## Week 5.1 - Classes

**Example** Defining a class, called MyFraction, for fractions

In [8]:
```
import math

class MyFraction():
    # contructor
    def __init__(self, num, den):
        # attributes
        self.num = num
        self.den = den
        self.simplify()

    # calculates the floating point value
    def calc_float(self):
        return self.num / self.den

    # simplify the fraction
    def simplify(self):
        # find the greatest common divisor
        gcd = math.gcd(self.num, self.den)

        # simplify the numerator and denomenator
        self.num = int(self.num / gcd)
        self.den = int(self.den / gcd)

    # prints the function nicely (the clunky way)
    def nice_print(self):
        print('\n ' + str(self.num) + '\n---\n' + ' ' + str(self.den) + '\n')

    # redefine Python's str function to work with MyFraction objects
    def __str__(self):
        return '\n ' + str(self.num) + '\n---\n' + ' ' + str(self.den) + '\n'

    # overload the multiplication operator *
    def __mul__(self, other):

        num = self.num * other.num
        den = self.den * other.den

        return MyFraction(num, den)
```

**Example**: Define a fraction

In [12]:
```
a = MyFraction(2, 4)
```

**Example**: Access the numerator and denominator of a fraction and print them

In [13]:
```
print(a.num)
print(a.den)
```

```
1
2
```

**Example**: Calculate the floating point approximation using the `calc_float` method

In [14]:
```
a.calc_float()
```

Out[14]:
```
0.5
```

**Example**: Print a fraction using the `nice_print` method

In [15]:
```
a.nice_print()
```

```
 1
---
 2
```

**Example**: Print a fraction using Python's `print` function

In [16]:
```
print(a)
```

```
 1
---
 2
```

**Example**: Multiply two fractions and print the result

In [19]:
```
a = MyFraction(1, 2)
b = MyFraction(4, 5)
c = a * b
print(c)
```

```
 2
---
 5
```

## Week 5.2 - Class inheritance

**Example**: Define a subclass of MyFraction, called NamedFraction, that has an extra attribute that provides the name of the fraction (e.g. one quarter, two thirds, etc.)

In [34]:
```
class NamedFraction(MyFraction):

    # the constructor
    def __init__(self, num, den, name):
        # call the constructor for the superclass to inherit attributes
        super().__init__(num, den)
        self.name = name

    # redefine the str function
    def __str__(self):
        return self.name + ' = ' + str(self.num) + '/' + str(self.den)

    # approximate the fraction with n digits of accuracy
    def sig_fig(self, n):
        return round(self.num / self.den, n)
```

**Example**: Define a NamedFraction object

In [35]:
```
a = NamedFraction(1, 3, 'One third')
print(a.name)
print(a.calc_float())
```

```
One third
0.3333333333333333
```

**Example**: Print a NamedFraction object and compare it with the output from printing a MyFraction object

In [37]:
```
print(a)

b = MyFraction(1,3)
print(b)
```

```
One third = 1/3

 1
---
 3
```

**Example**: Approximate the value of a NamedFraction object with 3 digits of accuracy:

In [38]:
```
a = NamedFraction(1, 3, 'One third')
a.sig_fig(3)
```

Out[38]:
```
0.333
```