Introduction to Object-Oriented Programming (OOP) II
Operator overloading

• We have learned some basic features of OOP
  – Constructor: `def __init__(self):`
  – String representation: `def __str__(self):`, or `def __repr__(self):`
  – Method within a class: `def tomorrow(self):`
  – Object attributes (object variables ...)
    `self.year, self.month, self.day`.

• We now will discuss 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
  - isEqual(), isBefore(), isAfter()
  - When comparing two Date objects, we’d say $d1.isEqual(d2), d1.isBefore(), d1.isAfter()$
- 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 isEqual() has been defined, we can do ...

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

class Date:
    ...
    def __eq__(self, other):
        return self.isEqual(other)
Overloading ‘>‘

```python
class Date:
    ...
    def __gt__(self, other):
        return self.isAfter(other)
```

Overloading ‘>='

```python
class Date:
    ...
    def __ge__(self, other):
        return self.isAfter(other) or \
        self isEqual(other)
```

Run demonstration of Date class
Given a Book class as follows, define methods to overload ‘>’, ‘<’, ‘>=’, ‘<=>’, and ‘==’, if the comparison is based on the attribute ‘pubYear’

class Book:

    def __init__( self, title, author, pubYear ):
        ...

        Create an object
        ...

        self.author = author
        self.title = title
        self.pubYear = pubYear     # an integer
Class exercises

If the comparison is based on the attribute ‘title’, write the method that overloads ‘>’ using string comparison.

class Book:

    def __init__( self, title, author, pubYear ):
        ...
        Create an object
        ...
        self.author = author     # a string
        self.title = title       # a string
        self.pubYear = pubYear   # an integer
Class exercises

If the comparison is based on the attribute ‘pubYear’, if ‘pubYear’ is the same, then check ‘title’, if title is the same, check ‘author’.

class Book:
    def __init__(self, title, author, pubYear):
        
        Create an object

        self.author = author  # a string
        self.title = title    # a string
        self.pubYear = pubYear # an integer
Other operator overload

- Python supports more operator overload
  - __ne__ : not equal
  - __contains__ : membership check
  - __add__ : add to the collection (+)
  - __iadd__ : for +=

- See BookShelf demonstration