Code Tracing

```
class G(object):
     def __init__(self, z=42):
          self.z = z if (isinstance(z, int)) else 42
     def __eq__(self, other):
          return (self.z == 42) or (other.z == 42) or (self.z ==
other.z)
     def __str__(self): return "G(%d)" % self.z
     def foo(self): return int(type(self)) == G)
     def bar(self): return int(isinstance(self, G))

class H(G):
     def __init__(self, z=99):
          super(H, self).__init__(z)
     def __str__(self): return "H" + super(H, self).__str__()

#Hint: this prints 20 values (4 lines with 5 values per line)
#Also: all the True/False values were converted to int values
(1's and 0's)

for obj in [G(1), G(), H(42), H()]:
     print(obj, int(obj == G(41)), int(obj == G(42)), obj.foo(),
obj.bar())
```

Free Response

Write the Q and PQ classes so the following test code works. Note that your PQ class may only have one method -- remove – and it does not have to be very efficient. Also, note that the class Q implements a "Queue", so Q.remove will return the least-recently-added value (so this is first-in first-out). Also, the class PQ implements a "Priority Queue", so PQ.remove will return the smallest value regardless of when it was added.

```
q = Q()
assert(str(q) == "<Q of size 0>")
q.add(5)
q.add(3)
assert(str(q) == "<Q of size 2>")
assert(q.remove() == 5) # first-in, first-out!
assert(str(q) == "<Q of size 1>")
assert(q.remove() == 3)
assert(str(q) == "<Q of size 0>")

q1 = Q()
q1.add(42)
q2 = Q()
q2.add(42)
q3 = Q()
q3.add(99)
assert(q1 == q2)
assert(q1 != q3)

pq = PQ()
assert(type(pq) == PQ)
assert(isinstance(pq, Q))

pq.add(4)
pq.add(1)
pq.add(2)
pq.add(3)
assert(str(pq) == "<PQ of size 4>")
assert(pq.remove() == 1)
```

```
assert(pq.remove() == 2)
assert(str(pq) == "<PQ of size 2>")
```

Recursion

Code Tracing

```
def h(x, depth = 0):
     #This uses depth-based indentation as in the course notes
     print("  "*depth, "h(%d)" % x)
     if (x < 3): result = x
     else: result = 2*h(x//2, depth+1) + 3*h(x//3, depth + 1)
     print("  "*depth, "-->", result)
     return result

print(h(8))

def t2(*args):
    if len(args) == 0:
        return []
    else:
        return ( t2(*args[-1:0:-1])+[args[0]])

print(t2(3,4,5,6,7,8,9))
```

Reasoning Over Code

```
def rc1(x):
    def f(x, depth=0):
        if (x==0): return 10**depth
        else: return x%10 + f(x//10, depth+1)
    return(f(x) == 117)

def rc2(L):
     def f(L, d):
          if (len(L) == 0):
               return [ ]
          else:
                assert(d < min(L))
               return [L[0]%d] + f(L[1:], d)
     return (f(L[1:], L[0]) == [2, 5, 3])
```

Free Response

solveWordLadder(wordList, word1, word2) - write the function from midterm2 called solveWordLadder. As a reminder, a word ladder is one that starts at word1, ends at word2, and each successive word differ's from the previous word by one index. Also, return the shortest possible wordLadder.

containsCopies(path) - Write the recursive function containsCopies(path) that takes a path to a folder, and returns True if any two text files in that folder or any subfolder of it are exact copies of each other, and False otherwise. Note that text files with different names and in different folders may be exact copies.

While you may use iteration, you must use recursion in a meaningful way (so, for example you may not use os.walk). Also, for full credit, you must solve this efficiently. In particular, you may not use flatten to create an unnecessary list, and you must use an efficient means of checking for copies.

You may assume that readFile(path) exists and returns the string contents of a text file.

Monte Carlo

Free Response **splitLineToMakeTriangle:** Let's say we give you a rope of size 1 (arbitrary unit). We are going to split the rope at two points. Give us the probability that the three lengths that result could form a triangle. Note: in order for three lines to form a triangle, two of the lines must be longer than the length of the longest side.