More On Classes

UW CSE 160
Spring 2015

Classes define objects

- What are objects we've seen?

Classes <u>define</u> objects

What are objects we've seen?

- String
- Int
- Float
- Dict
- List
- Set
- Graph
- File
- CSV Writer
- Others?

Objects can be created

- set_one = **set()**
- dict_one = **dict()** # dict_one = {}
- str_one = **str()**   # str_one = "
- list_one = **list()**   # list_one = []
- file_one = **open('data.csv')**
- import networkx as nx
- graph_one = **nx.Graph()**

Objects have methods

- set_one.**append**('purple')
- dict_one.**setdefault**('four',16)
- str_one.**capitalize**()
- list_one.**extend**([1,2,3,4])
- graph_one.**add_edges**([(1,2),(1,3),(2,4)])

Objects have internal state

```
str_one = 'purple'
str_two = 'spectrographically'

>> str_one.count('c')
0
>> str_two.count('c')
2
>> graph_one.nodes()
[1,2,3,4]
```

Classes define objects

- A class is a **blueprint** for an object.

```
class Vehicle:
```

Style Note: Classes use CamelCase. No spaces or underscore but the first letter of each word is capitalized. Usually keep class names to a single word if possible.

Classes define objects

- A class is a **blueprint** for an object.

```
class Vehicle:

    def __init__(self, make, color, passengers, wheels=4, tank=20):
        ''' Create a new Vehicle Object '''
        self.model, self.color, self.wheels = make, color, wheels
        self.seats = passengers
        self.gas = 0

if __name__ == '__main__':
    my_car = Vehicle('Honda', 'White', 4)
    your_motorcycle = Vehicle('Mazda', 'Red', 2, 2)
    semi = Vehicle('Mercedes', 'Black', 2, wheels=16)
```

Classes define objects

- A class is a **blueprint** for an object.

```
class Vehicle:

    def __init__(self, make, color, passengers, wheels=4, tank=20):
        ''' Create a new Vehicle Object '''
        self.model, self.color = make, color
        self.seats = passengers
        self.wheels, self.tank = wheels, tank
        self.gas = 0

if __name__ == '__main__':
    my_car = Vehicle('Honda', 'White', 4)
    your_motorcycle = Vehicle('Mazda', 'Red', 2, 2)
    semi = Vehicle('Mercedes', 'Black', 2, wheels=16)
```

__init__ is the constructor. This is a "**magic**" method. Means something special to python. In this case it defines how to create a new Vehicle object.

Classes define objects

```
class Vehicle:

    def __init__(self, make, color, passengers, wheels=4, tank=20):
        ''' Create a new Vehicle Object '''
        self.model, self.color = make, color
        self.seats = passengers
        self.wheels, self.tank = wheels, tank
        self.gas = 0

    def fill_tank(self,gallons):
        '''Add gallons to tank. Until it is full'''
        self.gas += gallons
        if self.gas > self.tank :
            self.gas = self.tank
```

Classes define objects

```
class Vehicle:

    def __init__(self, make, color, passengers, wheels=4, tank=20):
        ''' Create a new Vehicle Object '''
        self.model, self.color = make, color
        self.seats = passengers
        self.wheels, self.tank = wheels, tank
        self.gas = 0

    def __str__(self):
        return 'Gas remaining: ' + str(self.gas)

    def __hash__(self):
        return hash(self.make) + hash(self.color) + hash(self.seats) +\
                       hash(self.wheels) + hash(self.tank) + hash(self.gas)
```

More “**magic”** methods to convert object to a string or hash.

Let's Play With Vehicles

```
import vehicle
```

Why Use Classes

- Classes are blueprints for **objects**, objects model the real world. This makes programming easier.
- Have multiple objects with similar functions (methods) but different internal state.
- Provide a software abstraction for clients to use without needing to know the details of your program.

Why Use Classes

```
class Pizza:
    def __init__(self, name, toppings):
        self.name, self.toppings = name,toppings

    def is_vegetarian(self):
        for t in self.toppings:
            if not t.vegetarian:
                return False
            else
                return True

class Topping:
    def __init__(self, name, veg=False):
        self.name = name
        self.vegetarian = veg
```

Why Use Classes

```
#make toppings
from pizza import *
cheese, tomato = Topping('cheese',True), Topping('tomato',True)
pepper, pineapple = Topping('pepper',True), Topping('pineapple',True)
pepperoni, ham = Topping('pepperoni'), Topping('ham')

cheese_pizza =  Pizza('cheese',[cheese,tomato])
hawaiian = Pizza('hawaiian',[cheese,tomato,pineapple,ham])
combo = Pizza('combo',[cheese,tomato,pepper,pineapple])

>> combo.is_vegetarian()
    True
>> hawaiian.is_vegetarian()
     False
```

Text analysis module

(group of related functions)

representation = dictionary

```
def read_words(filename):
    """Return dictionary mapping each word in filename to its frequency."""
    wordfile = open(filename)
    word_list = wordfile.read().split()
    wordfile.close()
    wordcounts_dict = {}
    for word in word_list:
        count = wordcounts_dict.setdefault(word, 0)
        wordcounts_dict[word] = count + 1
    return wordcounts_dict


def word_count(wordcounts_dict, word):
    """Return count of the word in the dictionary. """
    if wordcounts_dict.has_key(word):
        return wordcounts_dict[word]
    else:
        return 0
```

```
# program to compute top 5:
wordcounts = read_words(filename)
result = topk(wordcounts, 5)
```

```
def topk(wordcounts_dict, k=10):
    """Return list of (count, word) tuples of the top k most frequent words."""
    counts_with_words = [(c, w) for (w, c) in wordcounts_dict.items()]
    counts_with_words.sort(reverse=True)
    return counts_with_words[0:k]


def total_words(wordcounts_dict):
    """Return the total number of words."""
    return sum(wordcounts_dict.values())
```

Problems with the implementation

```
# program to compute top 5:
wordcounts = read_words(filename)
result = topk(wordcounts, 5)
```

The **wordcounts** dictionary is exposed to the client:

•the user might corrupt or misuse it.

•If we change our implementation (say, to use a list),

•it may break the client program.

We prefer to

- –Hide the implementation details from the client
- –Collect the data and functions together into one unit

Class Implementation

```
class WordCounts:
  """Represents the words in a file."""
  # Internal representation:
  # variable wordcounts is a dictionary mapping words to frequency

  def __init__(self, filename):
    """Create a WordCounts object from the given file"""
    words = open(filename).read().split()
    self.wordcounts = {}
    for w in words:
      self.wordcounts.setdefault(w, 0)
      self.wordcounts[w] += 1

  def word_count(self, word):
    """Return the count of the given word"""
    return self.wordcounts[word]

  def topk(self, k=10):
    """Return a list of the top k most frequent words in order"""
    scores_with_words = [(c,w) for (w,c) in self.wordcounts.items()]
    scores_with_words.sort(reverse=True)
    return scores_with_words[0:k]

  def total_words(self):
    """Return the total number of words in the file"""
    return sum([c for (w,c) in self.wordcounts])
```

```
# program to compute top 5:
wc = WordCounts(filename)
result = wc.topk(5)
```

Alternate implementation

```
# program to compute top 5:
wc = WordCounts(filename)
result = wc.topk(5)
```

Exact same program!

```
class WordCounts:
  """Represents the words in a file."""
  # Internal representation:
  # variable words is a list of the words in the file

  def __init__(self, filename):
    """Create a WordCounts object from the given file"""
    self.words = open(filename).read().split()

  def word_count(self, word):
    """Return the count of the given word"""
    return self.words.count(word)

  def topk(self, k=10):
    """Return a list of the top k most frequent words in order"""
    scores_with_words = [(self.wordcount(w),w) for w in set(self.words)]
    scores_with_words.sort(reverse=True)
    return scores_with_words[0:k]

  def total_words(self):
    """Return the total number of words in the file"""
    return len(self.words)
```

A Card Game

Create the base classes that could be used by a client to create multiple card games.

- Blackjack
- Spades
- Poker
- Cribbage
- Euchre (24 cards!)

A Card Game: Design

What are some high level classes that might be useful?

A Card Game: Design

What are some high level classes that might be useful?

Deck

Holds a set of cards, can be shuffled and deal cards into Hands.

Hand

Holds cards and has basic methods for calculating properties. (has pair, sum ect)

Card

Takes a face value character, points value, and suit.

A Card Game: Design

- Useful functions for Card class

```
class Card:
```

A Card Game: Design

- Useful functions for Card class

```
class Card:

    def __init__(self, face, suit, value=1):
        '''Create a new card'''
        self.face, self.suit,  = face.upper()[0], suit.upper()[0]
        self.value = value

    def is_black(self):
        return self.suit == 'S' or self.suit == 'C'

    def is_face(self):
        return not self.face.isdigit()
```

A Card Game: Design

- More magic methods, comparing cards

(Also in class Card:)

```
    …
    def  __eq__(self,other):
        return self.value == other.value

    def __lt__(self,other):
        return self.value < other.value

    def __gt__(self,other):
        return self.value > other.value
```

See Also: __ne__, __le__, __ge__

A Card Game: Design

- Useful functions for the Hand class

```
class Hand:
```

A Card Game: Design

- Useful functions for the Hand class

```
class Hand:

    def __init__(self,cards):
        self.card = cards

    def value(self):
        return sum([c.value for c in self.cards])

    def has_pair(self):
        '''Returns True if hand has a pair'''
        for i,c in enumerate(self.cards):
            for c2 in self.cards[i+1:]:
                if c.face == c2.face:
                    return True
        return False
```

A Card Game: Design

- Useful functions for the Deck class

```
class Deck:
```

A Card Game: Design

- Useful functions for the Deck class

```
class Deck:

    def __init__(self,cards):
        self.cards = cards

    def shuffle(self):
        '''Randomize the order of internal cards list'''
        random.shuffle(self.cards)

    def deal(self,n=1):
        hand_cards = self.cards[0:n]
        del self.cards[0:n]
        return Hand(hand_cards)
```

A Card Game: Design

- Useful functions for the Deck class

(also in class Deck:)

```
    …
    def __len__(self):
        return len(self.cards)
```