**A team of doctors and scientists has developed a new medical test for a rare disease that they say is 99% accurate. However, they warn, that number is deceiving.

**To explore the implications of this, suppose we have a population of 10000 people, of whom 1%, or 100 people, have the disease.

write a function called simulate_tests, which predicts each of 4 quantities:

**The number of true positives: the number of sick people who are correctly diagnosed as sick. The number of false positives: the number of healthy people who are incorrectly diagnosed as sick. The number of false negatives: the number of sick people who are incorrectly diagnosed as healthy. The number of true negatives: the number of healthy people who are correctly diagnosed as healthy.

**Your function should print all 4 quantities mentioned above and return the fraction of incorrect positive results: that is, the number of incorrect positive results divided by the total number of positive results.

***To simulate a single person's test:

**Randomly choose whether the person is sick with a probability of 0.01.

**Randomly choose whether the test is correct for them with a probability of 0.99.

**Here's some sample output of the program (user input is italicized):

Number of people: 10000 Test accuracy: 0.99 Infection rate: 0.01 True positives: 93 False positives: 113 False negatives: 1 True negatives: 9793 54.85436893203883% of positive tests were incorrect

In [ ]:

In [13]:
```
import random
def simulateTests(num_people, test_accuracy, infection_rate):
    print('Number of people: '+str(num_people)+' Test accuracy: '+str(te
st_accuracy)+' infection rate: '+str(infection_rate))
    truPos=0
    falPos=0
    truNeg=0
    falNeg=0
    for x in range(0,num_people):
        inf = False
        test = False
        if random.randint(0,int(infection_rate*100)) == 0:
            inf=True
        if random.randint(0, int(test_accuracy*100)) != 0:
            test = True
        if inf ==True and test==True:
            truPos+=1
        elif inf==True and test==False:
            falNeg+=1
        elif inf==False and test==True:
            falPos+=1
        elif inf==False and test==False:
            truNeg+=1
    print('True positives: '+str(truPos)+'\nFalse positives: '+str(falPo
s)+ '\nTrue negatives: '+str(truNeg)+
          '\nFalse negatives: '+str(falNeg)+'\n'+str(100*falPos/(truPos+f
alPos))+'% of positive tests were incorrect')
simulateTests(1000, .99, .01)
```

```
Number of people: 1000 Test accuracy: 0.99 infection rate: 0.01
True positives: 508
False positives: 479
True negatives: 5
False negatives: 8
48.530901722391086% of positive tests were incorrect
```

Netflix Score

You've just begun working for a company whose goal is to help users make new friends based on what they like to watch and read. Users input information about themselves, such as their Netflix history and favorite books. Your program will use this information to calculate a 'compatibility score' between two people, which serves as an estimate of how likely those people are to get along with one another. The compatibility score of two people is calculated as follows:

compatibility = (books liked in common) + (shows on Netflix liked in common)

we'll represent the books and movies liked by a particular user as separate lists. Given the lists representing, for example, the books liked by two different users, we find the number of elements present in both lists and divide it by the sum of the lengths of the two lists.

implement the following function:

which takes in two lists of strings and returns the number of elements the two lists have in common divided by the total number of elements in both lists. For example, in_common(['a', 'b', 'c', 'd'], ['c', 'd', 'm', 'n', 'x', 'z']) would return 0.2, because both lists contain 'c' and 'd' and there are 4 elements in the first list and 6 in the second.

In [2]:
```
def in_common(list1, list2):
    counter=0
    for x in list1:
        for y in list2:
            if x==y:
                counter+=1
    return (counter/(len(list1)+len(list2)))

# test your method here

in_common(['a', 'b', 'c', 'd'], ['c', 'd', 'm', 'n', 'x', 'z'])
```

Out[2]:
```
0.2
```

def calc_score(netflix_history1, netflix_history2, fav_books1, fav_books2)

takes the names and preferences of two users and returns their compatibility score. The compatibility score between two users is the fraction of shows on Netflix in common + the fraction of books in common, You may assume that there are no repeated elements in any of the lists.

In [3]:
```
# you must use the in_common method in this function
def calc_score(netflix_history1, netflix_history2, fav_books1, fav_books
2):
    user1 = [netflix_history1, fav_books1]
    user2 = [netflix_history2, fav_books2]
    in_common(user1, user2)
```

def new_friend(name_list, compatibility_scores)

which takes in a list of names of all other users and a list of compatibility scores between the chosen user and all other users, and returns a list where the first element is the name of the user who is most compatible and the second element is their compatibility score. name_list stores the name for each user at the same index as the compatibility_scores list stores the corresponding compatabiity score. For example, for user Barack if we have name_list = ['Michelle', 'Joe'] and compability_scores = [1, 0.8], this means the the compatibility score between Barack and Michelle is 1 and the compatibility score between Barack and Joe is only 0.8. In this example, new_friend(name_list, compatability_scores) would return ['Michelle', 1]. You may break ties between equally-compatible users arbitrarily.

In [4]:
```
def new_friend(name_list, compatibility_scores):
    index=0
    last=0
    for x in range(0,len(compatibility_scores)):
        if capatibility_scores[x]>last:
            last = capatibility_scores[x]
            index = x
    final=[name_list[index], compatibility_score[index]]
    return final
```

In [ ]:
```

```