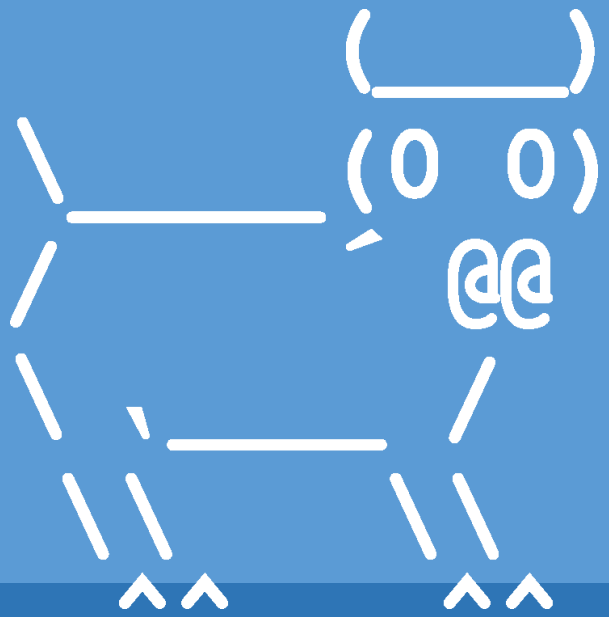
OBJECT-ORIENTED DESIGN

WEEK-AT-A-GLANCE



Introduction to Computer Science

Iris Howley

HAPPENING THIS WEEK

- There's a **quiz** this Friday, April 17
  - Check Glow!

- Homework 5 is due Monday, April 13
  - Homework 6 will be released Wednesday, April 15
- Lab 7 was released Friday, April 10
  - And it's due Thursday, April 16
- Lab 8 will be released Friday, April 17

LECTURES THIS WEEK

- Monday
  - Week Overview
  - Class Attributes
  - Inheritance Example
- Wednesday
  - Inheritance Syntax
  - Super Methods
  - Well-defined Classes
- Friday
  - Type conversion
  - Ciphers
  - Lab Intro

Prior to lecture videos…

Complete:

1. POGIL Activities: Inheritance
   - *available under Glow > Modules*
   - *also posted to the course website under Remote Lectures*

- Best done prior to watching lectures!
- Good for working with a partner (virtually, too!)
  - But will work without a partner, as well

Prior to this week's lessons…

Be able to:

1. Build & instantiate new classes & objects
2. …with attributes and methods

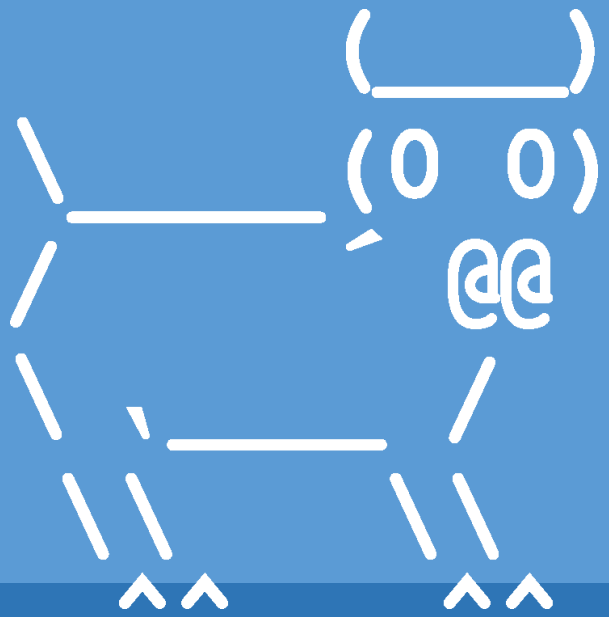
BOOK CHAPTER 18. INHERITANCE

Step through it!!!!

Highly recommended

Please contact me!

Class Attributes



Introduction to Computer Science

Iris Howley

TODAY'S LESSON

Class Attributes

(Some classes share some attributes across all instances)

Class Attributes Syntax

```
class EvilRobot:
   morality = 'evil'

   __slots__ = ['name']
   def __init__(self, nm):
      self.name = nm
```

Class attribute inside of the class

Restricting the instance attributes

Class methods

```
>>> er1 = EvilRobot('Herbert')
>>> er1.morality
'evil'
>>> er1.name
'Herbert'
```

Accessing the class attribute through the instance

Accessing the instance attribute

Class Attributes Syntax

```
class EvilRobot:
   morality = 'evil'

   __slots__ = ['name']
   def __init__(self, nm):
      self.name = nm

>>> er2 = EvilRobot('Pearl')
>>> er2.morality
'evil'
>>> er2.name
'Pearl'
```

Changing **Class** Attributes' Values

```
class EvilRobot:
   morality = 'evil'

   __slots__ = ['name']
   def __init__(self, nm):
      self.name = nm
```

Uses the CLASS name, not the instance name to re-assign!

```
>>> EvilRobot.morality = 'bad'
>>> er1.morality
'bad'
>>> er2.morality
'bad'
```

Changes the `morality` value for all objects of this type

Changing **Instance** Attributes' Values

```
class EvilRobot:
   morality = 'evil'

   __slots__ = ['name']
   def __init__(self, nm):
      self.name= nm
```

Uses the INSTANCE name, not the class name to re-assign!

```
>>> er1.name = 'Herbert the 2nd'
>>> er1.name
'Herbert the 2nd'
>>> er2.name
'Pearl'
```

Changes the `name` value for the specified instance

Does not change the `name` value for other instances

An Example

Image source: Fine Art America

Playing Cards

- Each card has a suit (one of 4)
  - Diamonds
  - Hearts
  - Clubs
  - Spades

Playing Cards

- Each card has a rank:
  - 2-10
  - Jack (11)
  - Queen (12)
  - King (13)
  - Ace

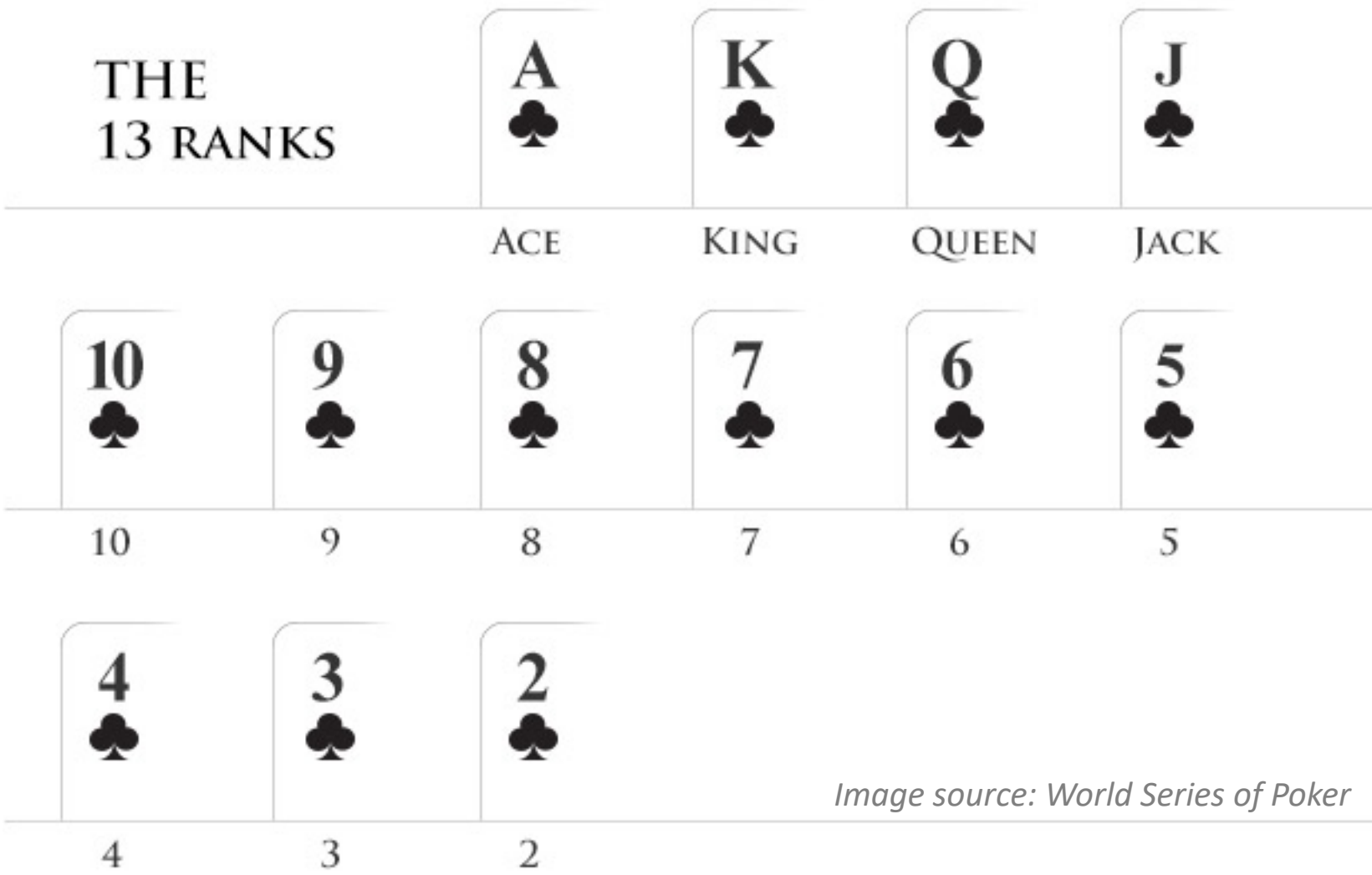


Image source: World Series of Poker

```
class Card:
    """ Represents a standard playing card. """
    # Instance Attributes
    __slots__ = ['suit','rank']

    # Class Attributes
    suit_names = ['Clubs','Diamonds','Hearts','Spades']
    rank_names =
[None,'Ace','2','3','4','5','6','7','8','9','10','Jack','Queen','King']



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

    def __str__(self):
        return '{} of {}'.format(Card.rank_names[self.rank],
Card.suit_names[self.suit])
```

```
if __name__ == '__main__':
    # Testing Card
    queenOfDiamonds = Card(1, 12)
    print(queenOfDiamonds) # Queen of Diamonds

    # See how c1.suit_names == c2.suit_names
    # But not c1.suit == c2.suit
    c1 = Card(2,11)
    print(c1)
    print("\tClass Attribute:", c1.suit_names)
    print("\tInstance Attribute:", c1.suit) # 2

    c2 = Card(3,5)
    print(c2)
    print("\tClass Attribute:", c2.suit_names)
    print("\tInstance Attribute:", c2.suit) # 3
```

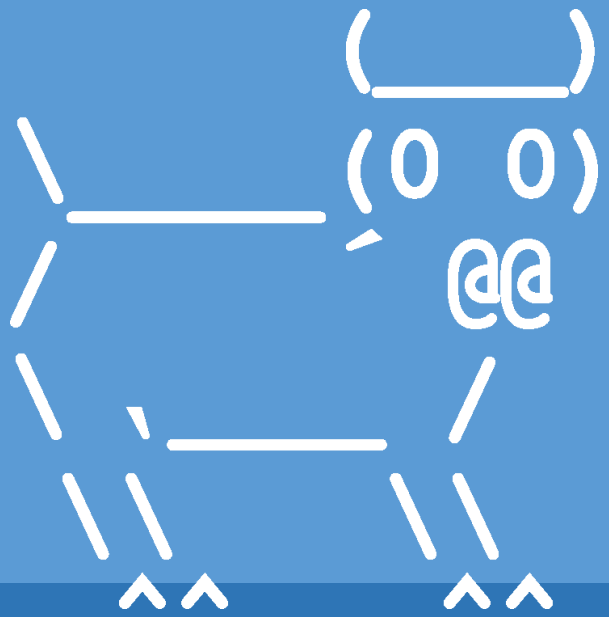
```
print("==", c1.suit_names == c2.suit_names)
# This will print '== True'
```

```
Queen of Diamonds
Jack of Hearts
Class Attribute: ['Clubs', 'Diamonds', 'Hearts', 'Spades']
Instance Attribute: 2
5 of Spades
Class Attribute: ['Clubs', 'Diamonds', 'Hearts', 'Spades']
Instance Attribute: 3
== True
```

```
print("==", c1.suit_names == c2.suit_names)
# This will print '== True'
```

Please contact me!

Inheritance: Example



Introduction to Computer Science

Iris Howley

TODAY'S LESSON

Inheritance

(A hierarchy of objects for leveraging parents' implementation)

Playing Card `Deck` (Poker)

- 52 cards
  - 13 ranks x 4 suits = 52
  - One of each card, every combination

Image source: Fine Art America

```
class Deck:
    """ Represents a playing card deck. """
    __slots__ = ['cards']

    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1,14):
                card = Card(suit, rank)
                self.cards.append(card)
    def __str__(self):
        res = [ str(card) for card in self.cards ]
        #res = []
        #for card in self.cards:
        #    res.append(str(card))
        return '\n'.join(res)
```

```
    def pop_card(self):
        return self.cards.pop()

    def add_card(self, card):
        self.cards.append(card)

    def shuffle(self):
        random.shuffle(self.cards)
```

```
if __name__ == '__main__':
    # Testing Deck
    thedeck = Deck()

    print('\t\t',thedeck, len(thedeck.cards))
```

Prints each card name on 50+ lines

Last line will be: `King of Spades 52`

```
    thedeck.pop_card()

    print('\t\t',thedeck, len(thedeck.cards))
```

Prints each card name on 50+ lines

Last line will be: `Queen of Spades 51`

Playing Card `Hand`

- The cards held by one player
  - Typically empty
  - One card at a time added *from* the Deck

Image source: Fine Art America

```
class Hand(Deck):
    """ Represents a playing hand """
    __slots__ = ['label']

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

```
if __name__ == '__main__':
    # Testing Hand
    hand = Hand('new hand')  []
    print('cards:',hand.cards)
    print('label:',hand.label)  'new hand'
```

Dealing a `Hand`

- Take one card from the `Deck`
- Add it to the player's `Hand`

Image source: Fine Art America

Dealing a `Hand`

- Take one card from the `Deck`
- Add it to the player's `Hand`

```
>>> deck = Deck()
>>> hand = Hand('player 1')
>>> card = deck.pop_card()
>>> hand.add_card(card)
>>> print(hand)
King of Spades
```

`hand` only contains one card, King of Spaces!

Image source: Fine Art America

Dealing a `Hand`

- Take *many* cards from the `Deck`
- Add them all to the player's `Hand`

Image source: Fine Art America

Dealing a `Hand`

- Take *many* cards from the `Deck`
- Add them all to the player's `Hand`

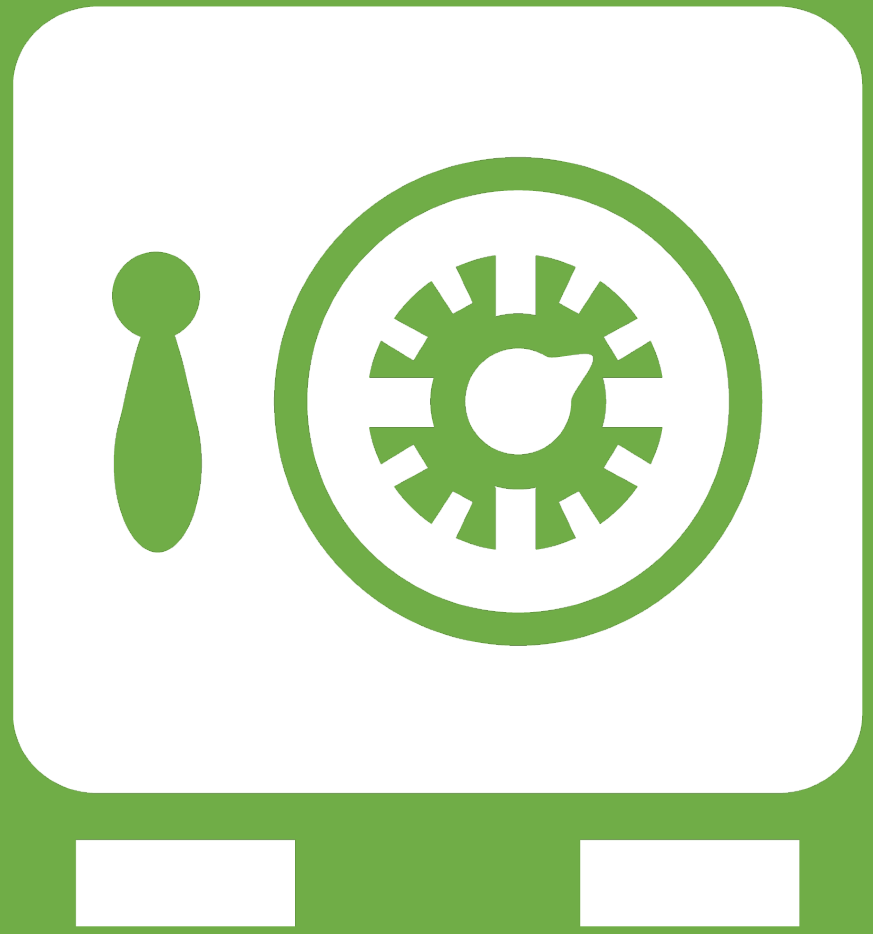
- Add to `Deck`:

```
def move_cards(self, hand, num):
    for i in range(num):
        hand.add_card(self.pop_card())
```

Image source: Fine Art America

```
if __name__ == '__main__':
# Testing Hand/Deck
    p1hand = Hand('player 1')
    newDeck = Deck()
    print("Starter Deck:", len(newDeck.cards)) 52
    print("Starter Hand:", p1hand)
    newDeck.shuffle()
    newDeck.move_cards(p1hand, 5)
    print("Final Deck:", len(newDeck.cards))  47
    print("Final Hand:", p1hand) 2 of Clubs
                                 7 of Spades
                                 9 of Spades
                                 8 of Diamonds
                                 2 of Spades
```

Please contact me!



Leftover Slides