Python Course 2023 - Assignment

Torkil Thomsen

Python Assignemnt 4

Added comments are in bold or cursive to increase the understanding of the problems.
To reduce the possibility of software assistance or other helping tools, the assignment must be completed only with the tools from chapter 1 to 6.

Exercise 5.8 Sieve of Eratosthenes

A prime number is an integer greater than 1 that's evenly divisible only by itself and 1. The Sieve of Eratosthenes is an elegant, straightforward method of finding prime numbers. The process for finding all primes less than 1000 is:

$a)$ create a 1000-element list primes with all elements initialized to True. List elements with prime indicies $(2, 3, 5, 7, 11, ..)$ will remain True. All other list elements will eventually be set to False.

$b)$ Starting with index 2, if a given element is True iterate through the rest of the list and set to False every element in primes whose index is a multiple of the index for the element we're currently processing. For the list index 2, all elements beyond element 2 in the list have indices which are multiples of $2(4, 6, 8, 10, ..., 998)$ will be set to False.

$c)$ Repeat Step (b) for the next True element. For the list index 3 (which was initialized to True), all elements beyond element 3 in the list that have indices which are multiples of $3(6, 9, 12, 15, ..., 999)$ will be set to False; and so on. A subtle observation (think about why this is true): The square root of 999 is 31.6, you'll need to test and set to False only all multiples of $2, 3, 5, 7, 9, 11, 13, 17, 19, 23, 29, 31$. This will significantly improve the performance of your algorithm, especially if you decide to look for large prime numbers.

Please carefully read the assignment. It states that we have a 1000 *element list which all values are True* ***Boolean value****. This means we test the list's index value whether it's a prime number*

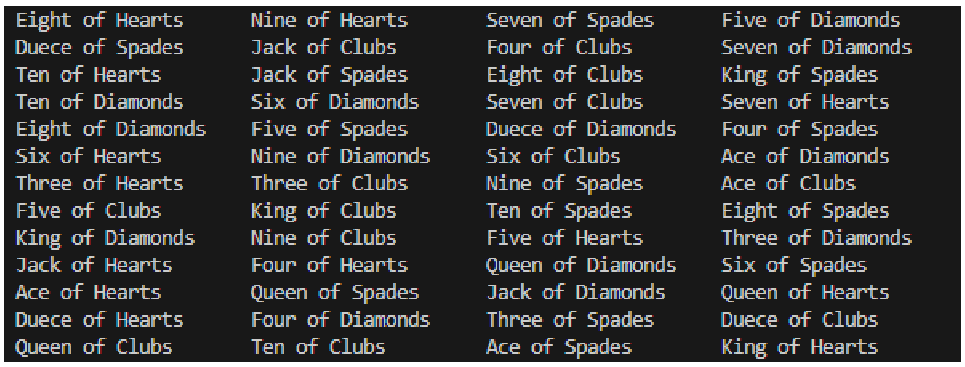
Exercise 5.24 Card Shuffling and Dealing (Page 205 – 206)

In Exercises 5.24 through 5.26, you'll use lists of tuples in scripts that simulate card shuffling and dealing. Each tuple represents one card in the deck and contains a face (e.g., "Ace", "Deuce", "Three",..., "Jack", "Queen", "King") and a suit (e.g., "Hearts", "Diamonds", "Clubs", "Spades").

Create an initialize_deck function to initialize the deck of card tuples with "Ace" through "King" of each suit, as in:

```
   deck = [('Ace', 'Hearts'), ..., ('King', 'Hearts'),
   ('Ace', 'Diamonds'), ..., ('King', 'Diamonds'),
   ('Ace', 'Clubs'), ..., ('King', 'Clubs'),
   ('Ace', 'Spades'), ..., ('King', 'Spades')]
```

Before returning the list, use the random module's **shuffle** function to randomly order the list elements. Output the shuffled cards in the following four-column format:



```
Eight of Hearts      Nine of Hearts       Seven of Spades      Five of Diamonds
Duece of Spades      Jack of Clubs        Four of Clubs        Seven of Diamonds
Ten of Hearts        Jack of Spades       Eight of Clubs       King of Spades
Ten of Diamonds      Six of Diamonds      Seven of Clubs       Seven of Hearts
Eight of Diamonds    Five of Spades       Duece of Diamonds    Four of Spades
Six of Hearts        Nine of Diamonds     Six of Clubs         Ace of Diamonds
Three of Hearts      Three of Clubs       Nine of Spades       Ace of Clubs
Five of Clubs        King of Clubs        Ten of Spades        Eight of Spades
King of Diamonds     Nine of Clubs        Five of Hearts       Three of Diamonds
Jack of Hearts       Four of Hearts       Queen of Diamonds    Six of Spades
Ace of Hearts        Queen of Spades      Jack of Diamonds     Queen of Hearts
Duece of Hearts      Four of Diamonds     Three of Spades      Duece of Clubs
Queen of Clubs       Ten of Clubs         Ace of Spades        King of Hearts
```

Figure 1: Information found on page 206

Exercise 6.11 Analyzing the Game of Craps (Page 237)

Modify the script of Fig. 4.2 to play 1.000.000 games of craps. Use a wins dictionary to keep track of the number of games won for a particular number of rolls. Similarly, use losses dictionary to keep track of the number of games lost for a particular number of rolls. As the simulation proceeds, keep updating the dictionaries.

A typical key-value pair in the wins dictionary might be

$4 : 50217$

indicating that 50217 games were won on the 4th roll. Display a summary of the results including:
a) the percentage of the total games played that were won.
b) the percentage of the total games played that were lost.
c) the percentages of the total games played that were won or lost on a given roll (colmun 2 of the sample output).
d) the *cumulative* percentage of the total games played that were won or lost up to and including a given number of rolls (colmun 3 of the sample out).
Your output should be similar to the one below 2.

In addition of this assignment, we're going to use a package called "matplotlib" on the output we've generated.

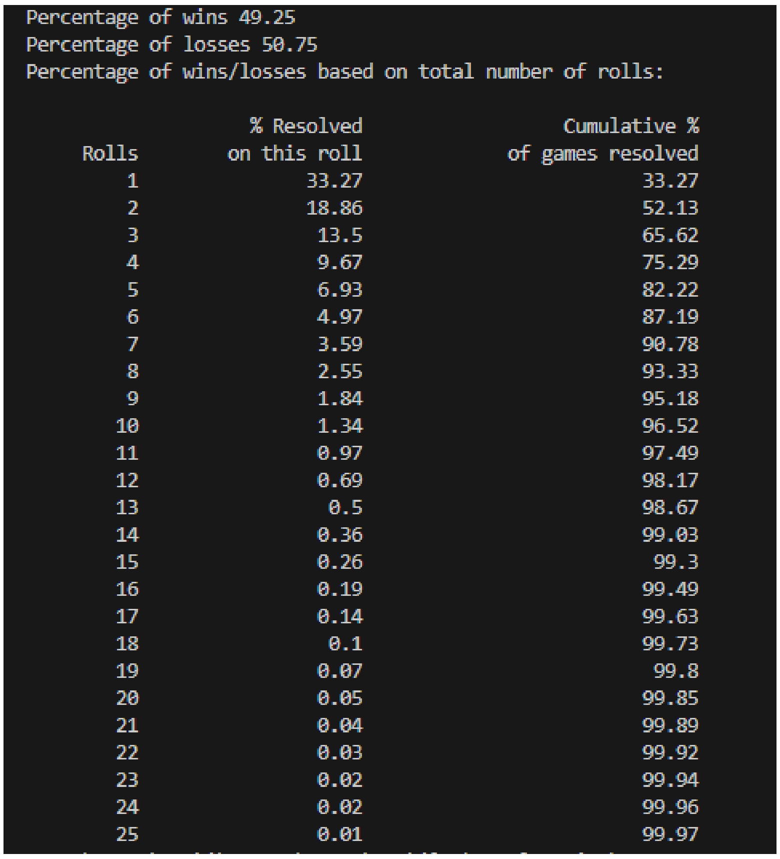
```
    from matplotlib import pyplot as plt
```

or

```
    import matplotlib.pyplot as plt
```

The generated output should look something like the below graphs 3

Hint: dictionaries are unordered, therefore it's required to have the data sorted and zipped for handling



```
Percentage of wins 49.25
Percentage of losses 50.75
Percentage of wins/losses based on total number of rolls:

                 % Resolved                  Cumulative %
  Rolls         on this roll              of games resolved
      1                33.27                          33.27
      2                18.86                          52.13
      3                 13.5                          65.62
      4                 9.67                          75.29
      5                 6.93                          82.22
      6                 4.97                          87.19
      7                 3.59                          90.78
      8                 2.55                          93.33
      9                 1.84                          95.18
     10                 1.34                          96.52
     11                 0.97                          97.49
     12                 0.69                          98.17
     13                  0.5                          98.67
     14                 0.36                          99.03
     15                 0.26                           99.3
     16                 0.19                          99.49
     17                 0.14                          99.63
     18                  0.1                          99.73
     19                 0.07                           99.8
     20                 0.05                          99.85
     21                 0.04                          99.89
     22                 0.03                          99.92
     23                 0.02                          99.94
     24                 0.02                          99.96
     25                 0.01                          99.97
```

Figure 2: Image can be found on page 237

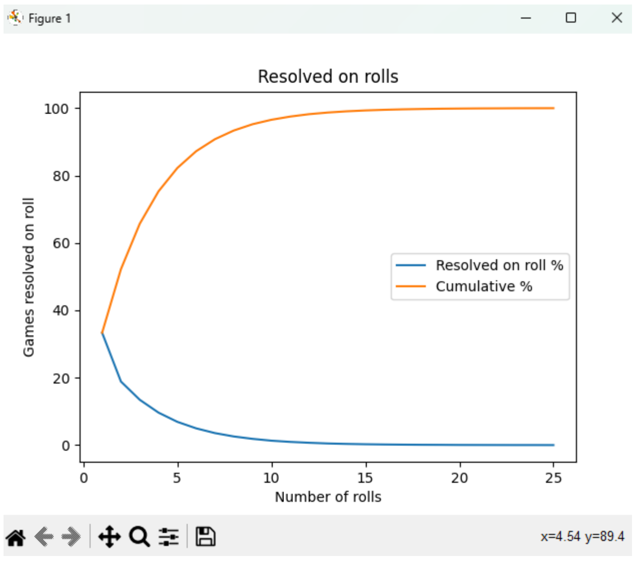


Figure 3: Graph of data