More On Classes

UW CSE 160
Winter 2017

Classes are a template for objects

- What are objects we've seen?

Classes are templates for objects

Examples of objects we've seen:

- Dict
- List
- Set
- Graph
- File
- Others?

Objects can be created with constructors

```
set_one = set()
dict_one = dict()   # dict_one = {}
str_one = str()     # str_one = ""
list_one = list()   # list_one = []

import networkx as nx
graph_one = nx.Graph()
```

Objects have methods

```
set_one.add('purple')

dict_one.setdefault('four', 16)

str_one.capitalize()

list_one.extend([1, 2, 3, 4])

graph_one.add_edge(1, 2)
```

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]
```

Classes are templates for 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 are templates for 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

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 are templates for 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 are templates for 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)
```

__str__ is a “**magic”** method to convert object to a string.

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 how the object is implemented.

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 etc)

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

```
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)
```