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?

```python
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)
Fill in the implementation for the Cat and Kitten classes. When a cat meows, it should say "Meow, (name) is hungry" if it is hungry, and "Meow, my name is (name)" if not. Kittens do the same thing as cats, except they say "i'm baby" instead of "meow", and they say "I want mama (parents name)" after every call to meow().

```python
>>> cat = Cat('Tuna')
>>> kitten = Kitten('Fish', cat)
>>> cat.meow()
meow, Tuna is hungry
>>> kitten.meow()
i'm baby, Fish is hungry
I want mama Tuna
>>> cat.eat()
meow
>>> cat.meow()
meow, my name is Tuna
>>> kitten.eat()
i'm baby
>>> kitten.meow()
meow, my name is Fish
I want mama Tuna

class Cat:
    noise = 'meow'
    def __init__(self, name):
        
        def meow(self):
            return
```
def eat(self):
    print(self.noise)
    self.hungry = False

class Kitten(Cat):

Check Your Understanding

1.1 Why do Foo.x and foo.x return different things?

1.2 Can we call the Foo.baz function on bar? How? What will it return?

1.3 What is tim.subjects_to_take after all the code is run?

1.4 What is the difference between a local variable, an instance variable, and a class variable? Give an example of each based on the code given.
2 Object Oriented Trees

Questions

2.1 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.

```python
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)
    """
```

2.2 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.

```python
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)])])
    """
```
Check Your Understanding

2.1 Why can we mutate trees using the Tree class? How does the Tree class differ from the Tree ADT?

2.2 How do you guarantee that your code does not recurse forever? Do we need an explicit base case?
3 Linked Lists

Questions

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

3.2 Draw a box and pointer diagram for the following:

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

3.3 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!

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

3.4 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).

   ```python
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
    """
```
Check Your Understanding

3.1 What can go in the first box of a linked list? What can go in the second?

3.2 For question 2, why do we need to store the linked list first in our code? Why can’t we just iterate through it? Why can we iterate through the linked list without storing it in question 3?