Homework 3

Problem 1

```
def convert_to_decimal(bits):
    exponents = range(len(bits)-1, -1, -1)
    nums = [num[0] * 2**num[1] for num in zip(bits,exponents)]
    return reduce(lambda acc, num: acc+num, nums)
```

Problem 2

a)

```
def parse_csv(lines):
    return [(lambda item : (item.split(',')[0], int(item.split(',')[1])))(item) for item in lines]
```

b)

```
def unique_characters(sentence):
    return set(sentence)
```

c)

```
def squares_dict(lower_bound, upper_bound):
    return {x: x**2 for x in range(lower_bound, upper_bound+1)}
```

Problem 3

```
def strip_characters(sentence, chars_to_remove):
    return ''.join([char for char in sentence if char not in chars_to_remove])
```

Problem 4

In the first function with num, num is a local variable that is created inside of the scope of the function, so when it is multiplied by 5, the local variable is set to 15 but is then garbage collected since it isn't used after. The original num variable stays 3.

In the box function, the box.value mutation affects the class object itself not just the instance of the class. Therefore, box.value can be mutated because it will be stored in the original object itself, and will thus reflect that when you print it.

Problem 5

Python's typing system does not enforce the restrictions on the types of objects that can be passed to a function; it is up to the implementation of the function itself to handle input validation or type checking.

When len(Foo()) is passed, the class has a definition for __len__() already defined so it outputs the result from it's type declaration. The same exists for any string.

When len(5) is passed, the class looks for a definition __len__() but the integer object doesn't have such a declaration because the type definition is immutable and has a fixed size in nature.

Problem 7

a)

```
def largest_sum(nums, k):
    if k < 0 or k > len(nums):
        raise ValueError
    elif k == 0:
        return 0

    max_sum = None
    for i in range(len(nums)-k+1):
        sum = 0
        for num in nums[i:i+k]:
            sum += num
        if max_sum is None or sum > max_sum:
            max_sum = sum
    return max_sum
```

Problem 8

a)

```
class Event:
    def __init__(self, start_time, end_time):
        self.start_time = start_time
```

```
        self.end_time = end_time

        if start_time > end_time:
            raise ValueError
```

b)

```
class Calendar:
    def __init__(self):
        self.events = list()

    def get_events(self):
        return self.events

    def add_event(self, event):
        if type(event) == Event:
            self.events.append(event)
        else:
            raise TypeError
```

c)

This happens because AdventCalendar does not inherit any of the member variables or functions from Calendar. Therefore, when get_events is called, no function is found that is attached to AdventCalendar.

To fix this, you could either define a new member variable get_events and create it's own get_events function. Or you could use the super().__init__ function to inherit Calendar's variables and functions.

Problem 9

```
def outf(x):
    def inf(y):
        return x - y
    return inf
```

This function displays that closure does exist. That is, the inner function is the closure and relies upon a variable x that is bound outside of inf(y) but can still utilize x and return the appropriate value we are looking for.