Introduction to Computer Programming

Week 5.2: Class inheritance

In [1]:
```
# This code is needed for the examples in the slides

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(' ' + str(self.num) + '\n---\n' + ' ' + str(self.den))

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

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

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

        return MyFraction(num, den)
```

Class inheritance

Class inheritance is mechanism to create a new class based on an existing class.

The new class is called the child class and the existing class is called the parent class.

In programming terminology, the child class is a **subclass** of the parent class.

The parent class is the child's **superclass**.

Class is inheritance is useful because the child class **inherits** the attributes and methods of its parent class.

Attributes can be added/modified to the child class without altering the parent class.

However, modifications to the parent class are automatically inherited by the child class.

Aims

In these slides, we'll learn how to use class inheritance to define a subclass.

Defining a subclass

Example: Define a class, called NamedFraction, that is a subclass of MyFraction. The NamedFraction subclass will:

- Have an extra attribute that stores the name of the fraction (e.g. one quarter, two thirds).
- Override the `__str__` function to print the name of the fraction
- Define a new function `sig_fig` that evaluates the fraction with a user-defined number of signficant figures

Before doing this, let's examine the definition of the MyFraction class:

```
class MyFraction():
    # code
```

The "argument" in a class definition is used to indicate its superclass, if there is one.

The empty round brackets () indicates that MyFraction does not have a superclass (so it is not a subclass).

Since NamedFraction **does** have a superclass (MyFraction), we must define it as:

```
class NamedFraction(MyFraction):
    # code
```

The constructor

As before, we must define the constructor for our new class.

This is done a little bit differently for a subclass, since we want it to inherit all of the attributes of its superclass.

To do this, we call the constructor for the superclass in the constructor for the subclass

In [2]:
```
class NamedFraction(MyFraction):
    # constructor
    def __init__(self, num, den):
        # call the constructor for the superclass to inherit attributes
        super().__init__(num, den)
```

Remember, even though the definition of `__init__` in the MyFraction class involves three arguments, we only pass two arguments when calling it

At this point, we can create NamedFraction objects with all of the features of MyFraction objects

In [3]:
```
a = NamedFraction(1, 4)
print(a)
```

```
 1
---
 4
```

Building the subclass: adding an attribute

We can add an attribute to the NamedFraction class to store the name of the fraction (given as input).

In [4]:
```
class NamedFraction(MyFraction):
    # constructor
    def __init__(self, num, den, name):
        # call the constructor for the superclass to inherit attributes
        super().__init__(num, den)
        self.name = name

a = NamedFraction(1, 3, 'One third')
print(a.name)
```

```
One third
```

Building the subclass: overwriting the str function

The NamedFraction class inherits the `__str__` method from MyFraction.

However, we'd like to modify this for the NamedFraction class so that it also prints the name and the fraction, e.g. One third = 1/3

This can be done by defining `__str__` in the NamedFraction class.

The changes here **will not** affect the MyFraction superclass

In [5]:
```
class NamedFraction(MyFraction):
    # 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)

# define and print a NamedFraction object
a = NamedFraction(1, 3, 'One third')
print(a)

# define and print a MyFraction object to show it remains the same
b = MyFraction(1, 3)
print(b)
```

```
One third = 1/3
 1
---
 3
```

Building the subclass: adding a new function

New functions can be added to subclasses.

These won't be available to objects belonging to the superclass.

Let's add a new function called `sig_fig` to the NamedFraction class that computes the floating point approximation to the fraction with n digits

In [6]:
```
class NamedFraction(MyFraction):
    # 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)

    # add a new function to approx. the fraction to n digits of accuracy
    def sig_fig(self, n):
        return round(self.num / self.den, n)

a = NamedFraction(1, 3, 'One third')
print(a.sig_fig(3))
```

```
0.333
```

Summary

- Class inheritance allows new classes to be defined that **inherit** the attributes of an existing class
- The new class is called a **subclass** of its parent class
- The parent class is the child's **superclass**

In this case, defining the NamedFraction subclass was relatively easy. This is because we could make use of all of the work we put into defining the MyFraction superclass