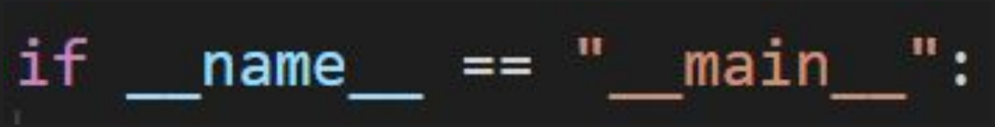
CPS 109 - Lab 6

Agenda

1. Classes
2. Testing

Agenda

Both concepts bring us back to this. You know how we've been droning on about using:


```python
if __name__ == "__main__":
```

Well now we have a use for it!

Classes

What’s a class? I know your profs haven’t gone over this, but if we’re introducing testing, we’ll introduce classes.

A class is simple: it’s a way to consistently define an object.

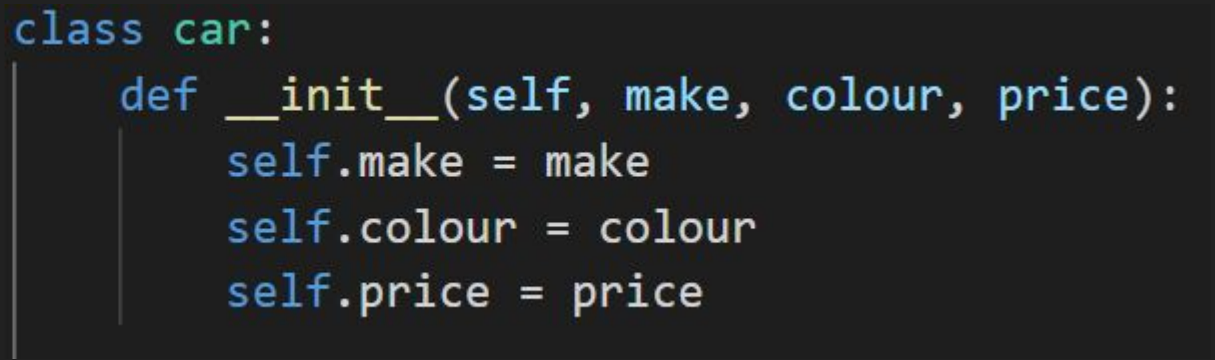
Classes

Recall: an object is a piece of data in your program. Built in objects are things like ints, lists, etc.

A class is a set of interlinked objects that make up a new object!

Classes

Let’s make a basic class:

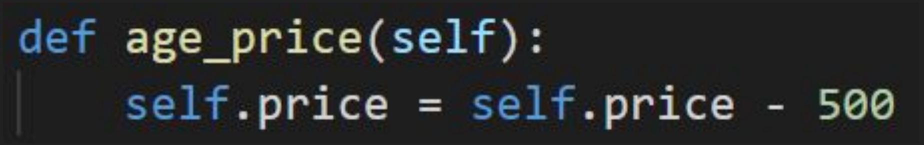


```python
class car:
    def __init__(self, make, colour, price):
        self.make = make
        self.colour = colour
        self.price = price
```

Notice that we use the __init__ here? It’s short for initialize! We use this for when we want to make a new object.

Classes

We can also give it methods! (Just built in functions):



```python
def age_price(self):
    self.price = self.price - 500
```

Instantiation

So it’s well and good that you’ve created a class, but now how do you create an instance of that class? It depends on how you’ve defined your __init__(), but generally:

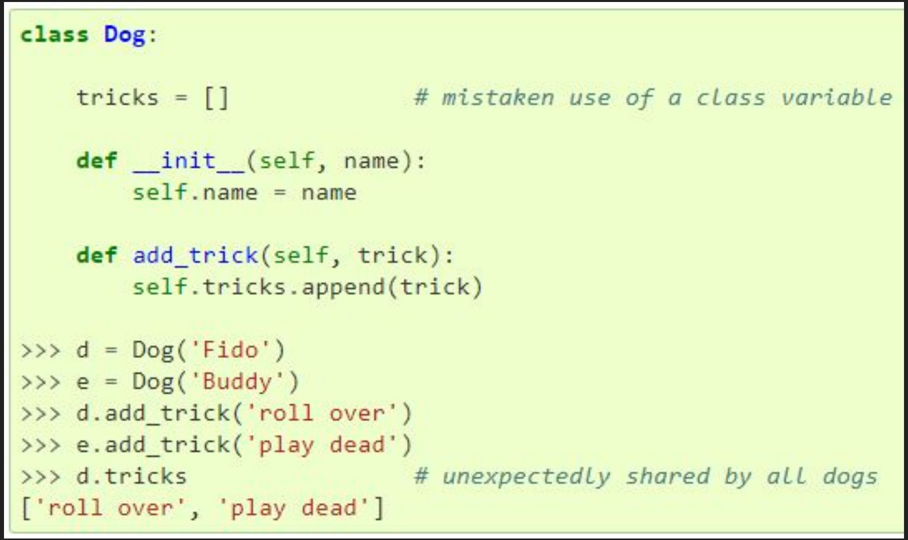
```
var = MyClassName(args)
```

(in the case of the car class)

```
my_car = car(“Tesla Model 3”, “black”, 60000)
```

Classes

Be careful about where you put your variables in classes!



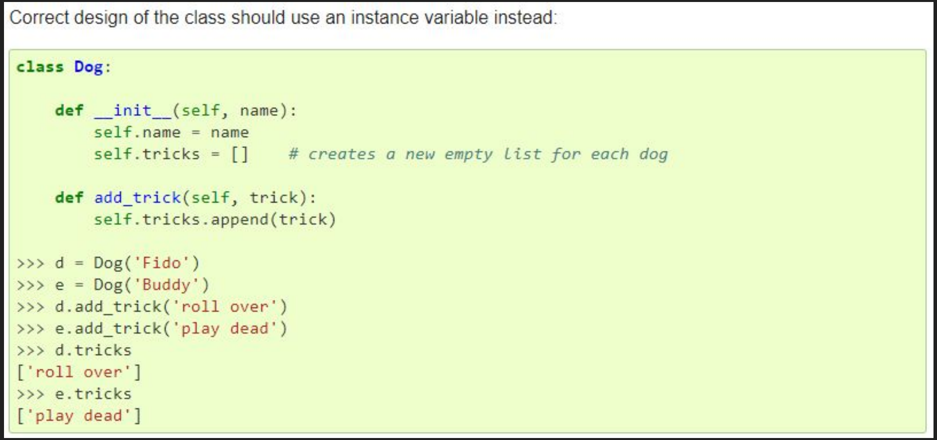
```
class Dog:

    tricks = []             # mistaken use of a class variable

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks                # unexpectedly shared by all dogs
['roll over', 'play dead']
```



Correct design of the class should use an instance variable instead:

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
```

Unit Testing

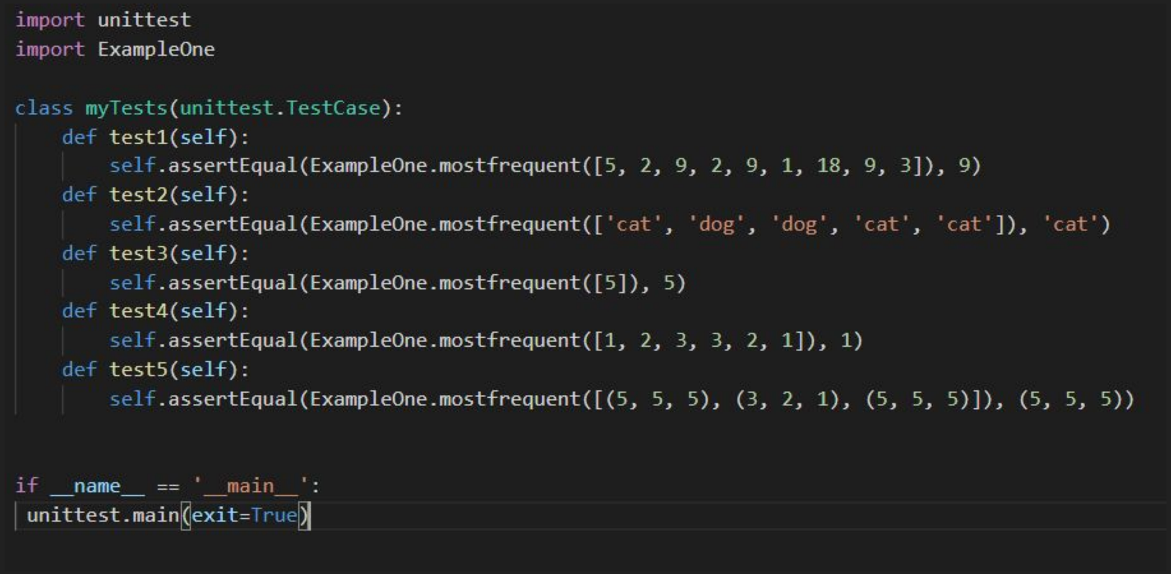
What is unit testing? Simply put: it’s a way to automate testing your code so you don’t have to run it a million times.

Unit Testing

So why did we introduce classes before showing testing? What does this have to do with that "__main__" business?!

Unit Testing

Let's look at an example from your prof:



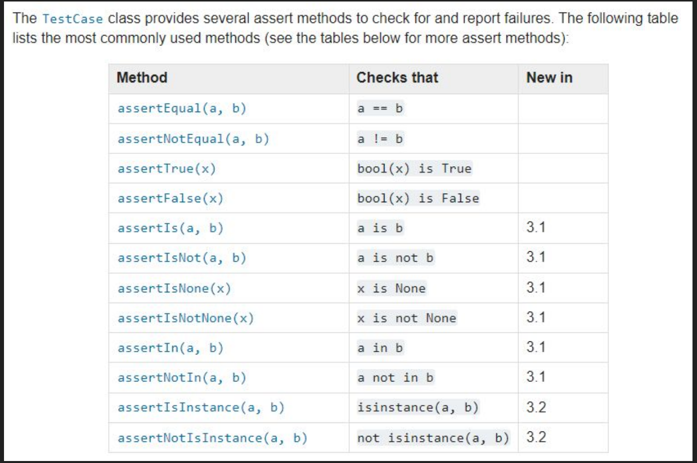
```python
import unittest
import ExampleOne

class myTests(unittest.TestCase):
    def test1(self):
        self.assertEqual(ExampleOne.mostfrequent([5, 2, 9, 2, 9, 1, 18, 9, 3]), 9)
    def test2(self):
        self.assertEqual(ExampleOne.mostfrequent(['cat', 'dog', 'dog', 'cat', 'cat']), 'cat')
    def test3(self):
        self.assertEqual(ExampleOne.mostfrequent([5]), 5)
    def test4(self):
        self.assertEqual(ExampleOne.mostfrequent([1, 2, 3, 3, 2, 1]), 1)
    def test5(self):
        self.assertEqual(ExampleOne.mostfrequent([(5, 5, 5), (3, 2, 1), (5, 5, 5)]), (5, 5, 5))


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

“Assert” your dominance

Note that the previous slide had a bunch of self.assertEqual(). There are many kinds of asserts out there:



The TestCase class provides several assert methods to check for and report failures. The following table lists the most commonly used methods (see the tables below for more assert methods):

Method	Checks that	New in
`assertEqual(a, b)`	`a == b`	
`assertNotEqual(a, b)`	`a != b`	
`assertTrue(x)`	`bool(x) is True`	
`assertFalse(x)`	`bool(x) is False`	
`assertIs(a, b)`	`a is b`	3.1
`assertIsNot(a, b)`	`a is not b`	3.1
`assertIsNone(x)`	`x is None`	3.1
`assertIsNotNone(x)`	`x is not None`	3.1
`assertIn(a, b)`	`a in b`	3.1
`assertNotIn(a, b)`	`a not in b`	3.1
`assertIsInstance(a, b)`	`isinstance(a, b)`	3.2
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	3.2

Unit Testing

That’s weird. The most frequent function is in a different file!

That’s because we’re importing it at the top. We can import from different files like we do with libraries. Speaking of libraries, notice how we import “unittest”?

Unit Testing

Long story short: make classes, call methods and classes outside of files you’ve already made, and then test them!