# Exercise #01

IT University of Copenhagen (ITU)
Data Mining KSD (DAMIN)
(Autumn 2025)

26 August 2025

**Introduction** The following exercise presents a few ideas that will be used throughout the DAMIN course. It does not have ready-made code; it only has starting code.

**General Concepts** In this course, you will primarily work with a dynamically typed language called Python. Python is likely the most widely used programming language due to its readability, ease of writing, and simplicity to learn. It also boasts one of the most extensive libraries available. Python offers two methods for writing and running code.

1. One can write a plain text file (or several calling for each other), saved with the `.py` extension, which is called Python script.

2. One can use what is known as "*notebook*," in an environment called Jupyter Notebook[1].

The first is close to writing a source code, and is sometimes referred as such, and is widely used to write programs and routines to do things in a development environment. The second is used more with the intent of doing research, calculations, tests and so on.

The advantages of notebooks are: (i) it can be read linearly; (ii) it lays down the problem's solution in a linear, train of thoughts fashion (think I do this, and them that); (iii) it encourages a step-by-step approach; and (iv) it uses a standard file extension and a development interface.

The disadvantages of notebooks are: (i) it can be read linearly; (ii) it lays down the problem's solution in a linear, train of thoughts fashion (think I do this, and them that); (iii) it encourages a step-by-step approach; and (iv) it uses a standard file extension and a development interface.

You are not reading it wrong, and sometimes advantages are, at the same time, disadvantages. You see, the notebook way is not the best way to actually write code for a program, as it discourages one to change things midway and create several files to work with each other. It is not impossible, just not standard, or effective.

So you say: "*–Whatever, dad. What are we doing here, anyways?*" Well, we're working on a little project. With a normal `.py` file(s). So let's go!

[1] See `https://jupyter.org`.

**Exercise 01.01.** *The Drunkard's Walk* (60-120 minutes) – We shall conduct some experiments involving (theoretical) drunken people. Also known as the "Random Walk." The idea is that a drunken person has a 50% chance of giving a (mis)step either to the left or right (negative or positive steps). We shall work on that by putting lots of drunken people with a fixed number of random steps and statistically see where they land.

- **Instructions:**
  - Read the article Random Walk[2] at Wikipedia®. Take some notes about:
    - The idea.
    - The distribution.
    - The conditions.
  - Look for the following files in the course's GitHub Repository: `drunken-1.py`, `drunken-2.py`, and `drunke-3.py`.
  - Open them and put them on the side.
  - Go to Project Jupyter, and open the `Console -> Python(Pyodide)`. You should get a simple console.
  - For the entire exercise, we shall use this interface.
- **Part 1 – Playing with predefined functions.** We are here to have fun, so no need to understand everything.
  1. Open the `drunken-1.py` file as plain text. (You can drop it in a browser tab or open it in a notepad). Take a breath and try reading the code. One has got used to it. Discuss the role of each line.
  2. Copy the content, paste it into the terminal input, and press `SHIFT + ENTER`. Remember to use `SHIFT + ENTER` to execute a line.
  3. It may seem that nothing has happened! It is because you just created a few functions, such as the ones shown in Listing 1 and 2.

  Listing 1: Function used to create $N$ drunken people start at position zero.

```python
def create_drunken(num: int) -> list:
    """Create a list of 'num'(ber) drunken people.

    Args:
        num (int): Number of drunken people.

    Returns:
        list: A list of drunken people.
    """
    s = []

    for _ in range(num):
        s.append(0)

    return s
```

[2] See `https://en.wikipedia.org/wiki/Random_walk`.

Listing 2: Function used to make these $N$ drunks take a (mis)step.

```python
def step(drunken: list) -> list:
    """Make a set of drunken people take a (mis)step.

    Args:
        drunken (list): A list of drunken people.

    Returns:
        list: A list of drunken people after the step.
    """
    ret = []

    for dr in drunken:
        move = random.uniform(0, 1)
        dr = dr + single_step(move)
        ret.append(dr)

    return ret
```

4. Chose a number, let's say 5, and create a list (as shown in Listing 3):

   Listing 3: Example of create a list with druken people.

```python
drunks_1 = create_drunken(5)
print(drunks_1)
```

   What do you see?

5. To make them take a first step, write the commands shown in Listing 4 in the console:

   Listing 4: Example of random walk with druken people.

```python
drunks_1 = step(drunks_1)
print(drunks_1)
```

   Every value should now be either 1 or $-1$. If you repeat this, they should change further.

6. Make every drunk take 100 (mis)steps, as shown in Listing 5.

   Listing 5: Example of walking 100 steps.

```python
for i in range(100):
    drunks_1 = step(drunks_1)
print(drunks_1)
```

   Did some of them go far? How spread is they are?

7. Explain what is going on, using what you learn from the article. (No ChatGPT! It is relatively easy to catch.)

- **Part 2 – Let's make the drunken people walk a lot.**

  1. Do steps 1 and 2 of Part 1, but with `drunken-2.py`.
  2. Now, we have two new functions, as shown in Listing 6. Have you read the file? We now have a function to make those drunkards walk as much as we like.

Listing 6: New functions added in `drunken-2.py` file.

```python
def random_walk(drunken: list, num: int) -> list:
    """Random walk with N steps.

    Args:
        drunken (list): A list of drunken people.
        num (int): Number of steps.

    Returns:
        list: A list of drunken people after the steps.
    """
    ret = drunken

    for _ in range(num):
        ret = step(ret)

    return ret


def plot_drunken(drunken: list, bins: int):
    """Plot the drunken people in a histogram.

    Args:
        drunken (list): A list of drunken people.
        bins (int): Number of bins.
    """
    plt.hist(drunken, bins)
    plt.show()
```

  3. We're now imposition of creating 1000 drunken people and making them take 1000 (mis)steps, as shown in Listing 7. You can choose a different number of drunkards and how many steps they shall take.

Listing 7: Example of creating 1000 drunken people and walk 1000 steps.

```python
drunks_2 = create_drunken(1000) # 1000 drunkards.
moved_drunks = random_walk(drunks_2, 1000) # Where they went?
```

  4. This time, it does not make sense to see (print) what is happening with individual drunkards. We have too many of them. Still, one can always make a histogram, as shown in Listing 8.

Listing 8: Example of showing a histogram with walk distribution.

```python
bins = 35 # number of bins.
plot_drunken(moved_drunks, bins) # Plot the histogram.
```

5. So what came out of it? What does it look like? Should it look like that?
6. Choose several different numbers of bins and repeat them. Why do you think they look different? What criterion should you use to select a good number of bins?

- **Part 3 – How does it compare with the theory?**

1. Load the third and final `drunken-3.py` file with the three new functions, as shown in Listing 9.

Listing 9: New functions added in `drunken-3.py` file.

```python
def export_data(data: list, filename: str):
    """Export the data to a CSV file.

    Args:
        data (list): List of data.
        filename (str): Name of the file.
    """
    df = DataFrame({ "Druken People final position": data })
    df.to_csv(filename)


def normal_drunkens(drunken: list) -> tuple:
    """Make a Gaussian fit of the drunken people.

    Args:
        drunken (list): A list of drunken people.

    Returns:
        tuple: Average and standard deviation.
    """
    mu, std = norm.fit(drunken)
    print(f"Median = {mu}\nStandard Deviation = {std}")
    return mu, std


def normal_fit_plot(drunken: list, bins: int, mu: np.float64, std: np.float64):
    """Plot the histogram and the Gaussian fit.

    Args:
        drunken (list): A list of drunken people.
        bins (int): Number of bins.
        mu (np.float64): Average.
        std (np.float64): Standard deviation.
    """
    plt.hist(drunken, bins, density=True)

    a, b = plt.xlim()
    x = np.linspace(a, b, 1000)
    fit = norm.pdf(x, mu, std)

    plt.plot(x, fit, "k", linewidth=2)
    plt.show()
```

2. (*Optional*) You can extract the drunkards' positions after they've taken 1000 (or whatever number you chose) (mis)steps, by exporting the generated data into a `.csv` file, as shown in Listing 10. You may use some other tool (Excel?) to load and calculate the average and standard deviation of their positions. The idea is

for you to use a familiar tool (it can also be Python) and give familiarity to what you have been doing so far.

Listing 10: Example of exporting a `DataFrame` into a `.csv` file.

```
export_data(moved_drunks, "where_drunks.csv")
```

3. Let's fit the thing. Start by calculating the mean and standard deviation of the generated distribution, as shown in Listing 11. Please note the values (maybe compare them to what you've got from the *optional* item).

Listing 11: Example of calculating the mean and standard deviation.

```
mu, std = normal_drunkens(moved_drunks)
```

4. Now, plot the histogram with the Probability Density Function (PDF) (as shown in Listing 12) and compare them.

Listing 12: Example of ploting the histogram and the Gaussian fit.

```
bins = 35
normal_fit_plot(moved_drunks, bins, mu, std)
```

How does it look? Vary the number of bins and see how many bins give a more good-looking fit. What is going on?

- **Part 4 – If a drunkard lives far away from where they live, can they get home?**

1. Now, that's a good question. According to our model, the answer should be "*Probably Not.*" Can you argue why? What can you do to check it with what we got? Good luck!
2. What about the theory? What should be the expected median after a long time? Why do you think it is only sometimes the case?
3. What else might be causing some of it regarding computers?