

# CSCA08 FALL 2017

## WEEK 11 - EXCEPTIONS & UNIT TEST

Brian Harrington & Marzieh Ahmadzadeh

University of Toronto Scarborough

November 20 - 24, 2017



UNIVERSITY OF  
**TORONTO**  
SCARBOROUGH

# ADMIN

- A2
- Week 12

# EXCEPTIONS

- What happens when it all goes wrong?
- We get an error
- Or rather... the user gets an error
- We can try to avoid them
  - Not always possible/practical
  - Sometimes better to deal with the errors after they occur

# TRY/EXCEPT

- A simple fix: Catch the error.

```
try:
```

```
    Block of code
```

```
except:
```

```
    Code to execute in case of an error
```

# TRY/EXCEPT

- Can also catch only exceptions of a certain type

```
try:
```

```
    Block of code
```

```
except ExceptionType:
```

```
    Code to execute in case of an ExceptionType error
```

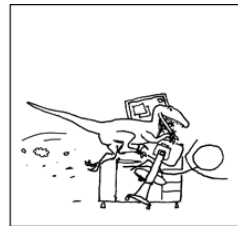
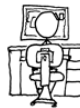
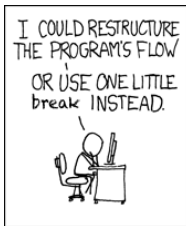
## ELSE AND FINALLY

- `else`: allows us to continue code after the `try` block
- `finally`: code that will be executed whether an exception is raised or not

# RAISE

- `raise`: raises a new exception
- Why would we want to do this?
- Fix leaky abstractions

# BREAK





# DOCTEST

- We've already been using this
- Good for quick sanity checks
- Tedious to do more than the basics
- Somewhat difficult for sets/dictionaries
- Doesn't work at all for i/o or OOP
- Really just testing "outside" the code

```
import unittest
import example_functions as func
class TestCommonChar(unittest.TestCase):
    def test_identical_single_char(self):
        self.assertEqual(
            func.common_chars('a', 'a'),
            (1,1),
            "identical single char"
        )
unittest.main(exit = False)
```

- One class per function to test.
- One method per test case (name must start with 'test')
- assertEquals method (fails if first two parameters aren't equal)
  - Example:
  - Return value of the function
  - Expected return value
  - Message for if/when error occurs

# BLACK BOX TESTING

- Imagine function as a “black box”
  - Can't see in/out
  - Can only see what goes in/what comes out
  - Try to cover all major test areas + boundary cases
  - Testing that could be done by external user

# WHITE BOX TESTING

- Now we can “see inside” the box
  - Shouldn't it be “clear box”?
- Can test for weaknesses specific to implementation details
- More focused testing
- When implementation details change, tests must change also
- Testing done by internal user

# COVERAGE TESTING

- Try to cover all possible scenarios
- Exhaustive testing too tedious/difficult
- Break up “test-space” into areas
- Pick representative examples from each area
- Pick examples from boundaries between areas

# ADVERSARIAL TESTING

- Try to break your (or preferably someone else's code)
- Usually white box
- Often monetary incentives
- Motivated to find bugs

# REGRESSION TESTING

- When you make a change, check that you haven't introduced a bug to other code
- Built new test cases on top of old, run everything
- Costly, but effective
- Individual cases can be derived through other methods

# TESTING LEVELS

- Unit - Test individual components
- Integration - Putting components together
- System - System as a whole
- Acceptance - Testing with users
- Release - Testing in the real world
  - alpha (select group, expecting buggy code)
  - beta (larger group, expect mostly working code)
  - full-release (it better be working by this point)
- Rule of 10: As we move ahead 1 level, difficulty/cost of repairing a logic error multiplies by  $\sim 10x$