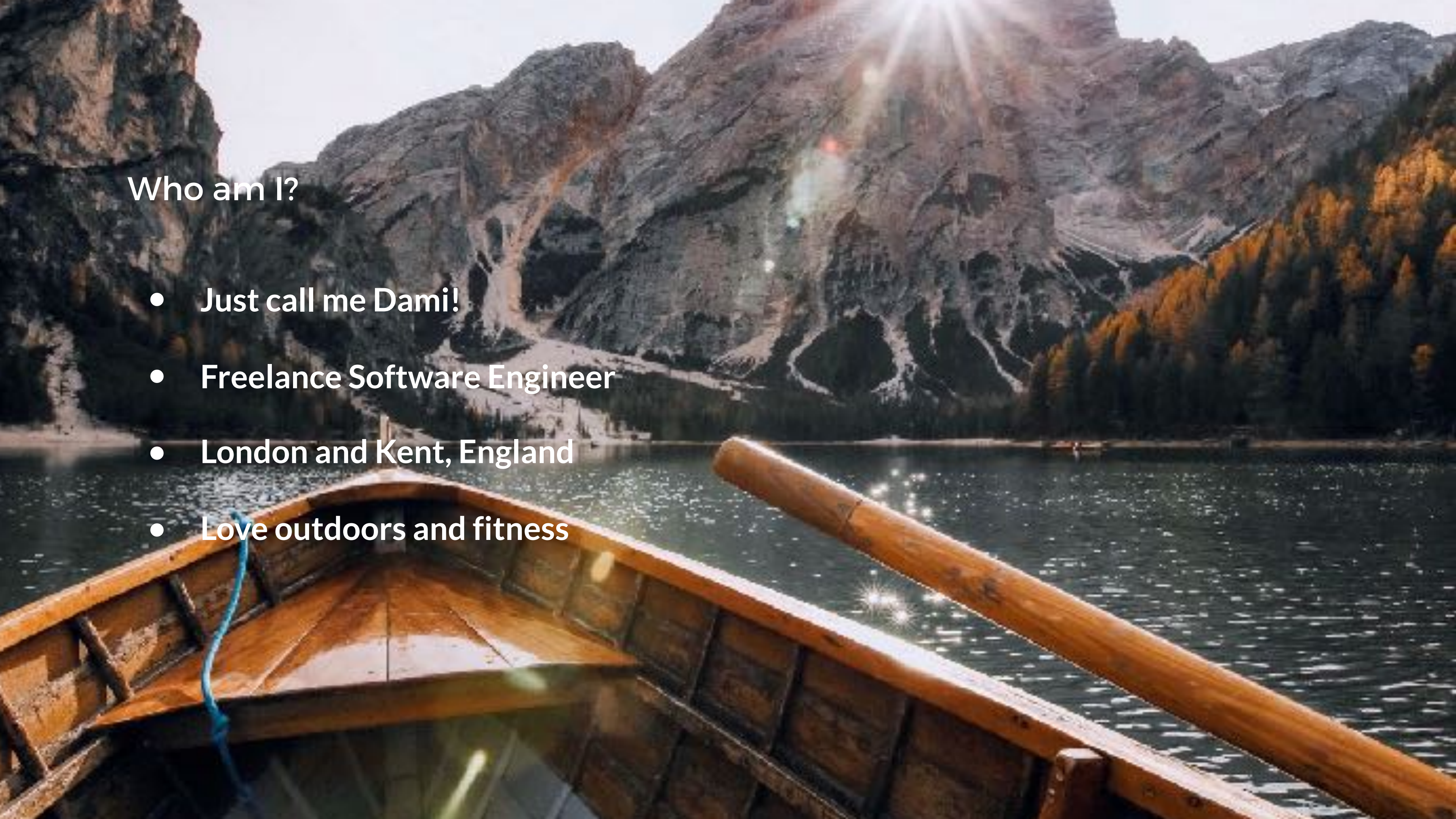
Debugging Python Application for Profit: Tools and Techniques

By Damilare Onajole
@damilare
PYCON NIGERIA 2018



Who am I?

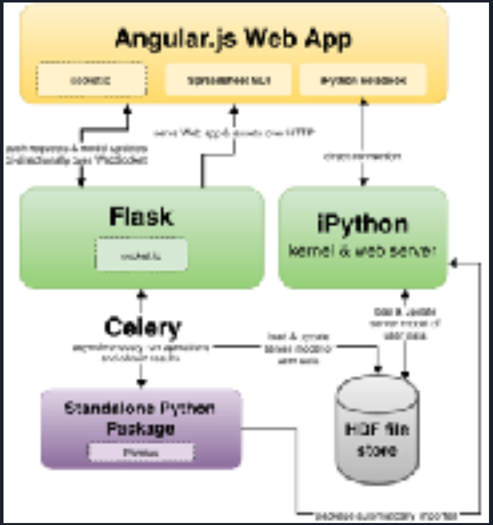
- Just call me Dami!
- Freelance Software Engineer
- London and Kent, England
- Love outdoors and fitness

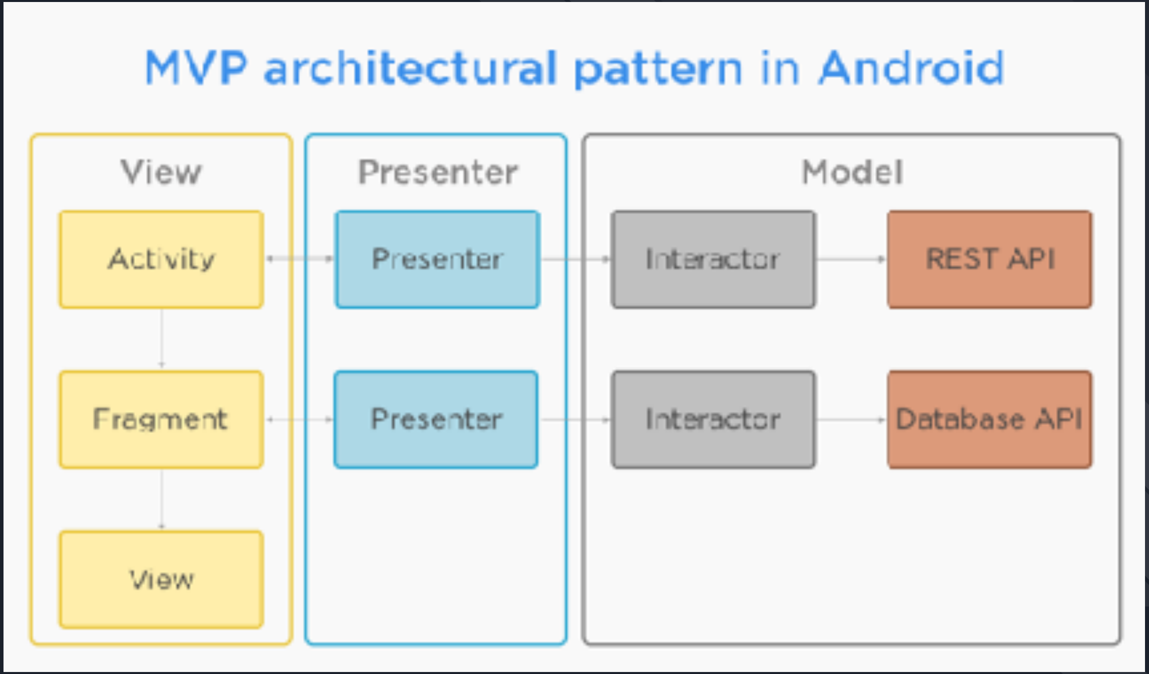
State of Software

What it looks like

State of Software

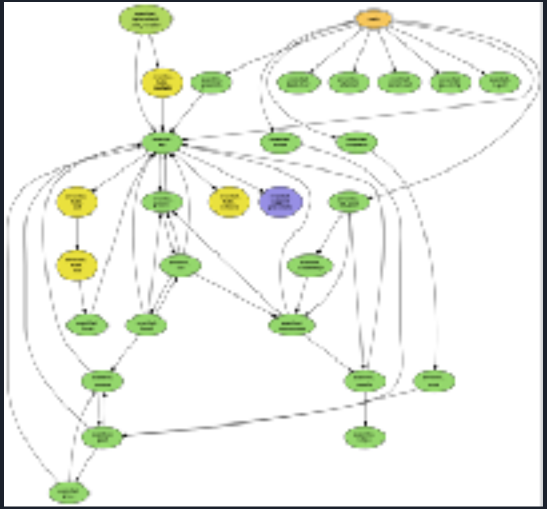
What you think it is

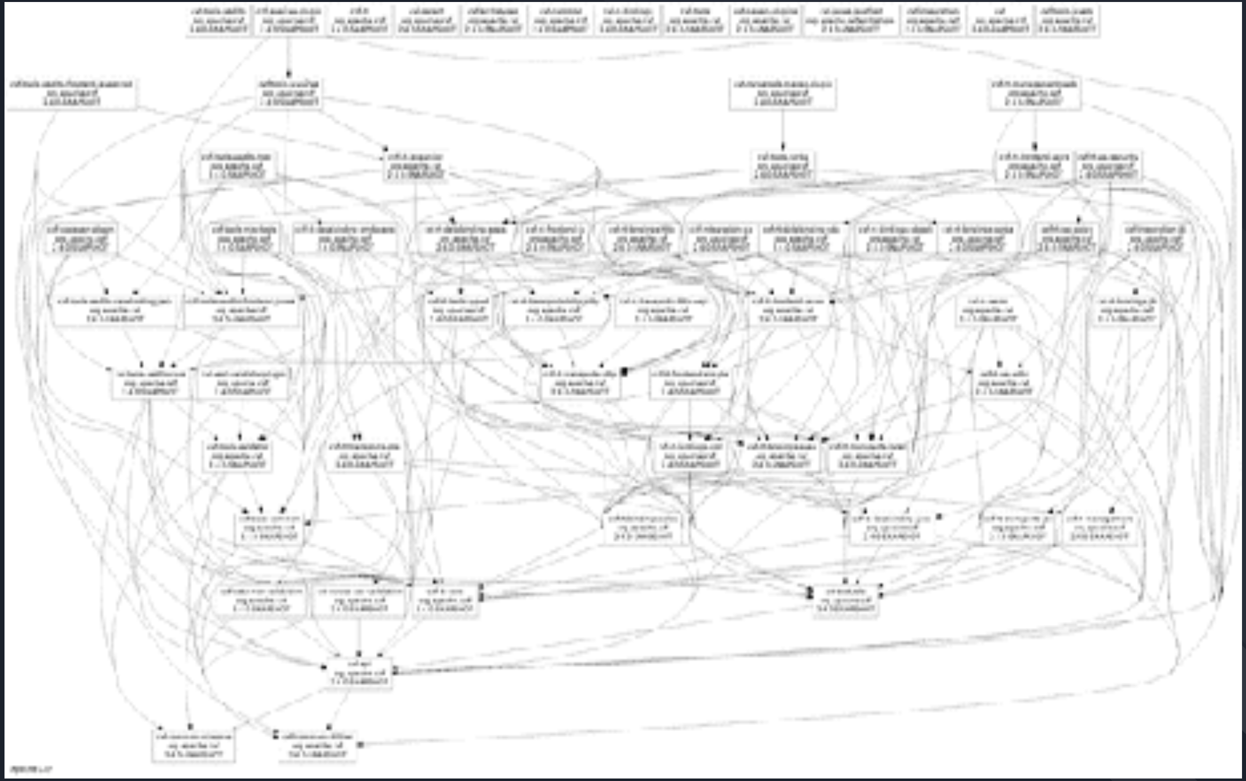




State of Software

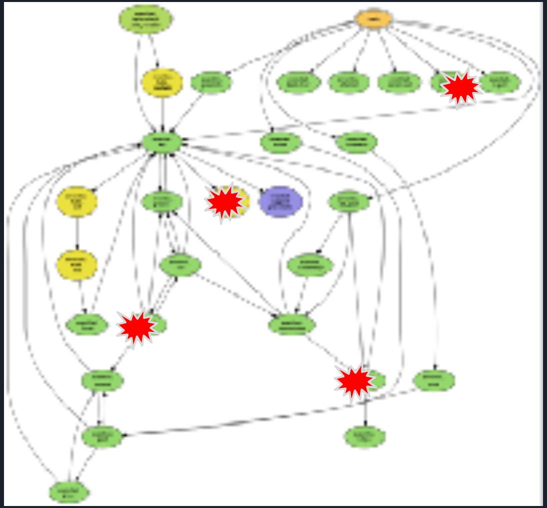
What it really is

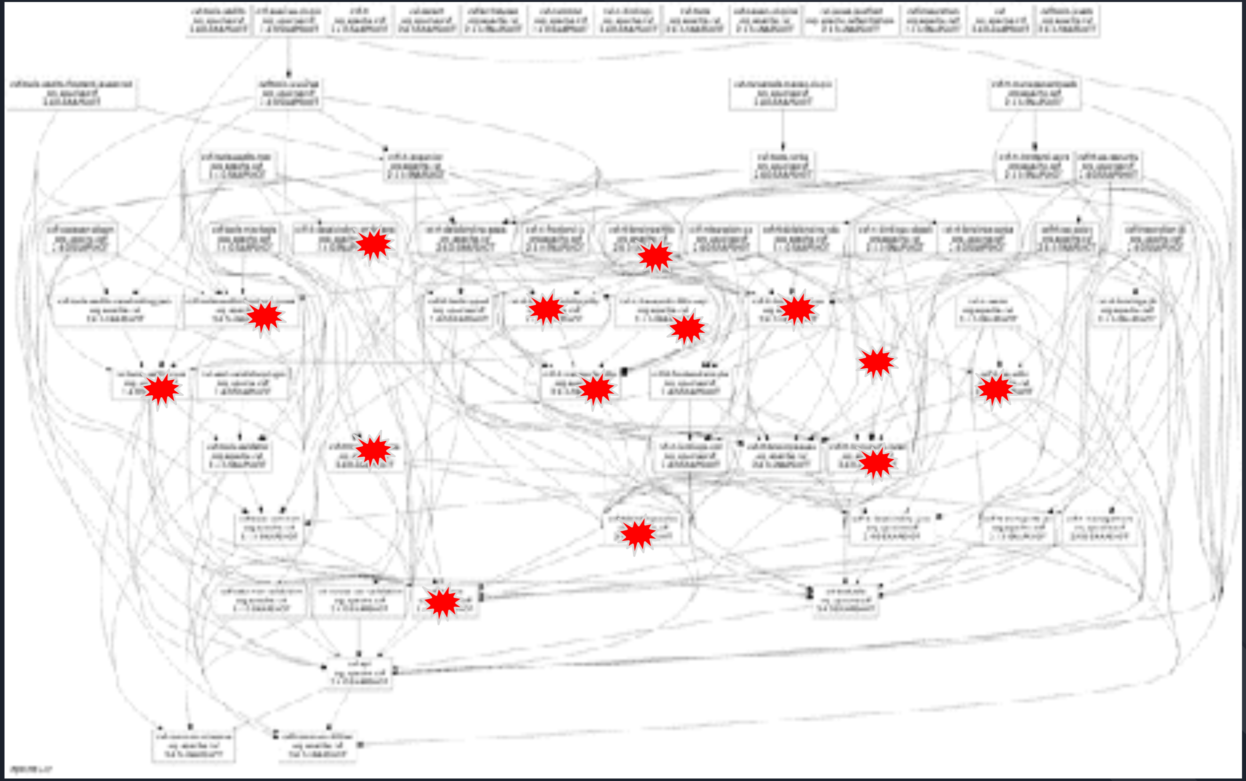




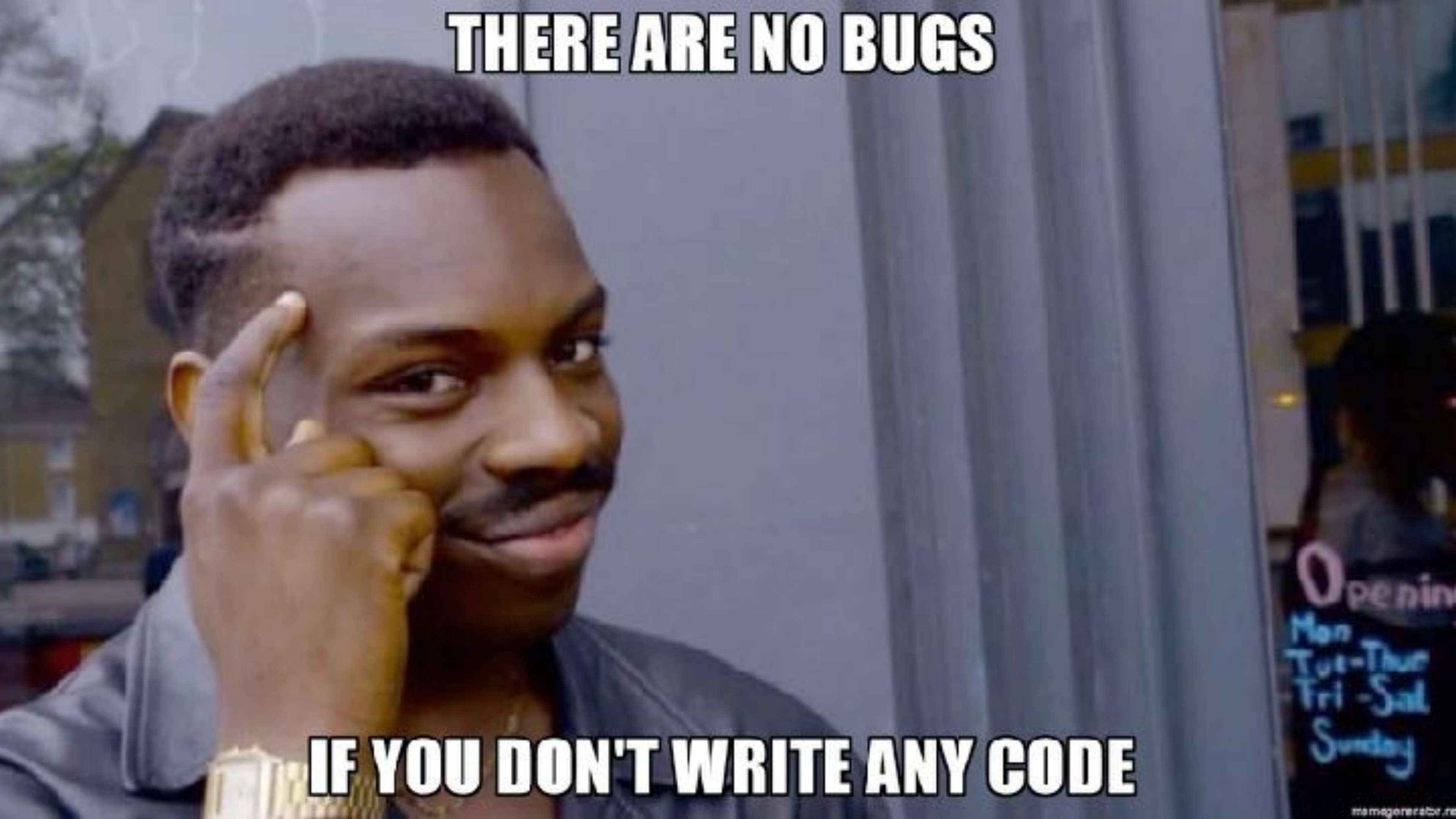
State of Software

And this happens!





As size grows, complexity increases, bugs get introduced


THERE ARE NO BUGS
IF YOU DON'T WRITE ANY CODE
Opening
Mon
Tue-Thur
Fri -Sat
Sunday

Bugs

- Bug cost money

Bugs

- Bug cost Money
- Bugs waste Time

Bugs

- Bug cost Money
- Bugs cost Time
- Bugs kill Morale

HARD LIFE

So what do we do?

1. Prevent
2. Find
3. Fix

Prevention is better than cure, errors are better than bugs

1. Testing
2. Logging
3. Error Handling

Testing

```
python -m unittest
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
python -m unittest tests/test_something.py

nostests --pdb

pytests --pdb
```

```python
import unittest

class TestStringMethods(unittest.TestCase):

   def test_upper(self):
      self.assertEqual('foo'.upper(), 'FOO')

   def test_isupper(self):
      self.assertTrue('FOO'.isupper())
      self.assertFalse('Foo'.isupper())

   def test_split(self):
      s = 'hello world'
      self.assertEqual(s.split(), ['hello', 'world'])
      # check that s.split fails when the separator is not a string
      with self.assertRaises(TypeError):
         s.split(2)


if __name__ == '__main__':
   unittest.main()
```

Logging

```
import logging

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)

# create a file handler
handler = logging.FileHandler('pycon.log')
handler.setLevel(logging.INFO)

# create a logging format
format_str = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
formatter = logging.Formatter(format_str)
handler.setFormatter(formatter)

# add the handler to the logger
logger.addHandler(handler)

# log!
logger.info('Hello Pythonistas')
```

```python
import logging

logger = logging.getLogger(__name__)

logger.debug(msg)
logger.info(msg)
logger.warning(msg)
logger.error(msg, exc_info=True, *args)
logger.exception(msg, *args)
```

Logging

```
import logging
logger = logging.getLogger(__name__)

try:
   1/0
except ZeroDivisionError as e:
   logging.exception(“Divide by zero traceback”)
```

Output:

```
ERROR:root:Divide by zero traceback
Traceback(most recent call last):
   File "/Users/dami/Pycon/main.py", line 2, in < module >
ZeroDivisionError: integer division or modulo by zero
```

Logging

```
import logging
from django.conf import settings

logger = logging.getLogger(__name__)

try:
   1/0
except ZeroDivisionError as e:
   logging.exception("Divide by zero traceback in %s " % settings.VERSION_INFO)
```

Output:

```
ERROR:root:Divide by zero traceback in my_app v1:ea3246404d2384504e052eb1c19f4575a840aad0

Traceback(most recent call last):
   File "<stdin>", line 2, in < module >
ZeroDivisionError: integer division or modulo by zero
```

Logging

```
import logging
from django.conf import settings

class CustomAdapter(logging.LoggerAdapter):
    """
    This example adapter expects the passed in dict-like object to have a
    'connid' key, whose value in brackets is prepended to the log message.
    """
    def process(self, msg, kwargs):
        return '[%s] %s' % (self.extra[‘app_version’], msg), kwargs

logger = logging.getLogger(__name__)
adapter = CustomAdapter(logger, {‘app_version’: settings.VERSION_INFO})

adapter.info("Log message”)
```

Error Handling

Python exceptions hierarchy

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
     +-- ArithmeticError
     |    +-- FloatingPointError
     |    +-- OverflowError
     |    +-- ZeroDivisionError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- ImportError
     |    +-- ModuleNotFoundError
     +-- LookupError
     |    +-- IndexError
     |    +-- KeyError
     +-- NameError
     |    +-- UnboundLocalError
     +-- OSError
     |    +-- TimeoutError
     +-- SyntaxError
     +-- SystemError
     +-- TypeError
     +-- ValueError
```

```
+-- Warning
     +-- DeprecationWarning
     +-- PendingDeprecationWarning
     +-- RuntimeWarning
     +-- SyntaxWarning
     +-- UserWarning
     +-- FutureWarning
     +-- ImportWarning
     +-- UnicodeWarning
     +-- BytesWarning
     +-- ResourceWarning
```

Django exceptions hierarchy

```
BaseException
 +-- Exception
      +-- FieldDoesNotExist
      +-- AppRegistryNotReady
      +-- ObjectDoesNotExist
      +-- MultipleObjectsReturned
      +-- SuspiciousOperation
      |    +-- SuspiciousMultipartForm
      |    +-- SuspiciousFileOperation
      |    +-- DisallowedHost
      |    +-- DisallowedRedirect
      |    +-- TooManyFieldsSent
      |    +-- RequestDataTooBig
      +-- PermissionDenied
      +-- ViewDoesNotExist
      +-- ImproperlyConfigured
      +-- FieldError
      +-- ValidationError
```

Custom exceptions hierarchy

```
BaseException
 +-- Exception
      +-- UserDoesNotExist
      +-- InvalidPasswordError
      +-- SocialNetworkError
      |    +-- TwitterTimeoutError
           +-- TwitterUnexpectedResultError
      |    +-- FacebookTimeoutError
      |    +-- FacebookUnexpectedResultError
      +-- PaymentError
      |    +-- PaymentGatewayTimeout
      |    +-- SuspiciousTransaction
      |    +-- BitcoinNodeNotFound
      |    +-- InvalidTransactionError
      |    +-- InvalidTransactionError
```

Be Assertive!

```
import logging
logger = logging.getLogger(__name__)


def do_something_with_a_resource():
    result = get_twitter_resource()
    try:
        assert isinstance(result, dict)
        do_something_with_result_dict(result)
    except AssertionError:
        raise TwitterUnexpectedResultError('Failure')
```

Exception Chaining, don’t lost the trace!

```
import logging
logger = logging.getLogger(__name__)


def do_something_with_a_resource():
   result = get_twitter_resource()
   try:
       assert isinstance(result, dict)
       do_something_with_result_dict(result)
   except AssertionError as e:
       raise TwitterUnexpectedResultError('failed') from e

# The above exception was the direct cause of the
following exception:
```

Code Contracts

```
from covenant import pre, post

# throws PreconditionViolationError
@pre(lambda x: x < 10)
def some_function(x):
    return 10 - x

# throws a PostconditionViolationError
@post(lambda r, x: r < x)
def some_function(x):
    return x - 20
```

```
# https://legacy.python.org/dev/peps/pep-0316/
# https://github.com/kisielk/covenant
```

Finding Bugs

To print or to pdb

pdb

1. Extensible
2. Customisable
3. Flexible
4. Powerful

You command, I obey!

```
Documented commands (type help <topic>):
========================================
EOF    bt                  cont      enable    jump  pp            run
           unt
a      c                   continue  exit      l          q         s
           until
alias  cl                  d         h                    list      quit
           step      up
args   clear               debug     help      n          r
tbreak     w
b      commands  disable   ignore    next  restart        u
whatis
break  condition down      j                   p          return
unalias    where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
```

withdrawls.py

```python
from handlers import (handle_withdrawal, handle_low_account, handle_overdraft)

account_balances = [2324, 0, 70, 409, -2]
account_details = {'name': 'Damilare Onajole', 'number': 9027303872}

def withdraw_funds():
   print("Welcome {}".format(account_details['name']))

   import pdb; pdb.set_trace()

for balance in account_balances:
       if balance < 0:
           handle_overdraft(balance)
       elif balance == 0:
           handle_low_account(balance)
       else:
           handle_withdrawal(balance)

withdraw_funds()
```

$ python withdrawls.py

```
Welcome Damilare Onajole
> /Users/dami/Talks/withdrawal.py(12)withdraw_funds()
-> for balance in account_balances:
(Pdb) l
  7
  8
  9     def withdraw_funds():
 10         print("Welcome {}".format(account_details['name']))
 11         import pdb; pdb.set_trace()
 12  ->     for balance in account_balances:
 13             if balance < 0:
 14                 handle_overdraft(balance)
 15             elif balance == 0:
 16                 handle_low_account(balance)
 17             else:
(Pdb)
```

```
$ python withdrawal.py
Welcome Damilare Onajole
> /Users/dami/Talks/withdrawal.py(12)withdraw_funds()
-> for balance in account_balances:
(Pdb) b 18
Breakpoint 1 at /Users/dami/Talks/withdrawal.py:18
(Pdb) c
> /Users/dami/Talks/withdrawal.py(18)withdraw_funds()
-> handle_withdrawal(balance)
(Pdb) s
--Call--
> /Users/dami/Talks/handlers.py(11)handle_withdrawal()
-> def handle_withdrawal(balance):
(Pdb) w
 /Users/dami/Talks/withdrawal.py(20)<module>()
-> withdraw_funds()
 /Users/dami/Talks/withdrawal.py(18)withdraw_funds()
-> handle_withdrawal(balance)
> /Users/dami/Talks/handlers.py(11)handle_withdrawal()
-> def handle_withdrawal(balance):
(Pdb) p balance
2324
```

handlers.py

```
def handle_overdraft(balance):
   print("Account balance of {} is below 0; add funds now."
         .format(balance))

def handle_low_account(balance):
   print("Account balance of {} is equal to 0; add funds soon."
         .format(balance))

def handle_withdrawal(balance):
   print("Account balance of {} is above 0.".format(balance))
```

handlers.py

```
> /Users/dami/Talks/handlers.py(11)handle_withdrawal()
(Pdb) pp account_details
*** NameError: name 'account_details' is not defined
(Pdb) u
> /Users/dami/Talks/debugging_python_applications/withdrawal.py(18)withdraw_funds()
-> handle_withdrawal(balance)
(Pdb) pp account_details
{'name': 'Damilare Onajole', 'number': 9027303872}
(Pdb)
```

Moving around

- n: Continue execution until the next line in the current function is reached or it returns
- s: Execute the current line and stop in a function that is called or in the current function.
- c: Continue execution and only stop when a breakpoint is encountered.
- b (or b <num>: List all breaks or set a breakpoint at this <num> in the current file.
- until: Continue execution until the line with a number greater than the current one is reached. With a line number argument, continue execution until a line with a number greater or equal to that is reached.

Jumping about

- w: Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame, which determines the context of most commands.

- u: Move the current frame count (default one) levels up in the stack trace (to an older frame).

- d: Move the current frame count (default one) levels down in the stack trace (to a newer frame).

Introspect

- p: Print the value of an expression.
- pp: Pretty-print the value of an expression.
- a: Print the argument list of the current function
- l: List 11 lines around the current line or continue the previous listing.
- ll: List the whole source code for the current function or frame.
- alias: Create an alias called name that executes command.
- unalias: Delete the specified alias

Extend

```
import pdb

class Epdb(pdb.Pdb):

  def store_old_history(self):
      ...
  def restore_old_history(self):
      ...
  def read_history(self, storeOldHistory=False):
      ...
  def save_history(self, restoreOldHistory=False):
      ...
  def do_savestack(self, path):
      ...
  def do_mailstack(self, arg):
      ...
  def do_printstack(self, arg):
      ...
  def complete(self, text, state):
```

```
# Put this in ~/.pdbrc

import rlcompleter
import pdb
pdb.Pdb.complete = rlcompleter.Completer(locals()).complete

# Print a dictionary, sorted. %1 is the dict, %2 is the prefix for the names.
alias p_ for k in sorted(%1.keys()): print "%s%-15s= %-80.80s" % ("%2",k,repr(%1[k]))

# Print the member variables of a thing.
alias pi p_ %1.__dict__ %1.

# Print the member variables of self.
alias ps pi self

# Print the locals.
alias pl p_ locals() local:

# Next and list, and step and list.
alias nl n;;l
alias sl s;;l
```

Other tools worth mentioning

- Django Debug Toolbar
- Django Shell
- django-extensions
- Weukzeug Debugger
- code.Interact
- Sentry

Fix It

Write Unit test

- Clearly reproduce your bugs and create exceptions for them, errors are better than bugs

Write you fix

- Fix your code
- Push

Any Questions?