National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2020/2021

Solutions for Recitation 9
Object-Oriented Programming

Problems

1. Write a `Food` class
   - Input properties is the `name`, `nutrition` value, and `good_until` time.
   - Additional property is the `age` of the food, initially 0.
   - Methods are:
     - `sit_there` - takes an amount of time, and increases the age of the food by the amount.
     - `eat` - return the nutrition if the food is still good; 0 otherwise.

```
class Food(object):
    def __init__(self, name, nutrition, good_until):
        self.name = name
        self.nutrition = nutrition
        self.good_until = good_until
        self.age = 0

    def sit_there(self, time):
        self.age += time

    def eat(self):
        if self.age < self.good_until:
            return self.nutrition
        else:
            return 0
```

2. Write an `AgedFood` class
   - Input property is the same as the `Food` class, with an additional property, which is the `good_after` time.
   - Should inherit from the `Food` class.
   - Methods are:
     - `sniff` - returns `True` if it has aged enough to be good, `False` otherwise.
     - `eat` - returns 0 if the food is not good yet; otherwise behaves like normal food.

```
class AgedFood(Food):
    def __init__(self, name, nutrition, good_until, good_after):
```

```
        super().__init__(name, nutrition, good_until)
        self.good_after = good_after

    def sniff(self):
        return self.age >= self.good_after

    def eat(self):
        if self.sniff():
            return super().eat()
        else:
            return 0
```

3. Write a `VendingMachine` class
   - Input property is the same as the `Food` class.
   - Additional property is `age` of the `VendingMachine`, initially 0.
   - Methods are:
     - `sit_there` - takes an amount of time, and increases the age of the vending-machine by *half* that amount (it's refridgerated!).
     - `sell_food` - returns a new food instance with the appropriate name, nutrition and good_until.

```
class VendingMachine(object):
    def __init__(self, name, nutrition, good_until):
        self.name = name
        self.nutrition = nutrition
        self.good_until = good_until
        self.age = 0

    def sit_there(self, time):
        self.age += time/2.0

    def sell_food(self):
        food = Food(self.name, self.nutrition, self.good_until)
        food.age = self.age
        return food
```

4. Write `mapn`, which allows an arbitrary number of input lists[1], for example:

```
mapn(lambda x,y,z: (z, x+y),
  ((1, 2, 3), (4, 5, 6), ('first', 'second', 'third')))

#Output:        (('first', 5), ('second', 7), ('third', 9))
```

   You may use the regular `map` in your implementation.

[1] It turns out that the regular `map` is pretty similar to the `mapn` you will write here!

```
def mapn(fn, tuples):
    if len(tuples) == 0 or len(tuples[0])==0:
        return ()

    first_col = tuple(map(lambda x:x[0], tuples))
    rest = tuple(map(lambda x:x[1:], tuples))
    return (fn(*first_col),)+ mapn(fn,rest)
```

5. **Homework:** How would you implement the vending machine so that it can sell both Food and AgedFood (and possibly other things too?).