ECE12: Introduction to Programming
Lecture 16

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 16: Overview

- Object-oriented Programming
  - Special methods
    - Destructors
      - `__del__` method
    - String representation of objects
      - `__str__` method
      - `__repr__` method
  - Operator overloading
    - Unary operators
    - Binary operators
    - Advanced operators

Object-Oriented Programming

- Special methods for objects
  - Destructors
    - **`__del__(self)`**
  - Special method for object destruction
    - Called implicitly
      - whenever reference count to the object reaches zero
      - by the garbage collector
    - Typical situations when destructor is called:
      - the object is explicitly deleted by use of **`del`**
      - the object goes out of scope
      - a temporary object is no longer needed
      - the python interpreter exits
    - Allows for special clean-up procedures
    - Must not return any value (**`None`**)

Object-Oriented Programming

- Special methods for objects
  - Destructors
    - **`__del__(self)`**
  - Example: **`class Time`** (file **`time3.py`**)

```
# time3.py: abstract data type for representation of time
#           (version 3)
...

class Time:
    """abstract data type for representation of time"""
    ...

    def __del__(self):        # destructor
        """cleans up and destroys a time object"""
        # nothing to do here, but let's print a message
        print "Time object being destroyed!“
...
```

Object-Oriented Programming

- Special methods for objects
  - String representation
    - built-in function **`repr(object)`**
      actually calls method **`__repr__(self)`**
      - formal representation from which the object can be replicated (if possible)
      - implicitly used in interactive mode to print results
      - Example:
        » **`t1 = Time(); print repr(t1)`**
    - string conversion function **`str(object)`**
      actually calls method **`__str__(self)`**
      - informal, human-readable string representation
      - implicitly used with **`print`**
      - Example:
        » **`t1 = Time(); print str(t1)`**

Object-Oriented Programming

- Special methods for objects
  - String representation
    - method **`repr(self)`**
    - method **`str(self)`**
  - Example: **`class Time`** (file **`time3.py`**)

```
...
    def __str__(self):
        """returns an informal string representation"""
        return "%02d:%02d:%02d" %    \
               (self.__hour,self.__minute,self.__second)

    def __repr__(self):
        """returns a formal string representation"""
        return "Time(%d,%d,%d)" %    \
               (self.__hour,self.__minute,self.__second)
...
```

Operator Overloading

- What is operator overloading?
  - An operator performs different functions depending on the types of the arguments
- Some built-in operators are overloaded
  - Example: binary **+** operator
    - for integer operands, performs integer addition
    - for floating point operands, performs floating point addition
    - for string arguments, performs string concatenation
- Operator overloading for user-defined types
    - for class objects, most operators can be overloaded to perform a specific function defined by the class
    - special methods are defined for each operator that can be overloaded

Operator Overloading

- Mathematical operations
  - Unary operations

• **+**	**__pos__(self)**	positive
• **-**	**__neg__(self)**	negative
• **~**	**__invert__(self)**	invert
• **abs()**	**__abs__(self)**	absolute value
• **int()**	**__int__(self)**	integer conversion
• **float()**	**__float__(self)**	floating conversion
• ...		

Operator Overloading

- Mathematical operations
  - Binary operations
    - `+` `__add__(self, other)` addition
    - `-` `__sub__(self, other)` subtraction
    - `*` `__mul__(self, other)` multiplication
    - `/` `__div__(self, other)` division
    - `%` `__mod__(self, other)` modulo
    - `**` `__pow__(self, other)` power
    - `<<` `__lshift__(self, other)` left shift
    - `>>` `__rshift__(self, other)` right shift
    - `&` `__and__(self, other)` logical and
    - `|` `__or__(self, other)` logical or
    - `^` `__xor__(self, other)` logical xor
    - ...

Operator Overloading

- Advanced operations
  - *NOT covered in this class!*
  - Methods for attribute access (overloading the dot-operator)
    - **`__getattr__(self, name)`** attribute read-access
    - **`__setattr__(self, name, value)`** write-access
    - **`__delattr__(self, name)`** attribute delete-access
    - **`__dict__(self, name)`** internal access to attributes
  - Methods for sequence and mapping operations
    - **`__len__(self)`** length of a sequence
    - **`__contains(self, object)`** membership test
    - **`__getitem__(self, index)`** element read-access
    - ...

Operator Overloading

- Example: **`class Time`** (file **`time3.py`**)

```
...
    def Advance(self, hour=0, minute=0, second=0):
        """advances the time by the given amounts"""
        s = self.__second + second
        m = s / 60
        s %= 60
        m += self.__minute + minute
        h = m / 60
        m %= 60
        h += self.__hour + hour
        h %= 24
        self.SetTime(h, m, s)

    def Add(self, other):
        """adds two times and returns result in new object"""
        result = Time(self.__hour, self.__minute,    \
                                        self.__second)
        result.Advance(other.__hour, other.__minute, \
                                        other.__second)
        return result
...
```

Operator Overloading

- Example: **`class Time`** (file **`time3.py`**)

```
...
    def __add__(self, other):
        """overloading the + operation for time objects"""
        return self.Add(other)
```

- Interactive Example:

```
% python
>>> from time3 import Time
>>> t1 = Time(10, 20)
>>> t2 = Time( 2, 11, 30)
>>> print t1, t2
10:20:00 02:11:30
>>> t3 = t1 + t2
>>> print t3
12:31:30
>>> del t1
Time object 10:20:00 being destroyed!
>>> ^d
Time object 12:31:30 being destroyed!
Time object 02:11:30 being destroyed!
%
```