CSCI 204: Data Structures & Algorithms

ADT: Operator Overloading

Quick review: operator overloading

- We have learned some basic features of OOP, e.g. the Date class
  - Constructor: `def __init__(self)`
  - String representation: `def __str__(self)`, or `def __repr__(self)`
  - Method within a class: `def is_leap_year(self)`
  - Object attributes (object variables …) `self.year, self.month, self.day.`
- We will discuss and practice the topic of operator overloading

What does it mean?

- An operator such as '==', '>' can be associated with a function to reflect its meaning.
- E.g., in our Date class, we have three functions
  - is_equal(), is_before(), is_after()
  - When comparing two Date objects, we'd say d1.is_equal(d2), d1.is_before(d2), d1.is_after(d2)
- If we implement operator overloads for the Date class, we could have said
  - d1 == d2, d1 < d2, d1 > d2

Overloading '=='

```
class Date:
    …
    def __eq__(self, other):
        if self.year == other.year and \
            self.month == other.month and \
            self.day == other.day:
            return True
        else:
            return False
```

If the function is_equal() has been defined, we can do one of the following…

```
class Date:
    …
    def __eq__(self, other):
        if self.is_equal(other):
            return True
        else:
            return False
```

```
class Date:
    …
    def __eq__(self, other):
        return self.is_equal(other)
```

Overloading '>'

```
class Date:
    …
    def __gt__(self, other):
        return self.is_after(other)
```

Overloading '>='

```
class Date:
    …
    def __ge__(self, other):
        return self.is_after(other) or \
               self.is_equal(other)
```

Other operator overload

- Python supports more operator overload
  - **__ne__** : not equal
  - **__contains__** : membership check
  - **__add__** : add to the collection (+)
  - **__iadd_** : for +=
- See album_app demonstration

album_app.py, album.app, song.py, date.py

Build a Rational ADT

- A rational is a fraction number such that both the enumerator and the denominator are integers, relatively prime to each other
- Build a Rational ADT such that
  - Support common arithmetic rational operations
  - x, y are two Rationals, x+y, x-y, x*y, x//y are all Rationals
  - Support comparisons
  - x,y are two Rationals, x < y, x <= y, x == y, x >= y, x > y returns a True or False

Help function gcd(a,b)

- You'd need a **gcd(a,b)** function that returns the greatest common divisor of two integers **a** and **b**

```
def gcd(self, a, b):
    """ Return the greastest common divisor of a and b
    """
    if b == 0:
      return a
    else:
      return self.gcd(b,  a % b)
```

Examples of Rational

- x = 2/3, y = 1/2
- x + y == 7 / 6
- x - y == 1/6, y - x == - 1/6
- x * y == 1/3
- x // y == 4/3
- x > y True
- x >= y True
- x == y False
- x < y False
- x <= y False
- Test program is on the course website, once finished your implementation, try it out.