Q1. Which two operator overloading methods can you use in your classes to support iteration?

The \_\_iter\_\_method

**The \_\_next\_\_ method**

Q2. In what contexts do the two operator overloading methods manage printing?

The \_\_iter\_\_ returns the iterator object and is implicitly called at the start of loops.

The \_\_next\_\_ method returns the next value and is implicitly called at each loop increment. \_\_next\_\_ raises a StopIteration exception when there is no more value to return, which is implicitly captured by looping constructs to stop iterating.

Q3. In a class, how do you intercept slice operations?

def \_getitem\_(self, index)

return self.data[index]

Q4. In a class, how do you capture in-place addition?

class Adder(object):

def \_\_init\_\_(self, num=0):

self.num = num

def \_\_iadd\_\_(self, other):

print 'in \_\_iadd\_\_', other

self.num = self.num + other

return self.num

Q5. When is it appropriate to use operator overloading?

Operator overloading is mostly useful when you're making a new class that falls into an existing "Abstract Base Class"