MATTHEW'S WEEK 5 PRACTICE PROBLEMS 5

COMPUTER SCIENCE 61A

July 27, 2014

1 What Would Python Print - OOP

1. What would Python print? (From Fall 2013 Midterm 2)

```
class Monster:
    vampire = {2: 'scary'}
    def werewolf(self):
        return self.vampire[2]

class Blob(Monster):
    vampire = {2: 'night'}
    def __init__(self, ghoul):
        vampire = {2: 'frankenstein'}
        self.witch = ghoul.vampire
        self.witch[3] = self

spooky = Blob(Monster)
spooky.werewolf = lambda self: Monster.vampire[2]
```

```
>>> Monster.vampire[2][3]


>>> len(spooky.with)


>>> spooky.witch[3] is not spooky


>>> spooky.witch[2][0:4]
```

```
>>> spooky.werewolf()

>>> Monster.werwolf(spooky)
```

2 Environment Diagram

1. (From Summer 2013 Midterm 2) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You only need to show the final state of each frame.
   A complete answer will:
   - Add all missing names, labels, and parent annotations to all local frames.
   - Add all missing values created during execution.
   - Show the return value for each local frame.

```
>>> n = 3
>>> def follow():
...     sunshine = 10
...     def x(n):
...         nonlocal sunshine
...         def y(fn):
...             nonlocal sunshine
...             sunshine = fn(sunshine) + n
...             return sunshine
...         sunshine += n
...         return y
...     return x(5)
>>> rain = follow()
>>> fall = rain(lambda x: x * n)
```

3 Generic Operators

1. (From Summer 2013 Final) Recall the `type_tag` function discussed in lecture:

```
def type_tag(generic_object):
    return type_tag.tags[type(generic_object)]
```

We will use this function to create a generic function. Because everyone likes pastries, you have created five different class to represent five different kinds of pastries:

- `SugarCookie`
- `SnickerdoodleCookie`
- `RedVelvetCookie`
- `VanillaCake`
- `CheeseCake`

You have two functions, `eat_cookie` and `eat_cake`, which you call on the appropriate pastry to consume it. However, each function only works on pastries of a particular type - `eat_cookie` only works on cookies, and `eat_cake` only works on cake. Tired of having to manually select the correct function, you attempt to define a generic `eat` function:

```
def eat(baked_good):
    return eat.implementations[type_tag(baked_good)](baked_good)
```

Complete the two dictionaries `type_tag.tags` and `eat.implementations` so that the `eat` function works as expected.

4 Iterators and Generators

1. Write an iterator that takes a string as input and returns each letter that is not a vowel. *Hint: you might want to have an instance variable that stores a list of all the vowels.*

```
class NoVowelString:
    """
    >>> s = NoVowelString("hello")
    >>> for char in s:
    ...     print(char)
    ...
    h
    l
    l
    """
```

2. Now try writing a generator that has the same functionality!

```
class NoVowelStringGen:
```

5 Binary Search Trees

Recall the BST class we learned about in discussion on Thursday.

```
class BinaryTree():
    def __init__(self, datum, left=None, right=None):
        self.datum, self.left, self.right = datum, left, right

class BST(BinaryTree):
    def __init__(self, datum, left=None, right=None):
        BinaryTree.__init__(self, datum, left, right)
        assert self.is_bst(), "Breaking Invariants"
```

Write a method for the `BST` class that takes in an integer and returns True if the tree contains that integer and False otherwise. Make sure the `find_elem` method takes advantage of the Binary Search Tree invariant.

```
    def find_elem(self, elem):
```