```
# OOP Tutorial Wk 7
# Object oriented programming, OOP principles
# author: B. Schoen-Phelan
# date: Nov 2020

# a simple class without behaviour, just a scaffold of
a class
# class MyDogs:
#   pass
#
# a = MyDogs()
# b = MyDogs()
# print(a)
# print(b)
#
# # in Python you can add attributes during runtime
# a.age = 5
# a.colour = "black"
#
# b.age = 2
# b.colour = "white"

# print(a.age)
# print(b.colour)

# a class with behaviours
# class MyDogs:
#   def bark(self):
#     print("wouff wouff")
#
# rex = MyDogs()
# rex.bark()

# add another method, use self
# class MyDogs:
#   def bark(self):
#     print("wouff wouff")
#
#   def new_puppy(self):
```

```
#     self.age = 0
#     self.eyes = "closed"
#
# rex = MyDogs()
# rex.new_puppy()
# print(rex.age)
# print(rex.eyes)

# class with more methods with more arguments
# class MyDogs:
#   def bark(self):
#     print("wouff wouff")
#
#   def new_puppy(self):
#     self.age = 0
#     self.eyes = "closed"
#
#   def run(self, x, y):
#     self.x = x
#     self.y = y
#
#   def come_when_called(self):
#     self.run(0,0)


# duke = MyDogs()
# rex = MyDogs()
# rex.run(10, 6)
# duke.come_when_called()
# print(rex.x, rex.y)
# print(duke.x, duke.y) # notice that these are not
initialised anywhere at the start
# # dog without x and y:
# luna = MyDogs()
# print(luna.x) #gives an error message that MyDogs
doens't have an x attribute

# a position initialised dog
# class MyDogs:
```

```
#   def __init__(self, x, y):
#     self.run(x,y)
#
#   def bark(self):
#     print("wouff wouff")
#
#   def new_puppy(self):
#     self.age = 0
#     self.eyes = "closed"
#
#   def run(self, x, y):
#     self.x = x
#     self.y = y
#
#   def come_when_called(self):
#     self.run(0,0)
#
# luna = MyDogs(10, 4)
# print(luna.x, luna.y)

# class with "private" variables and get and set
methods
# class MyDog:
#   def __init__(self, x, y, age=0): #default value of
age is 0
#     self.run(x,y)
#     self._age = age
#
#   # get method
#   def get_age(self):
#     return self._age
#
#   def set_age(self, value):
#     self._age = value # or with validation as follows
(comment out previous line if using validation part):
# if type(value) != int:
#   raise Exception("something wrong")
# elif value < self._age:
#   raise Exception("can't get younger")
# elif value > (self._age+1):
#   raise Exception("can only age one year at a time")
# else:
#   self._age = value


#
#   def run(self, x, y):
#     self.x = x
#     self.y = y
#
#   def come_when_called(self):
#     self.run(0,0)
#
# luna = MyDog(0, 0)
# print(luna.get_age())
# luna.set_age(7)
# print(luna.get_age())

# using Python Property()
# class MyDog:
#   def __init__(self, x, y, age=0): #default value of
age is 0
#     self.run(x,y)
#     self._age = age
#
#   # get method
#   def _get_age(self): #these functions are visible,
unless you write them with an _
#     return self._age  #nothing prevents you from
using them right without going via the attribute
#                       #without the _
#   def _set_age(self, value):
#     self._age = value #or with validation, as above
#
#   def _del_age(self):
#     del self._age
#
```

```
#   dog_age_attr = property(_get_age, _set_age,
_del_age)
#
#   def run(self, x, y):
#      self.x = x
#      self.y = y
#
#   def come_when_called(self):
#      self.run(0,0)
#
# luna = MyDog(0, 0)
# print(luna.dog_age_attr)
# luna.dog_age_attr = 6
# print(luna.dog_age_attr)
# del luna.dog_age_attr
# print(luna.dog_age_attr) # causes an error if run
after previous line

# luna.set_age(4)
# print(luna.get_age())

class MyDog:
 def __init__(self, age=0):
   self._age = age

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

 @age.setter
 def age(self, value): #or with validation, as above
   self._age = value

 @age.deleter
 def age(self):
   del self._age


luna = MyDog()
```

```
print(luna.age)
luna.age = 4
print(luna.age)
del luna.age
print(luna.age) #causes an error after the previous
line,
# look at the message though, it complains about the
# original attribute: _age
```