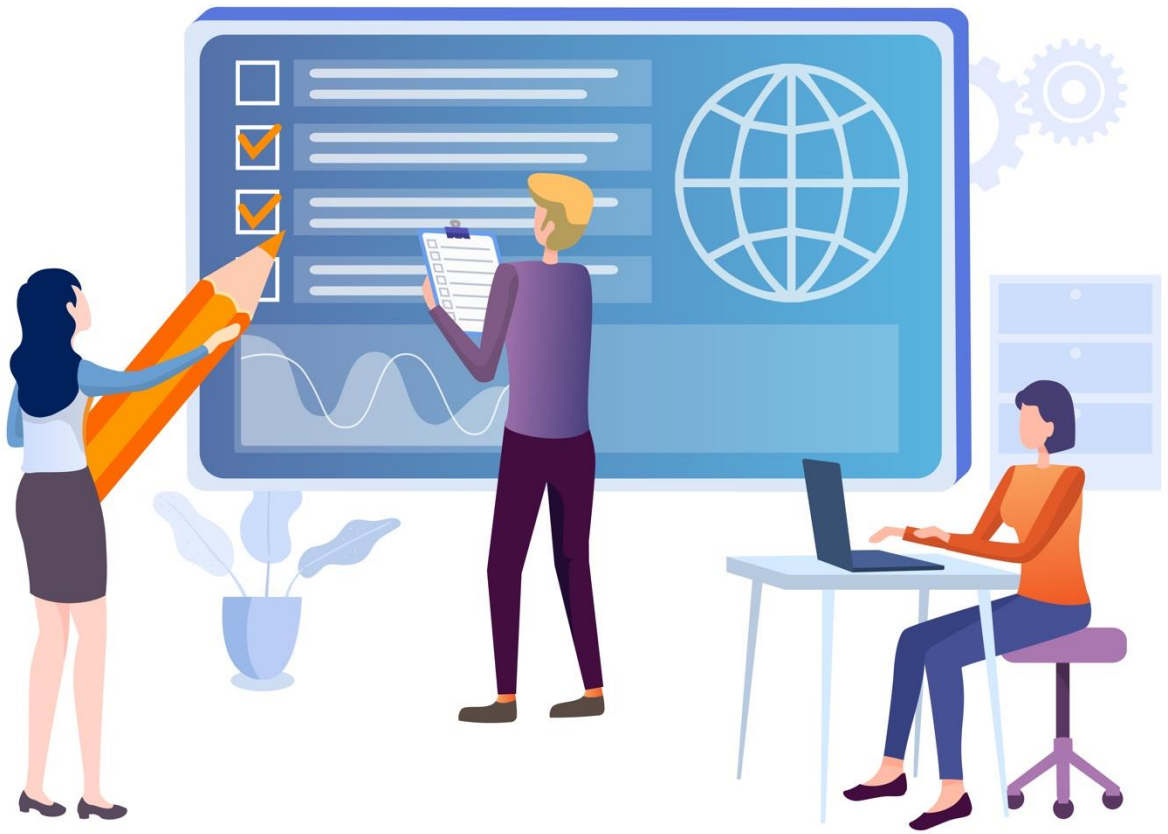
Stelios Sotiriadis

# 3. Sorting and Iterating

# Quiz of the day

Get ready!



# Agenda

- Sorting methods
- Working with text files
- Data streams
- Python `iterators` vs `generators`
- Eager vs Lazy evaluation
- Capstone project!
  - CSV Analysis with Python (Netflix Dataset)

# Sorting methods

# Bubble sort

8 5 3 1 4 7 9

# What is the complexity of bubble sort?

- Time:
  - Repeatedly traversing the array and comparing adjacent items, swapping them if they are in the wrong order.
    - Each complete pass through an array of length `n` compares elements in pairs, leading to `n-1` comparisons (first pass), `n-2` (second), and so on, down to `1` comparison in the last pass.
    - $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2} \approx O(n^2)$
- Space:
  - `O(1)` – in-place swapping, no extra space is used ☺

# Bubble sort

script

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-1-i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

# Insertion sort

6 5 3 1 8 7 2 4

# Insertion sort

- Iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
  - It is a stable sorting algorithm (elements with equal values maintain relative order).

# Merge sort

6 5 3 1 8 7 2 4

# What is $O(n \cdot \log n)$?

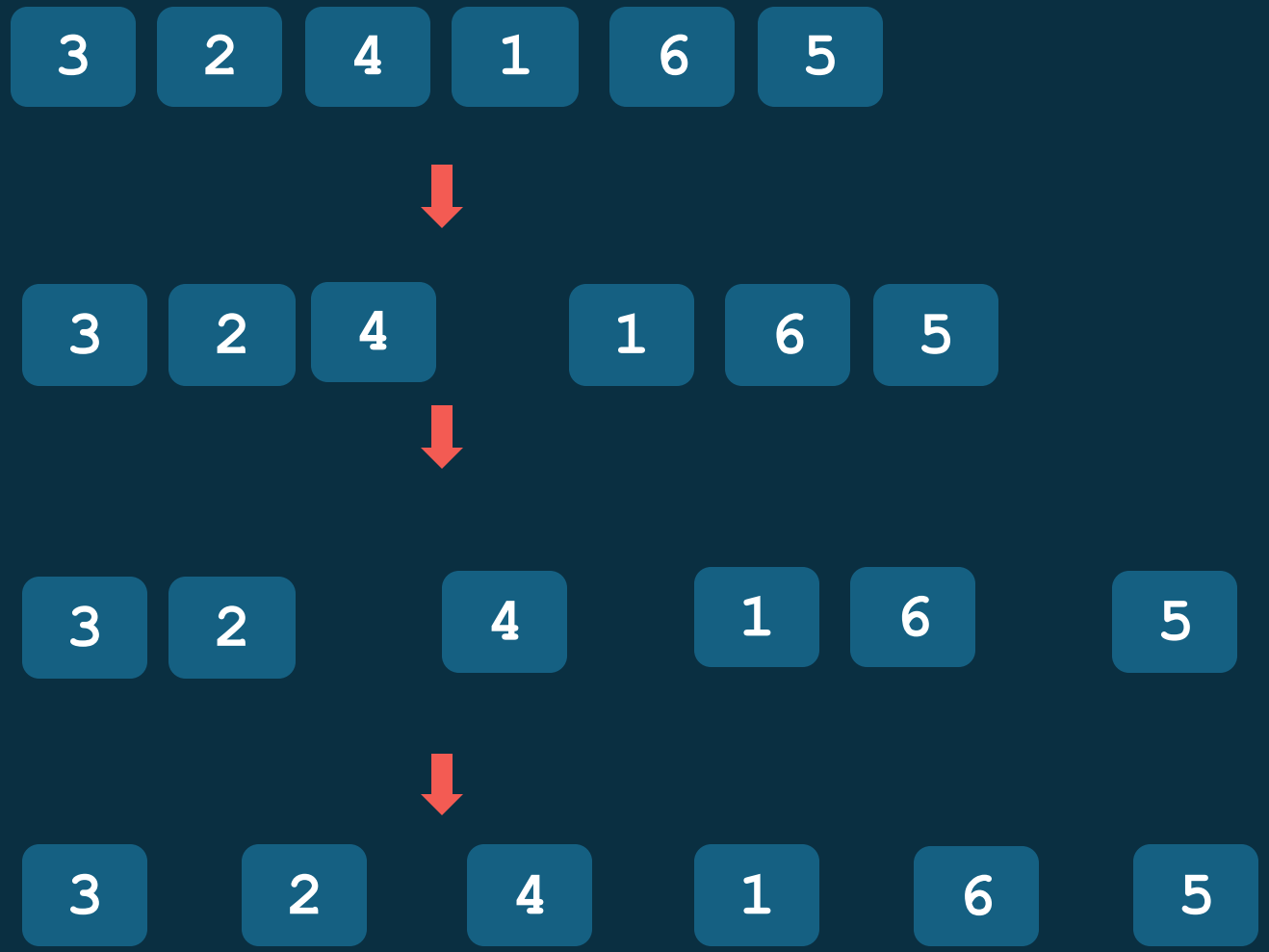
- Merge Sort:
  - Divides the array into two halves, sorts each half, and then merges the sorted halves.
  - Each divide (and subsequent merge) step takes linear time, and the division happens $\log n$ times (the depth of the recursion tree).

# Activity

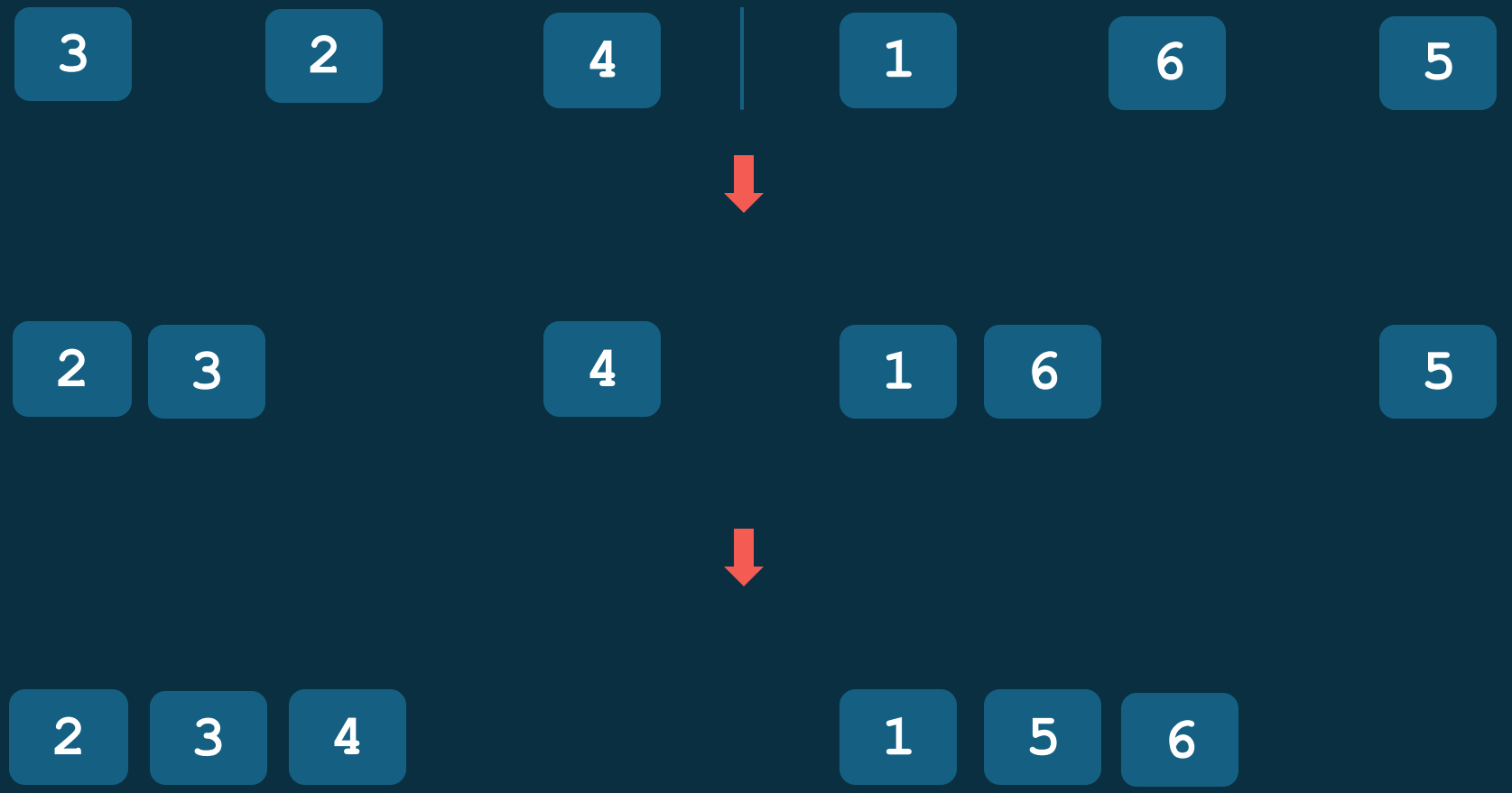
- Sort the following numbers using merge sort!


- First, split the array into smaller sub-arrays until each sub-array has only one element.
- Start merging them back together in a sorted order.
- Merge the two sorted sub-arrays

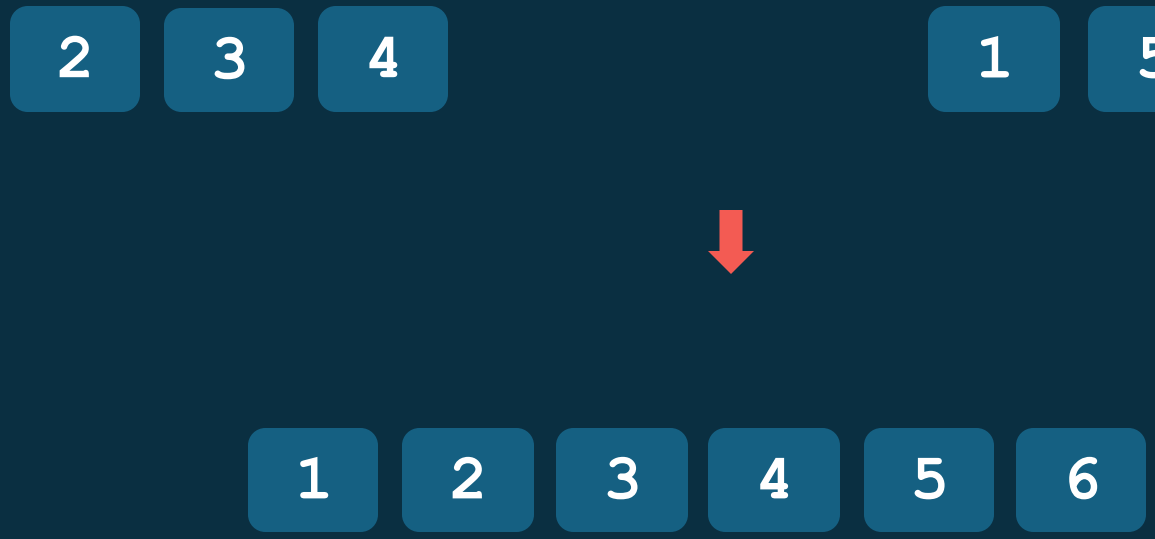
# Step 1: Split the Array



# Step 2: Merge and Sort



# Step 3: Final merge



# Insertion sort `O(n^2)`
# Can you do it better?

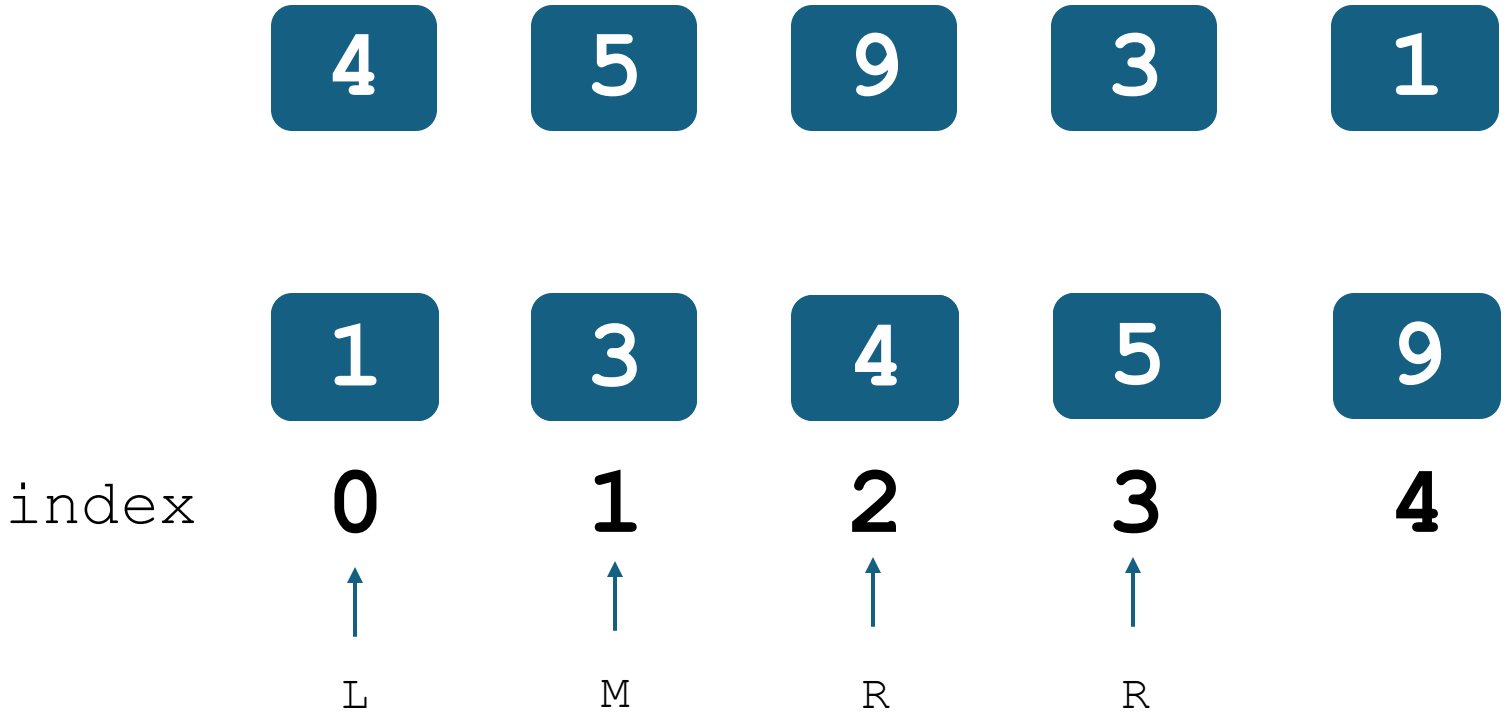
6 5 3 1 8 7 2 4

# Tim sort

- It was implemented by Tim Peters in 2002 for use in Python.
- Tim Sort is the default sorting algorithm used by Python's `sorted()` and `list.sort()` functions.
- Time:
  - `O(n·log(n))`
- Space
  - `O(n)`

# TimSort uses binary search insertion

- Instead of using linear search to find the location where an element should be inserted, it uses binary search.



# Tim sort steps

- Tim sort assumes that some data is already ordered…
- We call this a run:
  - **`arr = {5,7,8,9,3, … }`**
  - **`Run1`** is **`{5,7,8,9}`**
- Each run has a minimum length of elements (min run):
  - **`arr = {8,12,3,9,14, … }`**, with min run = **`3`**
    - Run is **`{8,12,3}`**
    - Tim sort performs binary insertion, so the run becomes **`{3,8,12}`**

# Tim sort steps

- Tim sort has some cool feature
  - If the min run is met, and the next element follows the same pattern (ascending order), then increase the size of the run…
  - **`arr = {5,7,8,9}, min run = 3`**
  - **`run1 = {5,7,8}`** → **`{5,7,8,9}`**
- If the run is in descending order, then blindly reverse!
  - **`run1 = {8,4,2,1} becomes {1,2,4,8}`**


Tim Sort - 236 comparisons, 723 array accesses, 34 ms delay
http://panthema.net/2013/sound-of-sorting

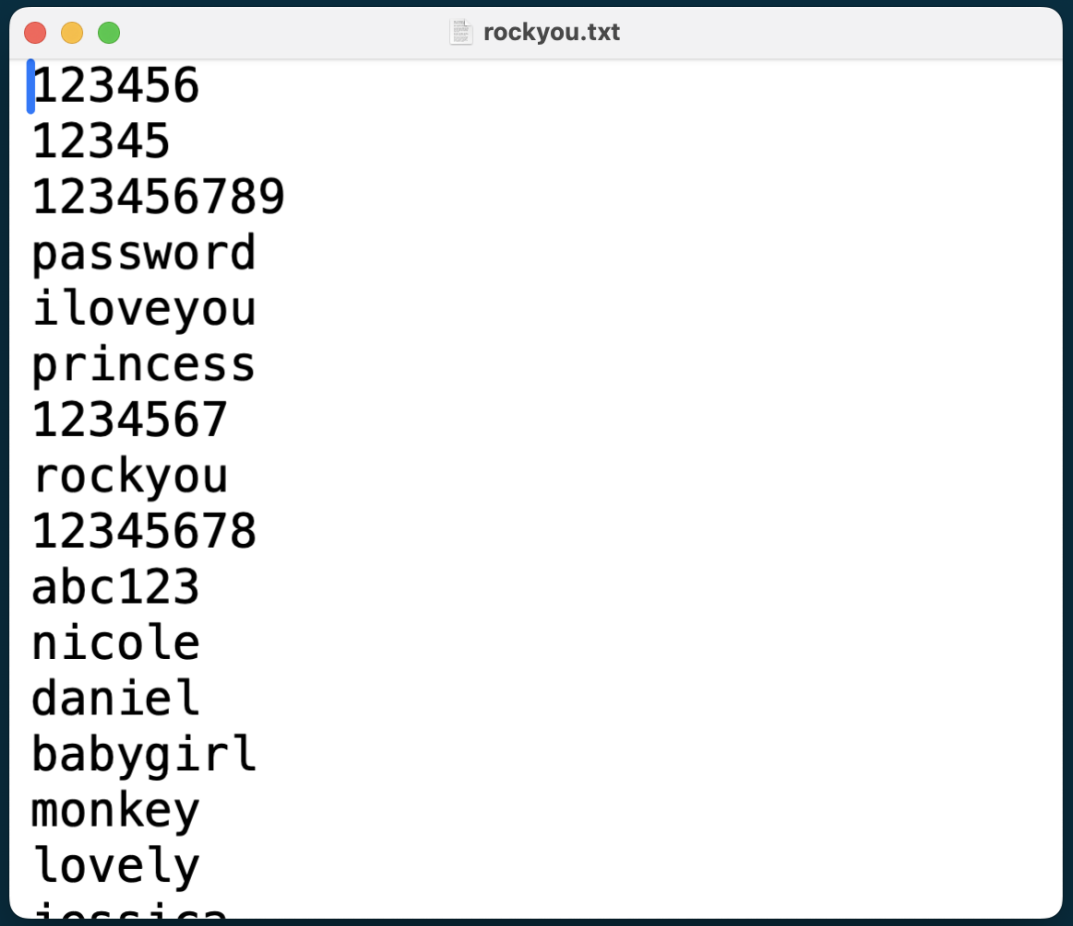
# Let's compare them!

| Algorithm | Time | Space |
|---|---|---|
| Linear search | O(n) | O(1) |
| Binary search | O(logn) | O(1) |
| Bubble sort | O(n^2) | O(1) |
| Insertion sort | O(n^2) | O(1) |
| Merge sort | O(nlogn) | O(n) |
| Tim sort | O(nlogn) | O(n) |

# Working with text files

# Pen & Paper

- Write the first 3 lines as Python sees them (raw input as a long String).




```
"123456\n12345\n123456789\n"
```

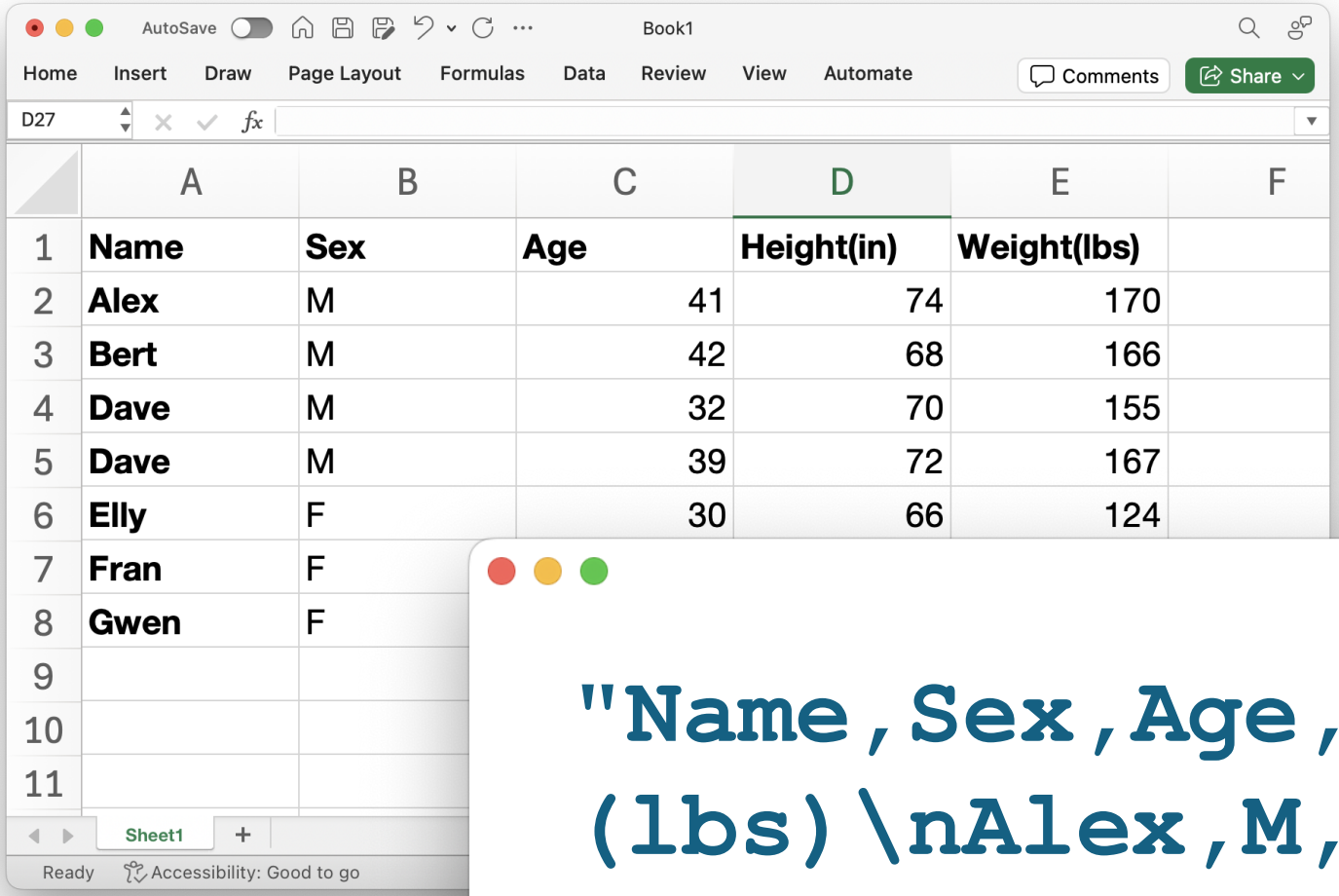
rockyou.txt

```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
nicole
daniel
babygirl
monkey
lovely
```

# How Python reads txt data!

script.py.md

```
"123456\n12345\n123456789
\npassword\niloveyou\nprincess\n
1234567\nrockyou\n12345678\nabc1
23\nnicole\ndaniel\nbabygirl\nmo
```




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Name** | **Sex** | **Age** | **Height(in)** | **Weight(lbs)** | |
| 2 | **Alex** | M | 41 | 74 | 170 | |
| 3 | **Bert** | M | 42 | 68 | 166 | |
| 4 | **Dave** | M | 32 | 70 | 155 | |
| 5 | **Dave** | M | 39 | 72 | 167 | |
| 6 | **Elly** | F | 30 | 66 | 124 | |
| 7 | **Fran** | F | | | | |
| 8 | **Gwen** | F | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |

# How Python reads csv data!

script.py.md

```
"Name,Sex,Age,Height(in),Weight(lbs)\nAlex,M,41,74,170\nBert,M,42,68,166\nDave,M,32,70,155\nDave,M,39,72,167\nElly,F,30,66,124\nFran,F,33,66,115\nGwen,F,26,64,121"
```

# Data streams

# Pen & paper!

- Write a Python script that checks if a leaked password (**`ilovepizza22`**) appears in **`rockyou.txt`**.

script.py.md

```
with open("rockyou.txt", "r", encoding="latin-1") as file:
    ...
```

# Pen & paper!

02:00

- Write a Python script that checks if a leaked password (**`ilovepizza22`**) appears in **`rockyou.txt`**.

script.py.md

```
with open("rockyou.txt", "r", encoding="latin-1") as file:
    ...
```

script.py.md

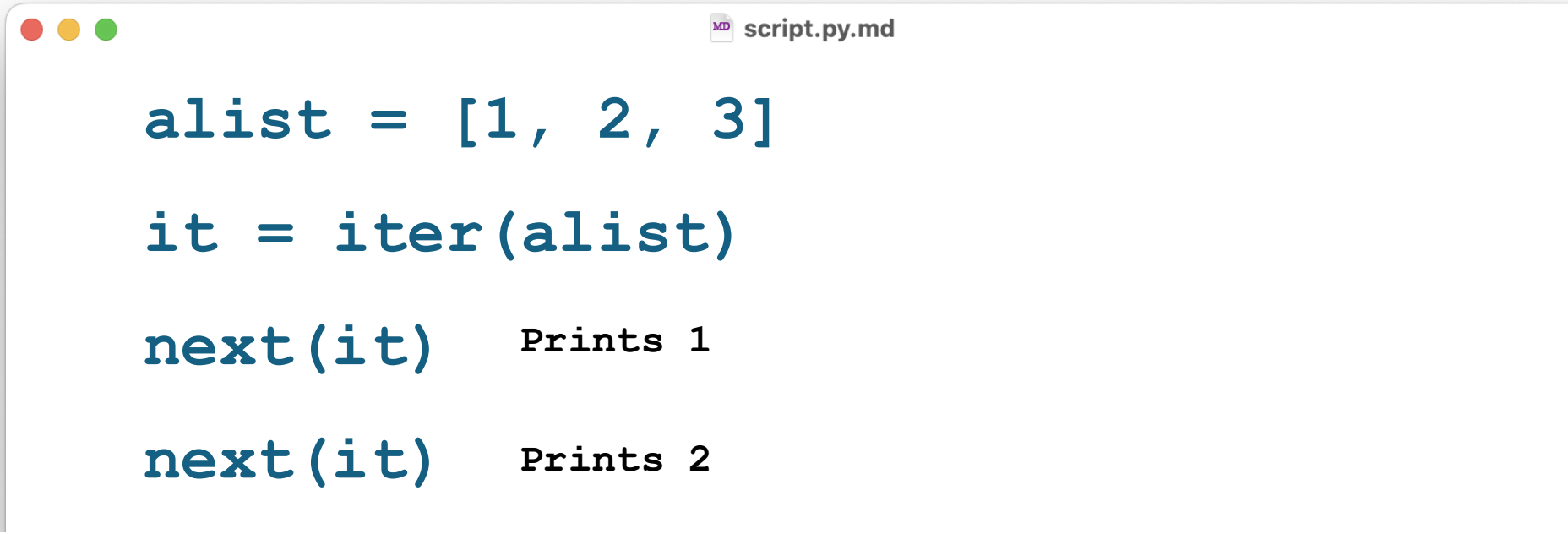
```
with open("rockyou.txt", "r", encoding="latin-1") as file:
    for line in file:
        if line.strip() == "ilovepizza22":
            print(True)
            break
    else:
        print(False)
```

# What is a stream of data?

- Data that is delivered one piece at a time, instead of all at once.
- Think of it like:
  - Watching a video on YouTube — the video plays while it's still downloading
  - Processing each line of a large file — without loading the whole file into memory
- Examples
  - Iterators
  - Generators (`yield`)
  - File reading line-by-line
  - Data from sensors, APIs, or network connections

# How does this works?

- `alist` is an *iterable* (you can loop over it).
- `iter(alist)` creates an iterator that tracks position as you go.
- `next(it)` retrieves the next item and remembers where it left off.



# File example

- Let's say you're reading a large file of customer records.
- You don't load the whole file into memory, just read one record at a time.

script.py.md

```
with open('customers.txt') as f:
    it = iter(f)
    print(next(it))  # prints first customer
    print(next(it))  # prints second customer
```

# Time complexity analysis

| Symbol | Meaning |
| --- | --- |
| `n` | Total number of lines in the file |
| `m` | Length of a **single line** (in characters or bytes) |

- You're not processing all `n` lines, just **2 lines**
- `next(it)` → `O(m)` – Reads one line from the file — time depends on the line length m)
- Total Time Complexity: `O(m)` per line

# Space complexity analysis

- Only **one line** is loaded into memory at a time.
- No lists or buffers are built — it's a true **stream**
- Total Space Complexity: **`O(max_m) or O(m) ~average`**
- Where:
  - n = total number of lines (not used here)
  - m = average size of each line
  - max_m = **worst-case time per line**

# Use Cases

- Reading logs line-by-line
- Streaming video or audio
- Processing data from sensors in real time
- Reading large CSVs without memory overload

# Python generators

# Big data processing problem

- Continuous goal to reduce the cost of:
  - Processing data → improve time complexity
  - Storing data → reduce space complexity

# Working with data

- Reading large files or datasets can use a lot of computer memory
- Functions like `return` store everything in memory
- **Goal**
  - Process one element, at a time, and then ignore
  - Keep memory usage low
  - Pause and resume execution

# Using generators

- A generator is a special function that uses **`yield`** instead of return

script.py.md

```
def count_up_to(n):
    for i in range(n):
        yield i
```

# How does it work?

- Each `yield` sends one value at a time
- Then, the function pauses, saving its state
- And resumes from the same place on the next call

# What is the time/space complexity?

```
import csv

def load_all_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        rows = []
        for row in reader:
            rows.append(row)
        return rows
data = load_all_rows('netflix_titles.csv')
print(data[0])  # First row
```

## Both O(n·m)

```
import csv

def load_all_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        rows = []
        for row in reader:
            rows.append(row)
        return rows
data = load_all_rows('netflix_titles.csv')
print(data[0])  # First row
```

## Memory efficient — ideal for big files or filtering on the fly.

```
import csv

def stream_rows(filename):
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            yield row  # yields one row at a time

# Usage
for i, row in enumerate(stream_rows('netflix_titles.csv')):
    print(row)       # One row at a time
    if i == 2: break  # Just show 3 rows
```

# `return` VS `yield`

| | *return (load all rows)* | *yield (stream one row at a time)* |
|---|---|---|
| **What it does** | Loads and returns the full list | Yields one row at a time (generator) |
| **Time Complexity** | O(n·m) *(read all n rows with m fields)* | O(n·m) *(same total work, row-by-row)* |
| **Space Complexity** | O(n·m) *(stores all rows in memory)* | O(m) *(just one row at a time)* |
| **Start-up cost** | Waits until all rows are loaded | Starts immediately |
| **Scalability** | Poor for large files | Excellent — memory usage stays low |
| **Use case** | Small datasets, fast access | Large datasets, streaming, filters |

# But in the end, is all the data held in memory?

- No — that's the beauty of yield!
  - Yields one value at a time.
  - After it's used, **it's gone from memory**.
  - Even if n = 1,000,000, only one number is ever in memory.

# Use `yield` when

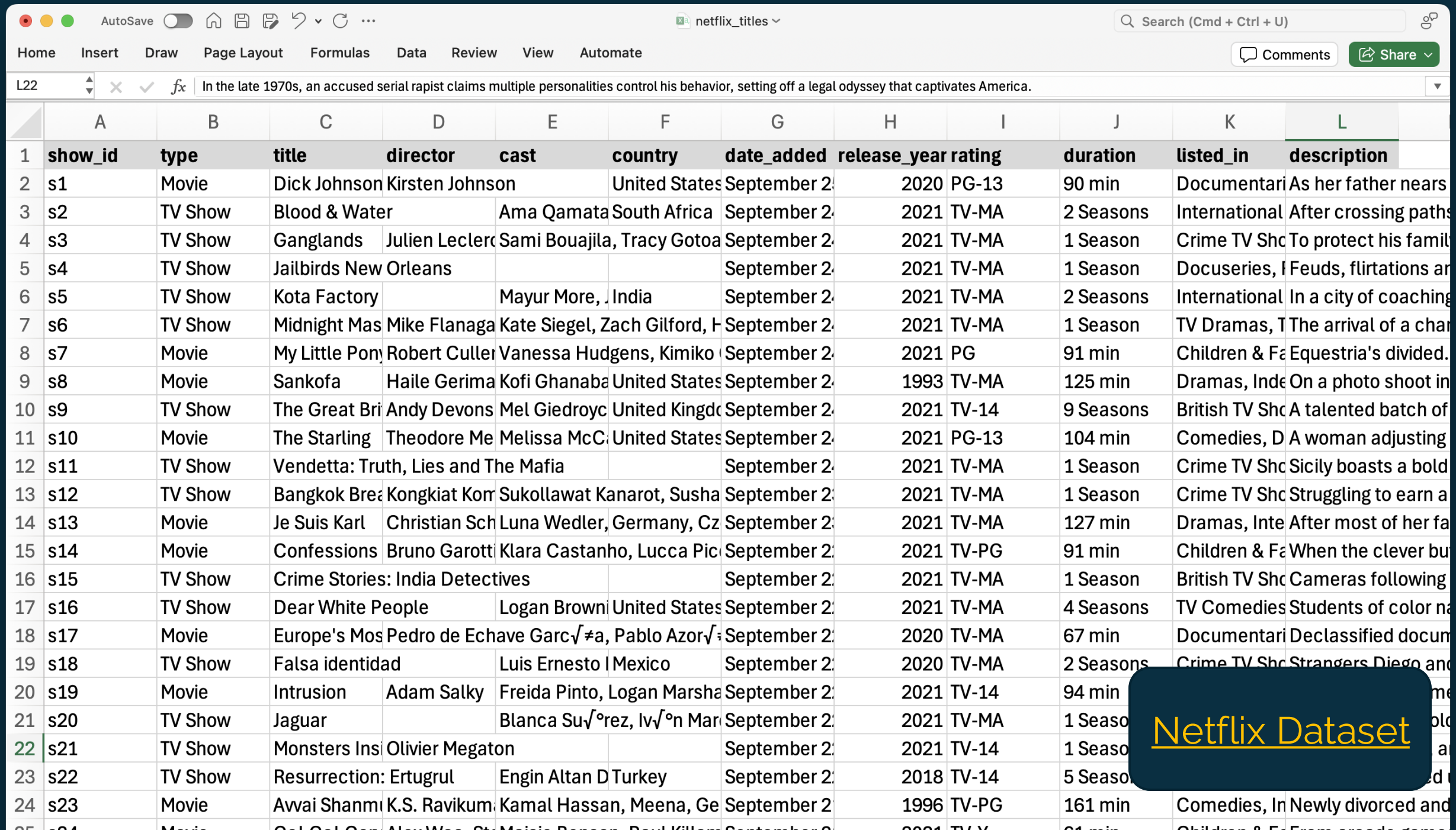
- You're working with large datasets or files
- You want to stream data one item at a time
- You don't need to access data by index
- You only need to iterate once
- You want to start processing data immediately

# Avoid `yield` when

- You need to access all data at once
- You need indexing or slicing (e.g. data[0])
- You want to reuse the data multiple times
- You plan to sort or filter the whole dataset
- You'll convert it to a list anyway (list(generator)) — defeats the purpose

# `Iterators` VS `Generators`

- A generator is a Pythonic way to implement an **`iterator`** that produces values one at a time, using **`yield`**, making it memory-efficient.
- **Generators are iterators**
  - Use yield to deliver values one at a time
  - Pause and resume execution
  - Are perfect for streaming data, **lazy evaluation**, and large/infinite sequences




| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | show_id | type | title | director | cast | country | date_added | release_year | rating | duration | listed_in | description |
| 2 | s1 | Movie | Dick Johnson | Kirsten Johnson | | United States | September 2 | 2020 | PG-13 | 90 min | Documentari | As her father nears |
| 3 | s2 | TV Show | Blood & Water | | Ama Qamata | South Africa | September 2 | 2021 | TV-MA | 2 Seasons | International | After crossing paths |
| 4 | s3 | TV Show | Ganglands | Julien Leclerc | Sami Bouajila, Tracy Gotoa | | September 2 | 2021 | TV-MA | 1 Season | Crime TV Sho | To protect his famil |
| 5 | s4 | TV Show | Jailbirds New | Orleans | | | September 2 | 2021 | TV-MA | 1 Season | Docuseries, I | Feuds, flirtations ar |
| 6 | s5 | TV Show | Kota Factory | | Mayur More, J | India | September 2 | 2021 | TV-MA | 2 Seasons | International | In a city of coaching |
| 7 | s6 | TV Show | Midnight Mas | Mike Flanaga | Kate Siegel, Zach Gilford, H | | September 2 | 2021 | TV-MA | 1 Season | TV Dramas, T | The arrival of a char |
| 8 | s7 | Movie | My Little Pony | Robert Culler | Vanessa Hudgens, Kimiko ( | | September 2 | 2021 | PG | 91 min | Children & Fa | Equestria's divided. |
| 9 | s8 | Movie | Sankofa | Haile Gerima | Kofi Ghanaba | United States | September 2 | 1993 | TV-MA | 125 min | Dramas, Inde | On a photo shoot in |
| 10 | s9 | TV Show | The Great Bri | Andy Devons | Mel Giedroyc | United Kingdo | September 2 | 2021 | TV-14 | 9 Seasons | British TV Sho | A talented batch of |
| 11 | s10 | Movie | The Starling | Theodore Me | Melissa McC | United States | September 2 | 2021 | PG-13 | 104 min | Comedies, D | A woman adjusting |
| 12 | s11 | TV Show | Vendetta: Truth, Lies and The Mafia | | | | September 2 | 2021 | TV-MA | 1 Season | Crime TV Sho | Sicily boasts a bold |
| 13 | s12 | TV Show | Bangkok Brea | Kongkiat Kom | Sukollawat Kanarot, Susha | | September 2 | 2021 | TV-MA | 1 Season | Crime TV Sho | Struggling to earn a |
| 14 | s13 | Movie | Je Suis Karl | Christian Sch | Luna Wedler, | Germany, Cz | September 2 | 2021 | TV-MA | 127 min | Dramas, Inte | After most of her fa |
| 15 | s14 | Movie | Confessions | Bruno Garotti | Klara Castanho, Lucca Pic | | September 2 | 2021 | TV-PG | 91 min | Children & Fa | When the clever bu |
| 16 | s15 | TV Show | Crime Stories: India Detectives | | | | September 2 | 2021 | TV-MA | 1 Season | British TV Sho | Cameras following |
| 17 | s16 | TV Show | Dear White People | | Logan Browni | United States | September 2 | 2021 | TV-MA | 4 Seasons | TV Comedies | Students of color na |
| 18 | s17 | Movie | Europe's Mos | Pedro de Ech | ave Garc√≠a, Pablo Azor√ | | September 2 | 2020 | TV-MA | 67 min | Documentari | Declassified docum |
| 19 | s18 | TV Show | Falsa identidad | | Luis Ernesto I | Mexico | September 2 | 2020 | TV-MA | 2 Seasons | Crime TV Sho | Strangers Diego and |
| 20 | s19 | Movie | Intrusion | Adam Salky | Freida Pinto, Logan Marsha | | September 2 | 2021 | TV-14 | 94 min | | |
| 21 | s20 | TV Show | Jaguar | | Blanca Su√°rez, Iv√°n Mar | | September 2 | 2021 | TV-MA | 1 Seaso | | |
| 22 | s21 | TV Show | Monsters Insi | Olivier Megaton | | | September 2 | 2021 | TV-14 | 1 Seaso | | |
| 23 | s22 | TV Show | Resurrection: Ertugrul | | Engin Altan D | Turkey | September 2 | 2018 | TV-14 | 5 Seaso | | |
| 24 | s23 | Movie | Avvai Shanmu | K.S. Ravikum | Kamal Hassan, Meena, Ge | | September 2 | 1996 | TV-PG | 161 min | Comedies, In | Newly divorced and |

# `yield` or `return`?

- What is the first movie rated “PG-13”?
  - yield – streaming example
- Build a report showing average age and total movies per country.
  - return – you need all data for aggregations
- Searching for the first 5 titles with the word “love”.
  - yield – streaming example
- Sort all Netflix titles by release year
  - return - you need all rows in memory to sort)

# What is a lazy evaluation?

# What is a lazy evaluation?

- One of the core building blocks of big data processing.
- Don't compute a value until **you actually need it**.
  - **Eager**: Do everything now
  - **Lazy:** Wait until asked
- **DVD Download vs Netflix Streaming**
  - Eager evaluation (DVD) = Download the entire movie before watching
  - Lazy evaluation (Netflix) = Start watching right away — stream it scene by scene

# Eager (loads everything at once)

- Memory is used for all 1,000,000 results.
- Even if you only need the first value!

script.py.md

```
def double_numbers():
    result = []
    for x in range(1_000_000):
        result.append(x * 2)
    return result
print(double_numbers()[0])
```

# Lazy (only creates what's needed)

- Only uses memory for one value at a time
- Much faster to start
- Ideal for large datasets or early exit

script.py.md

```
def double_numbers():
    for x in range(1_000_000):
        yield x * 2
lazy_result = double_numbers()
print(next(lazy_result))
```

# Lessons learned

- Lazy evaluation is useful when:
  - You're processing big files, logs, or API responses
  - You don't know how much data you need
  - You want to save memory and delay computation

# Thank you!

- The lab starts soon! (404-405)

O(no!)

# Lab 2

Big Data Analytics

# Lab activities

- Complete lab 2 activities and exercises.
- Use your preferred python IDE.