CS 61A More OOP, Linked Lists, Complexity
Spring 2021 Lost 6: March 12, 2021

1 Learning Goals

- Get some hands-on practice with OOP
- Understand the structure of linked lists (woohoo interview prep!)
- Begin to understand how to analyze the runtime of a program

2 OOP

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

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

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

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

foo.x

foo.baz()

Foo.baz()

Foo.baz(foo)

bar = Bar('ang')
Bar.x

bar.x

bar.baz()
```

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()

print(tim.subjects_to_take)

tim.partner.make_friends()

tim.take_course()

tim.partner.take_course()

tim.take_course()

tim.make_friends()
```

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 (parent’s 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):







     def meow(self):







     def eat(self):
          print(self.noise)
          self.hungry = False

class Kitten(Cat):
```

3 Linked Lists

3.1 Introductory

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!

```
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).

```
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
    """
```

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

3.2 Medium

3.1 Write a function that takes a sorted linked list of integers and mutates it so that all duplicates are removed.

```
def remove_duplicates(lnk):
    """
    >>> lnk = Link(1, Link(1, Link(1, Link(1, Link(5)))))
    >>> remove_duplicates(lnk)
    >>> lnk
    Link(1, Link(5))
    """
```

3.3 Hard

3.1 Define `reverse`, which takes in a linked list and reverses the order of the links. The function may *not* return a new list; it must mutate the original list. Return a pointer to the head of the reversed list.

```
def reverse(lnk):
    """
    >>> a = Link(1, Link(2, Link(3)))
    >>> r = reverse(a)
    >>> r.first
    3
    >>> r.rest.first
    2
    """
```

4 Growth

4.1 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)
```

4.2 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)
```

4.3 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)
```

4.4 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)
```

4.5 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)
```

4.6 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)
```