Inheritance

Design a Playing Card Object

- The *suit* of a card is 0 (clubs), 1 (diamonds), 2 (hearts), 3 (spades)
- The *rank* is its face value plus 11 (jack), 12 (queen), 13 (king), ace (1)

Design a Playing Card Object

```
class Card(object):
    """represents a standard playing card."""

    def __init__(self, suit=0, rank=2):
        self.suit = suit
        self.rank = rank
```


Design a Playing Card Object

- What cards would these represent?

```
card1 = Card(1, 12)
card2 = Card(3, 1)
card3 = Card(4, 3)
```

Class Attributes vs Instance Attributes

- *Class attributes* are defined inside the class but outside its methods. They are associated with the Class itself.
- *Instance attributes*, such as suit and rank, are associated with instances of the class.

```
# Inside the Card class:
    suit_names = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
    rank_names = [None, 'Ace', '2', '3', '4', '5', '6', '7',
          '8', '9', '10', 'Jack', 'Queen', 'King']
```


Design a Playing Card Object

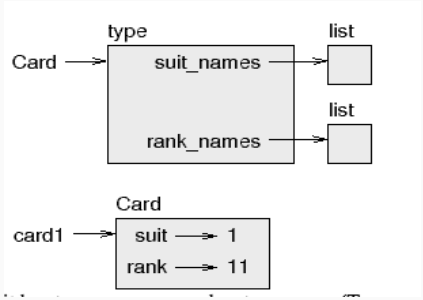
- What cards would these represent?

```
card1 = Card(1, 12)    # Queen of diamonds
card2 = Card(3, 1)     # Ace of spades
card3 = Card(4, 3)     # No such card
```


Creating and Printing a Card

- We can now create and print cards.

```
>>> card1 = Card(2, 11)
>>> print card1
Jack of Hearts
```



Class Attributes vs Instance Attributes

- Our __str__ method should extract the suit name and rank name from the class attributes

```
# Inside the Card class:
    suit_names = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
    rank_names = [None, 'Ace', '2', '3', '4', '5', '6', '7',
          '8', '9', '10', 'Jack', 'Queen', 'King']

    def __str__(self):
        return '%s of %s' % (Card.rank_names[self.rank],
                             Card.suit_names[self.suit])
```

Comparing Cards

•We can override the <, >, and == operators by defining the special __cmp__ method.

```
>>> card1 = Card(1,11)
>>> card2 = Card(1,6)
>>> card1 > card2
True
>>> card1 == card2
False
>>> card3 = Card(1,11)
>>> card3 == card1
True
>>>
```


Comparing Cards

•We can override the <, >, and == operators by defining the special __cmp__ method.

```
# inside class Card:
    def __cmp__(self, other):
        # check the suits
        if self.suit > other.suit: return 1
        if self.suit < other.suit: return -1

        # suits are the same... check ranks
        if self.rank > other.rank: return 1
        if self.rank < other.rank: return -1
        # ranks are the same... it's a tie
        return 0
```


Defining a Deck of Cards

• We can define a deck as a list of cards and use a ***nested loop*** to insert one of each card into the Deck.

```
class Deck(object):

    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)
```


Print the Deck

• The __str__ method constructs a list of the ***string representation*** of the cards.

• We convert the list to a string by using ***join()***.

```
#inside class Deck:

    def __str__(self):
        res = []
        for card in self.cards:
            res.append(str(card))
        return '\n'.join(res)
)
```

Deal, Add, Shuffle

• These methods are easy to write if we use built-in Python list methods and the random module.

```
#inside class Deck:

    def pop_card(self):
        return self.cards.pop()
    def add_card(self, card):
        self.cards.append(card)
    def shuffle(self):
        random.shuffle(self.cards)
```


Print the Deck

• The __str__ method constructs a list of the ***string representation*** of the cards.

• We convert the list to a string by using ***join()***.

```
>>> deck = Deck()
>>> print deck
Ace of Clubs
2 of Clubs
3 of Clubs
...
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
```


Inheritance

• Inheritance is the ability to define a new class (the ***child)*** that is a modified version of an existing class (the ***parent***).

• A hand in a game of cards is similar to a deck. Its a list of cards that we need to add, deal, shuffle, etc.

• This definition says a Hand inherits from Deck

```
class Hand(Deck):
    """represents a hand of playing cards"""
```


Initializing a Hand

• We need to override the __init__ method inherited from Deck.

• All the other methods remain unchanged.

```
class Hand(Deck):
    """represents a hand of playing cards"""

    def __init__(self, label=''):
        self.cards = []
        self.label = label
```

Inclass Exercises

- Write a __cmp__ method for Time objects. Hint: you might consider using integer subtraction.
- Write a Deck method named sort that uses the list method sort to sort the cards in a Deck. The list *sort* method will automatically use the __cmp__ method we defined to determine sort order.
- Write a function *deal(n)* that deals a hand of *n* cards and returns it.

Dealing a Hand

• Both Deck and Hand have the same methods, which can be used to deal a hand.

```
>>> hand = Hand('new hand')
>>> print hand.cards
[]
>>> print hand.label
new hand
>>> deck = Deck()
>>> card = deck.pop_card()
>>> hand.add_card(card)
>>> print hand
King of Spades
```