# Python, Day 7.5: Operator Overloading

Andrew Bydlon

January 17, 2019

# What is operator overloading?

Because we are able to define one operator/function name to mean multiple things using parent classes, we can **overload** an operator name.
The most obvious example of this that we have encountered from the beginning is "+":

## Example

```
> > > 2 + 3
5


> > > "Hello " + "friend"
"Hello friend"


> > > [3,4,5] + [6,7,8]
[3,4,5,6,7,8]
```

We say + is **overloaded** for int, str, and list.

# The method

The way that this is actually done is by using and underlying method:

__add__()

This is called a **special method** which corresponds to a specific operator symbol.

Note that you can call these methods explicitly:

## Example

```
> > > x,y,z,w = 4,5,"Hello ","friend"
> > > x.__add__(y)
9
> > > z.__add__(w)
"Hello friend"
```

# List of special methods I

Here are the standard mathematical operators:

1. +: __add__(self, object), Addition
2. -: __sub__(self, object), Subtraction
3. *: __mul__(self, object), Multiply
4. **: __pow__(self, object), Exponentiation
5. /: __truediv__(self, object), Division
6. //: __floordiv__(self, object), Integer Division
7. %: __mod__(self, object), Modulus/Remainder

Note there are 2 underscores on either side of the word.

# List of special methods II

Here are the standard boolean operators:

1. ==: __eq__(self, object), Equal to
2. !=: __ne__(self, object), Not Equal to
3. >: __gt__(self, object), Greater than
4. >=: __ge__(self, object), Greater than or equal
5. <: __lt__(self, object), Less than
6. <=: __le__(self, object), Less than or equal

# List of special methods III

Here are the standard positional operators:

1. in: __contains__(self, value), Membership
2. [index]: __getitem__(self, index), Not Equal to
3. len(): __len__(self), Number of
4. str(): __str__(self), Convert to string

**Note:** Similar to the case of the special operator __init__(self, object)

Let's test some code!

# An example

## Example

```
class Family:
    def __init__(self, members=[]):
        self.members = members

    def __add__(self,family):
        self.members += family.members
        family.members = self.members

    def __sub__(self,member):
        self.members.remove(member)

    def __mul__(self,member):
        self.members += ["Baby"]
```

# An example continued

## Example

```
> > > Smiths = Family(["Derek"])
> > > Jones = Family(["Kelly"])
> > > Smiths+Jones
> > > Smiths.members
['Derek', 'Kelly']
> > > Smiths*Jones
> > > Smiths.members
['Derek', 'Kelly', 'Baby']
> > > Smiths + Smiths
> > > Smiths.members
['Derek', 'Kelly', 'Baby', 'Derek', 'Kelly', 'Baby']
> > > Smiths - "Derek"
> > > Smiths.members
['Kelly', 'Baby', 'Derek', 'Kelly', 'Baby']
```

# Assignment 13

Create a class for Classes (as in school), with properties such as the title, the subject, a list of students, etc.
In addition, add the functions to

1. Add two classes: Concatenates the title and list of students. Uses the subject of the first class.
2. Multiply two classes: Uses the title and subject of the first class, but makes pairs of students, 1 from each class, up until you can no longer (maybe min and len will be useful?). You can omit or handle the remaining students, depending on ambition.
3. A check if 2 classes are equal: having the same list of students.

Then try running a few examples. **Upload the .py when you finish.**