Simulation of a Radom Walk

0. Introduction

This is a simulation of a Random Walk (RW) in Python through an exercise project done to complete the **Intermediate Python** course of DataCamp.

The game is about walking up the Empire State building from the ground floor (step number 'zero') throwing a die 100 times:

- if the result it's 1 or 2, you have to go one step down;
- if it's 3, 4 or 5, you have to go one step up;
- if it's 6, you have to throw the die again and walk up the resulting number of steps. Of course, it is not possible to go lower than step 0. To add some extra 'randomness' there is a 0.1% chance of falling down the stairs (because you are a bit clumsy) each time you make a move, meaning that you have to start again from step 0. Setted up the game, the question is: *will you reach step number **60***?

1. Code explanation

To do this we will need *Numpy* package, in particular the `random` module, in which there are the following functions:

- the Pseudo-random numbers generator in (0,1] is the function `np.random.rand()`;
- the function to set the seed `np.random.seed(seed_num)`;
- another generator is `np.random.randint(0,n)`, that generates a random integer between 0 and n-1.

Thus, we'll simulate the rolling of the dice using `dice = np.random.randint(1,7)` and we set the seed using `datetime` from *DateTime* package to simulate better the randomness (printing the used seed for reproducibility).

In [130…
```python
import numpy as np
from datetime import datetime
```

In [187…
```python
time = int(datetime.now().timestamp())
print(time)
```

```
1761080336
```

In [188…
```python
np.random.seed(time)
```

We now implement the 'movement' in the game using the `if` - `elif` - `else` construct below:

In [ ]:
```
if dice <= 2:
    step = max(0, step - 1)
elif dice > 2 and dice <= 5:
    step = step + 1
else :
    step = step + np.random.randint(1,7)
```

Notice that we used the `max` function to avoid going below zero in the subtraction in the `if` statement.

Then, we create a list `random_walk = [0]` initialised to zero (since we start from step 0) in which we will store the arriving step at each move using the `.append()` method. Thus, the `step` in the previous code will be the arriving step of the previous move, i.e. `step = random_walk[-1]`, for a maximum of 100 moves. The code becomes as follows:

In [199…
```
random_walk = [0]

for m in range(100):
    step = random_walk[-1]
    dice = np.random.randint(1,7)

    if dice <= 2:
        step = max(0, step - 1)
    elif dice > 2 and dice <= 5:
        step = step + 1
    else :
        step = step + np.random.randint(1,7)

    random_walk.append(step)
```

In this case this is the step sequence we have taken:

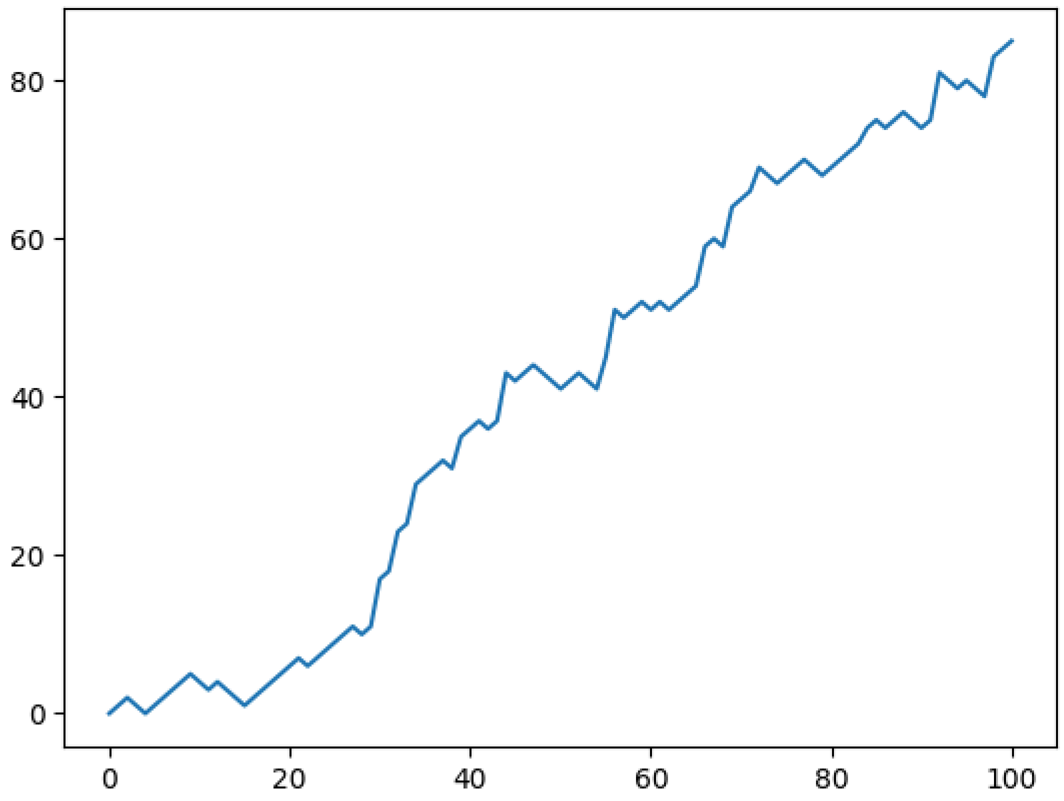
In [200…
```
print(random_walk)
```

```
[0, 1, 2, 1, 0, 1, 2, 3, 4, 5, 4, 3, 4, 3, 2, 1, 2, 3, 4, 5, 6, 7, 6, 7, 8, 9, 1
0, 11, 10, 11, 17, 18, 23, 24, 29, 30, 31, 32, 31, 35, 36, 37, 36, 37, 43, 42, 4
3, 44, 43, 42, 41, 42, 43, 42, 41, 45, 51, 50, 51, 52, 51, 52, 51, 52, 53, 54, 5
9, 60, 59, 64, 65, 66, 69, 68, 67, 68, 69, 70, 69, 68, 69, 70, 71, 72, 74, 75, 7
4, 75, 76, 75, 74, 75, 81, 80, 79, 80, 79, 78, 83, 84, 85]
```

Yes, we have reached step 60th! We can visualise the obtained rw using `pyplot` module of *Matplotlib* library:

In [4]:
```
import matplotlib.pyplot as plt
```

In [201…
```
plt.plot(random_walk)
plt.show()
```



Nevertheless, we still haven't added the clumsiness to the code! To simulate the 0.1% chance that I can fall down the stairs I can use the rng in [0,1) to generate a random float: if it's equal or less than 0.001, we fall, otherwise we are safe. Since it's the movement that makes me fall, I can check the fall chance after the movement, so that the step count is resetted to zero afterwards. Finally, the complete code is showed below.

In [202…

```
random_walk = [0]

for m in range(100):
    step = random_walk[-1]
    dice = np.random.randint(1,7)

    if dice <= 2:
        step = max(0, step - 1)
    elif dice > 2 and dice <= 5:
        step = step + 1
    else :
        step = step + np.random.randint(1,7)

    if np.random.rand() <= 0.001:
        step = 0

    random_walk.append(step)
```

In [203…

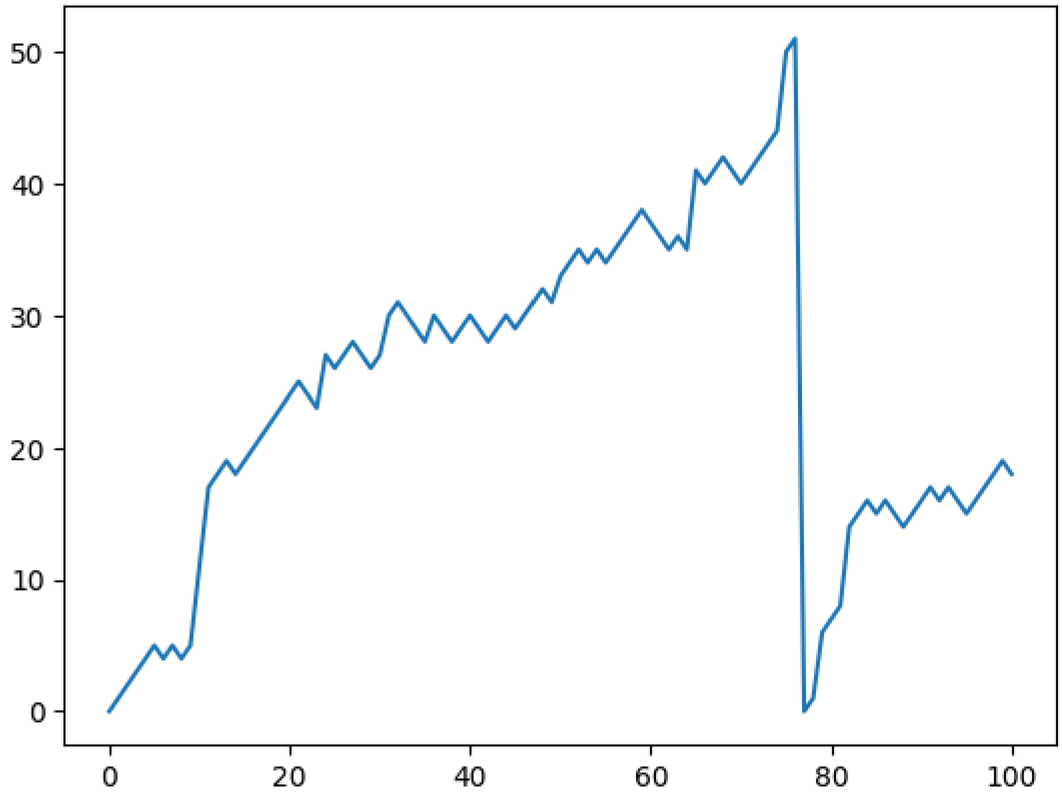
```
print(random_walk)
```

```
[0, 1, 2, 3, 4, 5, 4, 5, 4, 5, 11, 17, 18, 19, 18, 19, 20, 21, 22, 23, 24, 25, 2
4, 23, 27, 26, 27, 28, 27, 26, 27, 30, 31, 30, 29, 28, 30, 29, 28, 29, 30, 29, 2
8, 29, 30, 29, 30, 31, 32, 31, 33, 34, 35, 34, 35, 34, 35, 36, 37, 38, 37, 36, 3
5, 36, 35, 41, 40, 41, 42, 41, 40, 41, 42, 43, 44, 50, 51, 0, 1, 6, 7, 8, 14, 15,
16, 15, 16, 15, 14, 15, 16, 17, 16, 17, 16, 15, 16, 17, 18, 19, 18]
```

Unfortunately, this time we didin't reach the 60th step...

In [204…

```
plt.plot(random_walk)
plt.show()
```



2. Distribution

We implemented the code, but we can still ask ourselves the question: what were the odds of arriving at (least) the 60th step? We know that a single rw can't answer this question, but simulating this experiment 10.000 times for example we can try to understand (at least approximately) this probability by dividing the number of times we reachead a step equal or higher than 60 to the total amount of simulations. We will update the code from before adding a new empty list, `all_walks` , that will contain all the simulated rws.

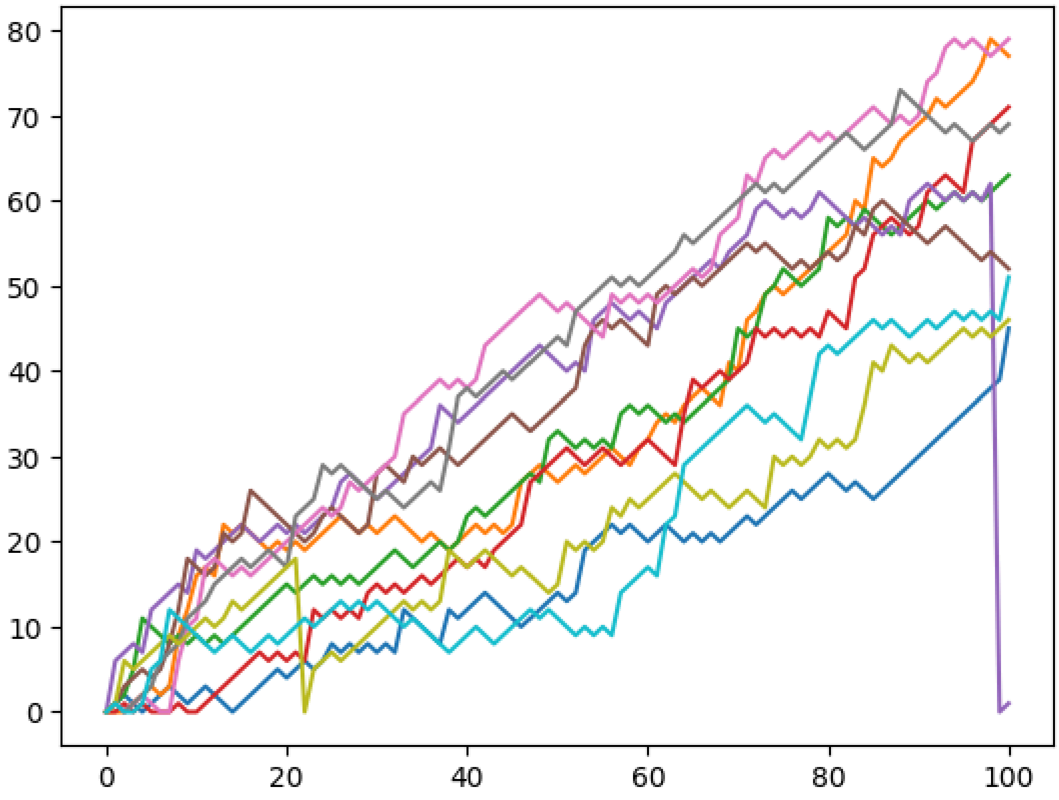
In [205…

```
all_walks = []

for t in range (10000):
    random_walk = [0]

    for m in range(100):
        step = random_walk[-1]
        dice = np.random.randint(1,7)

        if dice <= 2:
            step = max(0, step - 1)
        elif dice > 2 and dice <= 5:
            step = step + 1
        else :
            step = step + np.random.randint(1,7)
        if np.random.rand() <= 0.001:
            step = 0

        random_walk.append(step)

    all_walks.append(random_walk)
```

Then, we can convert `all_walks` from a list of lists to a Numpy array to show the plots of these simulated rws. We will show only the first ten simulations as example:

In [206…

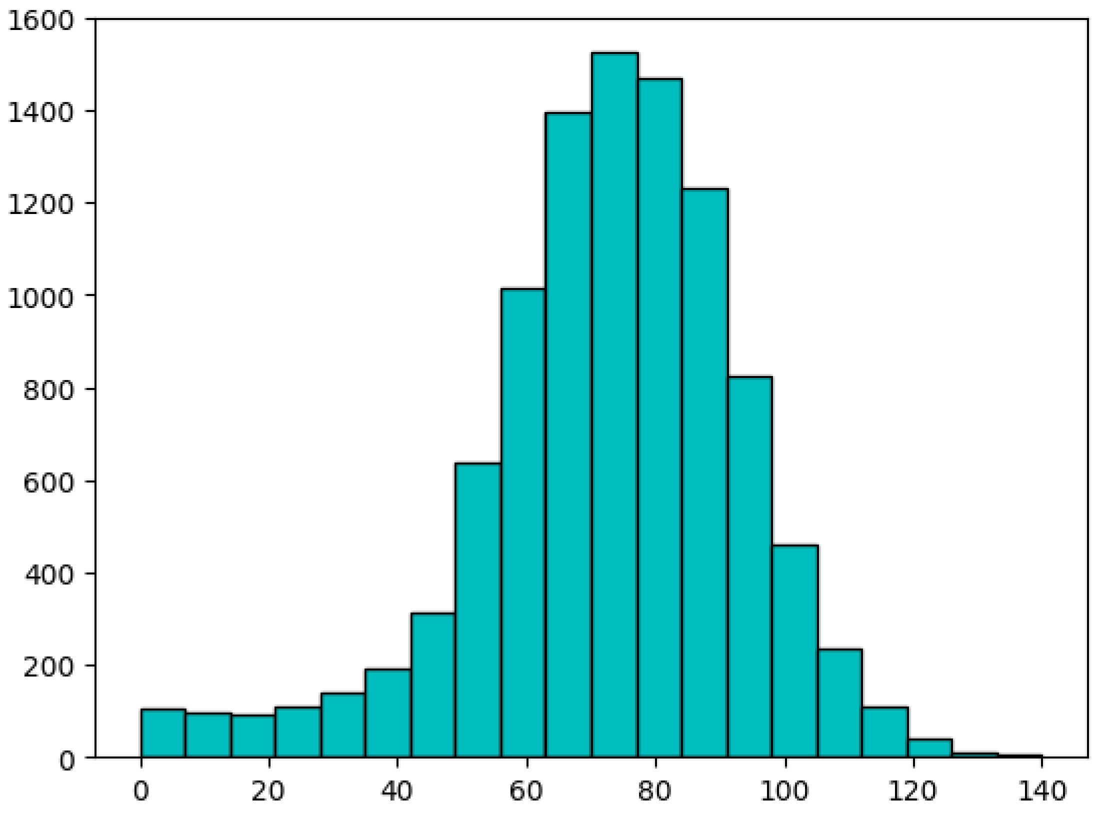
```
simulations = np.array(all_walks)

plt.plot(np.transpose(simulations)[:,:10])
plt.show()
```



Like we ware saying, we can then store the last element of every random walk into a list to see how much times we win the challenge:

In [207…

```
final = simulations[:,-1]

plt.hist(final, color='c', edgecolor='k', bins=20)
plt.show()
```



In [208…

```
wins = len(final[final >= 60])
print(wins)
prob = wins/10000
print('The probability to arrive at least at 60th step is '+ str(round(prob*100,
```

```
7794
The probability to arrive at least at 60th step is 77.94% .
```

So these are the odds using this seed: of course the probability changes a little changing the seed, but it seems to be steady around **80%**! It would have been an easy challenge!