Classes 2

CSE 231, Rich Enbody

AA = int( 45 )  # AA = 45

BB = str( "Help!" )  # BB = "Help!"

CC = Point( 3, 5 )

print( AA )  45

print( BB.upper() )  HELP!

print( CC )  ( 3, 5 )
class Point ( object ):

    def __init__ ( self, x, y ):
        self.x_coord = x
        self.y_coord = y

    def __str__ ( self ):
        return "( {}, {} )".format ( self.x_coord, self.y_coord )

Using a class definition

- To create a new instance (object) of a class, use the class name:

    CC = Point ( 3, 5 )

- To send a message to an instance, call class methods:

    CC.translate ( 0, 20 )
**Class constructor**

```python
class Point(object):
    def __init__(self, x, y):
        self.x_coord = x
        self.y_coord = y

    def __str__(self):
        return "{{{0}},{{1}}}".format(self.x_coord, self.y_coord)
```

- **Keyword:** defining a class
- **Class name:** follows Python name rules
- **Super class:** shown in parens
- **Methods:** indented under the class
- **Data attributes:**
Constructor

- The class constructor is called to make a new an instance of the class
- It provides the class designer the opportunity to set up the data attributes (by assignment to instance variables)

Special Python Keywords

- Python has special uses for keywords that begin and end with __
- For example: the __doc__ attribute of a function
- In classes, we will see more of these special values
The `__init__` method

- The class constructor – a template for creating instances of the class
- First parameter, `self`, is special: During execution of the constructor, `self` refers to the instance being constructed and returned
- Arguments are passed for other parameters, if any, in the usual manner
- Create data attributes by assignment

Calling a constructor

- Using the name of a class as a function being called invokes the class (i.e., adding () after the class name)

- Example: `CC = Point( 3, 5 )`
  - creates a new instance using `__init__` method defined inside class `Point`
class Point( object ):
    def __init__( self, x=0, y=0 ):
        self.x_coord = x
        self.y_coord = y

    def translate( self, x_shift=0, y_shift=0 ):
        self.x_coord += x_shift
        self.y_coord += y_shift

    def __str__( self ):
        return "( {}, {} )".format( self.x_coord, self.y_coord )

Constructing an instance

p0 = Point()
p1 = Point(1,0)
p1.translate(3, 2)
Constructing an instance

class Point(object):
    def __init__(self, x=0, y=0):
        self.x_coord = x
        self.y_coord = y
    def translate(self, x_shift=0, y_shift=0):
        ...
Constructing an instance

```python
class Point(object):
    def __init__(self,x=0,y=0):
        self.x_coord = x
        self.y_coord = y
    def translate(self,x_shift=0,y_shift=0):
        ...

p0 = Point()
p1 = Point(1,0)
p1.translate(3, 2)
```

```plaintext
Point instance

<table>
<thead>
<tr>
<th>self</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Constructing an instance

```python
class Point(object):
    def __init__(self, x=0, y=0):
        self.x_coord = x
        self.y_coord = y

    def translate(self, x_shift=0, y_shift=0):
        ...
```

Global N.S.

Point

p0

p1 = Point(1, 0)
p1.translate(3, 2)
class Point(object):
    def __init__(self, x=0, y=0):
        self.x_coord = x
        self.y_coord = y
    def translate(self, x_shift=0, y_shift=0):
        ...

p0 = Point()
p1 = Point(1,0)
p1.translate(3, 2)
Constructing an instance

```python
class Point(object):
    def __init__(self, x=0, y=0):
        self.x_coord = x
        self.y_coord = y

    def translate(self, x_shift=0, y_shift=0):
        ... 
```

```
p0 = Point()
p1 = Point(1, 0)
p1.translate(3, 2)
```
Constructing an instance

class Point(object):
    def __init__(self, x=0, y=0):
        self.x_coord = x
        self.y_coord = y

    def translate(self, x_shift=0, y_shift=0):
        ...

p0 = Point()
p1 = Point(1,0)
p1.translate(3, 2)

Global N.S.
P0
p0  p1

translate’s N.S.
self
x_shift
y_shift

Point instance
x_coord  0
y_coord  0

Point instance
x_coord  1
y_coord  0

difference in definition

- A method is defined inside the suite of a class
- A method binds its first parameter to the object that the method is called on
- This parameter can be named anything, but traditionally it is named self

class myClass(object):
    def myMethod(self, otherparams):
        ...

Michigan State University
CSE 231, Fall 2013
self is an important variable: it is bound to the object that called the method.

Also, through self we can access the internal structure of the instance.

Back to the example

class Point( object ):
    def __init__( self, x=0, y=0 ):
        ...

    def translate( self, xs=0, ys=0 ):
        self.x_coord += xs
        self.y_coord += ys

    def __str__( self ):
        return "({},{})".format( self.x_coord, self.y_coord)
Binding self

```python
p0 = Point()
p0.translate(2, 3)
```

```python
class Point(object):
    def __init__(self, x=0, y=0):
        ...

    def translate(self, xs=0, ys=0):
        self.x_coord += xs
        self.y_coord += ys
```

Use `self` to refer to the target object

```python
class Point(object):
    def __init__(self, x=0, y=0):
        ...

    def translate(self, xs=0, ys=0):
        self.x_coord += xs
        self.y_coord += ys
```
Examples of new data types

- Types to model mathematical objects
  (complex numbers, fractions, etc)
- Types to model scalar objects
  (books, students, automobile, etc)
- Types to model containers
  (lists, queues, sets, etc)

There are special methods that have pre-defined roles for all classes. The first we learn is the constructor.
A constructor is called when an instance is made. It provides the class designer the opportunity to set up the instance with variables (by assignment).

If you don’t provide a constructor, a default constructor is provided. The default constructor does “system stuff” to create the instance, nothing more. You cannot pass arguments to the default constructor.
Examine Card and Deck classes in cards.py

Exercise:
*Define* `calculate_hand(H)` that *returns* the total value (sum) of cards in a hand where hand H is a list of cards.

Next: add properties to Card class