Homework 4 Instructions

.

Included with this instruction file is the file `HW4.py`, containing the skeleton code for the homework.

Please read and follow all the directions carefully! Be sure to email me with any questions you have. **Note: There should be no `print` statements whatsoever!**

Put your name in the appropriate comment at the top of the program file `HW4.py`. Turn in a zipped file `HW4.zip` as described in the *About Homework* document.

Part 1. Person class (20 points)

Make a `Person` class containing `first_name` and `last_name` attributes that are passed into the `__init__` method with keyword names **as shown below**. (6 points)

- Add a **property** called `full_name` that **returns** the first and last names together with a space between them. (7 points)
- Add a **property** called `name` that **returns** the names together in the format of last name, followed by a comma and a space, followed by the first name. (7 points)

It should work **exctly** like this when you test it in the REPL:

```
>>> from HW4 import Person
>>> teacher = Person("Diane", "Chen")
>>> teacher.last_name
'Chen'
>>> the_name = teacher.full_name
>>> the_name
'Diane Chen'
>>> the_name = teacher.name
>>> the_name
'Chen, Diane'
>>> teacher.first_name = "D. D."
>>> teacher.full_name
'D. D. Chen'
>>> teacher.name
'Chen, D. D.'
>>> friend = Person(last_name='McMaster', first_name='Sonia')
>>> friend.name
'McMaster, Sonia'
>>> friend.full_name
'Sonia McMaster'
```

Part 2. Point class (30 points)

In the file `HW4.py`, there is a `Point` class defined as we saw in the lecture. It contains the method `get_magnitude` to return the magnitude value.

- (5 points each) Implement `__str__` and `__repr__` for the Point class **exactly** as shown here:

```
>>> from HW4 import Point
>>> point = Point(x=3.25, y=4.5)
>>> repr(point)
'Point(x=3.25, y=4.5)'
>>> str(point)
'Point at (3.25, 4.5)'
>>> point
Point(x=3.25, y=4.5)
>>> print(point)
Point at (3.25, 4.5)
```

- (8 points) Implement `x` and `y` defaults for Point of (0,0):

```
>>> point1 = Point()
>>> point1
Point(x=0, y=0)
>>> point2 = Point(y=9)
>>> point2
Point(x=0, y=9)
```

- (12 points) **Remove** the `get_magnitude` method and use its code to add a property method named `magnitude`:

```
>>> point1 = Point(3, 4)
>>> point1
Point(x=3, y=4)
>>> point1.magnitude
5.0
>>> point2 = Point(y=9)
>>> point2.magnitude
9.0
```

Part 3. Vehicle class (50 points)

Create a simple `Vehicle` class as shown below. (10 points) The inputs are `make`, `model`, `year`, `price`, and `color`, **in that order**. The values should be saved in the instance **under those names**. No defaults should exist for these values; they should all be required.

- (5 points each) Implement `__repr__` and `__str__` *exactly* as shown here. Note the formatting of the price:

```
>>> from HW4 import Vehicle
>>> car = Vehicle("Nissan", "Versa", 2018, 25000.5, "Silver")
>>> car.make
'Nissan'
>>> car.model
'Versa'
>>> car.year
2018
>>> car.price
25000.5
>>> car.color
'Silver'
>>> repr(car)
'Vehicle("Nissan", "Versa", 2018, 25000.50, "Silver")'
>>> car
Vehicle("Nissan", "Versa", 2018, 25000.50, "Silver")
>>> str(car)
'This is a 2018 Silver Nissan Versa costing $25000.50'
>>> print(car)
This is a 2018 Silver Nissan Versa costing $25000.50
>>> make, model, color = 'Toyota Camry White'.split() # See what I did there? Useful!
>>> car3 = Vehicle(make, model, 2020, 30000, color)
>>> str(car3)
'This is a 2020 White Toyota Camry costing $30000.00'
>>> repr(car3)
```

```
'Vehicle("Toyota", "Camry", 2020, 30000.00, "White")'
>>> car3.price
30000
```

- (10 points) Add a *property* `current_value` that calculates and **returns** the current value of the vehicle, based the vehicle's age and this completely arbitrary, silly, and unrealistic made-up formula:

Note: This formula needs to know the current year. I have provided a property attribute `current_year` so you can access the current year. I get the current year from a call to the `datetime` Python library's function `datetime.now()`, from which the `current_year` is extracted so you can calculate the age of the vehicle. Normally, I would not put this as a method on the object; but for our purposes, it's easier this way.

Because a vehicle loses some value immediately after purchase, calculate the vehicle's **age** as `self.current_year - self.year + 1`. If the vehicle's age is over 7 years old, then its current value is 10% of the price. Otherwise, its current value is the price minus 12.5% of the price for each year of age. **Note:** It should return a *number*, not a string.

```
>>> make, model, color = "Toyota Camry White".split()
>>> car2 = Vehicle(make, model, 2012, 30010.5, color)
>>> car2
Vehicle("Toyota", "Camry", 2012, 30010.50, "White")
>>> car2.current_value
3001.05
>>>
>>> car3 = Vehicle(make, model, 2021, 30010.5, color)
>>> car3
Vehicle("Toyota", "Camry", 2013, 30010.50, "White")
>>> car3.current_value
26259.1875
>>> car3 = Vehicle(make, model, 2020, 30010.5, color)
>>> car3.current_value
22507.875
>>> car3 = Vehicle(make, model, 2019, 30010.5, color)
>>> car3.current_value
18756.5625
>>> car3 = Vehicle(make, model, 2018, 30010.5, color)
>>> car3.current_value
15005.25
>>> car3 = Vehicle(make, model, 2017, 30010.5, color)
>>> car3.current_value
11253.9375
>>> car3 = Vehicle(make, model, 2016, 30010.5, color)
>>> car3.current_value
7502.625
>>> car3 = Vehicle(make, model, 2015, 30010.5, color)
>>> car3.current_value
3751.3125
>>> car3 = Vehicle(make, model, 2014, 30010.5, color)
>>> car3.current_value
3001.05
>>> car3 = Vehicle(make, model, 2013, 30010.5, color)
>>> car3.current_value
3001.05
>>>
>>> make, model, color = "Toyota Corolla Silver".split()
>>> car4 = Vehicle(make, model, 2016, 20000, color)
>>> car4
Vehicle("Toyota", "Corolla", 2016, 20000.00, "Silver")
>>> car4.current_value
5000.0
>>> car4 = Vehicle(make, model, 2017, 20000, color)
>>> car4.current_value
7500.0
>>> car4 = Vehicle(make, model, 2018, 20000, color)
>>> car4.current_value
10000.0
>>> car4 = Vehicle(make, model, 2019, 20000, color)
>>> car4.current_value
12500.0
>>> car4 = Vehicle(make, model, 2020, 20000, color)
>>> car4.current_value
15000.0
```

- (10 points each) Add checking in `__init__` to make sure that:

1. The year input is an integer
2. The price input is a number (either integer or float)

Raise a `TypeError` exception if either is not the case, with messages exactly as shown below.

Hint: Use the built-in function `isinstance()` - the documentation is here. The *classinfo* of `isinstance()` can be a type, or it can be a tuple of more than one type to test; `isinstance` will return `True` if the object is one of any of the types in *classinfo*.

There should be no `try/except` blocks in the code; you are *raising* an error to the calling code, not *handling* an error.

The *error messages* should look **exactly** like the lines starting with "TypeError":

```
>>> make, model, color = "Toyota Camry White".split()
>>> car3 = Vehicle(make, model, 201.2, 30000, color)
Traceback (most recent call last):
  [...]
TypeError: Input year must be an integer!
>>> car3 = Vehicle(make, model, 2012, color, color)
Traceback (most recent call last):
  [...]
TypeError: Input price must be a number!
>>>
```

Note: the `[...]` represents the traceback information (from Python), which may be different on each computer; do not try to print out the messages!

My email is dianechen.ucsdext@gmail.com. Please do not hesitate to email me if you have questions.