Ashanti Benons

```
def matrix_add_2d(a,b):
 result = []

 for i in range(len(a)):
    result.append([])
    for j in range(len(a[0])):
       result[i].append(a[i][j]+b[i][j])

return result;
```

Performs fast math on large array

1)

Besides speed, NumPy's addition ufunc allows for cleaner, more readable code. NumPy also has a built in functionality where it automatically checks for differences between the sizes of arrays, preventing certain errors that a manual function might not detect. NumPy also saves the user time as NumPy also allows you to add arrays of different shapes without explicit looping, so it is possible to add a one dimensional and two dimensional array without adding extra logic to the function.

2)

To check if my 2-dimensional lists are of the same size, I need to modify my matrix_add_2d function to first check if the two arrays passed as parameters have the same number of rows and columns. At the beginning of my function, I would need to add if len(a) != len(b). to check if the matrices have the same number of rows. If that evaluates true, I would then check if all rows have the same number of columns. (for in range(len(a)): if len(a[i]) != len(b[i]):) If any of these conditions fail, I could break out the program or raise a value error.

An error that would not occur in numPy but happens in python with 2-dimensional lists is an (#Error! Different row sizes). This wouldn't happen in NumPy because when you try to create a numPy array with different sized rows, a one dimensional array of lists is created instead. The user is also given a warning.

BroadCasting

1. a=np.array([[1,1,2],[3,2,1],[1,9,1]])
b=np.array([[1],[2],[3]])

Start: a.shape = (3,3) b.shape=(3,1)

Rule 1 doesn't need to be applied since both arrays are two dimensional.

But Rule 2 can be applied. Since b only has one column, it can be broadcasted across all the columns of a.

a.shape = (3,3) b.shape=(3,3)

a+b= [[1+1, 1+1, 2+1]
[3+2, 2+2, 1+2]
[3+1, 9+3, 1+3]]

Result of a+b: [[2 2 3
5 4 3
4 12 4]]

2. a=np.array([[[1,2],[3,1]],[[8,7],[9,1]]])
b=np.array([10,11])

Start: a.shape = (2,2,2) b.shape = (2,)

Rule 1 needs to be applied to b since the arrays have different dimensions.

b.shape = (1,1,2) a=(2,2,2)

Rule 2 is then applied to b to expand it to b.shape = (2,2,2), which is identical to a.

a+b = [[1+10, 2+11], [3+10, 1+11],
[8+10, 7+11], [9+10, 1+11]]

Result of a+b =

```
[
  [[11, 13],
   [13, 12]],
  [[18, 18],
   [19, 12]]
]
```

3. a = np.array([1,2,3])
b = np.array([[10],[11],[12]])

Start a.shape = (3,) b.shape = (3,1)
By applying Rule 1 to a to make it the same dimension as b, we get a.shape = (1,3)
Rule 2 is then applied to a. It is broadcast across the first dimension to match (3,3). b is then broadcasted across the second dimension to match (3,3).
a.shape = (3,3) b.shape = (3,3).

a+b =
```
[
  [10+1, 10+2, 10+3],
  [11+1, 11+2, 11+3],
  [12+1, 12+2, 12+3]
]
```

Result = [[11,12,13], [12,13,14], [13,14,15]]

4. a = np.array([[[1,2],[3,1]],[[8,7],[9,1]]])
b = np.array([10,11,12,13])

Start: a.shape = (2,2,2) b.shape = (4,)
First, Rule 1 can be applied to b so that both arrays end with the same dimensions.
b.shape = (1,1,4) a.shape = (2,2,2)
However, Rule 2 can't be applied since the last dimension of a is 2, while b is 4. One of these values must be 1 for NumPy to be able to stretch a dimension.
So the two arrays can't be added.

5. a=np.array([[[[1],]],[[[2]]]])
b=np.array([3,4])
Start: a.shape = (2,1,1) b.shape = (2,)
Using Rule 1, we can change b's shape to be three dimensional by adding two ones to the beginning. (1,1,2).
a.shape = (2,1,1)
b.shape = (1,1,2)
Using Rule 2, b can be expanded to (2,1,2) and a can be expanded to (2,1,2) to match b.

a+b = [[[1+3, 1+4], [2+3, 2+4]]]
a+b = [[[4,5]], [[5,6]]]

Answering numPy questions

1. 218,010 watts were generated all year.

2. The most watts in the morning was 446.4, while the least was 166.2. The most watts in the afternoon was 450.2, while the least was 0. The most in the evening was 247, while 0 watts was the least.

3. The avg watts generated on January afternoons is 211.4.

4. The system generated fewer than 500 watts total on 89 days.

5. An average of watts generated all year is 199 watts
Morning → 300.4 watts [North/south facing because more effective
Afternoon → 198.4 watts [East/west facing because not as effective
Evening → 98.5 watts [East/West facing because least effective
[Based on fact that North/south facing panels maximize sunlight exposure.