CS 61A OOP, Nonlocal, Trees, LLs, Growth
Spring 2019 Guerrilla Section 3: March 16, 2019

1 OOP

Questions

1.1 What is the relationship between a class and an ADT?

1.2 What is the definition of a Class? What is the definition of an Instance?

1.3 What is a Class Attribute? What is an Instance Attribute?

1.4 What Would Python Display?

```
class Foo():
    x = 'bam'
        def __init__(self, x):
                self.x = x

        def baz(self):
                return self.x

class Bar(Foo):
        x = 'boom'
        def __init__(self, x):
                Foo.__init__(self, 'er' + x)
        def baz(self):
                return Bar.x + Foo.baz(self)

foo = Foo('boo')

Foo.x

foo.x

foo.baz()
```

```
Foo.baz()

Foo.baz(foo)

bar = Bar('ang')
Bar.x

bar.x

bar.baz()
```

1.5 What Would Python Display?

```
class Student:
        def __init__(self, subjects):
                self.current_units = 16
                self.subjects_to_take = subjects
                self.subjects_learned = {}
                self.partner = None

        def learn(self, subject, units):
                print('I just learned about ' + subject)
                self.subjects_learned[subject] = units
                self.current_units -= units

        def make_friends(self):
        if len(self.subjects_to_take) > 3:
                        print('Whoa! I need more help!')
                        self.partner = Student(self.subjects_to_take[1:])
                else:
                        print('I'm on my own now!')
                        self.partner = None

        def take_course(self):
                course = self.subjects_to_take.pop()
                self.learn(course, 4)
                if self.partner:
                        print(?I need to switch this up!?)
                        self.partner = self.partner.partner
                        if not self.partner:
                                print('I have failed to make a friend :(')

tim = Student(['Chem1A', 'Bio1B', 'CS61A', 'CS70', 'CogSci1'])
tim.make_friends()

print(tim.subjects_to_take)
```

```
tim.partner.make_friends()

tim.take_course()


tim.partner.take_course()

tim.take_course()


tim.make_friends()
```

2 Nonlocal

Questions

2.1 Draw an environment diagram for the following code:

```
ore = "settlers"
def sheep(wood):
    def ore(wheat):
        nonlocal ore
        ore = wheat
    ore(wood)
    return ore
sheep(lambda wood: ore)("wheat")
```

2.2 Draw an environment diagram for the following code:

```
aang = 120
def airbend(zuko):
    aang = 2
    def katara(aang):
        nonlocal zuko
        zuko = lambda sokka : aang + 4
        return aang
    if zuko(10) == 1:
        katara(aang + 9)
    return zuko(airbend)
airbend(lambda x: aang + 1)
```

2.3 Write **make_max_finder**, which takes in no arguments but returns a function which takes in a list. The function it returns should return the maximum value it?s been called on so far, including the current list and any previous list. You can assume that any list this function takes in will be nonempty and contain only non-negative values.

```
def make_max_finder():
    """
        >>> m = make_max_finder()
        >>> m([5, 6, 7])
        7
        >>> m([1, 2, 3])
        7
        >>> m([9])
        9
        >>> m2 = make_max_finder()
        >>> m2([1])
        1
        """
```

3 Object Oriented Trees

Questions

3.1 Define **filter_tree**, which takes in a tree **t** and one argument predicate function **fn**. It should mutate the tree by removing all branches of any node where calling **fn** on its label returns **False**. In addition, if this node is not the root of the tree, it should remove that node from the tree as well.

```
def filter_tree(t, fn):
    """
    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4)]), Tree(6, [Tree(7)])])
    >>> filter_tree(t, lambda x: x % 2 != 0)
    >>> t
    tree(1, [Tree(3)])
    >>> t2 = Tree(2, [Tree(3), Tree(4), Tree(5)])
    >>> filter_tree(t2, lambda x: x != 2)
    >>> t2
    Tree(2)
    """
```

3.2 Fill in the definition for **nth_level_tree_map**, which also takes in a function and a tree, but mutates the tree by applying the function to every nth level in the tree, where the root is the 0th level.

```
def nth_level_tree_map(fn, tree, n):
   """Mutates a tree by mapping a function all the elements of a tree.
   >>> tree = Tree(1, [Tree(7, [Tree(3), Tree(4), Tree(5)]),
                Tree(2, [Tree(6), Tree(4)])])
   >>> nth_level_tree_map(lambda x: x + 1, tree, 2)
   >>> tree
   Tree(2, [Tree(7, [Tree(4), Tree(5), Tree(6)]),
            Tree(2, [Tree(7), Tree(5)])])
   """
```

4 Linked Lists

Questions

4.1 What is a linked list? Why do we consider it a naturally recursive structure?

4.2 Draw a box and pointer diagram for the following:

```
Link('c', Link(Link(6, Link(1, Link('a'))), Link('s')))
```

4.3 The Link class can represent lists with cycles. That is, a list may contain itself as a sublist. Implement **has_cycle** that returns whether its argument, a Link instance, contains a cycle. There are two ways to do this: iteratively with two pointers, or keeping track of Link objects we've seen already. Try to come up with both!

```
def has_cycle(link):
    """
    >>> s = Link(1, Link(2, Link(3)))
    >>> s.rest.rest.rest = s
    >>> has_cycle(s)
    True
    """
```

4.4 Fill in the following function, which checks to see if **sub_link**, a particular sequence of items in one linked list, can be found in another linked list (the items have to be in order, but not necessarily consecutive).

```
def seq_in_link(link, sub_link):
    """
    >>> lnk1 = Link(1, Link(2, Link(3, Link(4))))
    >>> lnk2 = Link(1, Link(3))
    >>> lnk3 = Link(4, Link(3, Link(2, Link(1))))
    >>> seq_in_link(lnk1, lnk2)
    True
    >>> seq_in_link(lnk1, lnk3)
    False
    """
```

5 Growth

Questions

5.1 What is the runtime of the following function?

```
def one(n):
    if 1 == 1:
        return None
    return n
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)

5.2 What is the runtime of the following function?

```
def two(n):
    for i in range(n):
        print(n)
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)

5.3 What is the runtime of the following function?

```
def three(n):
    while n > 0:
       n = n // 2
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)

5.4 What is the runtime of the following function?

```
def four(n):
    for i in range(n):
        for j in range(i):
            print(str(i), str(j))
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)

5.5 What is the runtime of the following function?

```
def five(n):
    if n <= 0:
        return 1
    return five(n - 1) + five(n - 2)
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)

5.6 What is the runtime of the following function?

```
def five(n):
    if n <= 0:
        return 1
    return five(n//2) + five(n//2)
```

a. Theta(1) b. Theta(log n) c. Theta(n) d. Theta(n^2) e. Theta(2^n)