1. The general syntax of class definition is:

```python
class MyClass [ ( superClass1 [, superClass2 ]* ) ]:
    """Document comment which becomes the __.doc__ attribute for the class"""
    def __init__(self, [param [, param]*):
        """Document comment for constructor method with self be referencing to the object itself"""
        #__init__body

    #defs of other class methods and assignments to class attributes

# end class MyClass
```

The three most important features of Object-Oriented Programming (OOP) to simplify programs and make them maintainable:

1. **encapsulation** - restricts access to an object's data to access only by its methods
   - helps to prevent indiscriminant changes that might cause an invalid object state (e.g., 6-side die with roll 8)
2. **inheritance** - allows one class (the subclass) to pickup data attributes and methods of other class(es) (the parents)
   - helps code reuse since the subclass can extend its parent class(es) by adding addition data and/or methods, or overriding (through polymorphism) a parent's methods
3. **polymorphism** - allows methods in several different classes to have the same names, but be tailored for each class
   - helps reduce the need to learn new names for standard operations (or invent strange names to make them unique)

```python
"""File: die_simple.py

This module defines the Die class.
"""

from random import randint
class Die(object):
    """This class represents a six-sided die."""
    def __init__(self):
        """The initial face of the die.""
        self._currentRoll = randint(1, 6)
    def roll(self):
        """Resets the die's value to a random number between 1 and 6.""
        self._currentRoll = randint(1, 6)
    def getRoll(self):
        """Returns the face value of the die.""
        return self._currentRoll
    def __str__(self):
        """Returns the string representation of the die.""
        return str(self._currentRoll)
"""
Consider the interface for a generalized AdvancedDie class that can have any number of sides.

<table>
<thead>
<tr>
<th>Method</th>
<th>Example Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>myDie = AdvancedDie(8)</td>
<td>Constructs a die with a specified number of sides and randomly rolls it</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Default of 6 sides if no argument supplied)</td>
</tr>
<tr>
<td><strong>cmp</strong></td>
<td>if myDie == otherDie:</td>
<td>Allows the comparison operations (&gt;, &lt;, ==, etc.) to work correctly for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AdvancedDie objects.</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>sum = myDie + otherDie</td>
<td>Allows the direct addition of AdvancedDie objects, and returns the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>integer sum of there current values.</td>
</tr>
<tr>
<td><strong>str</strong></td>
<td></td>
<td>Directly as: str(myDie) or indirectly as: print myDie</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns a string representation for the AdvancedDie. By overriding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the default <strong>str</strong> method the “print” statement will work correctly.</td>
</tr>
<tr>
<td>roll</td>
<td>myDie.roll()</td>
<td>Rolls the die randomly and return the value rolled</td>
</tr>
<tr>
<td>getRoll</td>
<td>myDie.getRoll()</td>
<td>Returns the current roll of the die</td>
</tr>
<tr>
<td>getSides</td>
<td>myDie.getSides()</td>
<td>Returns the number of sides on the die</td>
</tr>
<tr>
<td>show</td>
<td>myDie.show()</td>
<td>Displays the die’s value to standard output</td>
</tr>
</tbody>
</table>

Consider the following script and associated output:

```python
# testDie.py - script to test AdvancedDie class
from advanced_die import AdvancedDie

die1 = AdvancedDie(100)
die2 = AdvancedDie(100)
die3 = AdvancedDie()

die1 = 59
die2 = 49
die3 = 1
die1.show() = 59
die1.getRoll() = 59
die1.roll() = 53
die1.getRoll() = 53
die2.getRoll() = 49
die1 == die2: False
die1 < die2: False
die1 > die2: True
die1 <= die2: False
die1 >= die2: True
die1 != die2: True
die1.__str__(): 53
```

Notice that the testDie script needed to import AdvancedDie, but not the Die class.
File: advanced_die.py
Description: Provides a AdvancedDie class that allows for any number of sides
Inherits from the parent class Die in module die_simple

```python
from die_simple import Die
from random import randint

class AdvancedDie(Die):
    """Advanced die class that allows for any number of sides""

    def __init__(self, *args):
        """Constructor for any sided Die that takes an the number of sides
        as a parameter; if no parameter given then default is 6-sided.""
        # call Die parent class constructor
        Die.__init__(self)
        if len(args) == 0:
            self._numSides = 6
        elif len(args) == 1 and isinstance(args[0], int):
            self._numSides = args[0]
        else:
            print "Usage:  Die() or Die(numberOfSides)"
            return None

        self._currentRoll = randint(1, self._numSides)

    def roll(self):
        """Causes a die to roll itself -- overrides Die class roll""
        self._currentRoll = randint(1, self._numSides)
        return self._currentRoll

    def show(self):
        """Displays a Die by printing it""
        print self._currentRoll

    def __cmp__(self, rhs_Die):
        """Overrides the '__cmp__' operator for Dies, to allow for
to allow for a deep comparison of two Dice""
        if self._currentRoll < rhs_Die._currentRoll:
            return -1
        elif self._currentRoll == rhs_Die._currentRoll:
            return 0
        else:
            return 1

    def __add__(self, rhs_Die):
        """Returns the sum of two dice rolls""
        return self._currentRoll + rhs_Die._currentRoll

    def getSides(self):
        """Returns the number of sides on the die.""
        return self._numSides
```

---

a) What data attributes are inherited from the parent Die class?

b) What new data attributes are added as part of the subclass AdvancedDie?

c) Which Die class methods are used directly for an AdvancedDie object?

d) Which Die class methods are redefined/overridden by the AdvancedDie object?

e) Which methods are new to the AdvancedDie class and not in the Die class?