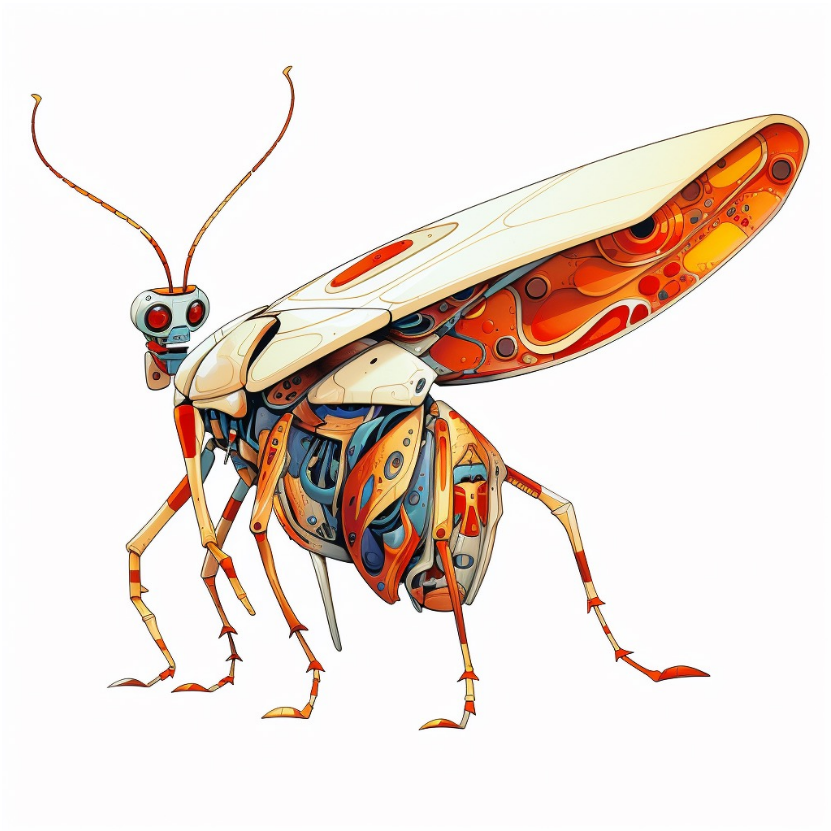
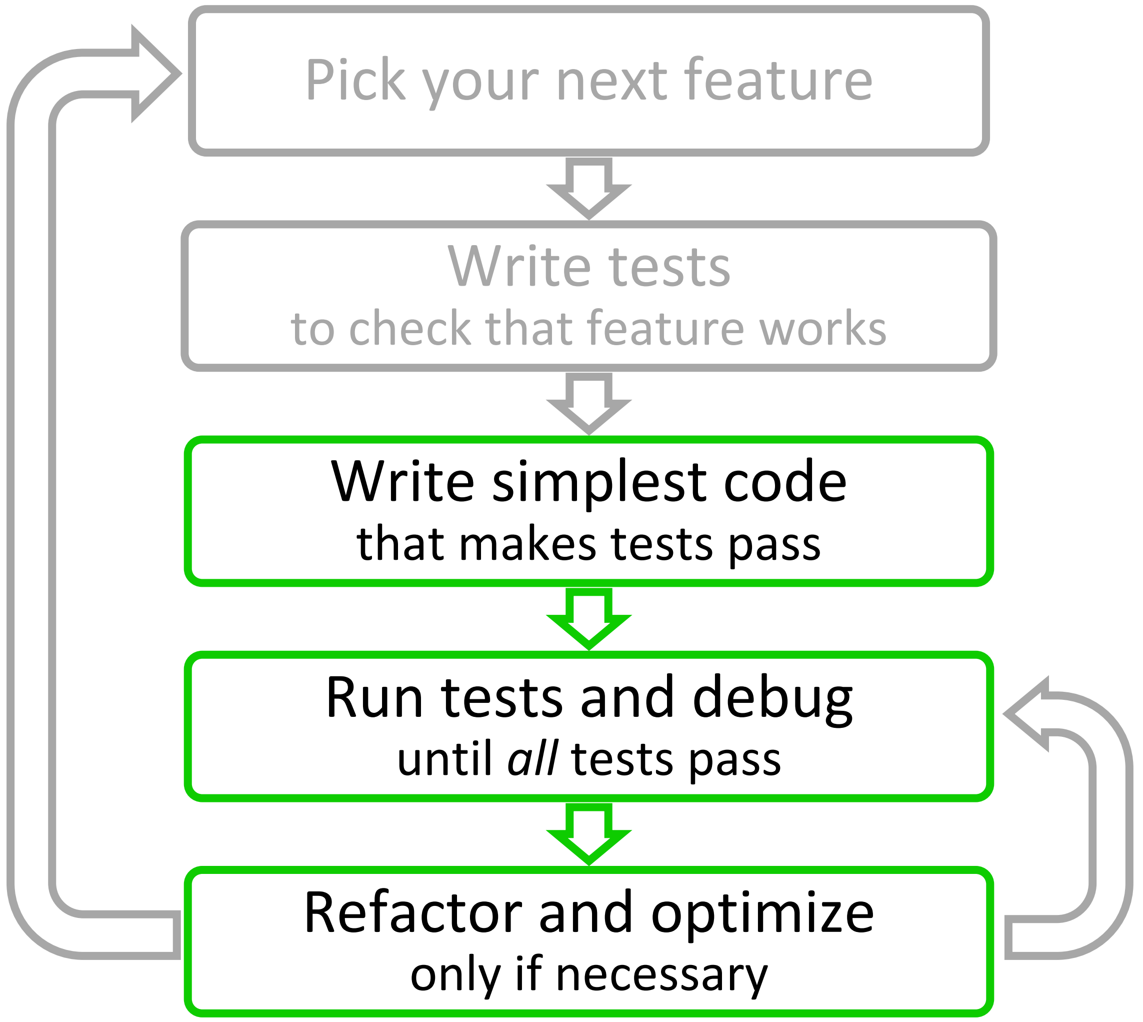
Debugging

Fix problems



Lisa Schwetlick and Pamela Hathway

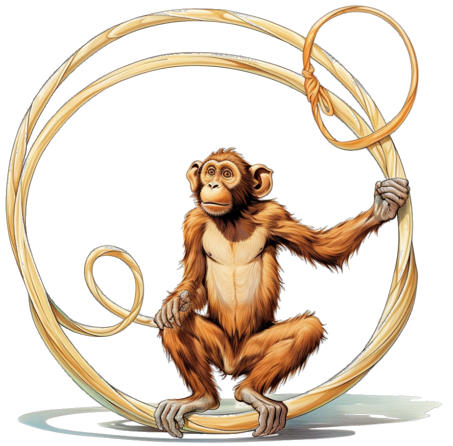
The agile development cycle



`pytest`

pdb

`timeit`
`cProfile`
`line_profiler`

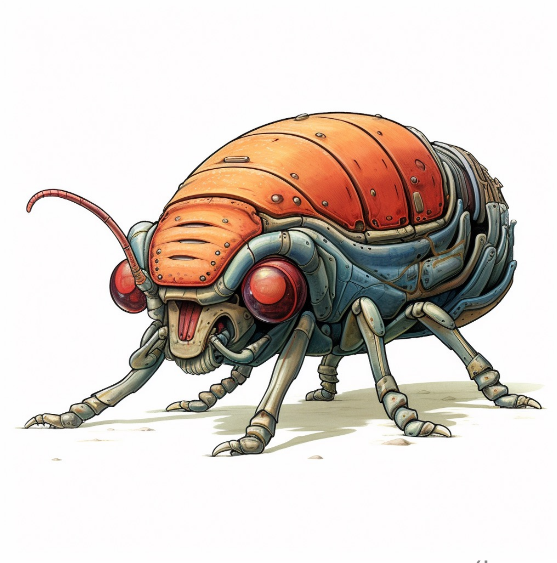


What is Debugging?

- The best way to debug is to avoid bugs
  - By writing tests, you *anticipate* the bugs
- Your test cases should already exclude a big portion of the possible causes
- Core idea in debugging: you can stop the execution of your application at any point, look at the state of the variables, and execute the code step by step

pdb, the Python debugger

- Command-line based debugger
- pdb opens an interactive shell, in which one can interact with the code
  - examine and change value of variables
  - execute code line by line
  - set up breakpoints
  - examine calls stack



Entering the debugger

- Enter debugger at the start of a file:
  ```
  python –m pdb myscript.py
  ```

- Enter at a specific point in the code (easy alternative to `print`):

```
# some code here
# the debugger starts here
breakpoint()
# rest of the code
```

Demonstration!

- Debugging with pdb

Hands-on! – Issue #2

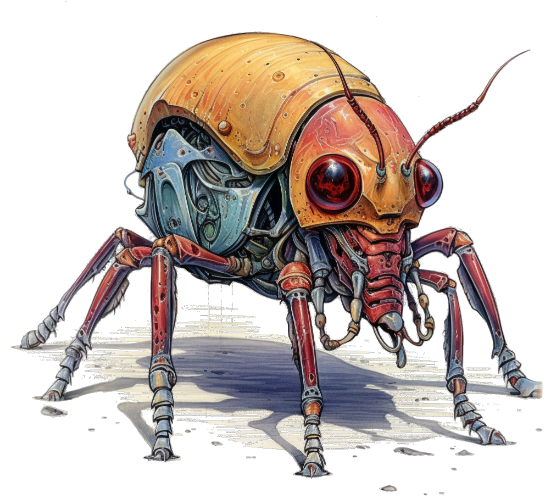
1) Go to `local_maxima_part3_debug/`**`local_maxima.py`** and execute it

2) Go to `local_maxima_part3_debug/`**`test_local_maxima.py`** and execute all the tests there using pytest

3) Fix the tests one by one and create a PR for each one you fix

pdb cheatsheet:

h (help) [*command*]	print help about *command*
n (next)	execute current line of code, go to next line
c (continue)	continue executing the program until next breakpoint, exception, or end of the program
s (step into)	execute current line of code; if a function is called, follow execution inside the function
l (list)	print code around the current line
w (where)	show a trace of the function call that led to the current line
p (print)	print the value of a variable
q (quit)	leave the debugger
b (break) [*lineno* | *function*[, *condition*]]	set a breakpoint at a given line number or function, stop execution there if *condition* is fulfilled
cl (clear)	clear a breakpoint
! (execute)	execute a python command
<enter>	repeat last command



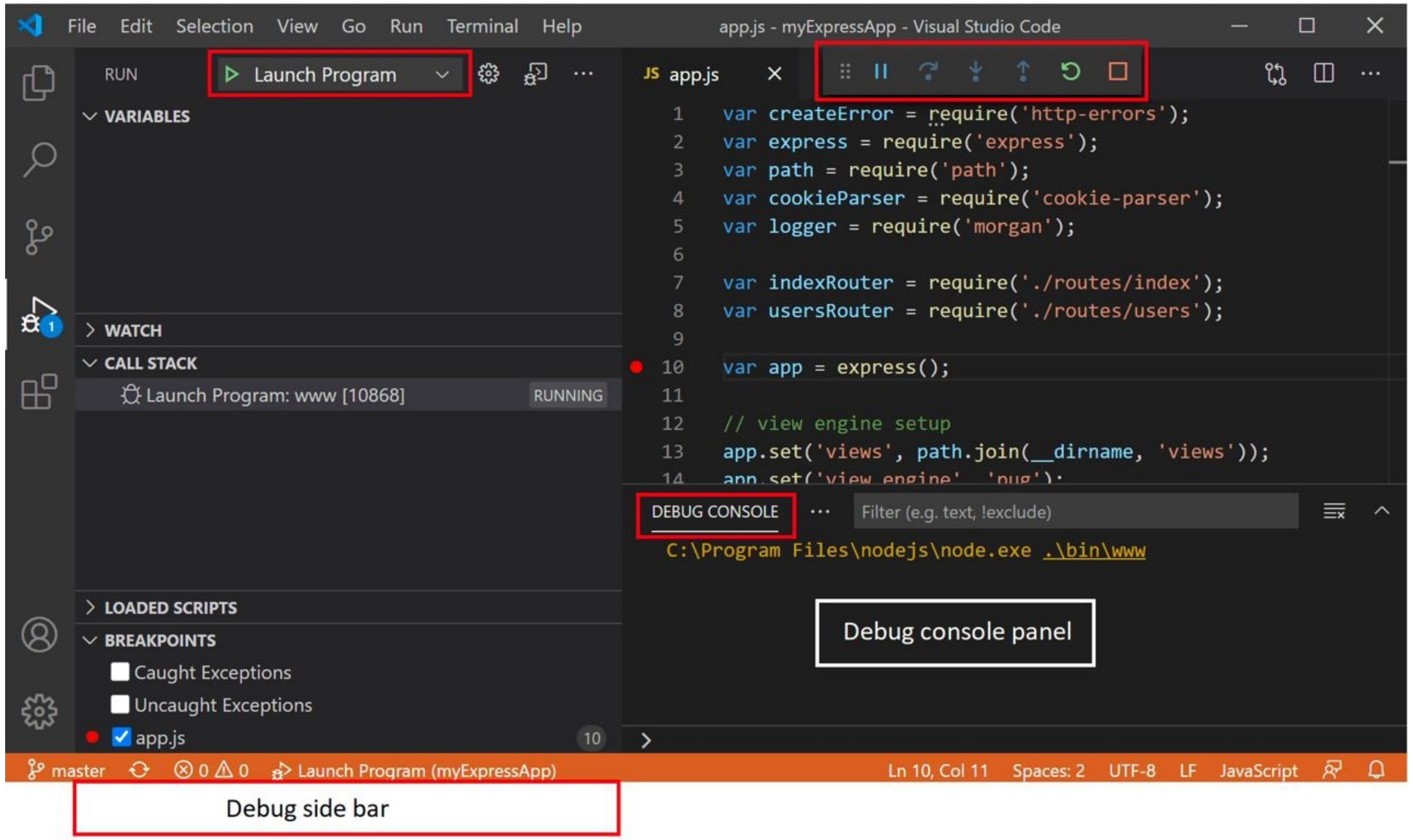
Demonstration!

- Debugging with pdb
- Debugging from an IDE

Entering the debugger from VSCode

Start debugging

Pause, step over, step in/out, restart, stop



```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

```
C:\Program Files\nodejs\node.exe .\bin\www
```

Debug console panel

Debug side bar

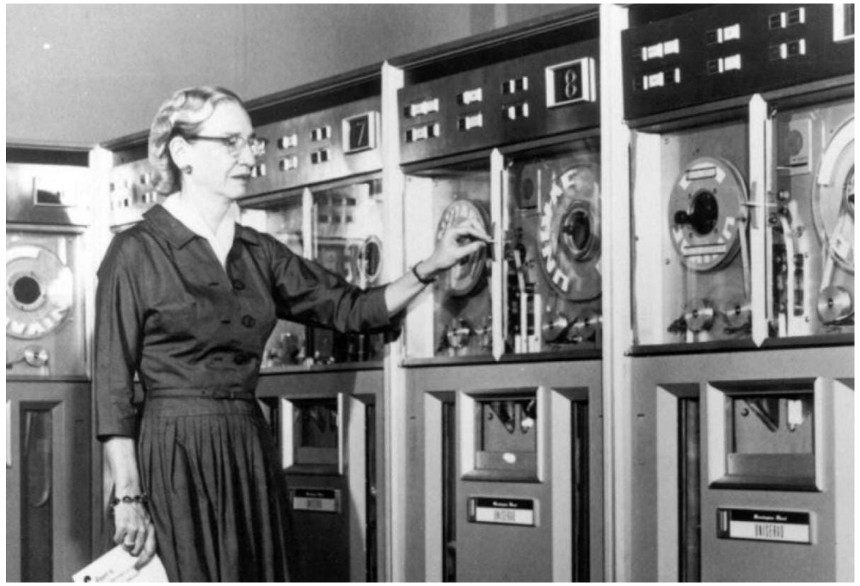
How to react to a bug

1. Add a test that matches the behavior you expect. It will fail and reproduce the bug
2. Debug and fix the the bug
3. Run the tests until they all pass (go back to 2 if necessary)

- Now your bug is fixed *and* it will never occur again!

Where does ”debug” come from

Team around Grace Hopper at Harvard found a moth in their computer in 1947 in maybe the first description of a computer bug.



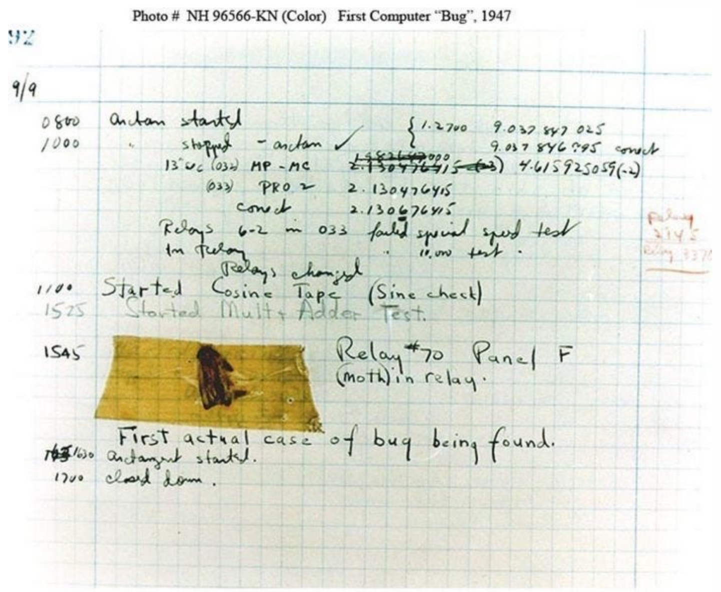


Photo # NH 96566-KN (Color) First Computer "Bug", 1947

92

9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025

9.037 846 995 conect

13"UC (032) MP - MC ~~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 Anctangent started.

1700 closed down.

Up next:
Continuous Integration