hw

1.

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

2.

a)

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

b)

```
def unique_characters(sentence):
    return {c for c in sentence}
```

c)

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

3.

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

4.

Here `num` is not mutated because it is, like all Python variables, an object reference. The `num` object reference is passed by **value** so a new reference is created for that function, meaning any change to the variable inside the scope of the function does not affect the original variable outside of the function scope (because a change means that the reference will be reassigned to the new value it is meant to be, rather than being the same reference and the value was changed). However the `value` attribute of `box` is modified because, while `box` is passed by value, it is still a reference to the original data which can be accessed (and modified) from any object that references it. So as long as we do not reassign our given `box` parameter we can continue to mutate the various references it contains.

5.

This is because of duck typing! Any object that implements the `__len__` method can be passed into the `len()` function and no `TypeError` will be thrown because that object's implementation for the `len()` method exists. An `int` object does not implement this method so when the program tries to call it on an `int` it throws an error.

6.

a)

```
class Squid:
    def __init__(self):
        pass

    def quack(self):
        print("Squid")
```

b)

```
class Dack(Duck):
    def __init__(self):
        pass
```

c)

The `try except` way is more Pythonic because, according to the article, Python favors to ask for forgiveness

rather than permission, which is certainly more akin to the `try except` method than the `isinstance` method.

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
```

8.

a)

```
class Event():
    def __init__(self, start_time, end_time):
        if start_time >= end_time:
            raise ValueError

        self.start_time = start_time
        self.end_time = end_time
```

b)

```
class Calendar():
    def __init__(self):
        self.__events = []

    def get_events(self):
        return self.__events

    def add_event(self, event):
        if not isinstance(event, Event):
            raise TypeError

        self.__events.append(event)
```

c)

Way 1

```
class AdventCalendar(Calendar):
    def __init__(self, year):
        self.year = year
        Calendar.__init__(self)

    def get_events(self):
        return super().get_events()
```

Way 2

```
class AdventCalendar(Calendar):
    def __init__(self, year):
        self.year = year
        Calendar.__init__(self)

    def get_events(self):
        return Calendar.get_events(self)
```

9.

Python **does** support closures: while it is different from many other languages in that its variables can be accessed outside of certain scopes like `if` and `while` blocks, its variables **are** isolated in scope to the functions they are defined in. Below is an example of closures affecting Python behavior.

```
def outer_func():
    def another_func():
        a = 5
```

```
        print(a)

    another_func()
    print(a)  # FAILS! `a` is only defined in `another_func`
```

This is an example of a closure because it displays the fact that the variable `a` is only valid for a certain scope, and cannot be used outside of that scope. This is a closure.