Quality Assurance and Testing

The Quest for Nice Things!

One-slide summary

- **Testing** is a fundamental way that we ensure our software is correct.
- There are numerous methods of testing, such as **unit testing**, **regression testing,** and **integration testing.**
- We can use **mocking** to test things that are otherwise difficult to test.
- Testing effectively requires planning.

Boring Technical Definition

Quality assurance - *The maintenance of a desired level of quality in a service or product, especially by means of attention to every stage of the process of delivery or production.*

- Oxford English Dictionary

Motivation

- Programs should be understandable and maintainable.
- Programs should “do the right thing.”

Maintainability

- How do we make sure that software is easy to maintain?
  - Human code review
  - Static analysis tools and linters
  - Use established programming idioms and design patterns
  - Follow your team’s coding standards
- (More on this in future lectures)

Do the right thing?

- Behave according to specification
  - Foreshadowing: How do we come up with a good spec?
- “Don’t do bad things”
  - Security issues, crashes, Blue Screen of Death
  - If some amount of failure is inevitable, do we handle it well?
- Robustness against regression
  - Do “fixed” bugs sneak back into the code?

Do the right thing: How?

- How about we just write a program that tells us if our software is correct?

Do the right thing: How?

- How about we just write a program that tells us if our software is correct?
  - Static analysis, linting, type checking, etc. can approximate

Practical Solution

TESTING

Testing in EECS Courses

- EECS 183 and 482:
  - 1 **main()** function == 1 test
  - Grading process:
    - For each test:
      - Run test against correct solution, save output
      - For each buggy solution:
        - Run test against buggy solution, diff output with result from correct solution
        - If output different, bug exposed

Testing in EECS Courses

- EECS 281:
  - 1 input file == 1 test
  - Grading process:
    - For each test:
      - Pipe input to correct solution, save output
      - For each buggy solution:
        - Pipe input to buggy solution, diff output with result from correct solution
        - If output different, bug exposed

Testing in EECS Courses

- EECS 280:
  - 1 function with **asserts()** == 1 test
  - Grading process:
    - For each test:
      - Run test against correct solution, throw out the test if it fails
      - For each buggy solution:
        - Run test against buggy solution
        - If assertion fails, bug exposed

Exercise: Testing in EECS Courses

- With your neighbor, discuss the pros and cons of each method of testing.
  - Summary:
    - 183/482: 1 **main()** function == 1 test, output diff
    - 281: 1 input file == 1 test, output diff
    - 280: 1 function with **asserts()** == 1 test, assertion failure == test failure

Exercise: Testing in EECS Courses

- The main difference:
  - For 183/281/482, students write program *inputs*, but **not** *expected outputs*.
  - For 280, students write inputs **and** expected outputs.
- For 183/281/482, goal is essentially high coverage.
- In real life, you probably don't have an already-correct implementation of your program…
- Note: Testing with random inputs (Fuzz testing) helps detect bugs of the the "bad things" variety (segfaults, memory errors, crashes, etc.)

Testing Buzzwords!

- Regression testing
- Unit testing
- xUnit
- Integration testing
- Mocking

Regression testing (in 1 slide!)

- Ever have one of those “I swear I fixed this bug!” moments?
  - Maybe you did, but then someone broke it again…
  - This is called a **regression** in the code.

- When you fix a bug, add a test that **specifically exposes that bug**.
  - This is called a **regression test.**

Regression testing (in 1 slide-ish!)

```
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 42
```

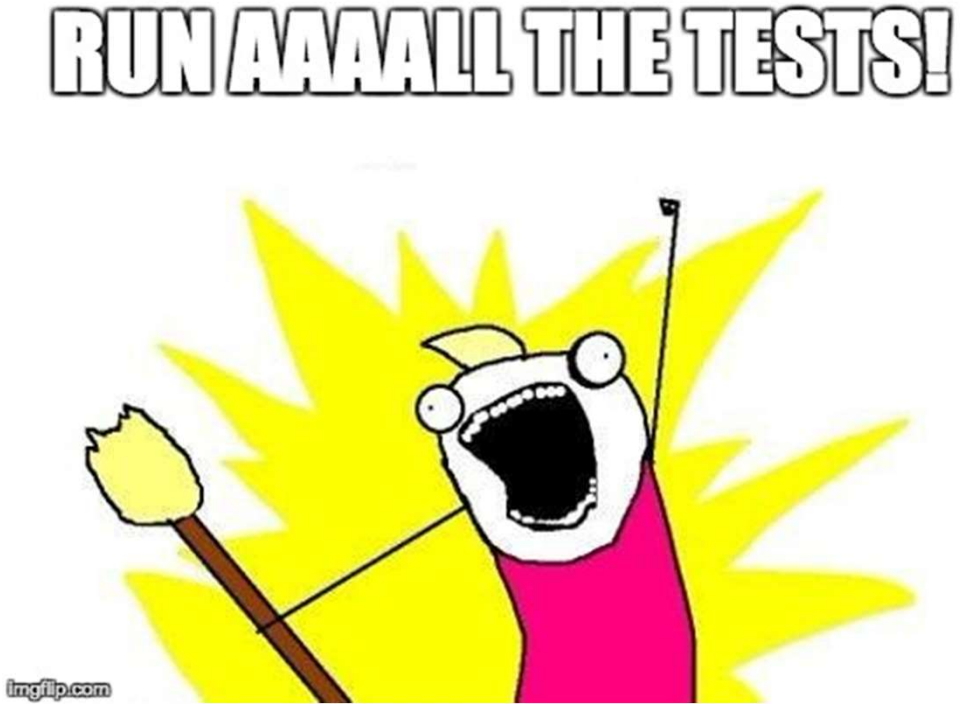
https://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered/482129#482129

Unit Testing Frameworks

- Often based on SUnit (Smalltalk), written by Kent Beck.
- JUnit, Python unittest, C++ googletest, etc.
- Collectively referred to as **xUnit**.

xUnit Features

- Test case discovery
- Test case runner
  - Choose which tests to run



xUnit Features

- Test case
  - A piece of code (usually a method) that establishes some preconditions, performs an operation, and asserts postconditions.
- Test fixture
  - Specify code to be run before/after each test case.
  - Each test is run in a “fresh” environment.
- Special assertions
  - Assert postconditions, helpful failure messages

Python unittest Example

```
import unittest
class NiceThing:
    def __init__(self, num_spams):
        self.num_spams = num_spams
    def zap(self):
        return self.num_spams + 42


class NiceThingTestCase(
         unittest.TestCase):
    def setUp(self):
        self.nice_thing = NiceThing(0)
    def test_zap(self):
        self.assertEqual(45, self.nice_thing.zap())

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

```
$ python3 unit_test_demo.py
F
======================================================================
FAIL: test_zap (__main__.NiceThingTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unit_test_demo.py", line 11, in test_zap
    self.assertEqual(45, self.nice_thing.zap())
AssertionError: 45 != 42

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Python unittest Example

- We'll cover this in more detail in discussion.
- See the Python unittest documentation for additional information:
  - https://docs.python.org/3/library/unittest.html

Unit Testing

- Test features in isolation
  - In the coding example, our test for **zap()** tested ***only*** the **zap()** method.
  - When a test fails, easier to locate the error.
- Tests are small
  - Small tests are easier to understand.
- Tests are fast
  - Slow tests are more expensive to run frequently.

Unit Testing

- Remember the Euchre project from EECS 280?
  - Card, Pack, and Player classes + top-level “play Euchre” application.

- Let’s say you wrote Card, Pack, and Player without testing, and then wrote “play Euchre.”
  - What do you do when you find a bug in “play Euchre”?
    - Wish you had used Test-Driven Development...

Test-Driven Development (in 1 slide)

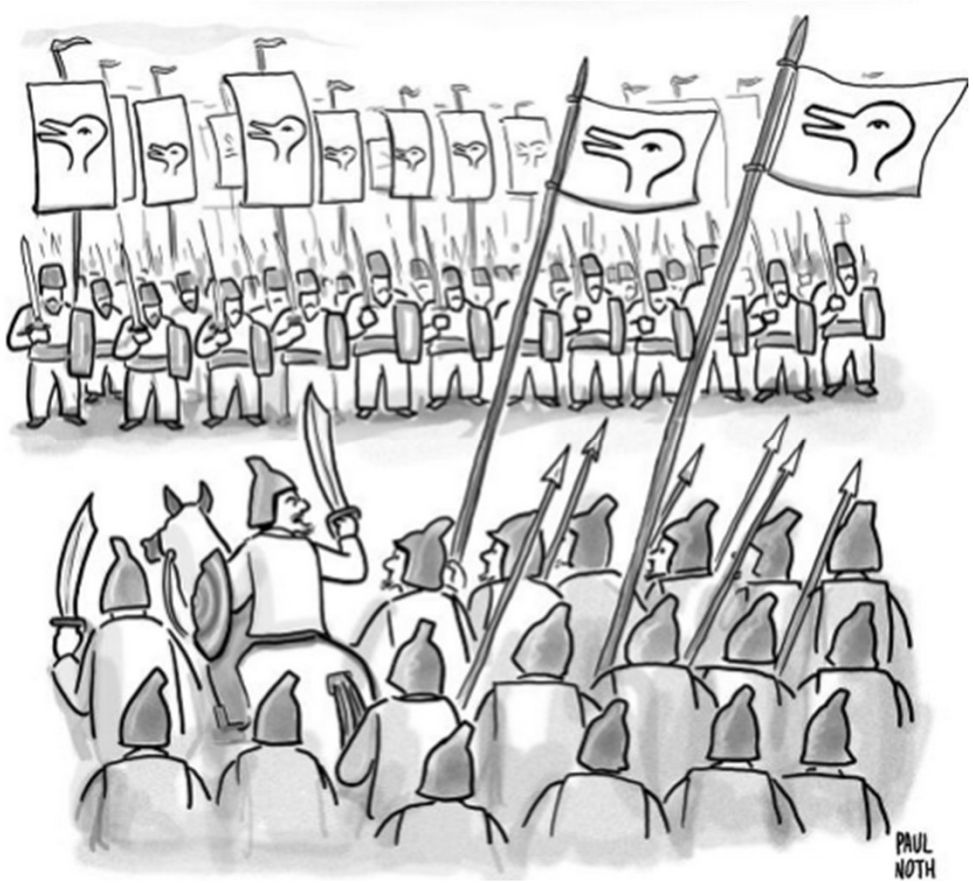
1. Write a unit test.
    a. When you run the test, it should fail.
2. Write the code that the test case tests.
3. Run ALL the tests.
    a. Fix anything that broke, repeat step 3 if any tests failed.
4. Go back to step 1.

Unit Testing vs. Integration Testing

- Aren’t those “unit tests” for Pack and Player actually integration tests???

Unit Testing vs. Integration Testing

“There can be no peace until they renounce their Rabbit God and accept our Duck God.”
- New Yorker cartoon



Unit Testing vs. Integration Testing

- Once you've unit-tested an ADT, you can build on top of it and write unit tests for new ADTs at a higher level of abstraction.
  - This also promotes modular, decoupled design.

"Does that mean that our tests that rely on integers aren't really unit tests? No. We can treat integers as a given and we do. Integers have become part of the way we think about programming." - Kent Beck https://www.facebook.com/notes/kent-beck/unit-tests/1726369154062608/

Integration Testing

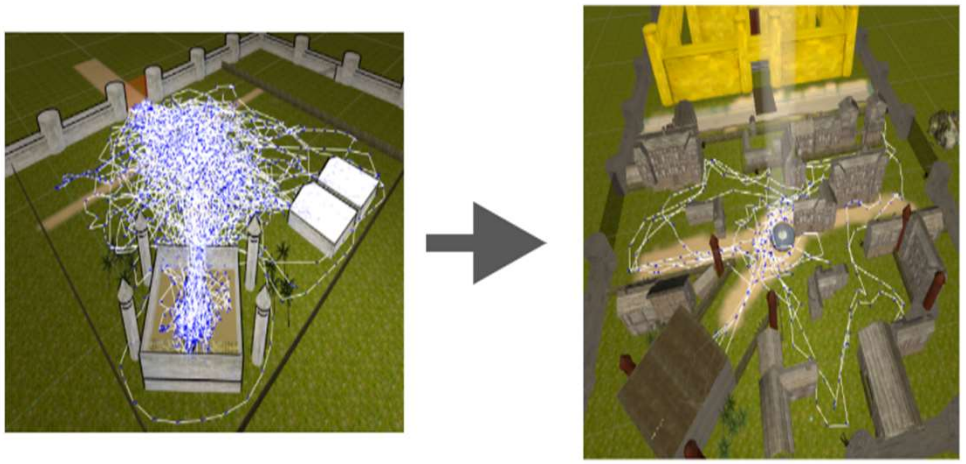
- Any feature will work in isolation.
- What happens when we try to put our unit-tested ADTs together?
- Does our application work from start to finish?
  - “End-to-end” testing

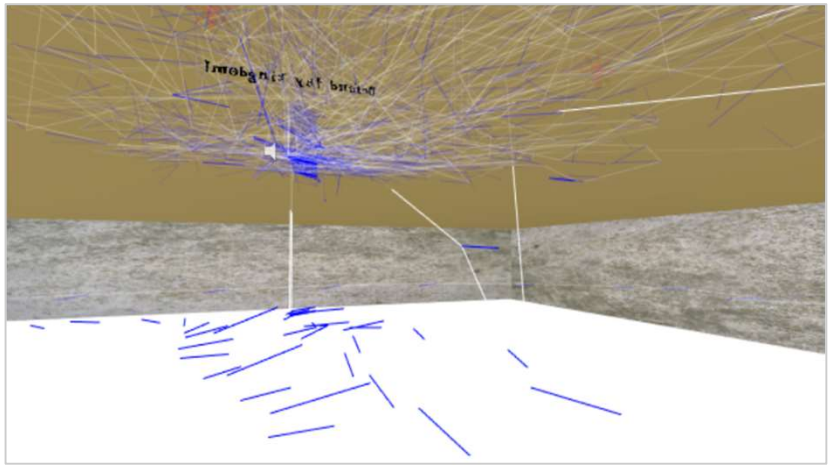
Integration Testing: Examples

- How? Depends on the application.
- EECS classes:
  - Run main program with input file, diff output.
- Web/GUI application:
  - Use a testing framework that lets you simulate user clicks and other input.
- Video games:
  - If you're really fancy, write an AI to play your game!
    - Bayonetta 2: https://www.platinumgames.com/official-blog/article/6968
    - Cloudberry Kingdom: https://www.gamasutra.com/view/feature/170049/how_to_make_insane_procedural_.php

Other Creative Testing Methods

- Gaze-detecting glasses: https://www.tobiipro.com/fields-of-use/user-experience-interaction/game-usability/
- Record everywhere the player goes.

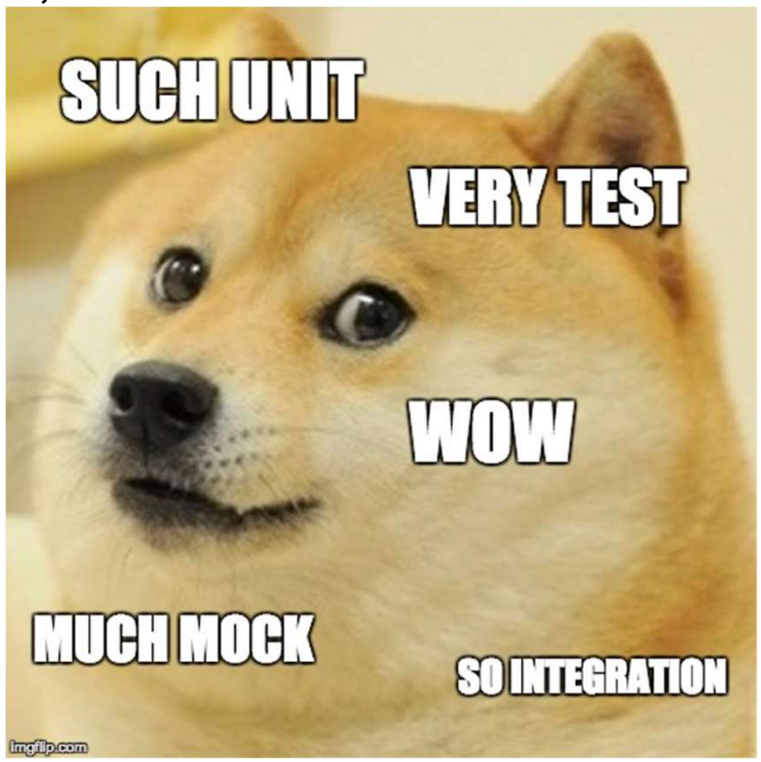




Special thanks to Austin Yarger for sending me these “testing in video game dev” examples.

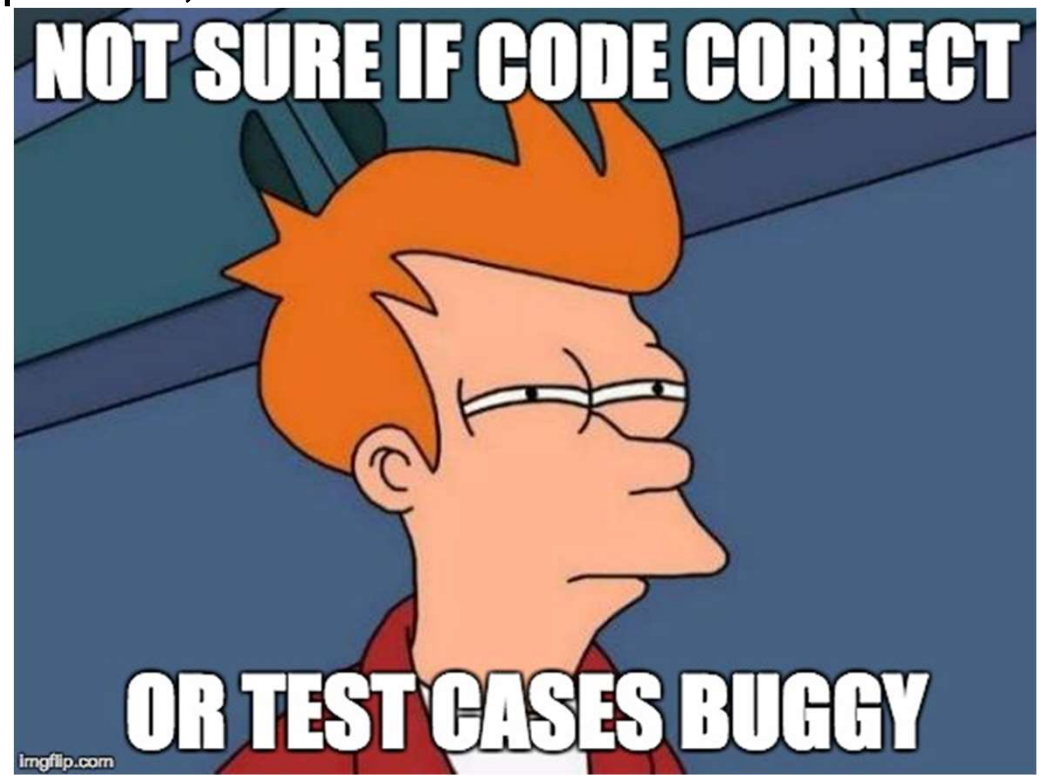
Break Time

I’m not a trivia person, so here’s another meme:



Break Time

I’m not a trivia person, so here’s another meme:



Mocking: Testing Hard-to-test Things

- What if we want to write unit (or integration) tests for some ADT, but the ADT has expensive dependencies?
- Exercise: Write down 2 examples of things that are hard to test because of their dependencies or other factors.

Scenario 1: Web API Dependency

- We're writing a single-page web application, but the web API we'll be using hasn't been implemented or costs money to use.
- We want to be able to write our frontend (website) code without waiting on the server-side devs or spending a bunch of money.
- What should we do?

Scenario 1: Web API Dependency

- Solution: Write our own “fake” version of the API.
- For each method that the API exposes, write a substitute for it that just returns some hard-coded data.
  - Why does this work? (Which concept(s) from 280?)
- I’ve used this technique to design parts of the autograder.io website.

Scenario 2: Uncommon Error Handling

- We're writing some code where certain kinds of errors will occur sporadically in production, but never in development.
  - e.g. Out of memory, network connection lost
- Can we use the same technique that we did for the web API?
  - i.e. Write a fake version of the function and substitute it in?
  - That sounds like a pain to do manually...

Scenario 2: Uncommon Error Handling

- Solution: Mocking libraries
- Provides a way to dynamically (at runtime) substitute objects, functions with fake versions.
  - For one test, we could use a mocking library to force a line of code *inside our function* to throw an exception when it's reached.

Scenario 2: Uncommon Error Handling

```
import unittest
from unittest import mock
def defrangulate():
    # Do some stuff that might cause an error
    pass

def spammify():
    try:
        defrangulate()
        return True
    except MemoryError:
        return False
```

Scenario 2: Uncommon Error Handling

```
# Same file as previous slide
class SpammifyTestCase(unittest.TestCase):
    def test_spammify_defrangulage_runs_out_of_memory(self):
        def throw_memory_error():
            raise MemoryError('WAAAAALUIGI')

        with mock.patch('__main__.defrangulate', throw_memory_error):
            self.assertFalse(spammify())


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

Scenario 2: Uncommon Error Handling

- Solution: Mocking libraries
- Provides a way to dynamically (at runtime) substitute objects, functions with fake versions.
  - For one test, we could use a mocking library to force a line of code *inside our function* to throw an exception when it's reached.
- Easier in languages with runtime reflection (Python, Java)
  - googletest used to require a special base class to enable mocking, now it uses macro shenanigans.

More fun with mocking libraries

- Other things you can use mocking libraries for:
  - Track how many times a function was called and/or with what arguments (“spying”).
  - Add or remove side effects (exceptions are considered a side effect by mocking libraries).
  - Test locking in multithreaded code (force a thread to stall after acquiring a lock).

More fun with mocking libraries

- Autograder.io example:
  - The code that grades submissions has retry logic.
  - Need to force errors to happen during testing.
  - Remove time delay between retries to speed up tests.

Downsides of Mocking

- Test cases that use mocking can be very fragile
  - What if someone moves or removes the call to **defrangulate()** that we mock.patch'd earlier?
- Good integration tests are a necessity
  - If we mock dependencies, we need to be extra careful that our ADTs play nicely together.
- Learning curve for mocking libraries
  - In Python, can be hard to determine the correct value for 'path' in mock.patch.
  - Error messages can be cryptic.

QA as Part of Dev Process

- How can we assure quality before, during, and after we write code?
- What if we don’t have enough resources?

Case Study: Autograder.io (my work)

- Web application with server and website code.
- 2014-2016, me.
- 2016-present me + 1-2 students.

- What are our QA goals?
- What do we do when reality prevents us from achieving those goals?

QA for Autograder.io: Server Code

1. On paper, design data abstractions/database tables & how they interact.
2. Write stubs w/ documentation (i.e. docstrings).
3. Write tests & run them.

   (Currently 1250 test cases, takes ~40 min to run.)
4. Implement code being tested.
5. Bugs, feature requests stored in issue tracker.
6. Refactor code to improve design, satisfy changing requirements.

QA for Autograder.io: Website Code

- Highly prototypical, requirements, tools often in flux
  - v1 (2015): JS + JQuery + a dinky template rendering library
  - v2 (2016): Dart + Angular 2 Beta
  - v3 (2016 - present): TypeScript + Angular 2
  - v4 (in development): TypeScript + Vue
- Project/build configuration was (is) a nightmare.
- Testing GUIs is HARD.
  - Tests often fragile, frameworks relatively new
  - Ask a 485 prof how much effort went into autograding their projects ;)
- Much harder to add tests to existing code than new code.

Risk Assessment for Autograder.io

Part of System	Worst-Case Failure(s)
Running student code (security)	`sudo rm -rf --no-preserve-root /`
Database	Data loss, Student given the wrong grade
Website Submit Page	Bug prevents student from submitting
Website Admin	Prof can’t edit some settings, they get annoyed

Other Thoughts

- Danger! It's easy for prototype code to wind up as the final product.
- Conveniently, I have 3 months every year when nobody uses autograder.io.
- Each part of autograder.io has been rewritten from scratch at least once.
- For me, UI design is the biggest challenge.

Further Watching

https://www.youtube.com/watch?v=ntpZt8eAvy0

https://www.youtube.com/watch?v=on7endO4lPY