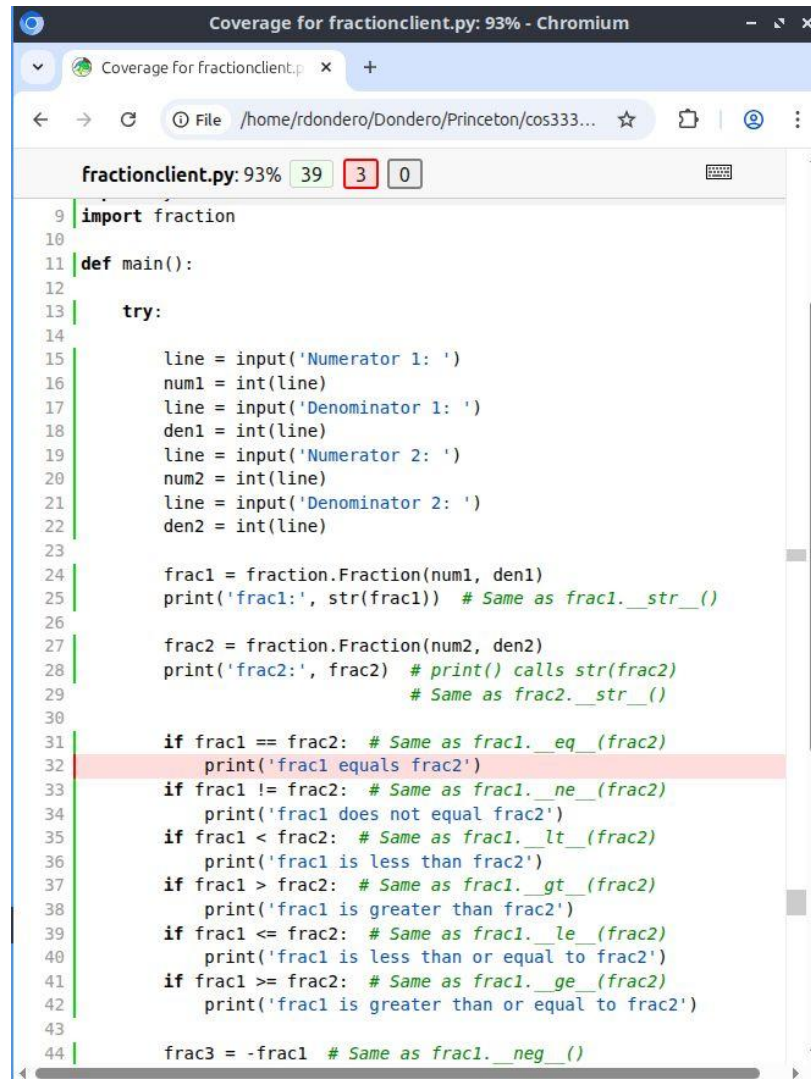
« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 13:37 -0400

# Aside: Statement Testing Tools

fraction.py: 90% 46 5 0

```
4 # fraction.py
5 # Author: Bob Dondero
6 #-----
7
8 import euclid
9
10 #-----
11
12 class Fraction:
13
14     def __init__(self, num=0, den=1):
15         if den == 0:
16             raise ZeroDivisionError('Denominator cannot be 0')
17         self._num = num
18         self._den = den
19         self._normalize()
20
21     def _normalize(self):
22         if self._den < 0:
23             self._num *= -1
24             self._den *= -1
25         if self._num == 0:
26             self._den = 1
27         else:
28             gcden = euclid.gcd(self._num, self._den)
29             self._num //= gcden
30             self._den //= gcden
31
32     def __str__(self):
33         if self._den == 1:
34             return str(self._num)
35         return '%d/%d' % (self._num, self._den)
36
37     def __eq__(self, other):
38         return (self._num == other._num) and (self._den == other._den)
39
40     def __ne__(self, other):
```

# Aside: Statement Testing Tools



fractionclient.py: 93% 39 3 0

```
9 import fraction
10
11 def main():
12
13     try:
14
15         line = input('Numerator 1: ')
16         num1 = int(line)
17         line = input('Denominator 1: ')
18         den1 = int(line)
19         line = input('Numerator 2: ')
20         num2 = int(line)
21         line = input('Denominator 2: ')
22         den2 = int(line)
23
24         frac1 = fraction.Fraction(num1, den1)
25         print('frac1:', str(frac1)) # Same as frac1.__str__()
26
27         frac2 = fraction.Fraction(num2, den2)
28         print('frac2:', frac2) # print() calls str(frac2)
29                               # Same as frac2.__str__()
30
31         if frac1 == frac2: # Same as frac1.__eq__(frac2)
32             print('frac1 equals frac2')
33         if frac1 != frac2: # Same as frac1.__ne__(frac2)
34             print('frac1 does not equal frac2')
35         if frac1 < frac2: # Same as frac1.__lt__(frac2)
36             print('frac1 is less than frac2')
37         if frac1 > frac2: # Same as frac1.__gt__(frac2)
38             print('frac1 is greater than frac2')
39         if frac1 <= frac2: # Same as frac1.__le__(frac2)
40             print('frac1 is less than or equal to frac2')
41         if frac1 >= frac2: # Same as frac1.__ge__(frac2)
42             print('frac1 is greater than or equal to frac2')
43
44         frac3 = -frac1 # Same as frac1.__neg__()
```

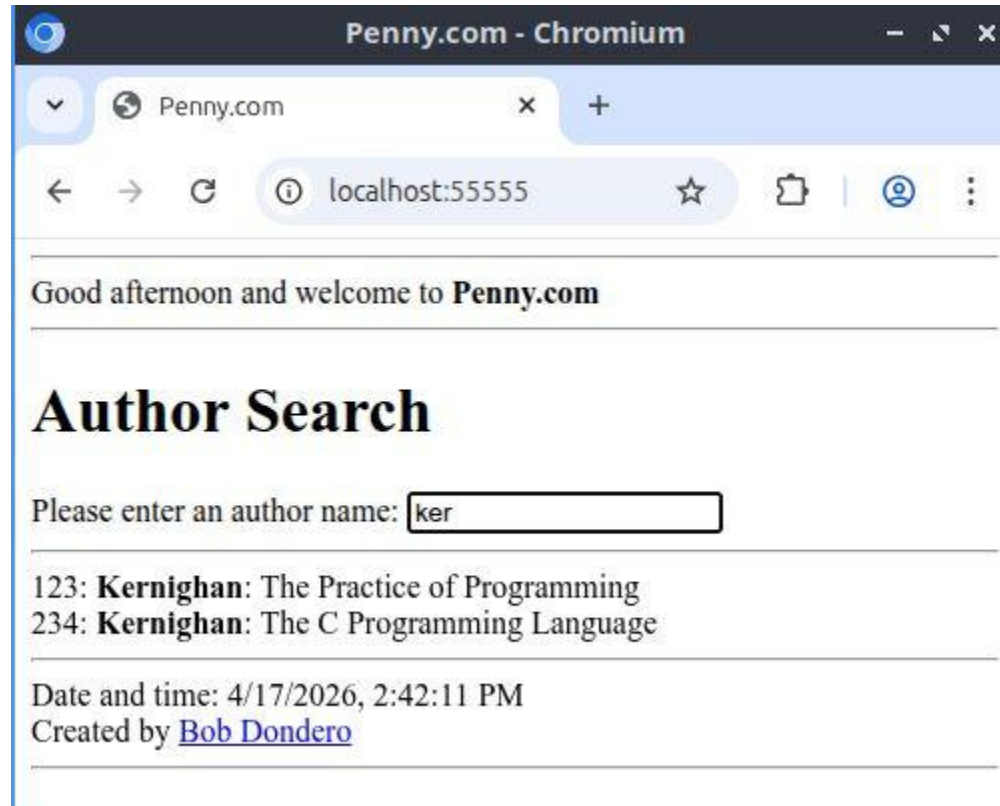
# Aside: Statement Testing Tools

- Python **web app** testing
  - See **PennyTest** app
    - **runserver.py**
    - **penny.sql**, **penny.sqlite**
    - **database.py**
    - **penny.py**
    - **index.html**

# Aside: Statement Testing Tools

```
$ python -m coverage run -p runserver.py 5555
* Serving Flask app 'penny'
* Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server
instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:55555
* Running on http://192.168.1.22:55555
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 457-747-552
```

# Aside: Statement Testing Tools

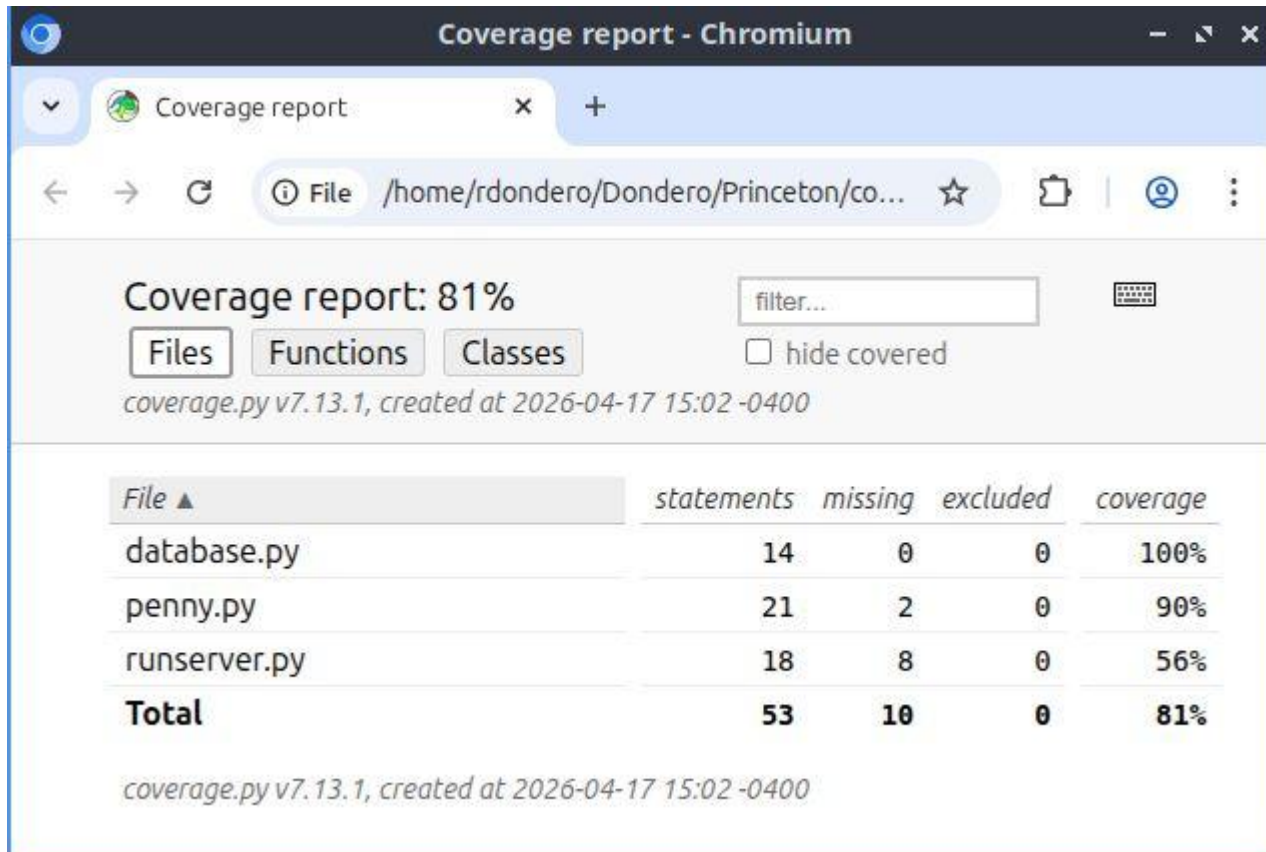


# Aside: Statement Testing Tools

```
$ python -m coverage combine  
Combined data file  
.coverage.rdondero-latitudee6430s.pid9985.XfaNCpcx.Hr66UMu8wAZh  
Combined data file  
.coverage.rdondero-latitudee6430s.pid9988.XjcZixRx.HiryNcoxZFyh  
$ python -m coverage html  
Wrote HTML report to htmlcov/index.html  
$
```

Then browse to [htmlcov/index.html](http://htmlcov/index.html)...

# Aside: Statement Testing Tools



Coverage report: 81%

filter...

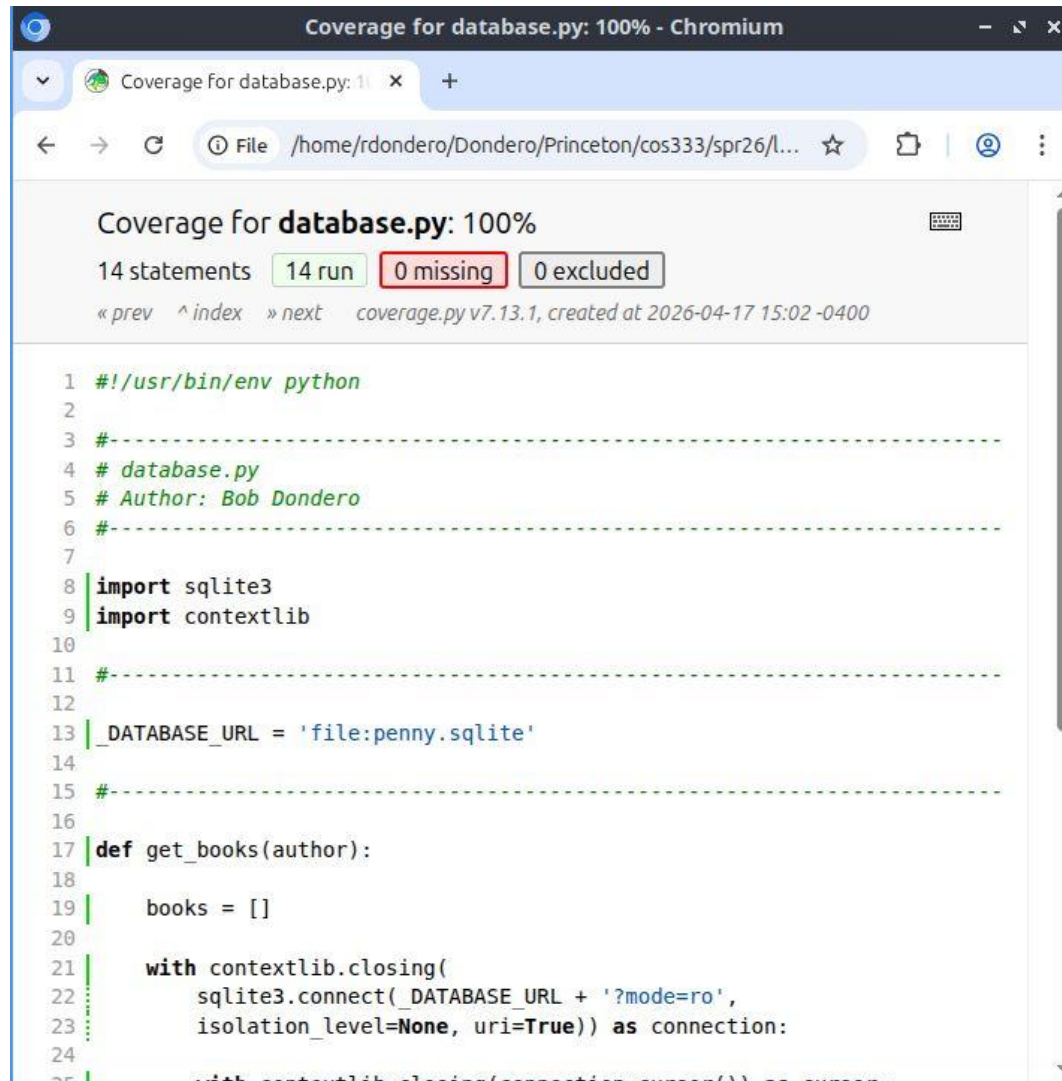
hide covered

coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

File ▲	statements	missing	excluded	coverage
database.py	14	0	0	100%
penny.py	21	2	0	90%
runserver.py	18	8	0	56%
<b>Total</b>	<b>53</b>	<b>10</b>	<b>0</b>	<b>81%</b>

coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

# Aside: Statement Testing Tools



Coverage for **database.py**: 100%

14 statements 14 run 0 missing 0 excluded

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

```
1  #!/usr/bin/env python
2
3  #-----
4  # database.py
5  # Author: Bob Dondero
6  #-----
7
8  import sqlite3
9  import contextlib
10
11 #-----
12
13 _DATABASE_URL = 'file:penny.sqlite'
14
15 #-----
16
17 def get_books(author):
18
19     books = []
20
21     with contextlib.closing(
22         sqlite3.connect(_DATABASE_URL + '?mode=ro',
23         isolation_level=None, uri=True)) as connection:
24
25     with contextlib.closing(connection.cursor()) as cursor:
```

# Aside: Statement Testing Tools

Coverage for penny.py: 90% - Chromium

Coverage for penny.py: 90%

21 statements 19 run 2 missing 0 excluded

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17

```
1 #!/usr/bin/env python
2
3 #-----
4 # penny.py
5 # Author: Bob Dondero
6 #-----
7
8 import json
9 import flask
10 import database
11
12 #-----
13
14 app = flask.Flask(__name__)
15
16 #-----
17
18 @app.route('/', methods=['GET'])
19 @app.route('/index', methods=['GET'])
20 def index():
21
22     return flask.send_file('index.html')
23
24 #-----
25
```

Coverage for penny.py: 90% - Chromium

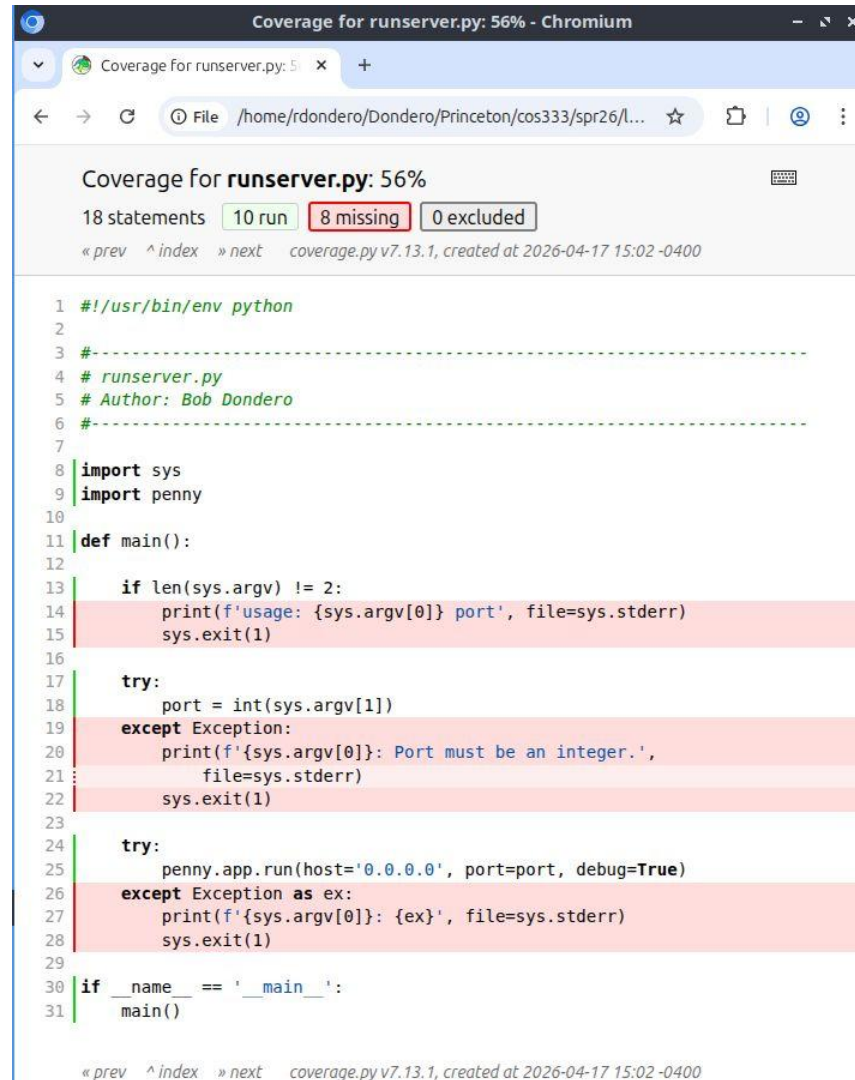
penny.py: 90% 19 2 0

```
18 @app.route('/', methods=['GET'])
19 @app.route('/index', methods=['GET'])
20 def index():
21
22     return flask.send_file('index.html')
23
24 #-----
25
26 @app.route('/searchresults', methods=['GET'])
27 def search_results():
28
29     author = flask.request.args.get('author')
30     if author is None:
31         author = ''
32         author = author.strip()
33
34     if author == '':
35         books = []
36     else:
37         books = database.get_books(author) # Exception handling omitted
38
39     json_doc = json.dumps(books)
40     response = flask.make_response(json_doc)
41     response.headers['Content-Type'] = 'application/json'
42     return response

```

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

# Aside: Statement Testing Tools



Coverage for runserver.py: 56%

18 statements 10 run 8 missing 0 excluded

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

```
1 #!/usr/bin/env python
2
3 #-----
4 # runserver.py
5 # Author: Bob Dondero
6 #-----
7
8 import sys
9 import penny
10
11 def main():
12
13     if len(sys.argv) != 2:
14         print(f'usage: {sys.argv[0]} port', file=sys.stderr)
15         sys.exit(1)
16
17     try:
18         port = int(sys.argv[1])
19     except Exception:
20         print(f'{sys.argv[0]}: Port must be an integer.',
21               file=sys.stderr)
22         sys.exit(1)
23
24     try:
25         penny.app.run(host='0.0.0.0', port=port, debug=True)
26     except Exception as ex:
27         print(f'{sys.argv[0]}: {ex}', file=sys.stderr)
28         sys.exit(1)
29
30 if __name__ == '__main__':
31     main()
```

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:02 -0400

# Testing: Heuristic 2.2

- **Do *black box external testing***
  - External testing without knowledge of structure of tested code
  - Done by *quality assurance (QA) engineers*

# Testing: Heuristic 2.2

- Black box external testing techniques
  - ***Use case testing***
    - Testing driven by use cases developed during design
  - ***Stress testing***
    - Testing with a large quantity of data
    - Testing with a large variety of (random?) data

# Aside: Testing Taxonomy

- Testing taxonomy
  - Internal testing
  - External testing
    - White box testing
      - Boundary testing
      - Path testing
      - Statement/coverage testing
    - Black box
      - Use case testing
      - Stress testing

# Testing: Heuristic 3

- Test incrementally
  - Use scaffolds and stubs
  - Do *regression testing*

# Testing: Heuristic 4

- Let debugging drive testing
  - Reactive mode
  - Proactive mode: do *fault injection*

# Testing: Heuristic 5

- Automate the testing

# Testing: Heuristic 5

- Kinds of automated testing:

	Unit Testing	System/Integration Testing
<b>What</b>	Test <b>modules</b> (classes, functions, methods)	Test <b>programs</b>
<b>When</b>	Often compose tests for module X <b>before</b> composing X	Typically compose tests for program X <b>after</b> composing X
<b>Who</b>	Often performed by a <b>single programmer</b> , or a small group	Often performed by a <b>QA organization</b>

# Aside: Automated Testing Tools

Language	Test Automation Tool
Python	<i>unittest</i> (formerly <i>PyUnit</i> )* <i>PyTest</i>
Java	<i>JUnit</i>
C	<i>CUnit</i>
JavaScript (browser)	<i>Playwright</i> * <i>Selenium</i>
JavaScript (Node.js)	<i>Jest</i> **

\* Described in the following slides

\*\* See me if you want an example

# Aside: Automated Testing Tools

- Python **module** testing
  - See **Fraction/**
    - euclid.py
    - fraction.py
    - fractionclient.py
    - **testfraction.py**

# Aside: Automated Testing Tools

Might as well generate a coverage report!

```
$ cd Fraction
$ python -m coverage run -p testfraction.py
test_add (testfraction.TestFraction.test_add) ... ok
test_constructor (testfraction.TestFraction.test_constructor) ... ok
test_eq (testfraction.TestFraction.test_eq) ... ok
test_ge (testfraction.TestFraction.test_ge) ... ok
test_gt (testfraction.TestFraction.test_gt) ... ok
test_le (testfraction.TestFraction.test_le) ... ok
test_lt (testfraction.TestFraction.test_lt) ... ok
test_mul (testfraction.TestFraction.test_mul) ... ok
test_ne (testfraction.TestFraction.test_ne) ... ok
test_neg (testfraction.TestFraction.test_neg) ... ok
test_str (testfraction.TestFraction.test_str) ... ok
test_sub (testfraction.TestFraction.test_sub) ... ok
test_truediv (testfraction.TestFraction.test_truediv) ... ok

-----
Ran 13 tests in 0.002s

OK
$
```

# Aside: Automated Testing Tools

Coverage report - Chromium

Coverage report: 95%

filter...

Files Functions Classes  hide covered

*coverage.py v7.13.1, created at 2026-04-17 15:26 -0400*

File ▲	statements	missing	excluded	coverage
euclid.py	14	6	0	57%
fraction.py	51	0	0	100%
testfraction.py	68	0	0	100%
<b>Total</b>	<b>133</b>	<b>6</b>	<b>0</b>	<b>95%</b>

*coverage.py v7.13.1, created at 2026-04-17 15:26 -0400*

# Aside: Automated Testing Tools

- Python **web app** testing
  - See **PennyTest** app
    - runserver.py
    - penny.sql, penny.sqlite
    - database.py
    - penny.py
    - index.html
    - **testdatabase.py**

# Aside: Automated Testing Tools

Order of method execution:

```
setUp  
test_abc  
tearDown  
  
setUp  
test_ern  
tearDown  
  
setUp  
test_ker  
tearDown  
  
setUp  
test_sed  
tearDown
```

# Aside: Automated Testing Tools

Might as well generate a coverage report!



```
$ python -m coverage run -p testdatabase.py
test_abc (testdatabase.TestDatabase.test_abc) ... ok
test_ern (testdatabase.TestDatabase.test_ern) ... ok
test_ker (testdatabase.TestDatabase.test_ker) ... ok
test_sed (testdatabase.TestDatabase.test_sed) ... ok

-----
-----
Ran 4 tests in 0.002s

OK
$
```

# Aside: Automated Testing Tools

```
$ python -m coverage combine  
Combined data file  
.coverage.rdondero-latitudee6430s.pid12382.X0bhNQdx.H  
5ZX2xeg4Y1h  
$ python -m coverage html  
Wrote HTML report to htmlcov/index.html  
$
```

# Aside: Automated Testing Tools

Coverage report - Chromium

Coverage report: 100%

filter...

Files Functions Classes  hide covered

coverage.py v7.13.1, created at 2026-04-17 15:43 -0400

File ▲	statements	missing	excluded	coverage
database.py	14	0	0	100%
testdatabase.py	26	0	0	100%
<b>Total</b>	<b>40</b>	<b>0</b>	<b>0</b>	<b>100%</b>

coverage.py v7.13.1, created at 2026-04-17 15:43 -0400

# Aside: Automated Testing Tools

- Python **web app** testing
  - See **PennyTest** app
    - runserver.py
    - penny.sql, penny.sqlite
    - database.py
    - penny.py
    - index.html
    - testdatabase.py
    - **testpenny.py**
      - Like assignments: uses Playwright
      - Unlike assignments: uses unittest, checks results

# Aside: Automated Testing Tools

Order of method execution:

```
setUpClass
```

```
setUp  
test_abc  
tearDown
```

```
setUp  
test_ker  
tearDown
```

```
setUp  
test_ker_bad_spaces  
tearDown
```

```
setUp  
test_ker_spaces  
tearDown
```

```
setUp  
test_sed  
tearDown
```

```
setUp  
test_spaces  
tearDown
```

```
tearDownClass
```

# Aside: Automated Testing Tools

Might as well generate a coverage report!



```
$ python -m coverage run -p runserver.py 5000
* Serving Flask app 'penny'
* Debug mode: on
WARNING: This is a development server. Do not
use it in a production deployment. Use a
production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.29:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 457-747-552
...
```

# Aside: Automated Testing Tools

```
$ python testpenny.py
test_abc (testpenny.PennyTestCase.test_abc) ... ok
test_ker (testpenny.PennyTestCase.test_ker) ... ok
test_ker_bad_spaces (testpenny.PennyTestCase.test_ker_bad_spaces) ... ok
test_ker_spaces (testpenny.PennyTestCase.test_ker_spaces) ... ok
test_sed (testpenny.PennyTestCase.test_sed) ... ok
test_spaces (testpenny.PennyTestCase.test_spaces) ... ok

-----

Ran 6 tests in 13.835s

OK
$
```

# Aside: Automated Testing Tools

```
$ python -m coverage combine
Combined data file
.coverage.rdondero-latitudee6430s.pid13007.X9kpz8Gx.Hx54BaeMYYzh
Combined data file
.coverage.rdondero-latitudee6430s.pid13008.XDbF2HUx.HuctYH0pjqzh
Skipping duplicate data
.coverage.rdondero-latitudee6430s.pid13564.XQabYdYx.Hx54BaeMYYzh
Skipping duplicate data
.coverage.rdondero-latitudee6430s.pid13565.XPG3zv3x.HuctYH0pjqzh
$ python -m coverage html
Wrote HTML report to htmlcov/index.html
$
```

# Aside: Automated Testing Tools

Coverage report - Chromium

Coverage report: 83%

filter...  hide covered

Files Functions Classes

coverage.py v7.13.1, created at 2026-04-17 15:56 -0400

File ▲	statements	missing	excluded	coverage
database.py	14	0	0	100%
penny.py	21	1	0	95%
runserver.py	18	8	0	56%
<b>Total</b>	<b>53</b>	<b>9</b>	<b>0</b>	<b>83%</b>

coverage.py v7.13.1, created at 2026-04-17 15:56 -0400

# Aside: Automated Testing Tools

Coverage for penny.py: 95% - Chromium

Coverage for penny.py: 95%

21 statements 20 run 1 missing 0 excluded

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:56 -0400

```
1 #!/usr/bin/env python
2
3 #-----
4 # penny.py
5 # Author: Bob Dondero
6 #-----
7
8 import json
9 import flask
10 import database
11
12 #-----
13
14 app = flask.Flask(__name__)
15
16 #-----
17
18 @app.route('/', methods=['GET'])
19 @app.route('/index', methods=['GET'])
20 def index():
21
22     return flask.send_file('index.html')
```

Coverage for penny.py: 95% - Chromium

penny.py: 95% 20 1 0

```
20 def index():
21
22     return flask.send_file('index.html')
23
24 #-----
25
26 @app.route('/searchresults', methods=['GET'])
27 def search_results():
28
29     author = flask.request.args.get('author')
30     if author is None:
31         author = ''
32         author = author.strip()
33
34     if author == '':
35         books = []
36     else:
37         books = database.get_books(author) # Exception handling omitted
38
39     json_doc = json.dumps(books)
40     response = flask.make_response(json_doc)
41     response.headers['Content-Type'] = 'application/json'
42     return response
```

« prev ^ index » next coverage.py v7.13.1, created at 2026-04-17 15:56 -0400

# Aside: Automated Testing Tools

- **Problem:**
  - What if the web app does authentication?
    - Via CAS, EntraID, Google, ...
  - How can your testing code access app endpoints that are protected?

# Aside: Automated Testing Tools

**Solution:** The general idea:

authstub.py

```
...
# Implement an "authentication" mechanism
# that simply asks the user for a
# username.
...
```

penny.py

```
...
TESTING = os.getenv('TESTING', 'no')
if TESTING == 'no':
    import auth
else:
    import authstub as auth
...
```

See me if you want an example

You've tested your code to make sure it meets your expectations. What's next?

Continued in  
Software Engineering (Part 4)...