Data Abstraction and OOP

CS Scholars

March 2, 2017

1 Data Abstraction

Take a moment to read through the data abstraction give below.

```
def length(question):          def num_questions(midterm):
    return question[0]             return midterm[0]

def name(question):            def questions(midterm):
    return question[1]             return midterm[1]

def difficulty(question):      def difficulty(midterm):
    return question[2]             return midterm[2]
```

```
def coding_question(blank_lines, name, difficulty):
    if difficulty > 100:
        blank_lines -= 1
    return  [blank_lines, name, difficulty]

def env_diagram(frames, name, difficulty):
    return [frames, nonlocal, difficulty]

def midterm(num_questions, questions, difficulty):
    if len(questions) > 10:
        questions = questions[:10]
    return [num_questions, questions, difficulty]
```

In the following code, find the data abstraction violations.

```
def proofread_midterm(midterm):
    questions = []
    difficulty = 0
    for question in midterm[1]:
        if question[0]  > 1:
            questions += [question]
            difficulty += 1
    return [len(questions), questions, difficulty]
def add_question(midterm, question):
    new_questions = midterm[1]  + [question]
    return [num_questions+1, new_questions, difficulty +
       question[2]
```

2 OOP

2.1 Introduction

Make sure you have the following terms memorized!

- **Object:** anything that has attributes and/or performs actions (think real life objects!)
- **Class:** a type or category of objects
- **Instance:** a single object belonging to some class
- **Instance attribute:** an attribute that is specific to each instance of a class
- **Class attribute:** an attribute that is the same for all instances of a class
- **Method:** an action that a single object can perform; a function that must be called on an instance

What is the point of OOP? It's very similar to data abstraction! OOP is used to collect pieces of data in one object. For example, a location has two components: latitude and longitude. How did we represent a location when we were using data abstraction?

```
def location(lat, long):
    return [lat, long]
def get_latitude(location):
    return location[0]
def get_longitude(location):
    return location[1]
```

```
class Location:
    def __init__(self, lat, long):
        self.lat = lat
        self.long = long
```

How are the two implementations different? How do you create a location in each case?

2.2 Questions

1. (Tammy Nguyen - http://tmmydngyn.com/cs61a/guides/oop.html)
   Below is the implementation of our `Puppy` class.

```
class Puppy:
    """Class representation of the best creatures to have
       walked our planet."""
    num_puppies = 0

    def __init__(self, name, fav_toys):
        self.name = name
        self.fav_toys = fav_toys  # list of favorite toys
        self.hungriness = 0
        self.happiness = 5
        Puppy.num_puppies += 1

    def bark(self):
        """Prints puppy's greeting. He just met you but he
           already loves
        you!"""
        print("Woof! My name is {0} and I love you!".format(
           self.name))

    def play(self, toy):
        if self.hungriness < 5:
            if toy in self.fav_toys:
                print(toy + " is my favorite toy EVER.")
                self.happiness += 2
            else:
                self.happiness += 1
            self.hungriness += 1
        else:
            print("I am too hungry to play :(")

    def eat(self, food):
        if self.hungriness:
            print(food + " is my favorite thing to eat EVER.")
            self.hungriness -= 1
            self.happiness += 1
        else:
            print("No thanks! Let's play!")
```

1. What are all the instance attributes of a Puppy? All the instance attributes of a Puppy are name, fav_toys, hungriness, happiness.
2. What is a puppys name initialized to? What are its hungriness and happiness levels initialized to? The name argument passed to the constructor, 0, 5 respectively.
3. What is the class attribute? Where does it change? What is it keeping track of?num_puppies, changes in constructor (when puppies are created), keeps track of total number puppies.
4. What common parameter do all methods have? All methods have a self parameter.
5. Does the bark method change the state of the Puppy?No, bark does not change any attributes.
6. What must be passed through to the play method?You must pass in string to play which will be assigned to toy
7. Under what condition will a Puppy play?A puppy will only play if his/her hungriness is less than 5.
8. How much does playing increase the puppys happiness?If toy is one of the puppies fav_toys, then happiness will increase by 2. Otherwise, it increases by 1.
9. How much does a puppys hungriness increase by after playing?hungriness increases by 1 after playing regardless of whether the toy is one of the Puppy's favorites.
10. What must be passed through to the eat method?You must pass in string to eat which will be assigned to food.
11. Under what condition will a Puppy eat?A puppy will only eat if hungriness is not 0.
12. Is it possible to get a Puppy to eat if his/her hungriness is not greater than 0?Yes, since eat only checks if self.hungriness is not 0 (remember, 0 is the only integer with a False-y value). If someone sneaky manually sets a puppy's hungriness to a negative number, then the puppy will eat!
13. How does the state of the puppy change after eating?After eating, a Puppy's hungriness decreases by 1 and happiness increases by 1

2. What will be displayed after the following code is executed? (Assume each statement is evaluated after the previous ones)

```
>>> puppy1 = Puppy('Hercules', ['squeaky duck'])
>>> puppy1.hungriness
0
>>> Puppy.num_puppies
1
>>> puppy1.play('stick')
>>> puppy2 = Puppy('Bruno', ['stick', 'ball'])
>>> puppy1.num_puppies
2
>>> puppy2.play('stick')
'stick is my favorite toy EVER.'
>>> puppy2.play('ball')
'ball is my favorite toy EVER.'
>>> puppy1.happiness
6
>>> puppy2.happiness
9
>>> for _ in range(4):
...     puppy1.play('ball')
I am too hungry to play :(
>>> puppy1.hungriness
5
>>> puppy1.eat('canned food')
canned food is my favorite thing to eat EVER.
>>> puppy1.hungriness
4
>>> puppy2.hungriness
2
>>> Puppy.num_puppies = 17
>>> Puppy('Goob', ['stuffed rabbit']).num_puppies
18
>>> puppy1.num_puppies
18
>>> puppy2.num_puppies = 10
>>> Puppy.num_puppies
18
>>> Puppy.__init__(puppy2, 'Chewie', ['squeaky ball']).bark()
Woof!  My name is Chewie and I love you!
```

3.
```
class Store:
    stores, open = [], True
    def __init__(self, name, open, items=[]):
        self.name, self.open = name, open
        Item.items[self.name] = {}
        Store.stores += [self]
        for item in items:
            Item.items[self.name][item.name] = item.count
    def sell(self, item, amount, person):
        if item.name in item.items[self.name]:
            item.items[self.name][item.name] -= amount
            if item.items[self.name][item.name] < 1:
                del item.items[self.name][item.name]
            print("Successfully sold " + item.name)
        else:
            print("Cannot sell items that don't exist")
class Item:
    items = {}
    def __init__(self, name, amount):
        Item.count, self.name = amount, name
class Customer:
    def __init__(self, name):
        self.name = name
        self.cart = []
    def enter_store(self, store):
        if hasattr(self, "store") and self.store == store:
            return "You're already in " + store.name
        if Store.open:
            self.store = store
            return "Welcome to " + store.name
        print(store.name + " is closed")
    def add_to_cart(self, item, amount=1):
        print("Adding " + item.name + " to cart")
        self.cart += [item]*amount
    def checkout(self):
        items = ""
        for item in self.cart:
            items += item.name + " "
            self.store.sell(item, 1, self)
        print(items)
        return "Have a nice day " + self.name
```

```
>>> eggs = Item("eggs", 12)
>>> milk = Item("milk", 1)
>>> banana =  Item("banana", 2)
>>> safeway = Store("safeway", True, [banana, milk, eggs])
>>> steve = Customer("Steve")
>>> steve.add_to_cart(banana)
Adding banana to cart
>>> steve.checkout()
Error
>>> Store.open = True
>>> steve.enter_store(safeway)
'Welcome to safeway'
>>> steve.add_to_cart(milk, 2)
Adding milk to cart
>>> steve.checkout()
Successfully sold banana
Successfully sold milk
Successfully sold milk
banana milk milk
'Have a nice day Steve'
>>> eggs.count
2
>>> Item.count = 1
>>> eggs.count
1
>>> steve.add_to_cart(eggs, 2)
Adding eggs to cart
>>> target = Store("target", False, [banana, milk, eggs])
>>> steve.enter_store(target)
'Welcome to target'
>>> target.open = True
>>> steve.enter_store(target)
"You're already in target"
>>> steve.checkout()
Successfully sold banana
Successfully sold milk
Cannot sell items that don't exist
Successfully sold eggs
Cannot sell items that don't exist
banana milk milk eggs eggs
'Have a nice day Steve'
```

3 Inheritance

Recall that when we inherit from a class, the subclass can access all attributes and methods of the super class. However, methods and attributes can also be **overwritten** in the subclass.

1. Examine the classes below:

```
class Foo():
    baz = 5
    def __init__(self):
        baz = 7
        self.baz = baz
class Bar(Foo):
    def __init__(self):
        Foo.__init__(self)
        Foo.baz = 3
        print(baz)
```

What will Python display if we execute the following code?

```
>>> f = Foo()
>>> f.baz
7

>>> Foo.baz
5


>>> b = Bar()
Error

>>> Foo.baz
3


>>> Bar.baz
3
```

2. Now we will define `Sneaky` and `Illusion`. Note that `pass` means do nothing. So Sneaky is an empty class.

```
class Sneaky: pass
class Illusion:
    def __init__(tricky, self):
        self.tricky = tricky
        print(tricky)
    def fool(you):
        print(you.tricky + " fooled you!")
```

What will Python display if we execute the following code?

```
>>> what = Illusion("what")
Error

>>> sneaky = Sneaky()

>>> tricky = Illusion.__init__("Python", sneaky)
Python

>>> tricky is sneaky
False

>>> sneaky.fool = lambda self: print(self.tricky + "is weird")
>>> sneaky.fool()
Error

>>> Illusion.fool = lambda wat: print(wat.tricky + "is wack")
>>> Illusion.fool(sneaky)
Python is wack


>>> gullible = Sneaky()
>>> gullible.tricky = "61a"
>>> sneaky.fool(gullible)
61a is weird
```