CS 61A Spring 2020 — Linked Lists, OOP

Guerrilla Section : March 13, 2020 Solutions

1 Linked Lists

Questions

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

A linked list is a data structure with a first and a rest, where the first is some arbitrary element and the rest MUST be another linked list (or Link.empty)

1.2 Draw a box and pointer diagram for the following:

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

1.3 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
    """

    # solution 1
    tortoise = link
    hare = link.rest
    while tortoise.rest and hare.rest and hare.rest.rest:
        if tortoise is hare:
            return True
        tortoise = tortoise.rest
        hare = hare.rest.rest
    return False

    # solution 2
    seen = []
    while link.rest:
        if link in seen:
            return True
        seen.append(link)
```

```
        link = link.rest
    return False
```

1.4 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
    """

    if sub_link is Link.empty:
        return True
    if link is Link.empty:
        return False
    if link.first == sub_link.first:
        return seq_in_link(link.rest, sub_link.rest)
    else:
        return seq_in_link(link.rest, sub_link)
```

2 OOP

Questions

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

In general, we can think of an abstract data type as defined by some collection of selectors and constructors, together with some behavior conditions. As long as the behavior conditions are met (such as the division property above), these functions constitute a valid representation of the data type.

There are two different layers to the abstract data type:

1) The program layer, which uses the data, and
2) The concrete data representation that is independent of the programs that use the data. The only communication between the two layers is through selectors and constructors that implement the abstract data in terms of the concrete representation.

Classes are a way to implement an Abstract Data Type. But, ADTs can also be created using a collection of functions, as shown by the rational number example. (See Composing Programs 2.2)

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

Class: a template for all objects whose type is that class that defines attributes and methods that an object of this type has.
Instance: A specific object created from a class. Each instance shares class attributes and stores the same methods and attributes. But the values of the attributes are independent of other instances of the class. For example, all humans have two eyes but the color of their eyes may vary from person to person.

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

Class Attribute: A static value that can be accessed by any instance of the class and is shared among all instances of the class.
Instance Attribute: A field or property value associated with that specific instance of the object.

2.4 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
```

'bam'

```
foo.x
```

'boo'

```
foo.baz()
```

'boo'

```
Foo.baz()
```

Error

```
Foo.baz(foo)
```

'boo'

```
bar = Bar('ang')
Bar.x
```

'boom'

```
bar.x
```

'erang'

```
bar.baz()
```

'boomerang'

2.5 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()
```

```
Whoa! I need more help!
```

```
print(tim.subjects_to_take)
```

```
['Chem1A', 'Bio1B', 'CS61A', 'CS70', 'CogSci1']
```

```
tim.partner.make_friends()
```

```
Whoa! I need more help!
```

```
tim.take_course()
```

```
I just learned about CogSci1
I need to switch this up!
```

```
tim.partner.take_course()
```

```
I just learned about CogSci1
```

```
tim.take_course()
```

```
I just learned about CS70
I need to switch this up!
I have failed to make a friend :(
```

```
tim.make_friends()

I'm on my own now!
```

2.6 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().

```
>>>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):

        self.name = name
        self.hungry = True
    def meow(self):

        if self.hungry:
            print(self.noise + ', ' + self.name ' + is hungry!')
        else:
            print(self.noise + ', my name is ' + self.name)
    def eat(self):
        print(self.noise)
        self.hungry = False

class Kitten(Cat):

    noise = "i'm baby"
    def __init__(self, name, parent):
```

```
        Cat.__init__(self, name)
        self.parent = parent
    def meow(self):
        Cat.meow(self)
        print('I want mama' + parent.name)
```