2.1. Writing a Proper Python Class

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.

- 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.

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.

2.1.1. A Basic implementation of the MSDie class

Lets 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.

Run | Load History | Show CodeLens

```
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()
```

(top/tree.html) (make_your_class

Activity: 2.1.1.1 ActiveCode (msdie_initial)

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.

Run Load History Show CodeLens

```
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()
```

Activity: 2.1.1.2 ActiveCode (msdie_initial1)

Notice that when we print a list of objects, the repr is used to display those objects. Having a good repr makes it easier to debug with simple print statements.

You have attempted 1 of 3 activities on this page

user not logged in

(toctree.html) (make_your_class