Introduction to Programming

Spring 2022

Functions

- Quick Review of Objects
- Example Program: Cannonball
- **Defining New Classes**
- **Data Processing with Class**
- Objects and Encapsulation
- Widgets
- Animated Cannonball

Defining New Classes

- Example:
  - A normal die (singular of dice) is a cube with six faces, each with a number from one to six.
  - Some games use special dice with a different number of sides.
  - Let's design a generic class `MSDie` (Multi-Sided Dice) to model multi-sided dice.

Defining New Classes

- Each `MSDie` object will know two things:
  - How many sides it has.
  - It's current value
- When a new `MSDie` is created, we specify n, the number of sides it will have.

Defining New Classes

- We have three methods that we can use to operate on the die:
  - `roll` – set the die to a random value between 1 and n, inclusive.
  - `setValue` – set the die to a specific value (i.e. cheat)
- In real work you will not include `setValue` for a real dice. Why?
  - `getValue` – see what the current value is.

Defining New Classes

- Using our object-oriented vocabulary, we create a die by invoking the `MSDie` constructor and providing the number of sides as a parameter.
- Our die objects will keep track of this number internally as an instance variable.
- Another instance variable is used to keep the current value of the die.
- We initially set the value of the die to be 1 because that value is valid for any die.
- That value can be changed by the `roll` methods, and returned by the `getValue` method.

Defining New Classes

- Class definitions have the form

```
class <class-name>:
    <method-definitions>
```

- Methods look a lot like functions! Placing the function inside a class makes it a method of the class, rather than a stand-alone function.
- The first parameter of a method is usually named `self`, which is a reference to the object on which the method is acting.

Defining New Classes

- Suppose we have a main function that executes `die1.setValue(8)`.
- Just as in function calls, Python executes the following sequence of steps:
  - `main` suspends at the point of the method application.
  - Python locates the appropriate method definition inside the class of the object to which the method is being applied.
  - Control is transferred to the `setValue` method in the `MSDie` class, since `die1` is an instance of `MSDie`.

Defining New Classes

- The formal parameters of the method get assigned the values supplied by the actual parameters of the call.
  - In the case of a method call, the first formal parameter refers to the object:
  - `self = die1`
  - `value = 8`
- The body of the method is executed.

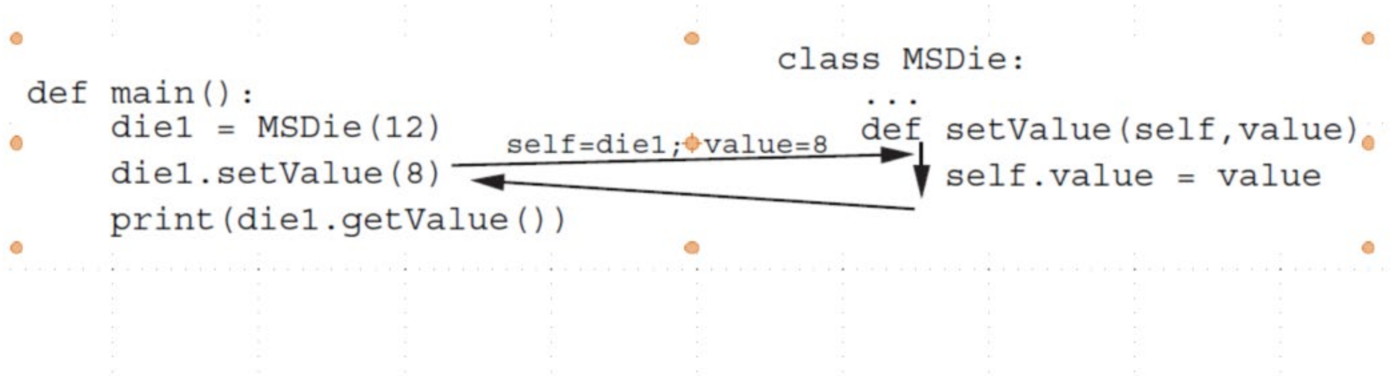
Defining New Classes

- Control returns to the point just after where the method was called. In this case, it is immediately following `die1.setValue(8)`.

- Methods are called with one parameter, but the method definition itself includes the self parameter as well as the actual parameter.

Defining New Classes

- The self parameter is a bookkeeping detail.
- We can refer to
  - the first formal parameter as the self parameter
  - other parameters as normal parameters.
- So, we could say `setValue` uses one normal parameter.



```
def main():                                  class MSDie:
    die1 = MSDie(12)                             ...
    die1.setValue(8)                             def setValue(self,value):
    print(die1.getValue())                           self.value = value
```

Defining New Classes

- Objects contain their own data.
- Instance variables provide storage locations inside of an object.
- Instance variables are accessed by name using our dot notation:
`<object>.<instance-var>`
- Looking at `setValue`, we see `self.value` refers to the instance variable value inside the object. Each `MSDie` object has its own value.
- We can also refer to `value` using the dot operator.

Defining New Classes

- Certain methods have special meaning. These methods have names that start and end with two _'s (underscore signs)
- `__init__` is the object contructor.
- Python calls this method to initialize a new `MSDie`.
- `__init__` provides initial values for the instance variables of an object.

Defining New Classes

- Outside the class, the constructor is referred to by the class name:
- `die1 = MSDie(6)`
- When this statement is executed, a new MSDie object is created and `__init__` is executed on that object.
- The net result is that `die1.sides` is set to 6 and `die1.value` is set to 1.

Defining New Classes

- Instance variables can remember the state of a particular object, and this information can be passed around the program as part of the object.
- This is different than local function variables, whose values disappear when the function terminates.

Data Processing with Class

- A class is useful for modeling a real-world object with complex behavior.
- Another common use for objects is to group together a set of information that describes a person or thing.
  - We need to keep track of student information like:
- Name, Credit Hours, Quality Points, etc
- A grouping of information like this is often called a record.

Data Processing with Class

- A grouping of information like this is often called a record.
- Suppose we have a data file that contains student grade information.
- Each line of the file consists of a student's name, credit-hours, and quality points.

```
Adams, Henry                127    228
Comptewell, Susan    100    400
DibbleBit, Denny     18     41.5
Jones, Jim           48.5   155
Smith, Frank         37     125.33
```

Data Processing with Class

- Our job is to write a program that reads this file to find the student with the best GPA and print out their name, credit-hours, and GPA.
- The place to start? Creating a `Student` class!
- We can use a `Student` object to store this information as instance variables.

Data Processing with Class

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
```

- The values for `hours` are converted to float to handle parameters that may be floats, ints, or strings.
- To create a student record:
- `aStudent = Student("Adams, Henry", 127, 228)`
- The coolest thing is that we can store all the information about a student in a single variable!

Data Processing with Class

- We need to be able to access this information, so we need to define a set of accessor methods.

```
 def getName(self):
        return self.name
def getHours(self):
        return self.hours
def getQPoints(self):
        return self.qpoints
def gpa(self):
        return self.qpoints/self.hours
```

Data Processing with Class

- For example, to print a student’s name you could write:

```
print aStudent.getName()
```

Data Processing with Class

- How can we use these tools to find the student with the best GPA?

```
Get the file name from the user
Open the file for reading
Set best to be the first student
For each student s in the file
   if s.gpa() > best.gpa
      set best to s
Print out information about best
```

Class Work

- Write a class to represent a square. It must be able to store its size and contain the following methods:
  - getSize
  - getArea
  - getParameter
- Write a class to represent a book
  - Data to keep track of: author, title, publisher
  - Include get and set methods for each of the instance variables.