hw02

0.1 Homework 2

Please import the following package.

```
In [ ]: import numpy as np
```

0.1.1 Loops

How to work with multiple indices?

Use the code from Homework 1 Question 1 to write a function `flatten` that takes as input an array with shape `(r,c)` and outputs an array with shape `(rc,)`. For example, input `np.array([[1,2], [3,4]])` would yield output `np.array([1,2,3,4])`. Note that the array has two indices. The indices indicate the row and column. We flatten the array by going across the columns in each row.

```
In [ ]: # Implement this
        def flatten(arr):
            raise NotImplementedError()
```

Modify the function from Question 1.
Add a parameter called `major`.
Make the default value of `major` be `"row"`
Rewrite the loop so
If `major` is `"row"`, then the array is flattened in row major order
If `major` is `"column"`, then the array is flattened in column major order

```
In [ ]: # Implement this, including modifying arguments
        def flatten_v2(arr):
            raise NotImplementedError()
```

Modify the function from Question 1 to allow for more than two indices. For example, if the input is an array with shape `(r,c,h)` then the output is an array with shape `(rch,)`. So input `np.array([ [[1,2], [3,4]], [[5,6], [7,8]] ])` which has shape `(2,2,2)` would yield output `np.array([1,2,3,4,5,6,7,8])` which has shape `(8,)`. You should approach the problem using recursion.

```
In [ ]: # Implement this
        def flatten_v3(arr):
            raise NotImplementedError()
```

0.1.2 Storage

How to compress an array with lots of zeros

Write a function called `dense_to_sparse` that inputs an array and outputs a dictionary with
Keys as tuples containing `(row,column)` of all non-zero entries
Values as the corresponding non-zero entries.
The resulting entries should be in row-major order.
For example, if the input is `np.array([[1,0], [0,4]])` then the output is `{(0,0):1, (1,1):4}`

```
In [ ]: # Implement this
        def dense_to_sparse(arr):
            raise NotImplementedError()
```

Write an inverse function called `sparse_to_dense`

```
In [ ]: # Implement this
        def sparse_to_dense(arr):
            raise NotImplementedError()
```