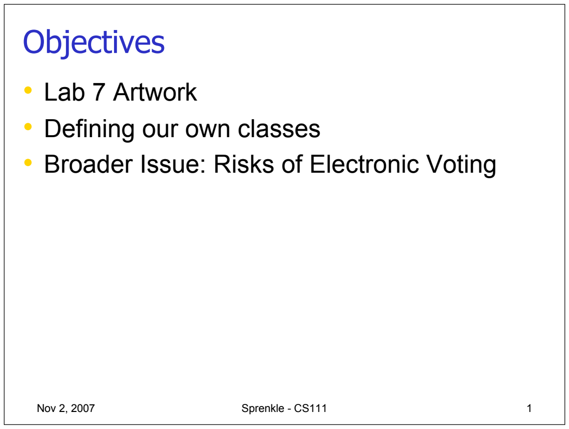
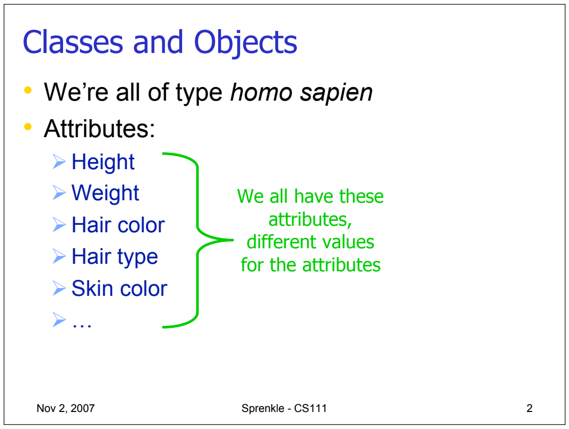


Objectives

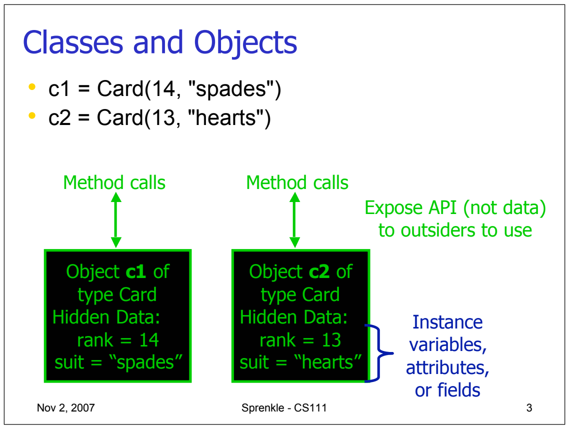
- Lab 7 Artwork
- Defining our own classes
- Broader Issue: Risks of Electronic Voting



Classes and Objects

- We're all of type *homo sapien*
- Attributes:
  - Height
  - Weight
  - Hair color
  - Hair type
  - Skin color
  - …

We all have these attributes, different values for the attributes



Classes and Objects

- c1 = Card(14, "spades")
- c2 = Card(13, "hearts")

Method calls

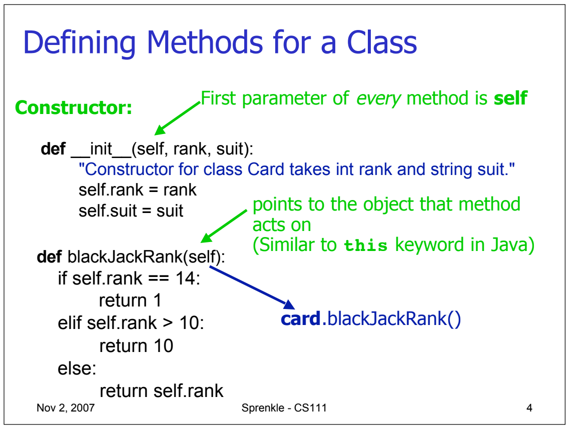
Method calls

Expose API (not data) to outsiders to use

Object **c1** of type Card
Hidden Data:
rank = 14
suit = "spades"

Object **c2** of type Card
Hidden Data:
rank = 13
suit = "hearts"

Instance variables, attributes, or fields



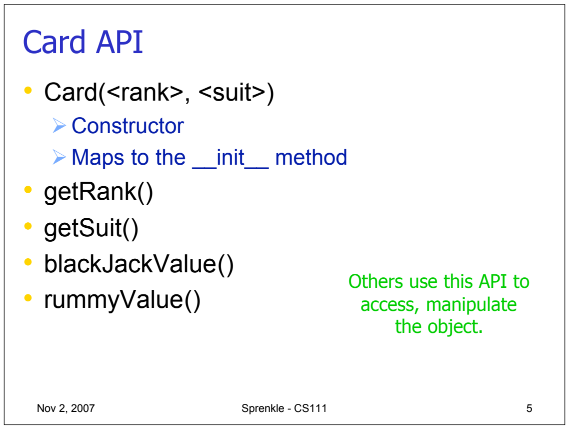
Defining Methods for a Class

Constructor: First parameter of *every* method is **self**

```
def __init__(self, rank, suit):
    "Constructor for class Card takes int rank and string suit."
    self.rank = rank
    self.suit = suit

def blackJackRank(self):
   if self.rank == 14:
        return 1
   elif self.rank > 10:
        return 10
   else:
        return self.rank
```

points to the object that method acts on
(Similar to **this** keyword in Java)

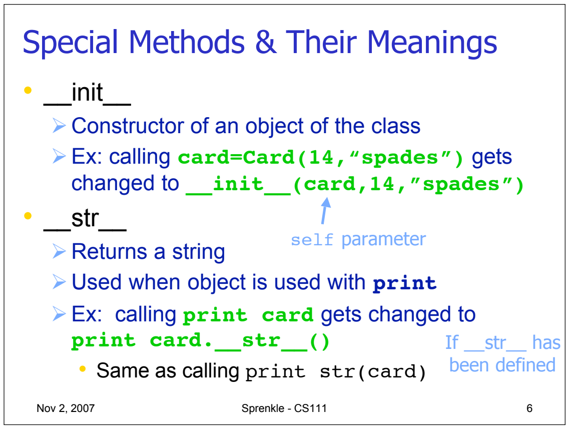
card.blackJackRank()



Card API

- Card(<rank>, <suit>)
  - Constructor
  - Maps to the __init__ method
- getRank()
- getSuit()
- blackJackValue()
- rummyValue()

Others use this API to access, manipulate the object.

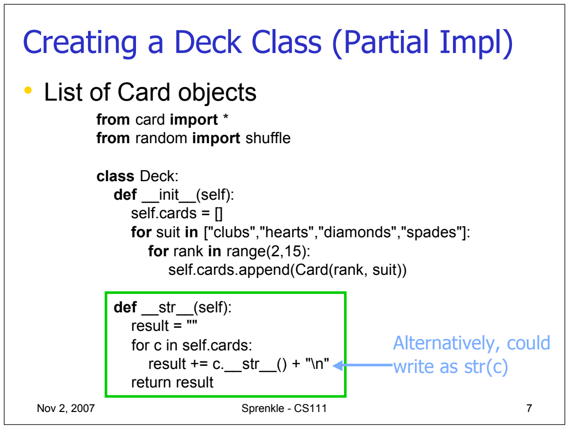


Special Methods & Their Meanings

- __init__
  - Constructor of an object of the class
  - Ex: calling `card=Card(14,"spades")` gets changed to `__init__(card,14,"spades")`

    `self` parameter
- __str__
  - Returns a string
  - Used when object is used with `print`
  - Ex: calling `print card` gets changed to `print card.__str__()`
    - Same as calling `print str(card)`

If __str__ has been defined

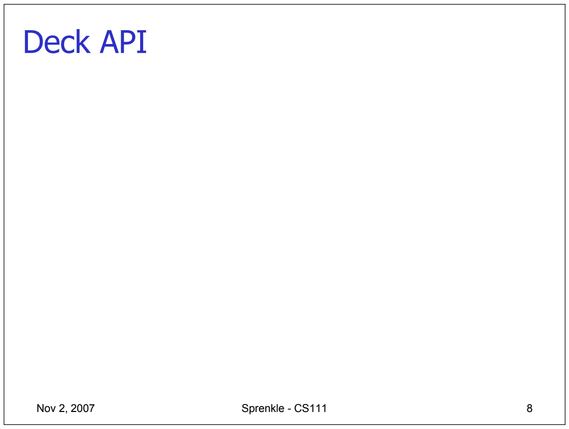


Creating a Deck Class (Partial Impl)

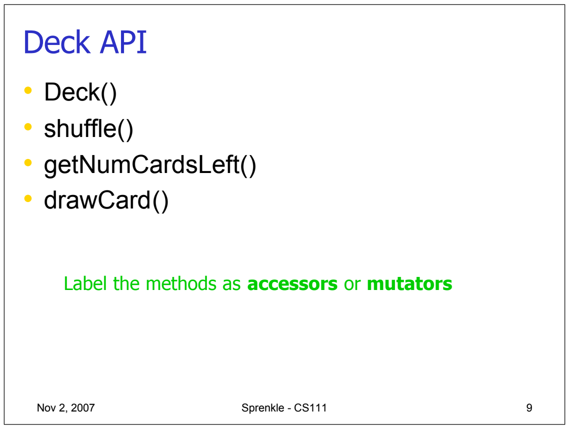
- List of Card objects

```
from card import *
from random import shuffle

class Deck:
  def __init__(self):
    self.cards = []
    for suit in ["clubs","hearts","diamonds","spades"]:
      for rank in range(2,15):
        self.cards.append(Card(rank, suit))

  def __str__(self):
    result = ""
    for c in self.cards:
      result += c.__str__() + "\n"
    return result
```

Alternatively, could write as str(c)



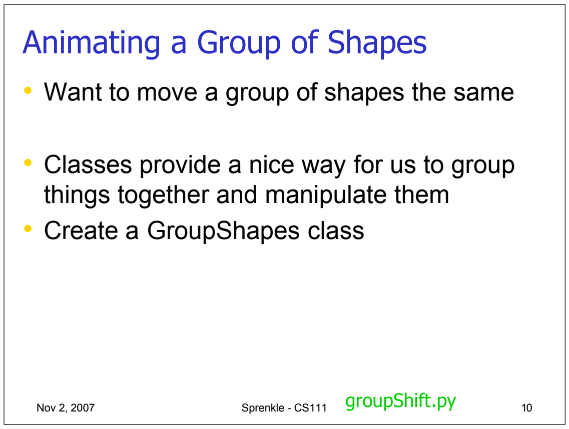
Deck API



Deck API

- Deck()
- shuffle()
- getNumCardsLeft()
- drawCard()

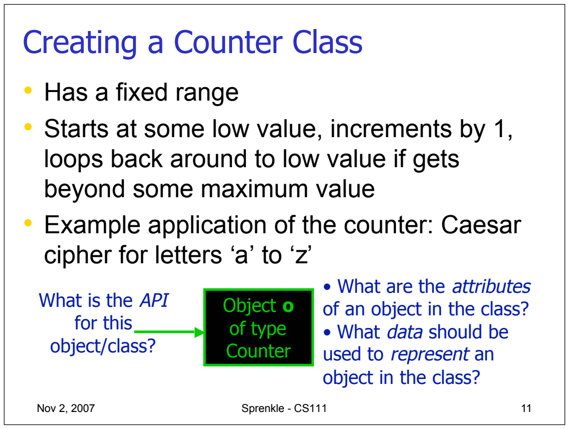
Label the methods as **accessors** or **mutators**



Animating a Group of Shapes

- Want to move a group of shapes the same
- Classes provide a nice way for us to group things together and manipulate them
- Create a GroupShapes class

groupShift.py

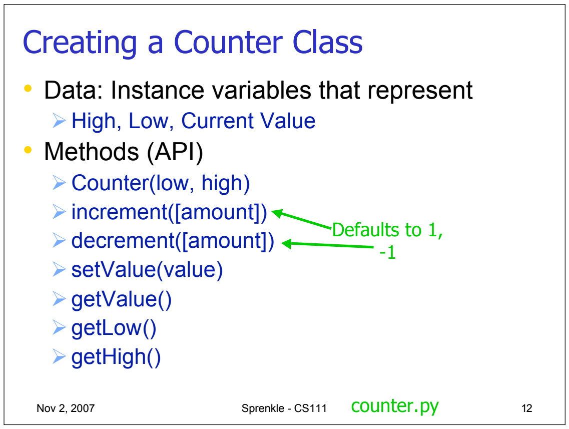


Creating a Counter Class

- Has a fixed range
- Starts at some low value, increments by 1, loops back around to low value if gets beyond some maximum value
- Example application of the counter: Caesar cipher for letters ‘a’ to ‘z’

What is the *API* for this object/class? → Object **o** of type Counter

- What are the *attributes* of an object in the class?
- What *data* should be used to *represent* an object in the class?



Creating a Counter Class

- Data: Instance variables that represent
  - High, Low, Current Value
- Methods (API)
  - Counter(low, high)
  - increment([amount])
  - decrement([amount])
  - setValue(value)
  - getValue()
  - getLow()
  - getHigh()

Defaults to 1, -1

counter.py

Applying the Counter Class

- To the Caesar Cipher program
  - Any restrictions on keys?

caesar2.py

Discussing Caesar Cipher

- Any drawbacks from using Counter class?

caesar2.py

Risks of Electronic Voting

- Why assigned?
  - Awareness of some of the issues
  - Assumed electronic voting is better than paper ballots
  - CS effect on society, policy
- Hope you’ll be knowledgeable when you read about these issues:
  - “Even *I* know that you should/shouldn’t do X”
  - Help you shape future policies

Risks of Electronic Voting

- What are the risks?
- For which of these risks would you (from just CS111) have been able to suggest a better solution?

Groups: Mod 3

Risks

- Encryption
- Checking boundaries
- Hardcoding passwords
- Not designing with security in mind
- Proprietary code (transparency of system)