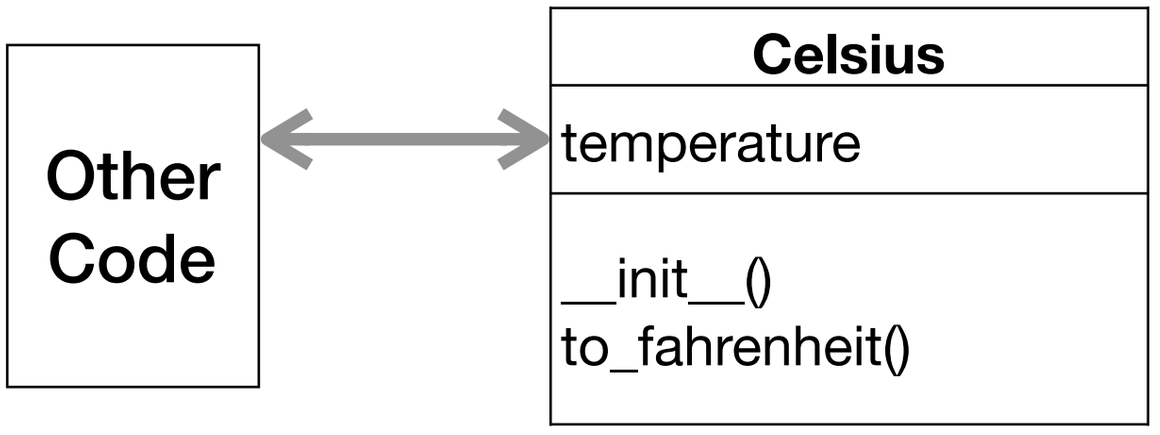
Getters & Setters

- With Getters and Setters
  - Class attributes only gettable and settable using get and set methods

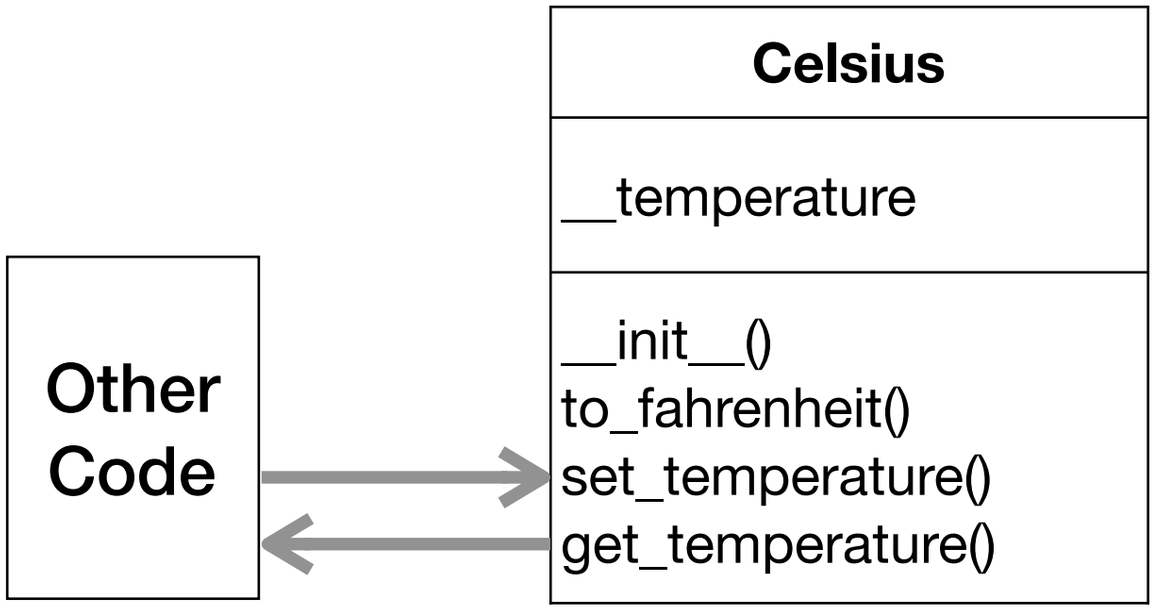
Overview

- Strict OOP philosophy
  - class data should not be directly accessible
  - Only through dedicated methods
    - Getters
    - Setters

Python allows this



Using get & set methods



Python allows this

- Class data (temperature) public
- No get and set functions

```
class Celsius:
    def __init__(self, temperature = 0):
        self.temperature = temperature

    def to_fahrenheit(self):
        return (self.temperature * 1.8) + 32
```

```
cold = Celsius(4)
In [3]:
cold.temperature
Out[3]:
4
In [4]:
cold.to_fahrenheit()
Out[4]:
39.2
In [7]:
cold.temperature = 26
cold.to_fahrenheit()
Out[7]:
78.80000000000001
In [8]:
cold.temperature = - 300
```

Slightly better

- But
  - data still public
  - `__init__` sets `temperature` directly

```
class Celsius:
    def __init__(self, temperature = 0):
        self.temperature = temperature

    def set_temperature(self, c):
        self.temperature = c

    def get_temperature(self):
        return self.temperature

    def to_fahrenheit(self):
        return (self.temperature*1.8)+32
In [6]:
hot = Celsius(31)
In [7]:
hot.set_temperature(32)
In [8]:
hot.get_temperature()
Out[8]:
32
```

The Pythonic Way

```
class Celsius:
    def __init__(self, temperature = 0):
        self.set_temperature(temperature)

    def to_fahrenheit(self):
        return (self.get_temperature()*1.8)+32

    # new update
    def get_temperature(self):
        return self._temperature

    def set_temperature(self, value):
        if value < -273:
            print("ERROR: Temp below -273")
        else:
            self._temperature = value
```

__init__ calls setter

some error handling

data ‘hidden’

```
In [25]:
vcold = Celsius(-273)
In [28]:
vcold.__dict__
Out[28]:
{'_temperature': -273}
In [27]:
vcold.set_temperature(-275)

ERROR: Temp below -273
In [29]:
vcold.get_temperature()
Out[29]:
-273
```

Exercise

- Write a class called KM that stores a distance in kilometres
- It should have `__init__`, getter and setter functions
- It should have `to_mile` and `to_yards` methods

MyDate & MyDateJ

- Alternative date classes (different name/constructors)
  - date1 = MyDate(6,11,2018)
  - dateJ2 = MyDateJ(6,11,2018)
- Methods
  - `print()` - prints date in *yyyy/mm/dd* format
  - `JulianDay()` - number of days since 1st Jan 4713 BC
  - `diff(other_day)` - number of days between this day and other_day
  - `weekday()` - day of week, e.g. 'Tuesday'
- MyDate stores date in day, month, year format
- MyDateJ stored Julian day number

Don't worry about the details of the conversion algorithms.

MyDate & MyDateJ Exercise

- Using a set of test dates, test the output of the JulianDay and weekday methods of both classes against each other.
  - e.g. testDates = [(29,2,2000),(13,5,2000),(13,4,2001),(15,5,2000), (13,5,2001),(13,5,1962),(3,11,2018)]
  - Loop through the test dates
    - Create MyDate and MyDateJ instances for each date
    - Compare the outputs of the JulianDay and weekday methods