Software Engineering (Part 3)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover these software engineering topics:
  - Requirements analysis
  - Design
  - Implementation
  - Debugging
  - Testing
  - Evaluation
  - Maintenance
  - Process models

Stages of SW dev

How to order the stages

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 YYY, 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_NDEBUG=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	***coverage****
Java	***JaCoCo***
C	***gcov***
JavaScript (Node.js)	***nyc/Istanbul**** ***Jest/Istanbul****
JavaScript (browser)	***Chrome Coverage***

* 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

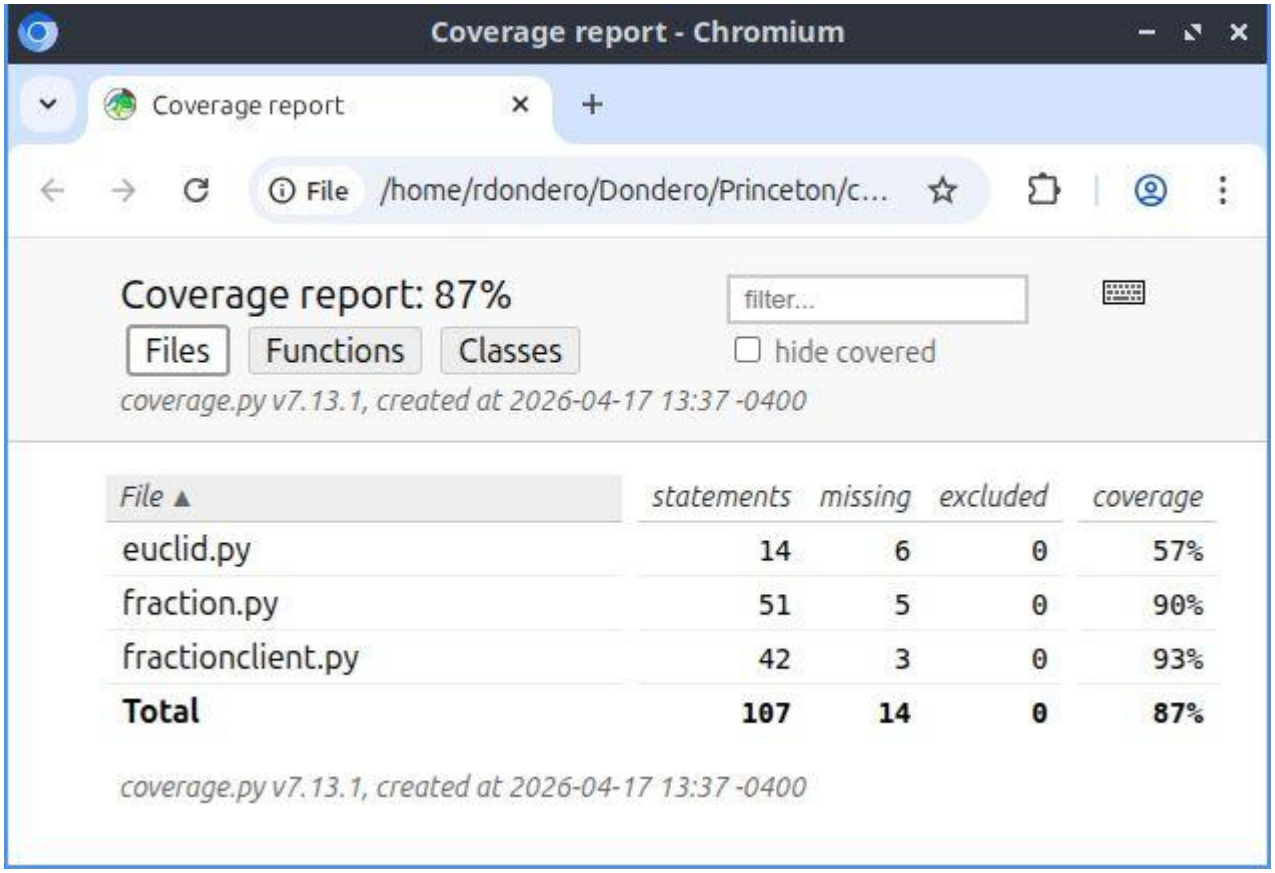
```
$ python -m coverage run -p fractionclient.py
Numerator 1: 3
Denominator 1: 4
Numerator 2: 1
Denominator 2: 2
frac1: 3/4
frac2: 1/2
frac1 does not equal frac2
frac1 is greater than frac2
frac1 is greater than or equal to frac2
-frac1: -3/4
frac1 + frac2: 5/4
frac1 - frac2: 1/4
frac1 * frac2: 3/8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

Aside: Statement Testing Tools

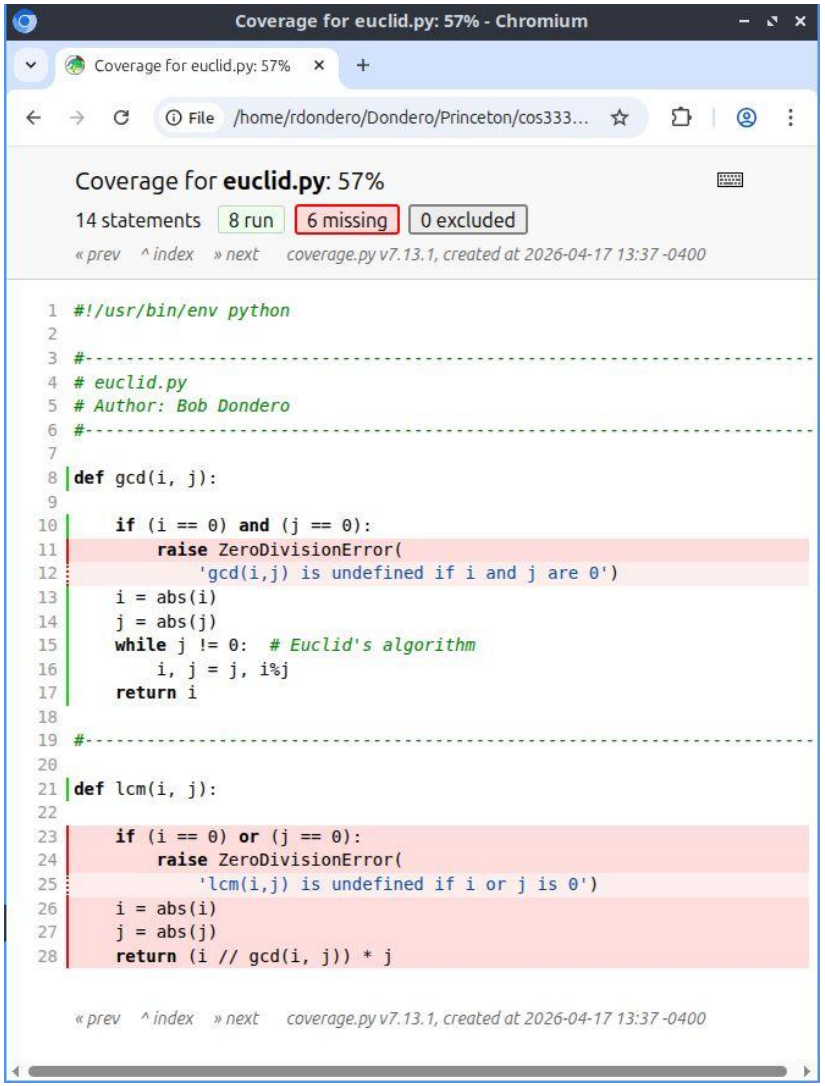
```
$ python -m coverage combine
Combined data file
   .coverage.rdondero-latitudee6430s.pid4800.XmWXSufx.HRcim92nc1Oh
Combined data file
   .coverage.rdondero-latitudee6430s.pid4805.X1f0Epbx.H0qhG6VTeC1h
$ python -m coverage html
Wrote HTML report to htmlcov/index.html
$
```

Then browse to htmlcov/index.html…

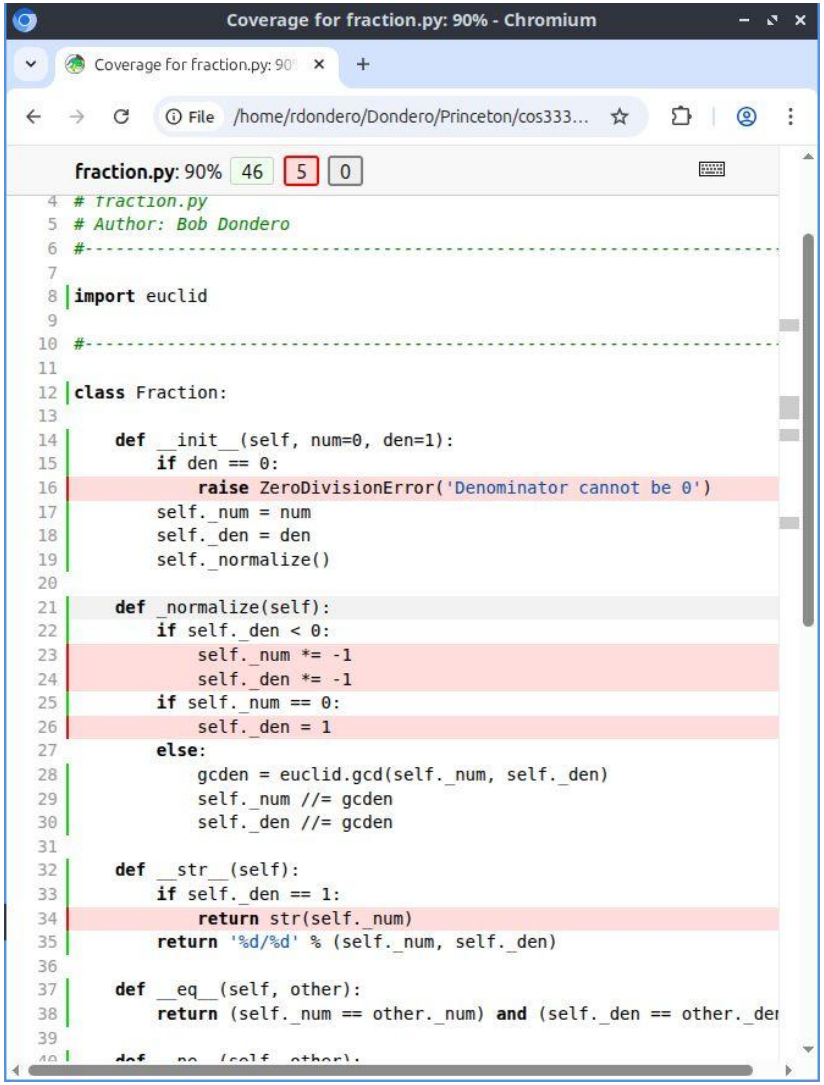
Aside: Statement Testing Tools



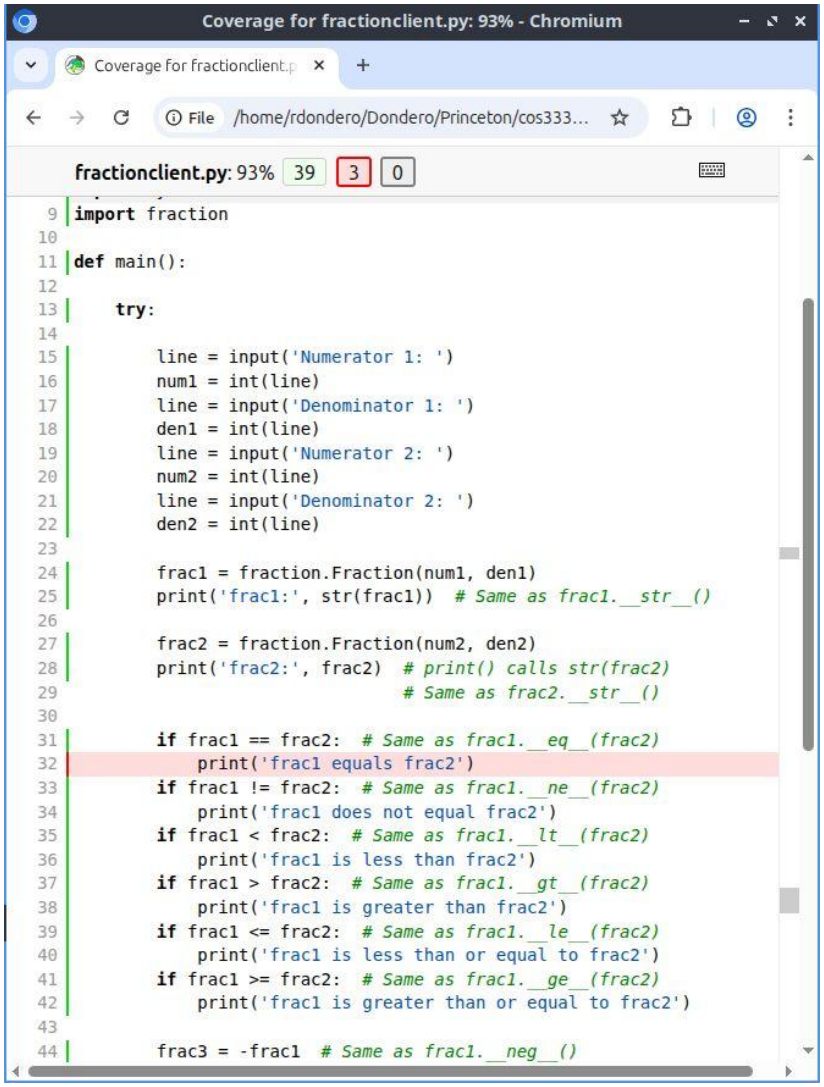
Aside: Statement Testing Tools



Aside: Statement Testing Tools



Aside: Statement Testing Tools



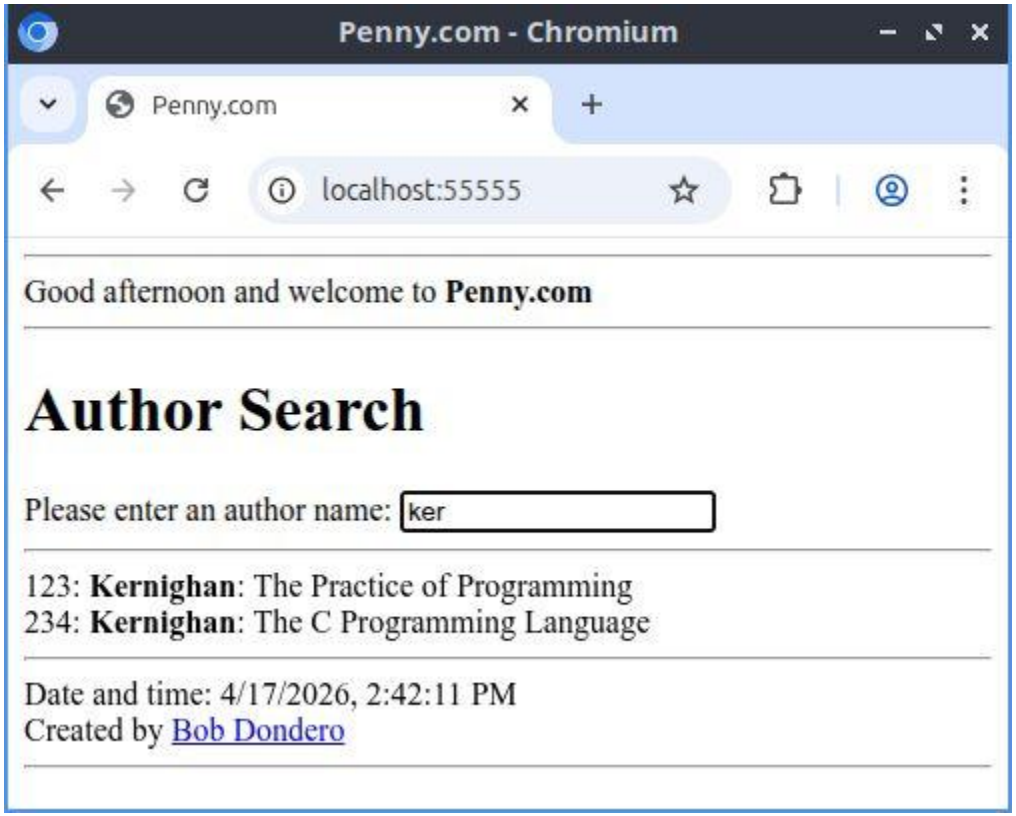
Aside: Statement Testing Tools

- 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 55555
 * 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

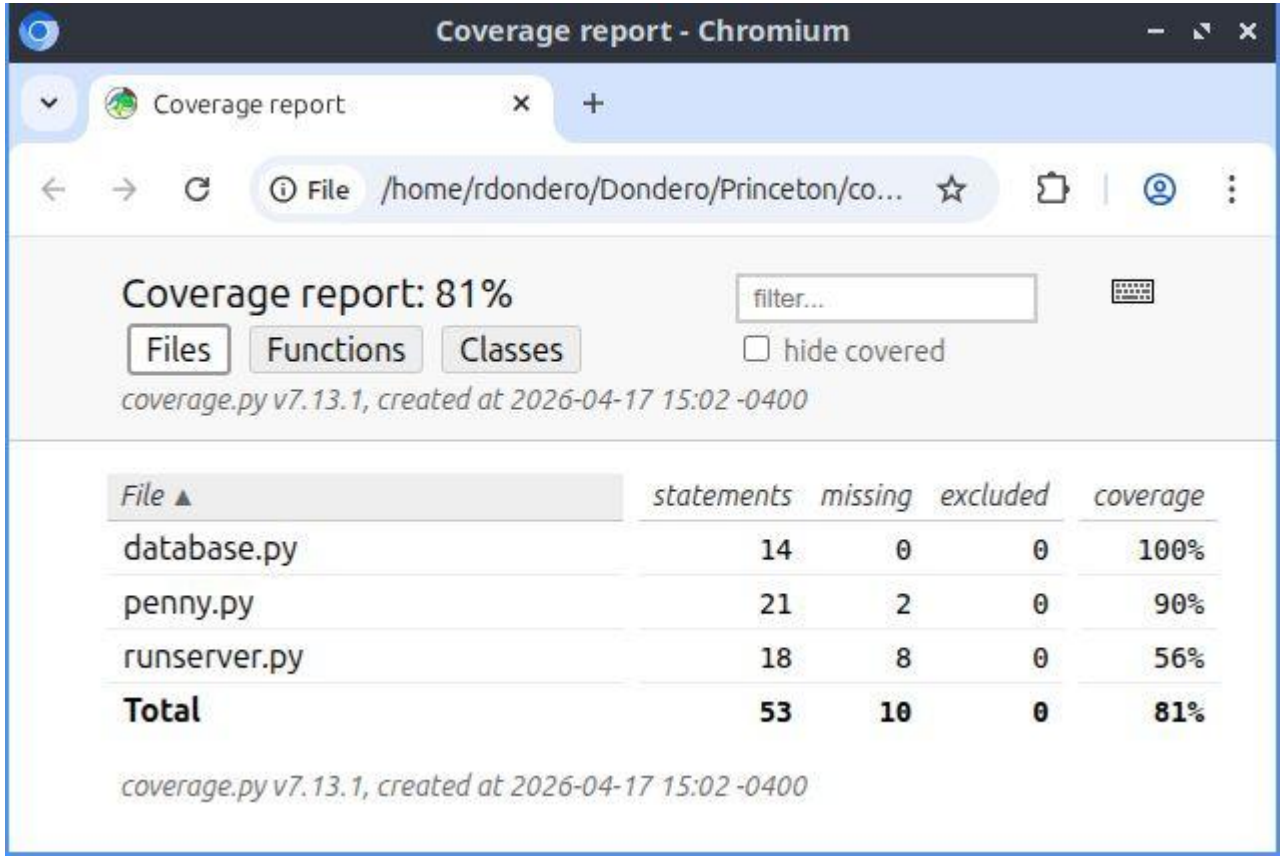


Aside: Statement Testing Tools

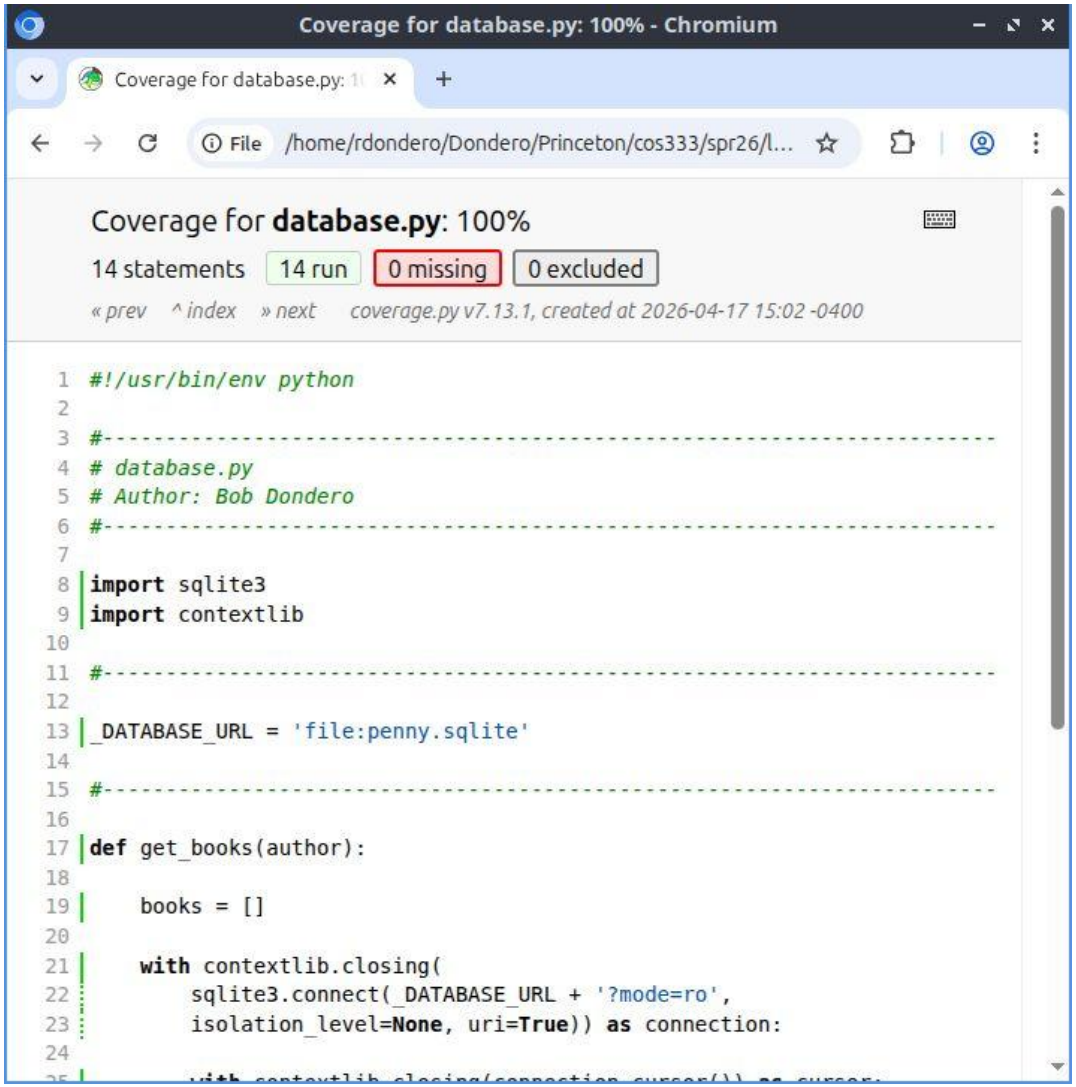
```
$ python -m coverage combine
Combined data file
.coverage.rdondero-latitudee6430s.pid9985.XfaNCpcx.Hr66UMu8wAZh
Combined data file
.coverage.rdondero-latitudee6430s.pid9988.XjcZixRx.HiryNcoxZFyh
$ python -m coverage html
Wrote HTML report to htmlcov/index.html
$
```

Then browse to htmlcov/index.html…

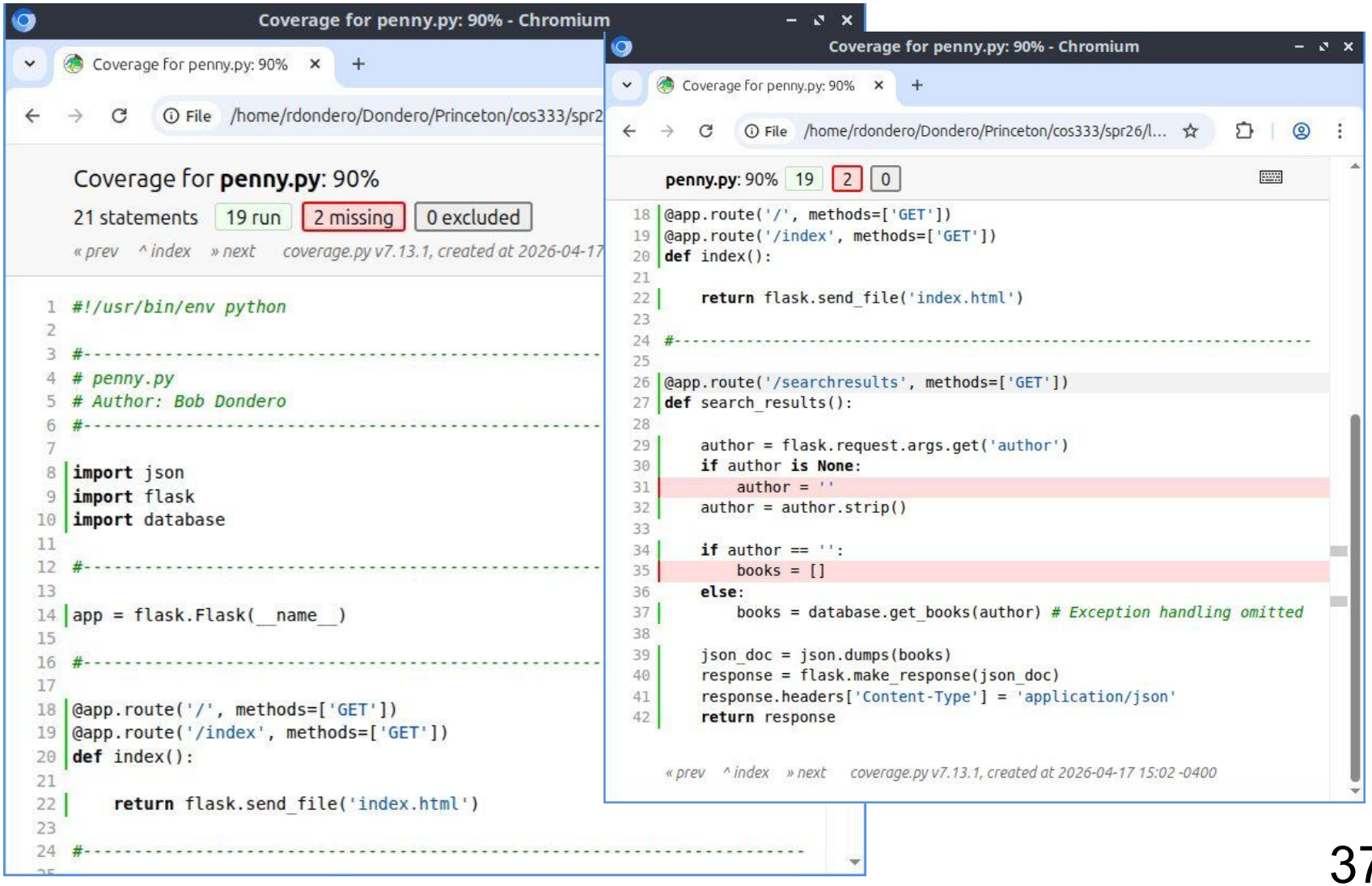
Aside: Statement Testing Tools



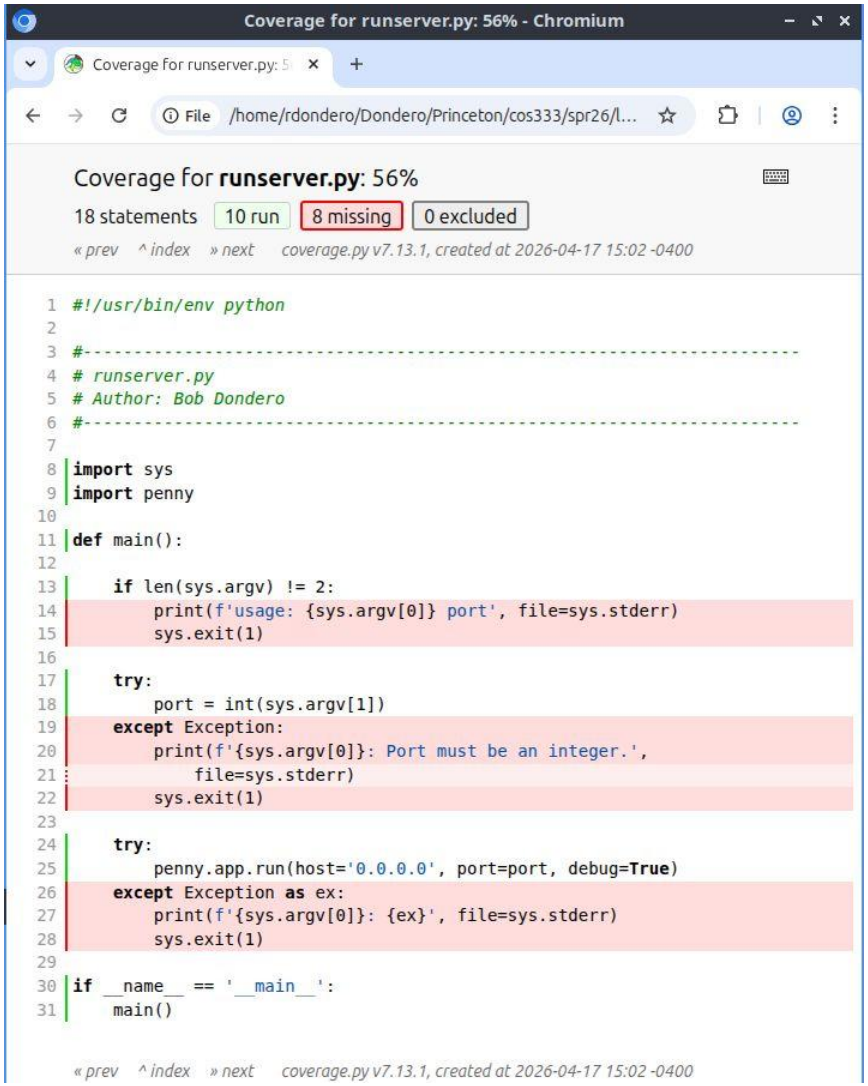
Aside: Statement Testing Tools



Aside: Statement Testing Tools



Aside: Statement Testing Tools



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
What	Test **modules** (classes, functions, methods)	Test **programs**
When	Often compose tests for module X **before** composing X	Typically compose tests for program X **after** composing X
Who	Often performed by a **single programmer**, or a small group	Often performed by a **QA organization**

Aside: Automated Testing Tools

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

* Described in the following slides
** See me if you want an example

Aside: Automated Testing Tools

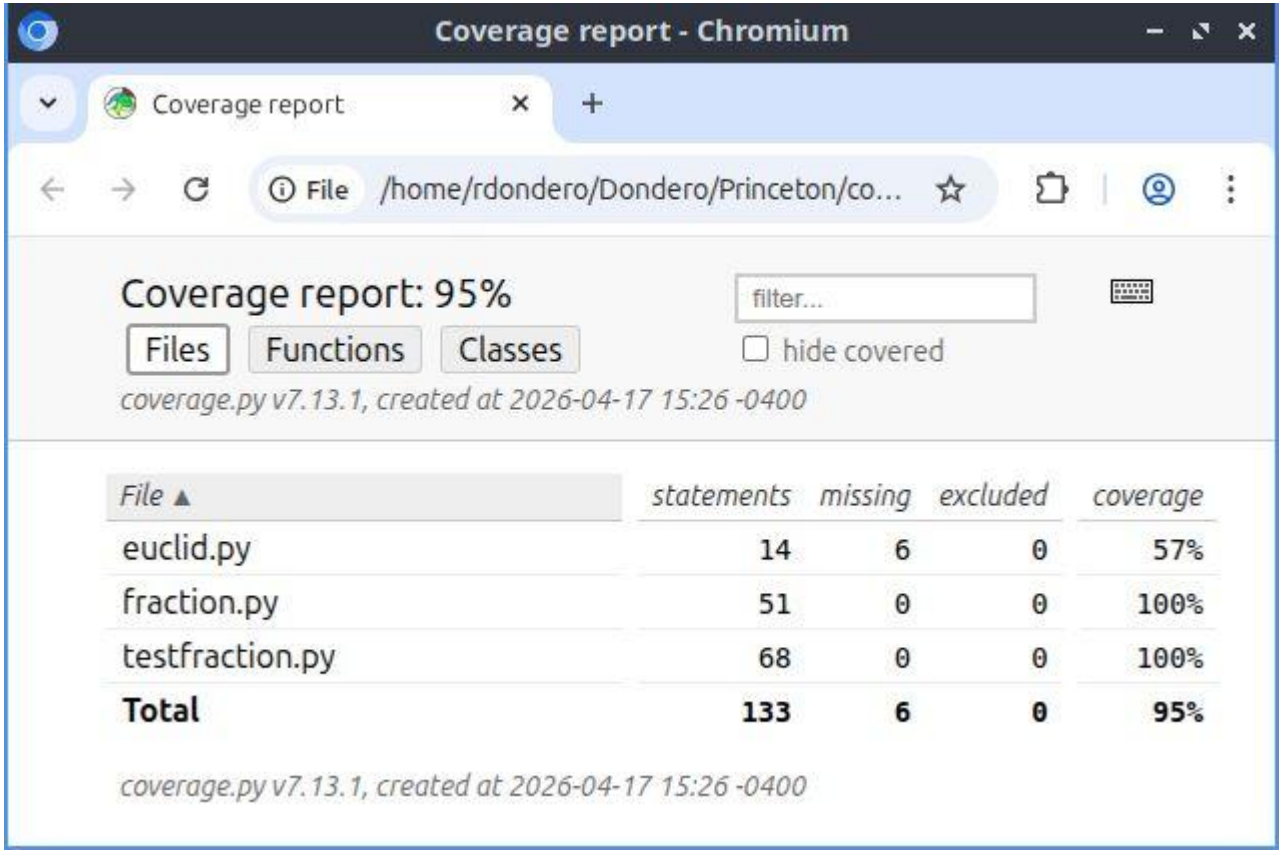
- Python **module** testing
  - See **Fraction/**
    - euclid.py
    - fraction.py
    - fractionclient.py
    - **testfraction.py**

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



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

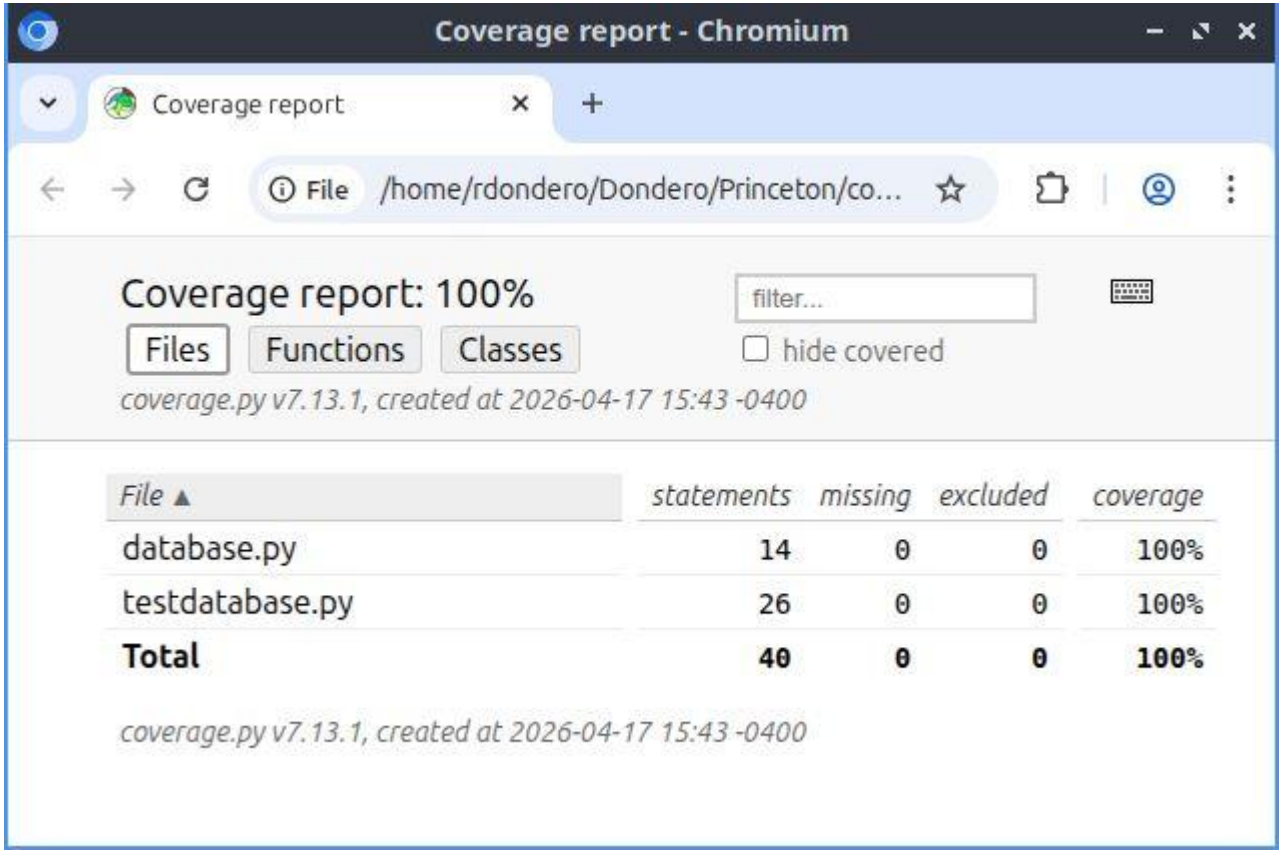
Might as well generate a coverage report!

```
$ 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



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
...
```

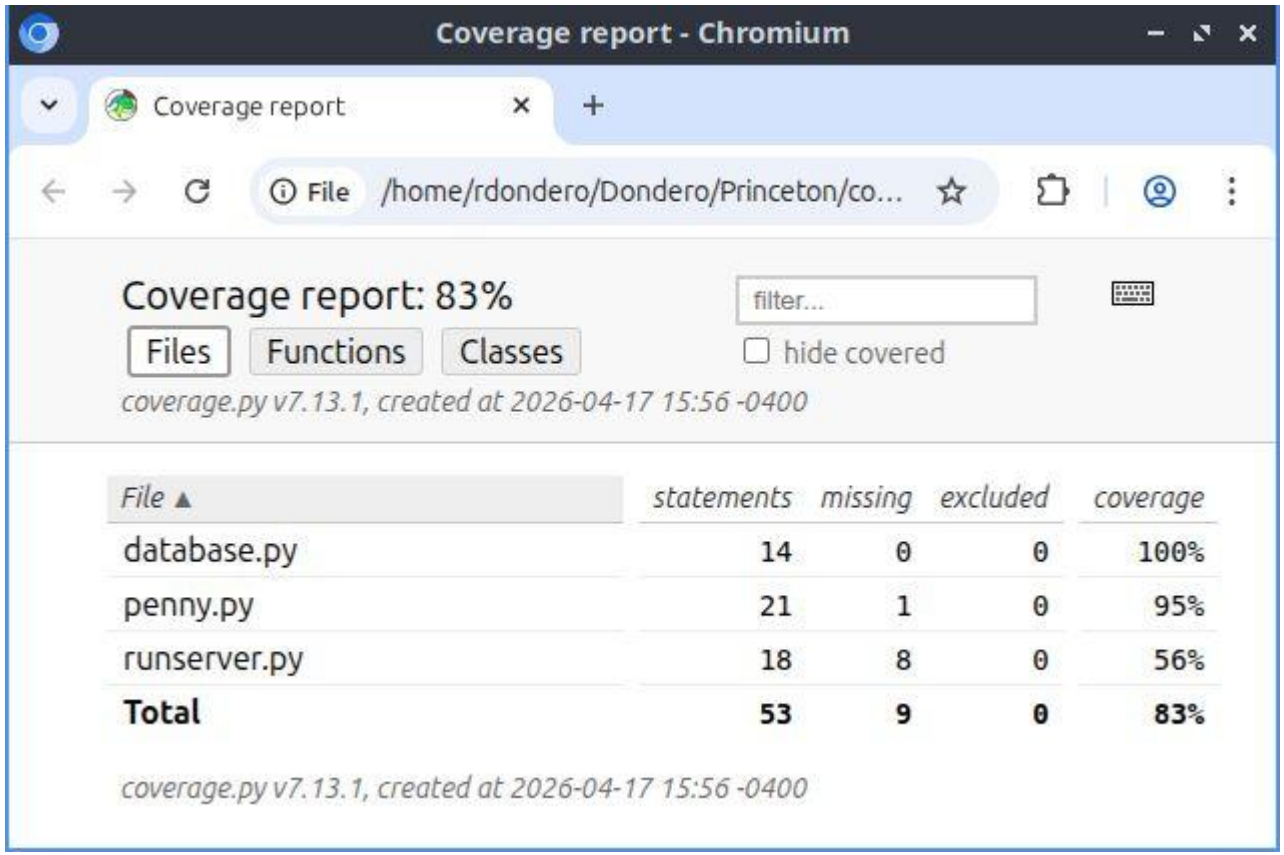
Aside: Automated Testing Tools

```
$ 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
$
```

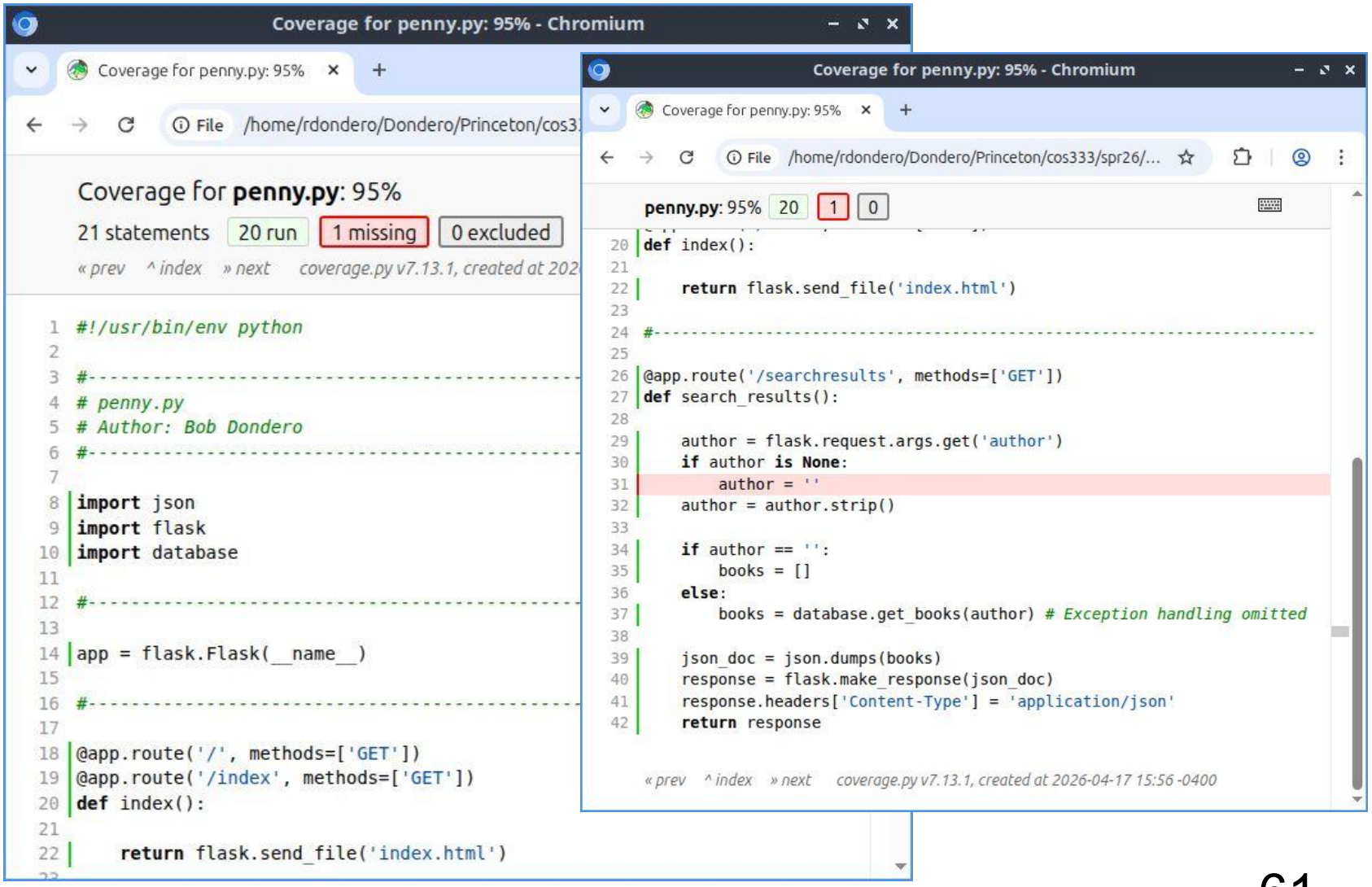
Aside: Automated Testing Tools

```
$ 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



Aside: Automated Testing Tools



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)…