Ch. 2 Python Class

Sunday, April 19, 2020 3:08 PM

Fundamentals Ch. 2

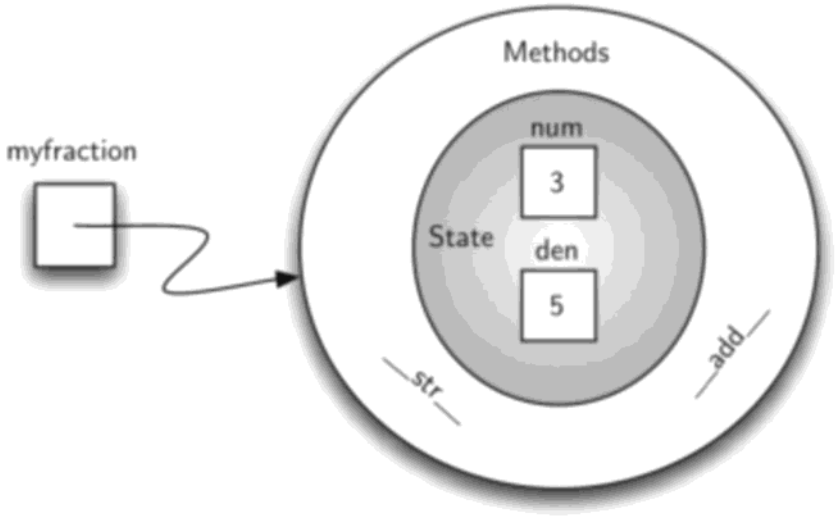

Figure 6: An Instance of the `Fraction` Class with Two Methods

2.1 Writing a Proper Python Class

Considerations for designing a class

- docstring provided on how to use the class
- Have a __st__ magic method to give it a meaningful string representation
- Have a __repr__ magic method for representation in the interactive shell
- It should be comparable with other instances (i.e. __eq__ and __lt__)
- Think about access control for each instance variable

Consideration for classes that's a container

- Find out how many things the container holds using len
- 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.

In [ ]:
```
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.current_value)
    my_die.roll()

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

In [ ]:
```
# Demonstrates what __str__ does

class Person:
    pass

# We tried creating a object from the class Person and printed it out
p = Person()
# It showed a Person instance at some memory reference
print(p)


# Next we again redefined the class overriding the __str__ method and when printe
class BetterPerson:
    def __str__(self):
        return 'BetterPerson Class'

g = BetterPerson()
print(g)
```

In [ ]:
```
# if we could just print(my_die) and have the value of the die show up without hc
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)
```

3.1 Algorithm Analysis

Algorithm: Step-by-step list of instructions for solving any instance of a problem. Program: An algorithm that has been encoded into some programming language.

A better algorithm

- More readable (i.e. better variable name)
- Efficient in use of computing resources (i.e. space, memory)