Introduction to Programming CS101

Spring 2012

Lecture #8

Last week we learned

- Default parameters
- Named parameters
- Formatting
- String methods
- Photo processing

This week we will learn

- Objects
  - object creation
  - object attributes

There are 52 cards. Each card has a face and a suit. The suits are clubs, spades, hearts, and diamonds. The faces are 2, 3, ..., 10, Jack, Queen, King, Ace.

The value of a card is the number for a number card, 11 for an Ace, and 10 for Jack, Queen, and King.

We can represent cards as a triple (`face, suit, value`).

If `card` is a card, then `card[0]` is the face, `card[1]` is the suit, and `card[2]` is the value.

Computing the value of a hand:

```
def hand_value(hand):
  total = 0
  for card in hand:
    total += card[2]
  return total
```

Printing a card nicely:

```
def card_string(card):
  article = "a "
  if card[0] in [8, "Ace"]: article = "an "
  return article + str(card[0]) + " of  " + card[1]
```

Easy to make mistakes: What does `card[2]` mean? What if somebody creates a card `("Ace", "Spades", 5)`?

Let us define a new object type with attributes for face, suit, and value:

```
class Card(object):
    """A Blackjack card."""
    pass


card = Card()          ←──────────  create Card object
card.face = "Ace"
card.suit = "Spades"   ←──────  set attributes of card
card.value = 11
```

card has a user-defined type:

```
>>> type(card)
<class '__main__.Card'>
```

Computing the value of a hand:

```python
def hand_value(hand):
    total = 0
    for card in hand:
        total += card.value
    return total
```

Printing a card nicely:

```python
def card_string(card):
    article = "a "
    if card.face in [8, "Ace"]: article = "an "
    return (article + str(card.face) +
            " of " + card.suit)
```
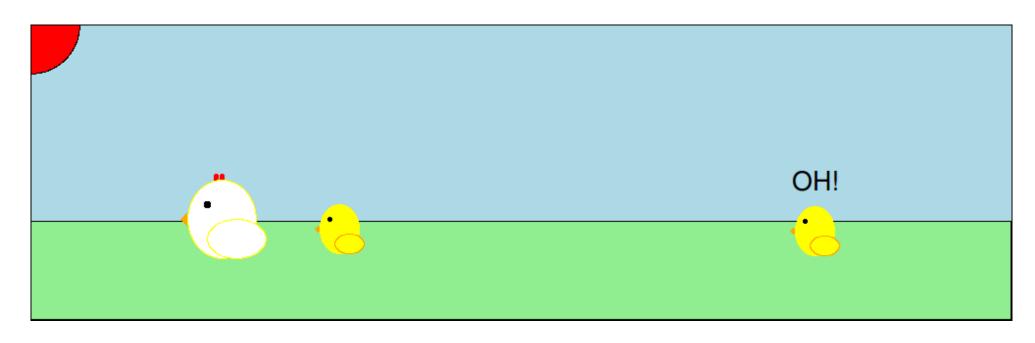
There is one big difference between tuples and Card objects:
Objects are mutable:

```
>>> card = Card()
>>> card.face = "Ace"
>>> card.suit = "Spades"
>>> card.value = 11
... and later ...
>>> card.suit = "Hearts"
```

An animation by Jeong-eun Yu and Geum-hyeon Song (2010 Freshmen).

Three `Layer` objects: mother hen, chick1, chick2.

Each chicken has `body`, `wing`, `eye`, and `beak`. Mother hen also has two red dots on the head.

The two chickens are exactly the same. Mother hen is larger and white.

The simplest method to make similar objects is to write the code once, and copy & paste it (with the necessary modifications).

Disadvantage: When you find a bug, you have to debug all copies of the code. It is not easy to change the appearance of all the chickens at once.

Let's try to implement the chicken as an object:

```python
class Chicken(object):
    """Graphic representation of a chicken."""
    pass
```

Our chicken will have attributes `layer`, `body`, `wing`, `eye`, and `beak`.

The function `make_chicken` creates a chicken object, with positioned at $(0,0)$.

```python
def make_chicken(hen = False):
  layer = Layer()
  if hen:
    body = Ellipse(70,80)
    body.setFillColor("white")
  else:
    body = Ellipse(40,50)
    body.setFillColor("yellow")
    body.move(0, 10)
  body.setBorderColor("yellow")
  body.setDepth(20)
  layer.add(body)
  # similar for wing, eye, beak, dots
```

Finally we create and return the Chicken object:

```python
def make_chicken(hen = False):
    # ... see previous page

    ch = Chicken()
    ch.layer = layer
    ch.body = body
    ch.wing = wing
    ch.eye = eye

    # return the Chicken object
    return ch
```

We use Chicken objects by accessing their attributes:

```
hen = make_chicken(True)
chick1 = make_chicken()
chick1.layer.move(120,0)

herd = Layer()
herd.add(hen.layer)
herd.add(chick1.layer)
herd.move(600, 200)

chick2 = make_chicken()
chick2.layer.move(800,200)
```