03-3
Classes and Methods

CSI 500

Spring 2018

Course material derived from:
Downey, Allen B. 2012. "Think Python, 2nd Edition". O'Reilly Media Inc., Sebastopol CA.

"How to Think Like a Computer Scientist" by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. Oct 2012
http://openbookproject.net/thinkcs/python/english3e/index.html

Object-oriented features

- Object-oriented programming
  - computation expressed as interaction among cooperating ensemble of objects
  - programs include class, attribute, and method definitions
- Class
  - User defined data type
  - an important real-world entity
- Objects
  - software instantiations of classes
- Attributes
  - data associated with classes and objects
  - correspond to real data in real-world things
- Methods
  - actions associated with classes and objects
  - correspond to the way real-world things interact
  - syntax for invoking a method is different than calling a function

```
class Time:
    """ Represents time of day
    attibutes: hour, minute, second
    """

time = Time()
time.hour = 11
time.minute = 59
time.second = 30

def print_time( t ):
    print( '%02d : %02d : %02d' % \
    (t.hour, t.minute, t.second)

print_time( time )
11 : 59 : 30
```

Printing objects

- Let's extend our earlier Time example
  - we'll include a revised method called "print" in the class definition
- Note use of "self" as the first parameter
  - This is a Python convention
  - Refers to the calling object

```
class Time:
    """ Represents time of day
    attibutes: hour, minute, second
    """
    def print_time( self ):
            print( '%02d : %02d : %02d' % \
            (self.hour, self.minute, self.second)

start = Time()
start.hour = 11
start.minute = 59
start.second = 30

# two ways to invoke the print_time functionality

# use the Time Class definition
 Time.print_time( start )
11 : 59 : 30

# use the method associated with the object
 start.print_time( )
11 : 59 : 30
```

Another example

- Let's revisit the Time implementation
  - convert time to integer
  - convert integer to time
- Note use of "self" parameter
  - Reference to invoking object
- Remember "divmod"
  - returns a tuple of the "div" and the "mod"
  - for example: divmod(7,3) returns (2, 1)

```
# continue with time examples
 class Time:
    """ Represents time of day
    attibutes: hour, minute, second
    """
 def print_time( self ):
       print( '%02d : %02d : %02d' % \
       (self.hour, self.minute, self.second))

   def time_to_int( self ):
       minutes = self.hour * 60 + self.minute
       seconds = minutes * 60 + self.second
       return seconds

   def int_to_time( self, seconds ):
       time = Time()
       minutes, time.second = divmod( seconds, 60)
       time.hour, time.minute = divmod( minutes, 60)
       return time
```

Another example

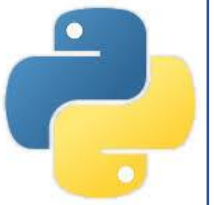
- Let's add on an "increment" method
  - this adds a specified number of seconds to the Time object

```
# continue our Time Class example…

# put this in your Time class after the
# last bit we just wrote…
#

# NOTE: Downey's book code doesn't work, so
# use my example here instead!

 def increment( self, seconds ):
       seconds += self.time_to_int()
       return self.int_to_time( seconds )
```



A more complicated example

- We can use our object tools to add a more complicated method
  - compare two Time objects
  - return True if first occurs later than second

```
# continue with our Time Class example

# this bit comes after the last bit

 def is_after( self, other):
       return self.time_to_int() > other.time_to_int()
```

The __init__ method

- by convention, the __init__ method is used to "initialize" an object
  - keyword parameters are used to correspond to attributes
- You may specify 0 or more parameters at initialization
  - any params you specify will override the defaults
  - any params you don't specify are filled in with defaults

```
# continue our Time Class example…

# put this in your Time class after the
# last bit we just wrote…
#

 def __init__( self, hour=0, minute=0, second=0 ):
     self.hour = hour
     self.minute = minute
     self.second = second



 t = Time(9)
 t.print_time()
09 : 00 : 00

 p = Time(9, 45)
 p.print_time()
09 : 45 : 00

 q = Time( hour=7, minute=13, second=47)
 q.print_time()
07 : 13 : 47
```

The __str__ method

- by convention, the __str__ method is used to "print" an object
  - returns a String object that you format as you see fit

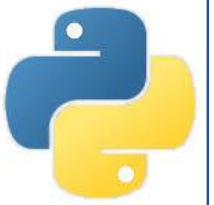
```
# continue our Time Class example…

# put this in your Time class after the
# last bit we just wrote…
#

 def __str__( self ):
     return '%02d : %02d : 02d' % \
     (self.hour, self.minute, self.second)

 q = Time( hour=7, minute=13, second=47)
 q.print_time()
07 : 13 : 47

 print( q )
07 : 13 : 47
```



Operator overloading

- Python allows you to "overload" operators such as "+" and "*"
  - you can define methods to handle these operators
- Here's an example of how we could "add" two Time objects together
  - the method is named __add__
  - the method is invoked using "+"

```
# continue our Time Class example…

# put this in your Time class after the
# last bit we just wrote…
#

 def __add__( self, other ):
     seconds = self.time_to_int() + \
                 other.time_to_int()
     return self.int_to_time( seconds )     # bk error!

# here's how to use it…

 p = Time( hour=3, minute=17, second=10)
 q = Time( hour=7, minute=13, second=47)
 print(p)
03 : 17 : 10

 print(q)
07 : 13 : 47

 print( p + q )
10 : 30 : 57
```

Debugging

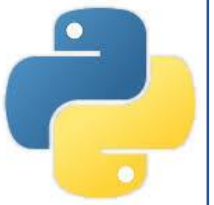
- The "vars" method is handy for debugging
  - returns a dictionary of attribute names and values
- You can wrap it in a handy function
  - print out attribute names and values

```
# continue our Time Class example…

 t1 = Time( 7, 43)
 vars( t1 )
{'second': 0, 'hour': 7, 'minute': 43}

 def print_attributes( obj ):
     for attr in vars( obj ):
                print( attr, getattr( obj, attr) )

# here's how to use it
 print_attributes( t1 )
second 0
hour 7
minute 43
```



Summary

- Object Oriented Programming
  - style of software in which problems are expressed as ensembles of collaborating objects
  - class defines object structure, attributes, and methods
  - operators can be overloaded (such as + or *) for user defined class types
- Python classes by convention include
  - __init__(self, [params]) to initialize an object (may specify params)
  - __str__(self) to print an object via user defined formatted printing