COMP1531

1.3 - Testing, Teamwork, Project

C-Style Testing

How did you test in COMP1511?

ctest.c

```c
#include <stdio.h>
#include <assert.h>

double sum(double a, double b) {
    return a + b;
}

int main() {
    assert(sum(1, 2) == 3);
    assert(sum(2, 2) == 4);
    assert(sum(3, 2) == 5);
    printf("All tests passed\n");
}
```

C-Style Testing

Let's first look at python functions

```
1 double sum(double a, double b) {
2     return a + b;
3 }
```

```
1 def sum(a, b):
2     return a + b
```

Q. What are the key differences?

C-Style Testing

Let's first look at python functions

```
1 double sum(double a, double b) {
2     return a + b;
3 }
```

```
1 def sum(a, b):
2     return a + b
```

Q. What are the key differences?

- No semi-colons
- No braces
- No typing
- "def" to say define function

C-Style Testing

Q. How would we test this python function?

```
1 def sum(a, b):
2     return a + b
```

C-Style Testing

Q. How would we test this python function?

cstyletest.c

```
1 def sum(a, b):
2     return a + b
3
4 assert sum(1, 4) == 3
```

```
:~/teaching/cs1531/19T3-lectures/week1$ python3 cstyletest.py
Traceback (most recent call last):
  File "cstyletest.py", line 4, in <module>
    assert sum(1, 2) == 3
AssertionError
```

C-Style Testing

Let's clean this up and wrap it in a function, though!

```
1 def sum(a, b):
2     return a + b
3
4 def testSmallNumbers():
5     sum(1, 4) == 3
6
7 testSmallNumbers()
```

Basic Python testing

Let's take a look at **pytest**

What is pytest?

- pytest is a library that helps us write small tests, but can also be used to write larger and more complex tests
- pytest comes with a binary that we run on command line
- pytest detects any **function** prefixed with **test** and runs that function, processing the assertions inside

pytest - basic

test1_nopytest.py

```
1 def sum(x, y):
2     return x * y
3
4 def test_sum1():
5     assert sum(1, 2) == 3
6
7 test_sum1()
```

```
1 $ python3 test1_nopytest.py
```

test1_pytest.py

```
1 import pytest
2
3 def sum(x, y):
4     return x * y
5
6 def test_sum1():
7     assert sum(1, 2) == 3, "1 + 2 == 3
```

```
1 $ pytest-3 test1_pytest.py
```

pytest - more complicated

A more complicated test

test_multiple.py

```
 1 import pytest
 2
 3 def sum(x, y):
 4     return x + y
 5
 6 def test_small():
 7     assert sum(1, 2) == 3, "1, 2 == "
 8     assert sum(3, 5) == 8, "3, 5 == "
 9     assert sum(4, 9) == 13, "4, 9 == "
10
11 def test_small_negative():
12     assert sum(-1, -2) == -3, "-1, -2 == "
13     assert sum(-3, -5) == -8, "-3, -5 == "
14     assert sum(-4, -9) == -13, "-4, -9 == "
15
16 def test_large():
17     assert sum(84*52, 99*76) == 84*52 + 99*76, "84*52, 99*76 == "
18     assert sum(23*98, 68*63) == 23*98 + 68*63, "23*98, 68*63 == "
```

pytest - prefixes

If you just run

$ pytest-3

Without any files, it will automatically look for any files in that directory in shape:

- test_*.py
- *_test.py

pytest - particular files

You can run specific functions without your test files with the **-k** command. For example, we if want to run the following:

- **test_small**
- **test_small_negative**
- ~~test_large~~

We could run

$ pytest-3 -k small

or try

$ pytest-3 -k small -v

pytest - markers

We can also use a range of **decorators** to specify tests in python:

```python
import pytest

def pointchange(point, change):
        x, y = point
        x += change
        y += change
        return (x, y)

@pytest.fixture
def supply_point():
        return (1, 2)

@pytest.mark.up
def test_1(supply_point):
        assert pointchange(supply_point, 1) == (2, 3)

@pytest.mark.up
def test_2(supply_point):
        assert pointchange(supply_point, 5) == (6, 7)
```

```python
@pytest.mark.up
def test_3(supply_point):
        assert pointchange(supply_point, 100) == (101, 102

@pytest.mark.down
def test_4(supply_point):
        assert pointchange(supply_point, -5) == (-4, -3)

@pytest.mark.skip
def test_5(supply_point):
        assert False == True, "This test is skipped"

@pytest.mark.xfail
def test_6(supply_point):
        assert False == True, "This test's output is muted
```

pytest - more

There are a number of tutorials online for pytest.
This is a very straightforward one.

importing and modules

calmath.py

```python
def daysIntoYear(month, day):
    total = day
    if month > 0:
        total += 31
    if month > 1:
        total += 28
    if month > 2:
        total += 31
    if month > 3:
        total += 30
    if month > 4:
        total += 31
    if month > 5:
        total += 30
    if month > 6:
        total += 31
    if month > 7:
        total += 30
    if month > 8:
        total += 31
    if month > 9:
        total += 30
    if month > 10:
        total += 31
    return total

def quickTest():
    print(f"month 0, day 0 = {daysIntoYear(0,0)}")
    print(f"month 11, day 31 = {daysIntoYear(11,31)}")

#if __name__ == '__main__':
#    quickTest()

quickTest()
```

importto.py

```python
import sys

import calmath

if len(sys.argv) != 3:
    print("Usage: importto.py month dayofmonth")
else:
    print(calmath.daysIntoYear(int(sys.argv[1]), \
                               int(sys.argv[2])))
```

Teamwork

- **Why** do we want to work in teams?

Teamwork

- What are **benefits** of working in teams?

Teamwork

- What are **downsides** of working in teams? (as opposed to by yourself)

Teamwork

- What are **downsides** of working in teams? (as opposed to by yourself)

Teamwork

You do not scale.

David Whiteing, ex-CIO of Combank

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

Teamwork

Scenario

Your 4th group member hasn't turned up for 1.5 weeks and isn't replying to their emails. What do you do?

Teamwork

Scenario

Your group is split into two pairs of people. One pair wants to build the navigation bar at the top of the page. One wants it on the side. How do you decide what to do?

Team-based Project

The project is a 9 week timeframe where your team has been contracted as backend developers to provide a web server for a client.

- The front end has been outsourced to another contractor, and you've been told it will not be completed until mid-October
- Specifications may change over that period

Team-based Project

Project schedule

Week	Topic
1	
2	Iteration 1 released
3	
4	Iteration 1 review; Iteration 2 released
5	
6	
7	Iteration 2 review; Iteration 3 released
8	
9	
10	Iteration 3 review

- Iteration 1:
  - Requirements
  - Testing
- Iteration 2:
  - Web-server
  - Development
  - Testing
- Iteration 3:
  - More features
  - Deployment

Team-based Project

Project schedule

- Groups formed during your week 2 tutorial
- Project iteration 1 released Sunday night, along with marking criteria.
  - Discussed in Tuesday's lecture
- Marks awarded for each part of the iteration

Team-based Project

In your groups of 4-5

You must all:

- Contribute equally (via git)
- Write code
- Write documentation (e.g. user stories)

If students don't contribute equally, marks will be deducted for individuals.

- We will use a peer assessment tool