Debugging and Interpreting Exceptions

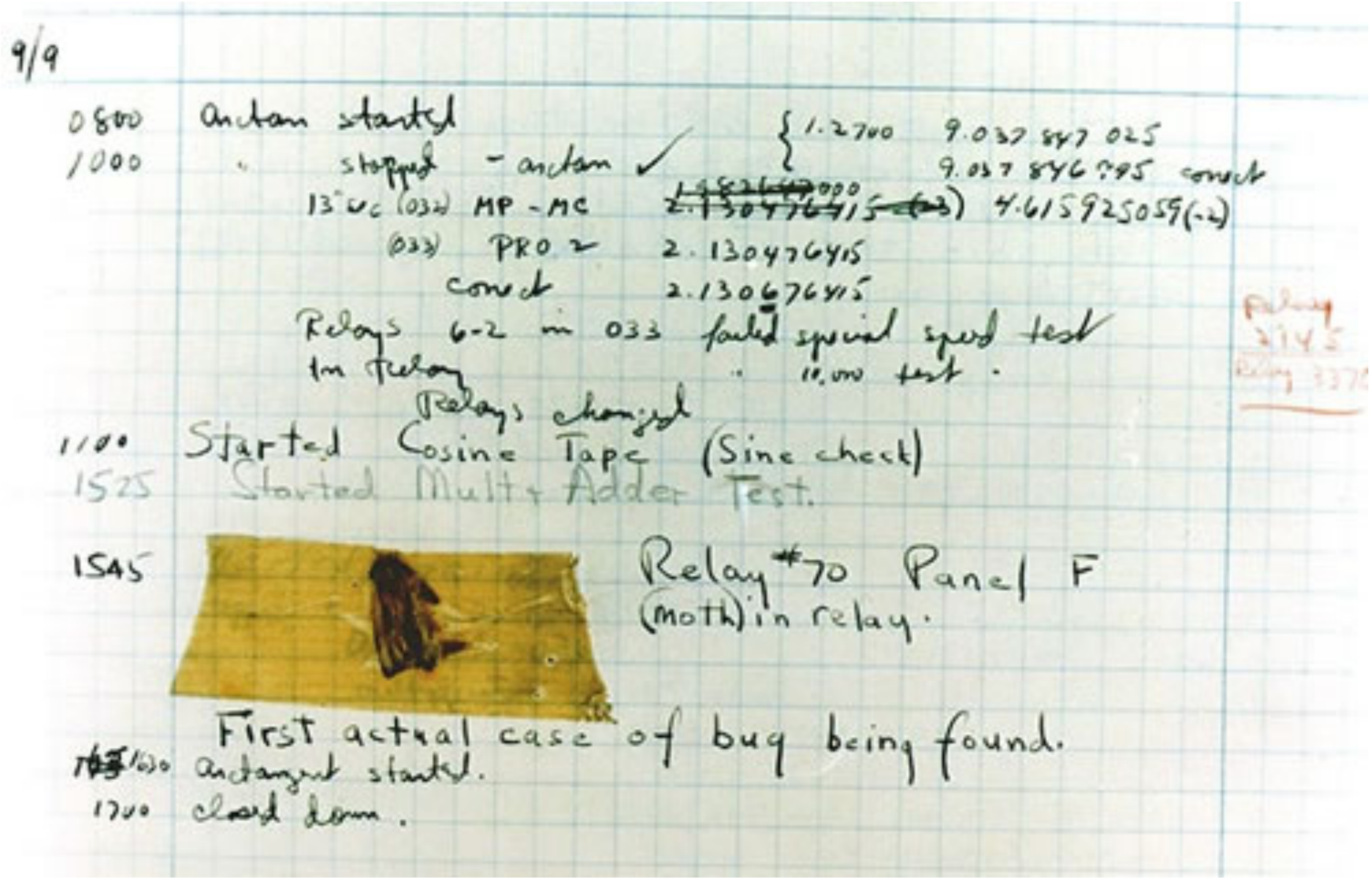
UW CSE 190p

Summer 2012

Review

```
>>> print "foo"
foo
>>> x = "foo"
>>> print x
foo
>>>
```

Debugging



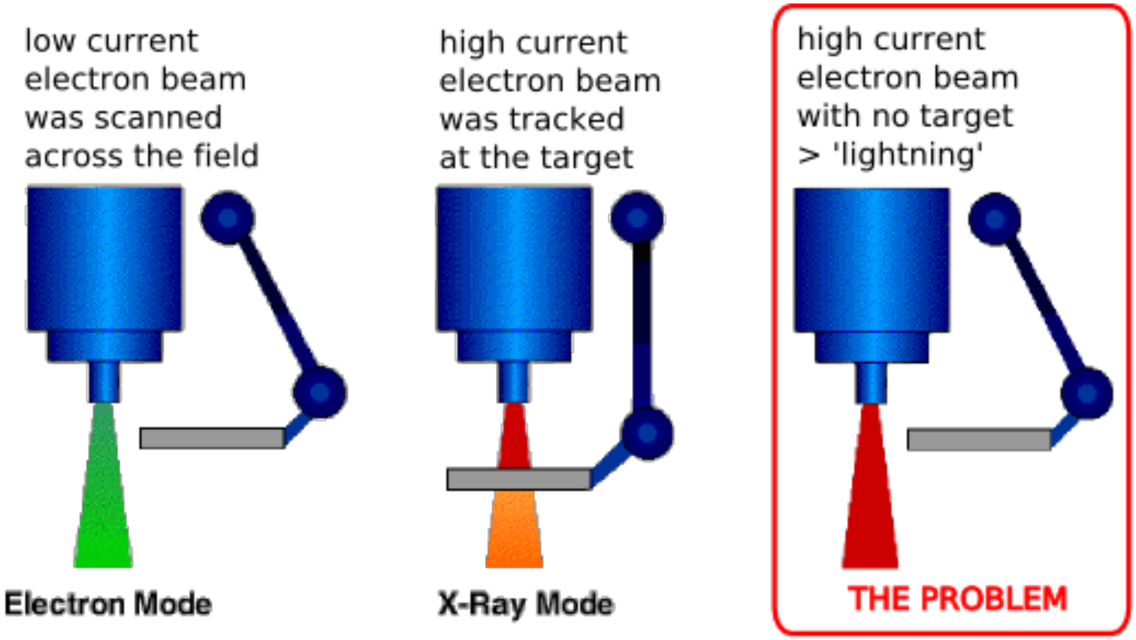
9/9

0800 arctan started
1000 " stopped - arctan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11,000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
1630 arctangent started.
1700 closed down.

Relay 2145
Relay 3370

Debugging Matters

Ariane 5, 1996



low current electron beam was scanned across the field

high current electron beam was tracked at the target

high current electron beam with no target > 'lightning'

Electron Mode

X-Ray Mode

THE PROBLEM

tray including the target, a flattening filter, the collimator jaws and an ion chamber was moved OUT for "electron" mode, and IN for "photon" mode.

Therac 25, 1980s

The Way I Think About Debugging

Apologies for the mixed metaphors….

If it doesn’t work as expected, then by definition you don’t understand what is going on.

- You’re lost in the woods.
- You’re behind enemy lines.
- All bets are off.
- Don’t trust anyone or anything.

Don’t press on into unexplored territory -- go back the way you came!

(and leave breadcrumbs!)

You’re trying to “advance the front lines,” not “trailblaze”

My Favorite Time-Saving Trick: Make Sure you're Debugging the Right Problem

- The game is to go from "working to working"
- When something doesn't work, STOP!
  - It's wild out there!
- FIRST: go back to the last situation that worked properly.
  - Rollback your recent changes and verify that everything still works as expected.
  - Don't make assumptions – by definition, you don't understand the code when something goes wrong, so you can't trust your assumptions.
  - You may find that even what previously worked now doesn't
  - Perhaps you forgot to consider some "innocent" or unintentional change, and now even tested code is broken

Timeline

- A works, so celebrate a little
- Now try B
- B doesn't work
- Change B and try again
- Change B and try again
- Change B and try again

…

Timeline

- A works, so celebrate a little
- Now try B
- B doesn't work
- *Rollback to A*
- Does A still work?
  - Yes: Find A' that is somewhere between A and B
  - No: You have unintentionally changed something else, and there's no point futzing with B at all!

These "innocent" and unnoticed changes happen more than you would think!

- You add a comment, and the indentation changes.
- You add a print statement, and a function is evaluated twice.
- You move a file, and the wrong one is being read
- You're on a different computer, and the library is a different version

Once you're on solid ground you can set out again

- Once you have something that works and something that doesn't work, it's only a matter of time
- You just need to incrementally change the working code into the non-working code, and the problem will reveal itself.
- Variation: Perhaps your code works with one input, but fails with another. Incrementally change the good input into the bad input to expose the problem.

Scientific Method

By definition, unexpected behavior means you don’t understand the code.

How do you learn about something you don’t understand?

1) Form a hypothesis

2) Make a prediction

3) Test and analyze

Simple Debugging Tools

print

- shows what's happening whether there's a problem or not
- does not stop execution

assert

- Raises an exception if some condition is not met
- Does nothing if everything works
- Use this liberally! Not just for debugging!

raw_input

- Stops execution
- (Designed to accept user input, but I rarely use it for this.)

assert statement

```
assert len(rj.edges()) == 16
```

```
Traceback (most recent call last):
  File "assertion.py", line 28, in <module>
    assert len(rj.edges()) == 16
AssertionError
```

Recommendation 2:
Read the error message!

- As unhelpful as they sometimes can be, they are your best (and often only) starting point for diagnosis.
- The developers went through a lot of trouble to provide these messages – use them.
- You need to master

  1) the literal meaning of the error

  2) the underlying problems certain errors tend to suggest

```
def friends_of_friends(graph, user):
    """Returns a set of friends of friends of the given user, in
the given graph. The result does not include the user nor their
friends """
    fof = set()
    f = friends(graph, user)
    for fren in f:
        friend = set(graph.neighbors(fren))
        fof = fof | friend
    g = (fof - f) - user
    return g
```

Mecutio -> Romeo -> Juliet

```
  Traceback (most recent call last):
    File ”social_network.py", line 20, in <module>
      friends_of_friends(g, 2)
    File ”social_network.py", line 14, in friends_of_friends
      g = (fof - f) - user
  TypeError: unsupported operand type(s) for -: 'set' and 'int'
```

```
def friends_of_friends(graph, user):
    """Returns a set of friends of friends of the given user, in
the given graph. The result does not include the user nor their
friends """
    fof = set()
    f = friends(graph, user)
    for fren in f:
        friend = set(graph.neighbors(fren))
        fof = fof | friend
    f.add([user])
    g = (fof - f)
    return g
```

```
Traceback (most recent call last):
  File "unhashable_type.py", line 21, in <module>
    friends_of_friends(g, "Mercutio")
  File "unhashable_type.py", line 14, in friends_of_friends
    f.add([user])
TypeError: unhashable type: 'list'
```