Properties

- The Pythonic way to do Encapsulation is through Properties

- Unfortunately:
  - To come to grips with Properties we have to engage with some of the more esoteric aspects of the language.

Properties

- The Pythonic way to do Encapsulation is through Properties

- Unfortunately:
  - To come to grips with Properties we have to engage with some of the more esoteric aspects of the language.
- What is Syntactic Sugar
- Inner Functions
- Decorator Functions

Syntactic Sugar

IMO: Not a great metaphor

- **Syntactic sugar** is syntax within a programming language that is designed to make things easier to read or to express.
  - It makes the language "sweeter" for human use
  - things can be expressed more clearly, more concisely,
  - or in an alternative style that some may prefer.

```
dd = {"K1":"V1","K2":"V2"}
In [9]:
dd.get("K2")
Out[9]:
'V2'
In [11]:
dd["K2"]
Out[11]:
'V2'
```

The Hard Way

Syntactic Sugar

```
el = [3,4,5]
In [24]:
el.append(6)
el
Out[24]:
[3, 4, 5, 6]
In [26]:
el += [7]
el
Out[26]:
[3, 4, 5, 6, 7, 7]
```

More Syntactic Sugar

- Python List Comprehensions

The Hard Way

```
squares = []
for x in range(10):
    squares.append(x**2)
squares
Out[28]:
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Syntactic Sugar

```
squares = [x**2 for x in range(10)]
squares
Out[30]:
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Inner Functions

- Inner function not defined until outer function is called

```
def outer(num):
    def inc(x):
        return x + 1

    def dec(x):
        return x - 1

    if num == 1:
        return inc
    else:
        return dec

outer(1)(3)
Out[35]:
4

newInc = outer(1)
newInc(3)
Out[36]:
4
```

Wrapper/Decorator Functions

```
def my_decorator(func):
    def wrapper():
        print("==========")
        func()
        print("==========")
    return wrapper

def NineTT():
    for i in range(1,10):
        print(i, "x 9 = ",i*9)
In [53]:
NineTT()

1 x 9 =  9
2 x 9 =  18
3 x 9 =  27
4 x 9 =  36
5 x 9 =  45
6 x 9 =  54
7 x 9 =  63
8 x 9 =  72
9 x 9 =  81
```

- Changing the behaviour of `NineTT()` function

```
NineTT = my_decorator(NineTT)
In [51]:
NineTT()

==========
1 x 9 =  9
2 x 9 =  18
3 x 9 =  27
4 x 9 =  36
5 x 9 =  45
6 x 9 =  54
7 x 9 =  63
8 x 9 =  72
9 x 9 =  81
==========
```

Syntactic Sugar

- ‘Nicer’ syntax for applying a decorator function

```
@my_decorator
def EightTimesTables():
    for i in range(1,10):
        print(i, "x 8 = “,i*8)
```

equivalent to

```
EightTimesTables =
my_decorator(EightTimesTables)
```

```
@my_decorator
def EightTimesTables():
    for i in range(1,10):
        print(i, "x 8 = ",i*8)
In [55]:
EightTimesTables()

===*****=====
1 x 8 =  8
2 x 8 =  16
3 x 8 =  24
4 x 8 =  32
5 x 8 =  40
6 x 8 =  48
7 x 8 =  56
8 x 8 =  64
9 x 8 =  72
===*****=====
```

Object Attributes

- Attribute
  - General idea: *SomeObj.SomeAttr*
  - Simple case: *SomeAttr* is an ***instance variable*** of SomeObj

```
class Employee():
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
ff = Employee("Fred Flinstone", 45)
ff.__dict__
Out[65]:
{'name': 'Fred Flinstone', 'age': 45}
```

age is an instance variable

```
ff.age += 1

ff.__dict__
Out[65]:
{'name': 'Fred Flinstone', 'age': 46}
In [71]:
```

Promote ‘age’ to be a Property

- age now a property
- age is still an attribute of the form *SomeObj.SomeAttr*
  - has a setter and getter and error checking
  - backwardly compatible
    - `ff.age += 1`

```
ff = Employee("Fred Flinstone", 45)
In [68]:
ff.age += 1
In [69]:
ff.age
Out[69]:
46
In [70]:
ff.age = 90

Employees must be between 18 and 65.
```

```
class Employee():
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if value < 18 or value >66:
            print("Employees must be between 18 and 66.")
        else:
            self._age = value
```

Software Development Strategy

- Start off implementing attributes as local variables
- As system develops some attributes will be reimplemented as properties
- Backward compatibility maintained

@property - what you need to know

- Syntactic sugar that allows us to set up attributes as properties.

```
    @property
    def age(self):
        return self._age
```

```
    @age.setter
    def age(self, value):
        …
```

  - age attribute is accessible and assignable like a local variable
  - but getters and setters are used.

Python Decorators

- This @property syntax is a bit odd…
- This is the syntax for decorators
- Commonly used decorators that are built-ins in Python are `@classmethod`, `@staticmethod`, and `@property`.

- Primer on Decorators:
  - https://realpython.com/primer-on-python-decorators/

Celsius again

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

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

    @property
    def temperature(self):
        print("Getting value")
        return self._temperature

    @temperature.setter
    def temperature(self, value):
        if value < -273:
            print("ERR: Temp <-273 not possible")
        else:
            print("Setting value")
            self._temperature = value
```

```
In [57]:
mild = Celsius(18)

Setting value
In [60]:
Celsius.temperature.
Out[60]:
<property at 0x1104bcae8>
In [11]:
mild.temperature

Getting value
Out[11]:
18
In [13]:
mild.temperature = 21

Setting value
In [12]:
mild.temperature = -500
ERR: Temp <-273 not possible
```

Exercise

- Change the month attribute so that it is a property.
- Code the setter function so that it produces a message if the month is not an integer between 1 and 12.

```
class my_date():
    def __init__(self, d, m, y):
        self.day = d
        self.month = m
        self.year = y
    def show(self):
        print(self.day,'/',self.month,'/',self.year)
In [12]:
td = my_date(22,8,2018)
In [13]:
td.show()

22 / 8 / 2018
```

This is what you want...

```
td = my_date(22,8,2018)
In [17]:
td.show()
22 / 8 / 2018

td.month = 0

Month must be between 1 and 12.
```