CS111 Jeopardy: The Home Version
The game that turns CS111 into CSfun11!

Fall 2017

This is intended to be a fun way to review some of the topics that will be on the CS111 final exam. These questions are not indicative of the style and difficulty of the questions that will be on the final.

QUESTIONS

List One-Liners

List One-Liners [1] This is an expression whose value is the number of elements in the list L.

List One-Liners [2] This is a one-line expression whose value is a list of all even numbers between 1 and 100 (inclusive).

List One-Liners [3] This is a one-line statement in the body of

```python
with open('temp.txt', 'r') as infile:
    that prints the total number of words in the file.
```

List One-Liners [4] Given a list L of single-character digit strings, this is a one-line expression whose value is the integer that corresponds to concatenating the digits in reverse order. For example:

- if L is the list ['3', '4', '5'], the code should compute 543
- if L is the list ['5', '3', '7', '2'], the code should compute 2735

List One-Liners [5] Given a list L, this is a one-line statement that moves the last element of the list to the beginning of the list (shifting all other elements to the right).

Iteration & Recursion

Iteration & Recursion [1] This is the definition of a function that satisfies the following contract:

```python
# Returns the sum of the lengths of all
# the strings in the given list of strings
def sumLengths(listOfStrings):
```
Iteration & Recursion [2] This is the definition of a recursive function that satisfies the following contract:

# Returns the sum of all integers in the range between the
# 1st argument, inclusive, and the 2nd argument, exclusive
def sumRange(start, end):

For example:

- `sumRange(5, 7)` returns 11
- `sumRange(-2, 4)` returns 3
- `sumRange(3,3)` returns 0
- `sumRange(4,3)` returns 0

Iteration & Recursion [3] Consider the following function.

def mystery(x):
    print(x)
    if x < 5:
        mystery(3 * x - 1)
    elif x > 5:
        mystery(x - 3)

This is printed by the invocation `mystery(10)`

Iteration & Recursion [4] This is the definition of a recursive function that satisfies the following contract:

# Returns the power for a given power of 2.
def whichPowerOf2(n):

For example:

- `whichPowerOf2(1)` returns 0 (because 2**0 = 1)
- `whichPowerOf2(8)` returns 3 (because 2**3 = 8)
- `whichPowerOf2(16)` returns 4 (because 2**4 = 16)

Iteration & Recursion [5]

These statements read the content of tweets retrieved from the Twitter API, and print this list of hashtags: ["ruhlman16", "redClass16", "wellesley", ...]. (The list may contain duplicates.)

tweets = [
{"source": "Come to SCI 104 for our Ruhlman! #ruhlman16", "entities": {"urls" : [], "hashtags": ["ruhlman16"], ...},
{"source": "Congratulations to our seniors! #redClass16, #wellesley", "entities": {"urls" : [], "hashtags": ["redClass16", "wellesley"]}, ...}]
Dictionaries & Sets

Dictionaries & Sets [1]  This is the value of the expression `len(set('abracadabra'))`

Dictionaries & Sets [2]  These are **all** the following types that **cannot** be keys in a dictionary:

- `int`
- `bool`
- `string`
- `list`
- `tuple`
- `set`
- `dict`

Dictionaries & Sets [3]  These are **all** of the following boolean variables that are not necessarily **True**:

```python
bool1 = {'x':1, 'y':2, 'z':3} == dict([( 'x', 1), ( 'y', 2), ( 'z', 3)])
bool2 = {'x':1, 'y':2, 'z':3}.items() == [( 'x', 1), ( 'y', 2), ( 'z', 3)]
bool3 = set({'x':1, 'y':2, 'z':3}.items()) == {( 'x', 1), ( 'y', 2), ( 'z', 3)}
```

Dictionaries & Sets [4]  This **one** variable has a value that is **not** equal to the others:

```python
d1 = {'a':3, 'b':2, 'c':1}
d2 = dict([( 'b', 2), ( 'c',1), ( 'a', 3)])
d3 = {key: 'bacaba'.count(key) for key in 'cab'}
d4 = dict(zip('cba', [1, 2, 3]))
d5 = json.loads('{"b":2, "a":3, "c":1}')
d6 = {('a', 3), ('b', 2), ('c', 1)}
```

Dictionaries & Sets [5]  This is a printed representation of the value returned by `mysteryDict('a bat ate an oval berry')` given the following definition:

```python
def mysteryDict(sentence):
    dct = {}
    for word in sentence.split():
        c = word[0]
        dct[c] = dct.get(c, 0) + len(word)
    return dct
```
Objects

**Objects [1]** A class declaration typically includes these entities, used to keep track of an object’s state.

**Objects [2]** Consider the following classes:

```python
class Food:
    def info(self):
        print('Good to eat')

class Dessert(Food):
    def calories(self):
        print('Lots of calories')

class Cake(Dessert):
    def flavor(self):
        print('I like chocolate')
```

This is the number of methods that a Cake object has.

**Objects [3]** This will be printed by the following program.

```python
class A():
    def number(self):
        print(8)

class B(A):
    def number(self):
        A.number(self)
        print(9)

class C(B):
    def number(self):
        B.number(self)

C().number()
```

**Objects [4]** This will be printed by the following program.

```python
import math
class RightTriangle:
    def __init__(self, base, height):
        self.base, self.height = base, height

    def scale(self, factor):
        self.base, self.height = self.base*factor, self.height*factor

    def hypotenuse(self):
        return math.sqrt(self.base**2 + self.height**2)

    def perimeter(self):
        return self.base + self.height + self.hypotenuse()

tri = RightTriangle(6, 8)
tri.scale(0.5)
print(tri.perimeter())
```
Objects [5] These statements in a MinionYoga step method make its instances rotate by 90 degrees half the time:

```python
class MinionCeiling(Minion):
    def step(self):
        # most of body omitted for space reasons
        else: self.minionLayer.move(self.deltax, -self.deltay)

class MinionYoga(MinionCeiling):
    ''' Rotate the minion by 90 degrees 50\% of the time '''
    def step(self):
        # FILL IN THE MISSING STATEMENTS
```

Higher-order Functions

*Note: Problems 3 and 5 in this category use the reduce function, which was not covered in Fall 2017 and will not be on the Fall 2017 final. For this reason, this category was removed from the in-class game. But we leave it in the handout for completeness.*

Higher-order Functions [1] A function is said to be higher-order when it satisfies at least one of these two conditions (give both).

Higher-order Functions [2] This is one line of code that, without using list comprehensions, generates a list consisting of the result of adding 10 to each element in a given list of numbers L.

Higher-order Functions [3] What is the result of the following expression?

```python
reduce(lambda ans, num: ans*10 + num, [1,7,2,9])
```

Higher-order Functions [4] This is a function that satisfies the following contract with a one-line body using a higher-order list operator:

```python
# Returns a new list of strings excluding all the words
# that end in the letter ‘s’ in the given list of strings
def dropPlurals(listOfStrings):
    For example:
    dropPlurals(['ant', 'ants', 'bunny', 'bunnies', 'horse', 'horses',
                  'goose', 'geese', 'albatross', 'albatrosses'])
    returns ['ant', 'bunny', 'horse', 'goose', 'geese']
```

Higher-order Functions [5] Using higher-order list operators and lambda, write a function myProduct that takes as a single argument a list of numbers and returns the product of the cubes of all the numbers in the list that are divisible by 5. For example:

- myProduct(range(0,5)) returns 1
- myProduct(range(0,10)) returns 125
- myProduct(range(0,15)) returns 125000
Bugs That Bite

Bugs That Bite [1] This is a bug in the following function definition:
```python
def compare(a, b):
    if a == b:
        return 'equal'
    else:
        return 'not equal'
```

Bugs That Bite [2] This is a bug in the following class definition:
```python
class Animal:
    def __init__(self, numLegs):
        numberOfLegs = numLegs
    
    def isBiped(self):
        return self.numberOfLegs == 2
```

Bugs That Bite [3] Recall that `random.randint(a,b)` returns a random integer `i` such that `a ≤ i ≤ b`. This is a bug in the following function definition:
```python
import random
def chooseRandom(aList):
    if len(aList) > 0:
        randomIndex = random.randint(0, len(aList))
        return aList[randomIndex]
```

Bugs That Bite [4] This is a bug in the following code:
```python
vowelDict = {}
for vowel in 'aeiou ':
    vowelDict[vowel] = vowel.upper()
print(vowelDict['E'])
```

Bugs That Bite [5] The following definition of the `areAllPositive` function does not satisfy the contract specified in its comment. Show this by giving a sample input on which it returns an incorrect answer.
```python
def areAllPositive(numbers):
    '''Returns True if all elements in the list of numbers are positive, and False otherwise.'''
    if len(numbers) == 0:
        return True
    else:
        for num in numbers:
            if num <= 0:
                return False
        else:
            return True
Potpourri

Potpourri [1] When the below program is executed, this will be printed.

```python
x = 3
y = 8
def f():
    x = 6
    y = 7
f()
print(x)
print(y)
```

Potpourri [2] This is the definition of a function `swap` that takes three arguments (a list `L` and two list indices `i` and `j`) and modifies `L` by swapping the contents of its slots at indices `i` and `j`.

Potpourri [3] Consider the function below:

```python
def appendages(L):
    if len(L) == 0:
        return []
    else:
        return L + appendages(L[1:])
```

This is the list returned by the invocation `appendages([1,2,3,4])`.

Potpourri [4] This is (1) the buggy expression and (2) the corrected expression in the following function definition:

```python
# Returns a dictionary mapping each string in the given list
# of strings to the number of times it appears in the list.
def frequenciesBuggy(strings):
    freqDict = {}
    for s in strings:
        freqDict[s] = 1 + freqDict[s]
    return freqDict
```

Potpourri [5] This is a function that satisfies the following contract:

```python
# Returns the number of times the most commonly occurring character
# in the string s occurs.
def countOfMostCommonCharacter(s):
```

For example:

- `countOfMostCommonCharacter('eerie')` returns 3
- `countOfMostCommonCharacter('Mississippi')` returns 4
List One-Liners

List One-Liners [1]  len(L)

List One-Liners [2]
Here are some of many possible answers:

- [i for i in range(1, 101) if i % 2 == 0]
- filter(lambda i: i % 2 == 0, range(1, 101))
- range(2, 101)[::2]
- range(2, 101, 2)

List One-Liners [3]  len(infile.read().split())

List One-Liners [4]  int(''.join(L)[::-1])

List One-Liners [5]  L.insert(0, L.pop())

Iteration & Recursion

Iteration & Recursion [1]
Here are some of many solutions:

Solution 1:
def sumLengths(listOfStrings):
    sum = 0
    for s in listOfStrings:
        sum += len(s)
    return sum

Solution 2:
def sumLengths(listOfStrings):
    return sum([len(s) for s in listOfStrings])

Solution 3:
def sumLengths(listOfStrings):
    return sum(map(len, listOfStrings))

Solution 4:  Note: reduce was not covered in Fall 2017.
def sumLengths(listOfStrings):
    return reduce(lambda total, s: total + len(s), listOfStrings, 0)

Iteration & Recursion [2]
def sumRange(start, end):
    if start >= end:
        return 0
    else:
        return start + sumRange(start + 1, end)
Iteration & Recursion [3]

def whichPowerOf2(n):
    if n==1:
        return 0
    else:
        return 1 + whichPowerOf2(n/2)

Iteration & Recursion [4]

Solution 1:
print [hashtag for tweet in tweets
    for hashtag in tweet["entities"]['hashtags']]

Solution 2:
hashtags = []
for tweet in tweets:
    for ht in tweet["entities"]['hashtags']:
        hashtags.append(ht)
print hashtags

Solution 3:
hashtags = []
for tweet in tweets:
    hashtags.extend(tweet["entities"]['hashtags'])
print hashtags

---

Dictionaries & Sets

Dictionaries & Sets [1] 5. set('abracadabra') is {'a', 'b', 'c', 'd', 'r'}, which has 5 elements.

Dictionaries & Sets [2] list, set, and dict These are mutable object types that cannot be dictionary keys.

Dictionaries & Sets [3] bool2. Although bool2 could be True, it can also be False, because the ordering of pairs returned by .items() is unpredictable. bool1 and bool3 are necessarily true.

Dictionaries & Sets [4] d6. It is a set of tuples, not a dictionary. But dict(d6) would be dictionary equal to the others.

Dictionaries & Sets [5] {"a": 6, "b": 8, "o": 4}. The order of key/value pairs is arbitrary, as is using double or single quotes for the string keys.
**Objects**

**Objects [1]** instance variables, state variables, or data attributes. (instance variables is standard across object-oriented programming languages; state variables is a more generic term that means the same thing in an object-oriented context. The term data attributes is specific to Python.)

**Objects [2]** 3 methods (info, calories, flavor). This does not include default object methods like __repr__, and does not include __init__ (which is used to create the instance, but not operate on it after it has been created).

**Objects [3]**

```
7
8
9
```

Joke: Why was 6 afraid of 7? Because 7 8 9!

**Objects [4]** 12.0 The hypotenuse of 3.0,4.0 triangle is 5.0, and 3.0 + 4.0 + 5.0 = 12.0. The result is necessarily a float, not an integer, both because of the multiplication by 0.5 (which returns a float) and the use of math.sqrt (which always returns a float).

**Objects [5]**

```python
def step(self):
    MinionCeiling.step(self) # move up toward ceiling
    if random.randint(0,1) == 0:
        self.minionLayer.rotate(90)
```

**Higher-order Functions**

**Higher-order Functions [1]**

1. it takes a function as an argument;
2. it returns a function as a result

**Higher-order Functions [2]** map(lambda x: x+10, L)

**Higher-order Functions [3]** 1729

**Higher-order Functions [4]**

```python
def dropPlurals2(L):
    return filter(lambda item: item[-1]!= 's', L)
```

**Higher-order Functions [5]**

```python
def myProduct(nums):
    return reduce(lambda a,b: a*b,
                  map(lambda n: n**3,
                       filter(lambda n: n%5==0, nums)),
                  1)
```
Bugs That Bite

Bugs That Bite [1]  a = b should be a == b

Bugs That Bite [2]  In the __init__ method, numberOfLegs = numLegs assigns to the local variable numberOfLegs in the execution frame for the __init__ method, but does not create an instance variable in the new Animal instance. This can be fixed by changing this line to self.numberOfLegs = numLegs.

Bugs That Bite [3]  random.randint(0, len(aList)) is inclusive on its second argument. So in the case where randomIndex is len(aList), the error “list index out of range” will be raised. The correct expression is random.randint(0, len(aList)-1).

Bugs That Bite [4]  vowelDict['E'] raises the error “Key Error 'E' ” because 'E' is not a key in vowelDict (but 'e' is).

Bugs That Bite [5]  areAllPositive is wrong because its return value is based only on the first element of the list. Any list with a positive first element and some later nonpositive element will be incorrect. For example, brokenAreAllPositive([3,-2] returns True.

Potpourri

Potpourri [1]  The assignments to x and y in the body of the function f create local variables in the execution frame for f and do not change the values of the global variables x and y. So the answer is:

3
8

Potpourri [2]

Solution 1:

def swap (L, i, j):
    ival = L[i]
    L[i] = L[j]
    L[j] = ival

Solution 2:

def swap (L, i, j):
    ival, jval = L[i], L[j]  # simultaneous assignment
    L[i] = jval
    L[j] = ival

Solution 3:

def swap (L, i, j):
    L[j], L[i] = L[i], L[j]  # simultaneous assignment to list slots!

Potpourri [3]  [1, 2, 3, 4, 2, 3, 4, 3, 4, 4]

Potpourri [4]

1. 1 + freqDict[s] (because when the string s is not yet in freqDict, there is a key error)
2. 1 + freqDict.get(s, 0) (because this evaluates to 1 when s is not in freqDict)
Potpourri [5]

Solution 1:

```python
def countOfMostCommonCharacter(s):
    countDict = {}
    for ch in s:
        if ch not in countDict:
            countDict[ch] = 1
        else:
            countDict[ch] += 1
    return max(countDict.values())
```

Solution 2:

```python
def countOfMostCommonCharacter(s):
    countDict = {}
    for ch in s:
        countDict[ch] = countDict.get(ch, 0) + 1
    return max(countDict.values())
```

Solution 3:

```python
def countOfMostCommonCharacter(s):
    maxCount = 0
    for ch in s:
        maxCount = max(maxCount, s.count(ch))
    return maxCount
```

Solution 4:

```python
def countOfMostCommonCharacter(s):
    return max([s.count(ch) for ch in s])
```

Solution 5:

```python
def countOfMostCommonCharacter(s):
    return max(map(lambda ch: s.count(ch), s))
```