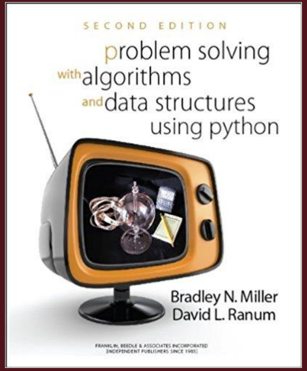
Problem Solving with Algorithms and Data Structure Using Python


SECOND EDITION
problem solving with algorithms and data structures using python
Bradley N. Miller
David L. Ranum

Unit 2: Classes

LECTURE 1: PROPER CLASSES

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- Write a class
- Make the class comparable
- Make the class iterable

Class Design

LECTURE 1

Class Design

- When you write a class there are a lot of things to consider. Especially if you are going to release your class for others to use. In this section we will build a simple class to represent a die that you can roll, and a cup to contain a bunch of dice. We will incrementatlly improve our implementations to take into consderation the following aspects of desiging a class that works well in the Python ecosystem.

Class Design

- Each class should have a docstring to provide some level of documentation on how to use the class.
- Each class should have a __str__ magic method to give it a meaninigful string representation.
- Each class should have a proper __repr__ magic method for representation in the interactive shell, the debugger, and other cases where string conversion does not happen.
- Each class should be comparable so it can be sorted and meaningfully compared with other instances. At a minimum this means implementing __eq__ and __lt__.
- You should think about access control each instance variable. Which attributes do you want to make public, which attributes do you want to make read only, and which attributes do you want to control or do value checking on before you allow them to be changed.

Class Design

If the class is a container for other classes then there are some further considerations:

- You should be able to find out how many things the container holds using len
- You should be able to iterate over the items in the container.
- You may want to allow users to access the items in the container using the square bracket index notation.

A Basic implementation of the MSDie class

Let’s start with a really simple implementation of the MSDie class, and we’ll improve it one step at a time. We want to make our die a bit flexible so the constructor will allow us to specify the number of sides.

MSDie.py

```python
import random
class MSDie:
    """
    Multi-sided die

    Instance Variables:
        current_value
        num_sides

    """
    def __init__(self, num_sides):
        self.num_sides = num_sides
        self.current_value = self.roll()
    def roll(self):
        self.current_value = random.randrange(1,self.num_sides+1)
        return self.current_value

my_die = MSDie(6)
for i in range(5):
    print(my_die, my_die.current_value)
    my_die.roll()

d_list = [MSDie(6), MSDie(20)]
print(d_list)
```

```
<__main__.MSDie object at 0x0000016E5CD2D470> 1
<__main__.MSDie object at 0x0000016E5CD2D470> 5
<__main__.MSDie object at 0x0000016E5CD2D470> 6
<__main__.MSDie object at 0x0000016E5CD2D470> 3
<__main__.MSDie object at 0x0000016E5CD2D470> 4
[<__main__.MSDie object at 0x0000016E5CD2D860>,
<__main__.MSDie object at 0x0000016E5CD2D550>]
```

A Basic implementation of the MSDie class

- This is a nice starting point. In fact, for some assignments this might be all you need. We have a class, we can construct a die, and roll it, and print out the current value. Sort of… It would be nicer if we could just print(my_die) and have the value of the die show up without having to know about the instance variable called current_value.
- Lets fix up the representation to make printing and interacting with the die a bit more convenient. For this we will implement the __str__ and __repr__ magic methods.

MSDie2.py

```python
import random
class MSDie:
    """
    Multi-sided die

    Instance Variables:
        current_value
        num_sides

    """
    def __init__(self, num_sides):
        self.num_sides = num_sides
        self.current_value = self.roll()
    def roll(self):
        self.current_value = random.randrange(1,self.num_sides+1)
        return self.current_value
    def __str__(self):
        return str(self.current_value)
    def __repr__(self):
        return "MSDie({}) : {}".format(self.num_sides, self.current_value)

my_die = MSDie(6)
for i in range(5):
    print(my_die)
    my_die.roll()
d_list = [MSDie(6), MSDie(20)]
print(d_list)
```

Make the Class Comparable

LECTURE 2 (TO BE ADDED)

Make the Class Iterable

LECTURE 3 (TO BE ADDED)