# Data Structures

Name:_____

1. **Classes:** A *class* definition is like a blueprint (receipe) for each of the objects of that class

- A class specifies a set of data attributes and methods for the objects of that class
- The values of the data attributes of a given object make up its state
- The behavior of an object depends on its current state and on the methods that manipulate this state
- The set of a class's methods is called its *interface*

A simple class definition example is a 6-sided Die:

```
"""
File: simple_die.py
Description:  This module defines a six-sided 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 following script to test the Die class and its associated output:

```
# testDie.py - script to test Die class
from simple_die import Die

die1 = Die()
die2 = Die()
print 'die1 =', die1    #calls __str__
print 'die2 =', die2
print
print 'die1.getRoll() = ', die1.getRoll()
print 'die2.getRoll() = ', die2.getRoll()
die1.roll()
print 'die1.getRoll() = ', die1.getRoll()
print 'str(die1): ' + str(die1)
print 'die1 + die2:', die1.getRoll() + die2.getRoll()
```

```
>>>
die1 = 2
die2 = 5

die1.getRoll() =  2
die2.getRoll() =  5
die1.getRoll() =  3
str(die1): 3
die1 + die2: 8
>>>
```

Classes in Python have the following characteristics:

- all class attributes (data attributes and methods) are *public* by default, unless your identifier starts with a single underscores, e.g, `self._currentRoll`
- all data types are objects, so they can be used as inherited base classes
- **objects are passed by reference when used as parameters to functions**
- all classes have a set of standard methods provided, but may not work properly (__str__, __doc__, etc.)
- most built-in operators (+, -, *, <, >, ==, etc.) can be redefined for a class. This makes programming with objects a lot more intuitive.

# Data Structures

Name:_____

Three important features of *Object-Oriented Programming* (OOP) to simplify programs and make them maintainable are:

1. *encapsulation* - restricts access to an object's data to its own methods
   ⇨ helps to prevent indiscriminant changes that might cause an invalid object state (e.g., 6-side die with a of 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 attributes 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)

Consider using inheritance to extend the Die class to a generalized AdvancedDie class **that can have any number of sides**. The interface for the AdvancedDie class are:

| Detail Descriptions of the AdvancedDie Class Methods | | |
|---|---|---|
| **Method** | **Example Usage** | **Description** |
| `__init__` | `myDie = AdvancedDie(8)` | Constructs a die with a specified number of sides and randomly rolls it (Default of 6 sides if no argument supplied) |
| `getRoll` | `myDie.getRoll()` | Returns the current roll of the die |
| `getSides` | `myDie.getSides()` | Returns the number of sides on the die |
| `roll` | `myDie.roll()` | Rerolls the die randomly |
| `__cmp__` | `if myDie == otherDie:` | Allows the comparison operations (>, <, ==, etc.) to work correctly for AdvancedDie objects. |
| `__add__` | sum = myDie + otherDie | Allows the direct addition of AdvancedDie objects, and returns the integer sum of their current roll values. |
| `__str__` | Directly as: `str(myDie)` or indirectly as: `print myDie` | Returns a string representation for the AdvancedDie. |

Consider the following script and associated output:

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

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

print 'die1 =', die1    #calls __str__
print 'die2 =', die2
print 'die3 =', die3
print 'die1.getRoll() = ', die1.getRoll()
print 'die1.getSides() =', die1.getSides()
die1.roll()
print 'die1.getRoll() = ', die1.getRoll()
print 'die2.getRoll() = ', die2.getRoll()
print 'die1 == die2:', die1==die2
print 'die1 < die2:', die1<die2
print 'die1 > die2:', die1>die2
print 'die1 <= die2:', die1<=die2
print 'die1 >= die2:', die1>=die2
print 'die1 != die2:', die1!=die2
print 'str(die1): ' + str(die1)
print 'die1 + die2:', die1 + die2

help(AdvancedDie)
```

```
die1 = Number of Sides=100 Roll=32
die2 = Number of Sides=100 Roll=76
die3 = Number of Sides=6 Roll=5
die1.getRoll() =  32
die1.getSides() = 100

die1.getRoll() =  70
die2.getRoll() =  76
die1 == die2: False
die1 < die2: True
die1 > die2: False
die1 <= die2: True
die1 >= die2: False
die1 != die2: True
str(die1): Number of Sides=100 Roll=70
die1 + die2: 146
Help on class AdvancedDie in module
advanced_die:
...
```

The AdvancedDie class that inherits from the Die superclass.

```
"""
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
"""
from simple_die import Die
from random import randint

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

    def __init__(self, sides = 6):
        """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)
        self._numSides = sides
        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)

    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 __str__(self):
        """Returns the string representation of the AdvancedDie."""
        return 'Number of Sides='+str(self._numSides)+' Roll='+str(self._currentRoll)

    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?