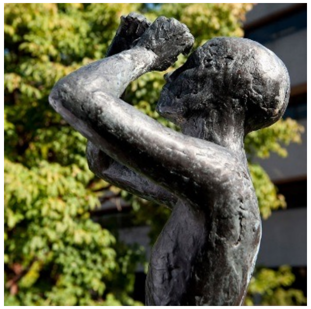


Lecture 21 Unit Testing and Inheritance Revisited

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

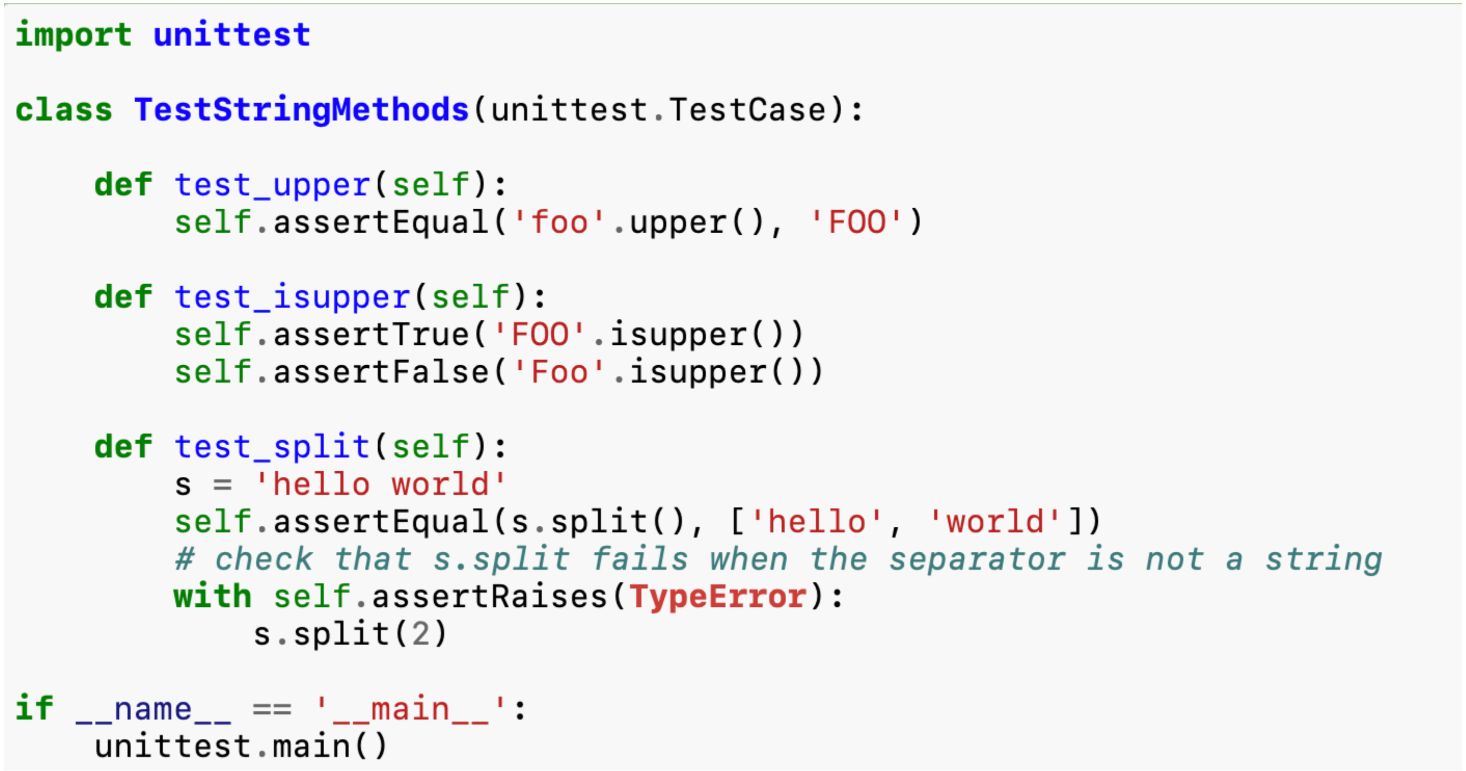
UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Unit Testing

- A software development process in which the smallest testable parts of an application, called units, are individually scrutinized for proper operation.
- Unit tests are can be automated tests or manual tests, written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.
- They are often performed by the developer who originally wrote the code, as a first line of defense before conducting further testing.
- Benefits:
  - **Early detection of problems in the development cycle**
  - **Reduced cost**
  - **Detects changes which may break a design**
  - **Test-driven development -** The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves.

Unit Testing

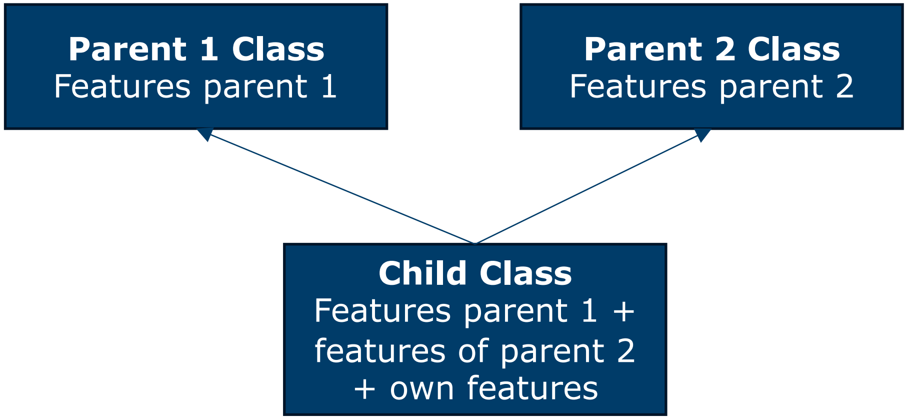


```python
import unittest

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()
```

https://docs.python.org/3/library/unittest.html

Inheritance Revisited

- We have studied inheritance from the perspective of inheriting from a single super or parent class. In many languages, this is the only option.
- Multiple inheritance is when you can inherit from more than one super/parent class.
- Python allows for multiple inheritance.
  - Just because you can doesn't mean you should! Multiple inheritance is absent from some more modern languages because it is considered to cause potential unexpected behaviours that can be difficult to resolve.

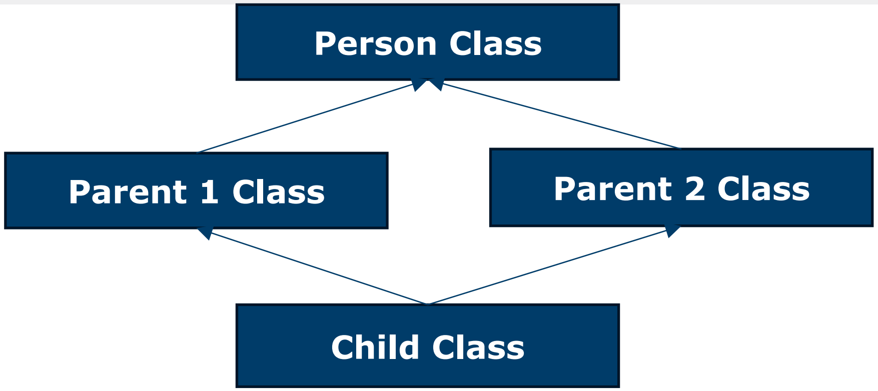


Parent 1 Class
Features parent 1

Parent 2 Class
Features parent 2

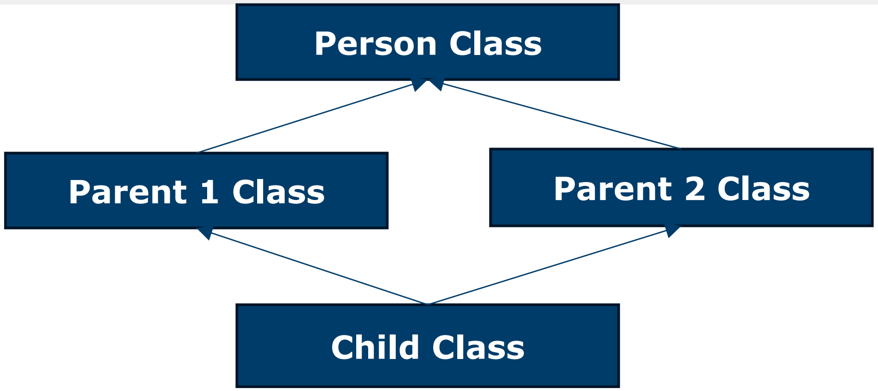
Child Class
Features parent 1 +
features of parent 2
+ own features

Inheritance Revisited



- This picture can get more complex. A well known example is the ‘diamond problem’. Here the the parents in the previous example also have a parent. And all classes implement a method called skill.
- Python must now decide which method to call.
- This is done by applying Method Resolution Order (MRO) – a simple rule that says if method names conflict in the parent classes, the method in the first inherited class is called.
- We can check this order by querying a classes __mro__ order
  - E.g. chlld_obj.__mro__

Inheritance Revisited



- But what happens if:
  - Child implements a display method? No problem – interpreter will look inside the child class first and execute any specialisation of display() found there before looking at the parent classes.
  - What if Parent1 doesn't implement the display class? More interesting! Doesn't it inherit from Person so it has an inherited display()? Python sees this as Person's display() – when we look at the MRO order, Parent2 comes before Person and so Parent2's display method is called.
  - Now imagine we are working as part of a team - if a newer version of Parent2 is released with a display() method, then it could require refactoring of the Child class since it will now inherit Parent1's display and so perhaps use different results than expected.

Next Time:

Regular Expressions