Operator Overloading

Thomas Schwarz, SJ

Operator Overloading

- Operator overloading allows you to use expressions such as `a+b, a+=b, a==b, a<b,`
  - even if the operands are not numbers
  - but objects of a user defined class
- Python defines an association between operators and dunder functions
  - https://docs.python.org/3/reference/datamodel.html

Class Rational

- First a function that is not overloaded
  - inverse is an object method
  - returns the inverse of a rational number
  - $\left(\frac{p}{q}\right)^{-1} = \frac{q}{p}$

Class Rational

- Implementation:

```
def inverse(self):
        return Rational(self.den, self.num)
```

- Calling it:

```
>>> a = Rational(3,5)
>>> print( a.inverse() )
5/3
```

Class Rational

- Addition:
  - Use `__add__( )` dunder
- += operator:
  - Use `__iadd__( )` dunder
    - Needs to return self

Class Rational

- Implementation:
  - $\frac{p}{q} + \frac{r}{s} = \frac{ps + rq}{sq}$
    - Traditionally: use self and other as the variable names
    - Need to return the result

```
def __add__(self, other):
        return Rational(self.num*other.den+other.num*self.den,
                        self.den*other.den)
```

Class Rational

- += uses the __iadd__ dunder
  - Needs to return self (reason in the python manual)

```
def __iadd__(self, other):
        self.num, self.den =
self.num*other.den+other.num*self.den, self.den*other.den
        divisor = Rational.gcd(self.num,self.den)
        self.num = self.num//divisor
        self.den = self.den//divisor
        if self.den < 0:
            self.num = - self.num
            self.den = - self.den
        return self
```

Class Rational

- This part should be in a class method
  - def clean(self)

```
def __iadd__(self, other):
        self.num, self.den =
self.num*other.den+other.num*self.den, self.den*other.den
        divisor = Rational.gcd(self.num,self.den)
        self.num = self.num//divisor
        self.den = self.den//divisor
        if self.den < 0:
            self.num = - self.num
            self.den = - self.den
        return self
```

Class Rational

- Define multiplication

```
def __mul__(self, other):
        return Rational(self.num*other.num,
self.den*other.den)
```

Class Rational

- An example calculation
  - Heron's formula for the square root
    - Start with a guess $x = 1$
    - Improve the guess $x \leftarrow \frac{1}{2}(x + 2 \cdot \frac{1}{x})$
    - Repeat several times

Class Rational

```
def heron(n):
    result = Rational(1,1)
    for i in range(n):
        result = Rational(1,2) * (result +
                    Rational(2,1)*result.inverse())
    return result
```

```
>>> print(heron(10))
...
4584286909472409228225666455952521669217309116252649701885681720745376712709535405241807263985770088869113
4361639497243771528176716782876457748796945284249594870200383920805322146999779237833195077272838504228313
3099156077778147669481592899633555182948656598838967121365018560046135181125504905510456807418457335837652
0574859330076222527770624297071982141468006020346747954392375022095276441 7/3241580360592661071790620999021
5925889160576452720547724429585547651493040781333832362975891811042370960101005436793459936444123023843707
1808639277664637257090987314791426159312811991060429452743544130062958250208034504512010719340758722866368
0244724075107209602237068051342092557181515995460807316232483500965712150921959960316712288257851732340772
6312472428238351392267283960361495336307712
```