Part 1

2a) it would output an array with 1000 entries, which the values are numbers in [0,1000].

b) **a** would be a 3 by 3 n-array with the first row being [1,2,3], second row being [4,5,6], and third row being [7,8,9]. **b** is the third row of the n-array -- in Python 0 represents first, 1 represents second, and so on.

c) **a** is the same as part b., but **b** becomes just a 1 by 9 array: [1,2,3,4,5,6,7,8,9].

d) **f** is a 5 by 1 array, which the values are numbers in [0,1], and **g** is transpose(**f**).

e) **x** is a 1 by 10 array, which every entry equals to 0.5, and **y** an array by the same size/shape and values of **x**. **z**, as the sum of **x** and **y**, is a 1 by 10 array, which every entry equals to 1.

f) **a** is [1,2,3,4,.....,99], and **b** is [99,98,97,.....,1].

3a)

```
> a=[6] #the number of sides of this die
> outcomes= []
> for i in a:
>   n=10 #the number of trials
>   out = 1+np.random.randint(i, size = n ) #I have to use 1+ because the
output is in [0,5]
>  outcomes.append(out)

> print(outcomes)
```

b)

```
> y = np.array([1, 2, 3, 4, 5, 6])
> z = y.reshape([3,2])
> print(z)
```

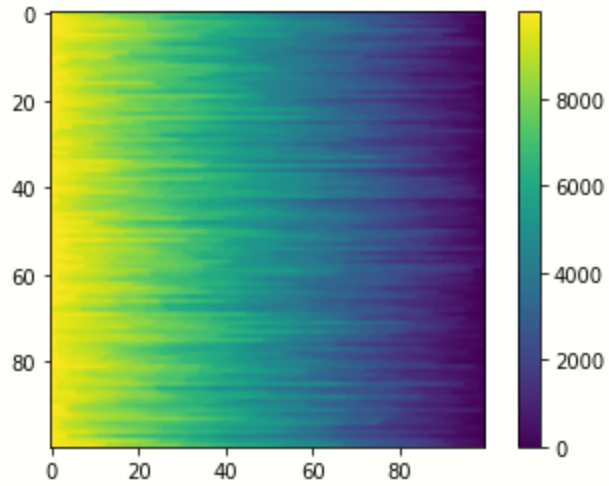
c)

```
> x = np.max(z)
> r = np.where(z==x)[0]
> c = np.where(z==x)[1]
> print(x,r,c)
```

d)

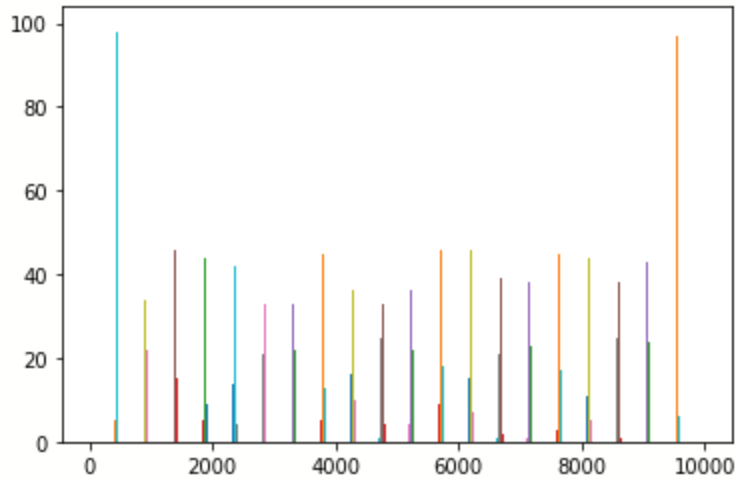
```
> v = np.array([1, 8, 8, 2, 1, 3, 9, 8])
> x = np.count_nonzero(v==1)
> print(x)
```

4a)

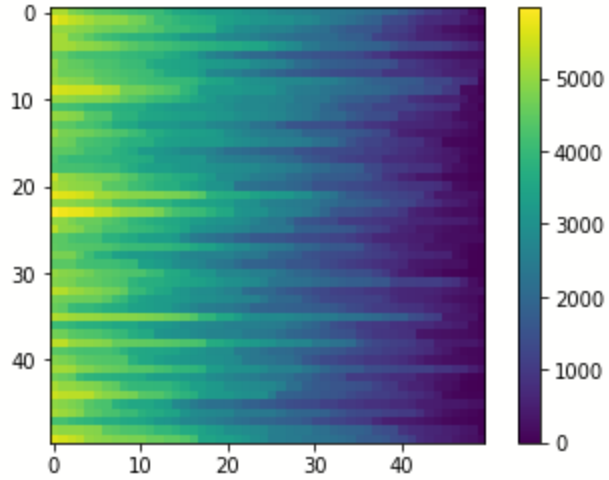


(the description is a bit unclear about whether it is the entire matrix in descending order, or just each row/column in descending order. I made each row in descending order)

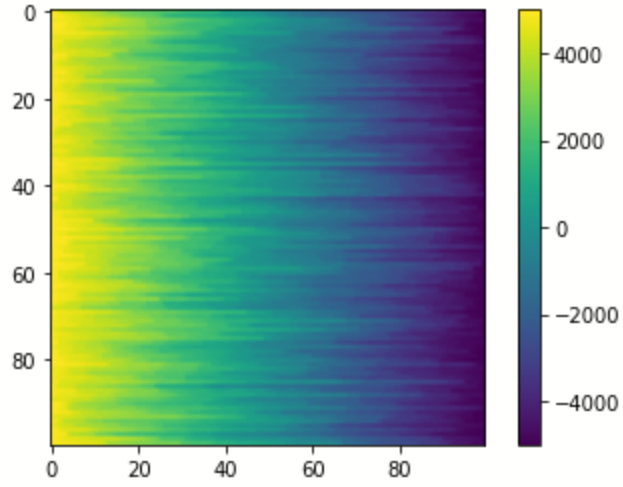
b)



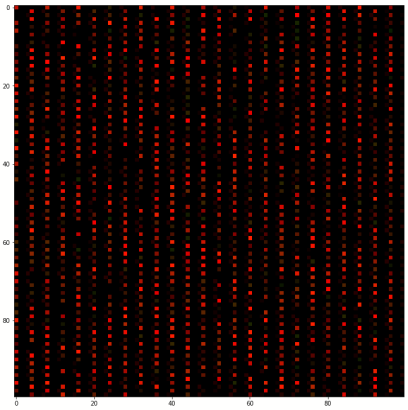
c)



d)



e)



(the attached zip has the image)