Final Exam Practice Solutions

CS 111

1.

a = [1, 2, 6, 4] b = [1, 2, 6, 4]

2.

The first assignment creates a new list in memory and assigns a reference

to that list to the variable a:

\_

a |-|-----------> [1, 2, 3, 4]

-

The second assignment simply assigns the reference held in a to the

variable b. Therefore, no new list is created, but a and b share

a reference to the same list:

\_

a |-|-----------> [1, 2, 3, 4]

- ^

|

\_ |

b |-|--------------

-

Therefore, the third assignment modifies the list that both a and

b reference:

\_

a |-|-----------> [1, 2, 6, 4]

- ^

|

\_ |

b |-|--------------

-

And thus printing either a or b will show the changed list.

3.

food = [4, 9, 6, 11]

4.

The eat function accepts a reference to a mutable list, and thus any changes

that are made to the list inside of eat will also be reflected when

we return to the global scope, since there is only one list in memory.

When prob3() is called, the following list is set up in its stack frame:

-------------

| \_ |

prob3 | food |-|--+--------> [4, 5, 6, 7]

| - |

-------------

Then the call eat(food) is made. Note that eat is passed a reference to

the list, so we get the following picture in memory:

-------------

| \_ |

eat | x |-|--+-----------

| - | |

------------- |

------------- |

| \_ | v

prob3 | food |-|--+--------> [4, 5, 6, 7]

| - |

-------------

In the eat function, elements 1 and 3 in the list are changed:

-------------

| \_ |

eat | x |-|--+-----------

| - | |

------------- |

------------- |

| \_ | v

prob3 | food |-|--+--------> [4, 9, 6, 11]

| - |

-------------

When the eat function returns, the list is still changed since

there is only one list in memory between the two functions:

-------------

| \_ |

prob3 | food |-|--+--------> [4, 9, 6, 11]

| - |

-------------

# Problem 5

# first option: gradually build up the 2-D list

def create\_2d(height, width):

grid = []

for r in range(height):

row = []

for c in range(width):

row += [r\*c]

grid += [row]

return grid

# second option: start with a 2-D list of 0s,

# and then replace the 0s with the correct values

def create\_2d(height, width):

grid = [[0]\*width for i in range(height)]

for r in range(height):

for c in range(width):

grid[r][c] = r\*c

return grid

# Problem 6

def add\_one(grid):

""" we assume that the input grid is a rectangular 2D list --

i.e., that the number of columns is the same in each row

"""

num\_rows = len(grid)

num\_cols = len(grid[0])

for r in range(num\_rows):

for c in range(num\_cols):

grid[r][c] += 1

# Problems 7-10

class Phonebook:

# problem 7

def \_\_init\_\_(self):

self.entries = {}

# problem 8

def add\_entry(self, name, number):

""" note that we assume that each name should have only one number"""

self.entries[name] = number

# problem 9

def contains(self, name):

if name in self.entries:

return True

else:

return False

# here's an alternate version of contains:

#

# def contains(self, name):

# return (name in self.entries)

# problem 10

def number\_for(self, name):

if self.contains(name):

return self.entries[name]

else:

return -1

# Problems 11-13

class Triangle:

# problem 11

def \_\_init\_\_(self, base, height):

self.base = base

self.height = height

# also problem 11

def area(self):

return 0.5 \* self.base \* self.height

# problem 12

def \_\_repr\_\_(self):

s = 'triangle with '

s += 'base ' + str(self.base)

s += ' and height ' + str(self.height)

return s

# problem 13

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

return self.base == other.base \

and self.height == other.height

# Here's an alternate version of \_\_eq\_\_:

#

# def \_\_eq\_\_(self, other):

# if self.base != other.base:

# return False

# elif self.height != other.height:

# return False

# else:

# return True

# Problem 14

def main():

tri1 = Triangle(3, 4)

tri2 = Triangle(6, 6)

tri3 = Triangle(3, 4)

# note: the \_\_repr\_\_ method takes care of the dimensions

print('tri1:', tri1, '(area =', str(tri1.area()) + ')')

print('tri2:', tri2, '(area =', str(tri2.area()) + ')')

print('tri3:', tri3, '(area =', str(tri3.area()) + ')')

if tri1 == tri2:

print('tri1 and tri2 are equal')

else:

print('tri1 and tri2 are not equal')

if tri1 == tri3:

print('tri1 and tri3 are equal')

else:

print('tri1 and tri3 are not equal')

# Problems 15 and 16

class EquilateralTriangle(Triangle):

# problem 15

def \_\_init\_\_(self, side):

super().\_\_init\_\_(side, 0.866\*side)

# problem 16

def \_\_repr\_\_(self):

s = 'equilateral triangle with side '

# we don't have an attribute called side,

# but the side length is stored in the base

# attribute that we inherited

s += str(self.base)

return s

17.

