CSC 220 Data Structures

Object Oriented Programming II – Python specific

Parkland College Spring 2016

20160127
Operator Overloading & Special Methods

• Operator Overloading philosophy
  • Classes are types just like integers and strings.
    • Those have operators, so user-defined types should
• Special methods like `len()` for sequences
• Define either with `__opname__` format
• Never use that form anywhere else, use the actual operator or special method
• Some might be automatically defined for you
**vector.py**

- Robust constructor using exceptions

<table>
<thead>
<tr>
<th>Special method/operator</th>
<th>Use in code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>len</strong></td>
<td>len(v)</td>
</tr>
<tr>
<td><strong>getitem</strong></td>
<td>a = v[3]</td>
</tr>
<tr>
<td><strong>setitem</strong></td>
<td>v[3] = 22.3</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>u = v + w</td>
</tr>
<tr>
<td><strong>eq</strong></td>
<td>u == v</td>
</tr>
<tr>
<td><strong>ne</strong></td>
<td>u != v</td>
</tr>
<tr>
<td><strong>str</strong></td>
<td>print(v)</td>
</tr>
<tr>
<td><strong>neg</strong></td>
<td>-v</td>
</tr>
<tr>
<td><strong>lt</strong></td>
<td>u &lt; v</td>
</tr>
<tr>
<td><strong>le</strong></td>
<td>u &lt;= v</td>
</tr>
</tbody>
</table>
Element accessors

• `__getitem__`, `__setitem__`
• These will be used by most of the data structures we look at this semester
• Similar to `operator[]()` in C++
  • Would **never** do `v.operator[](3) = 22.3f;`
  • Therefore **never** do `v.__setitem__(3) = 22.3`
Inheritance

• Abstraction
  • Extract commonalities from similar classes into a different class
    • called super, base, or parent class

• New classes can derive from base class
  • Adaptability, Reusability, Encapsulation
Python Inheritance

• Example: Rectangle.py and Square.py
• from to avoid module namespace qualifier
• class Derived(Base)
• explicit call to base constructor
Super constructor

• if sub class does additional work in \_\_init\_, call super.\_\_init\_ at top of definition
• if sub uses super’s ctor exactly, don’t even define \_\_init\_ in sub.
• if sub needs a different ctor, define \_\_init\_, but don’t call super’s (rare in CSC 220).
Non-trivial inheritance example

• progressions.py
• essentially defining new generators
  • generator is a noun, so that makes sense
Contrast w/ C++ & Java classes

• No member access privileges
  • But follow the member naming convention!
• No inheritance access privileges
• No friendship
• No native abstract classes
  • No pure virtual methods
  • abc “abstract base class” module
Shallow & Deep Copyys

• Everything is an object
• Everything uses references
• Assignments create an alias: shallow copy
• To get actual duplicates: deep copy
  • import copy.deepcopy

• (See figures 2.9, 2.10, 2.11 in textbook)