

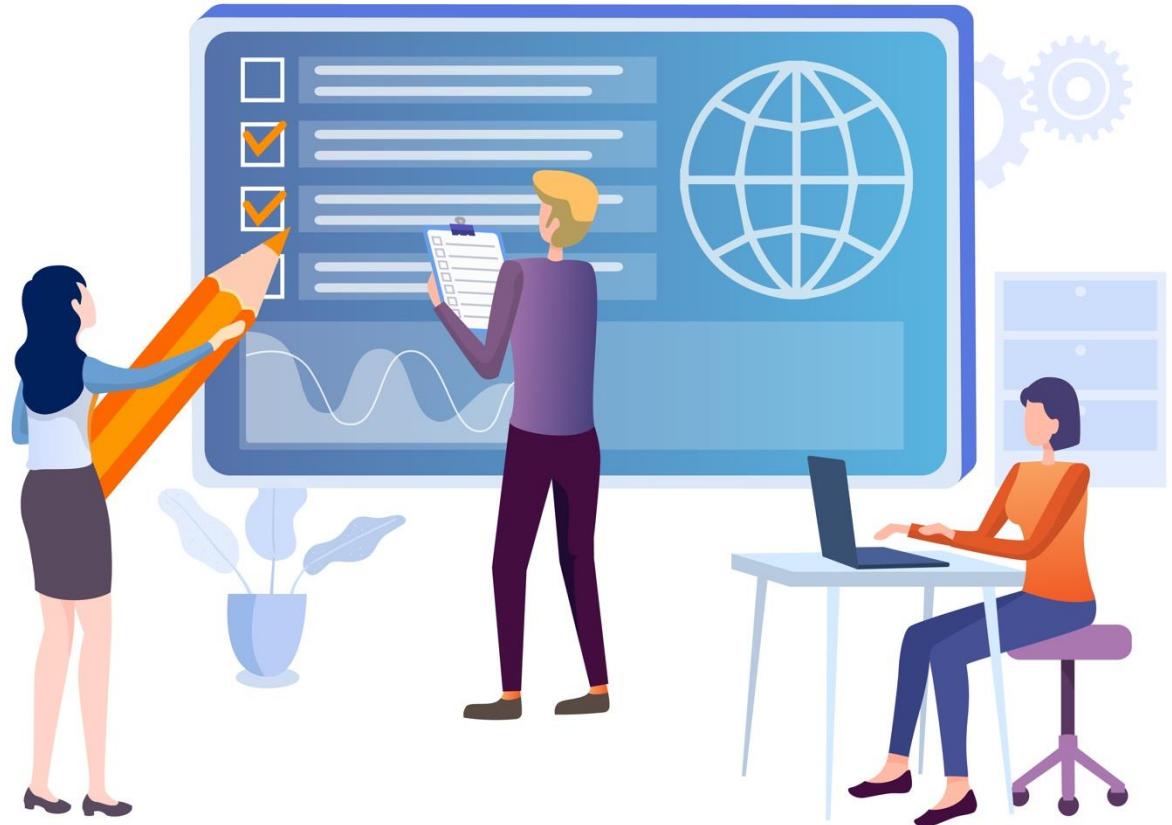


Stelios Sotiriadis

3. Sorting and Data Streams

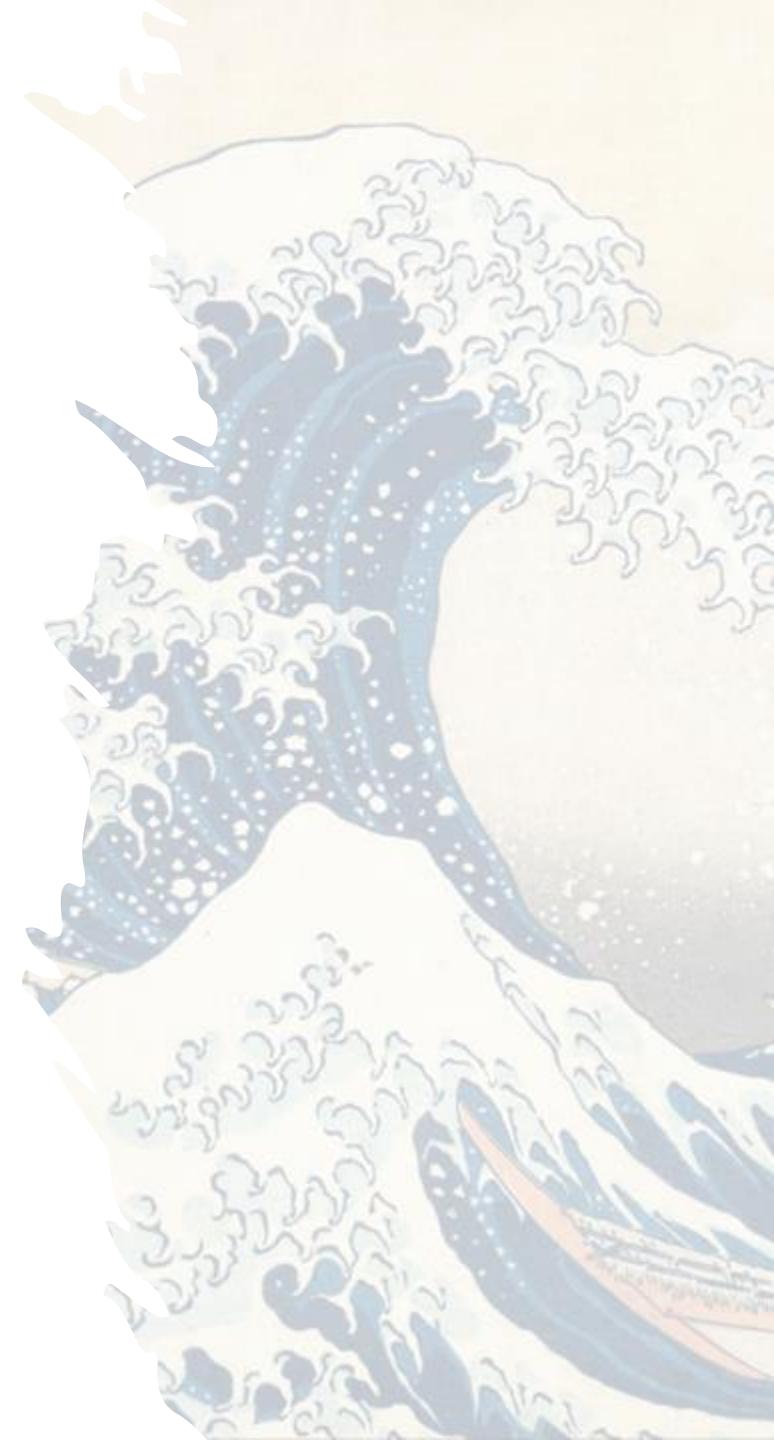
Quiz of the day

Get ready!



Agenda

- ▶ Sorting methods
- ▶ Working with text files
- ▶ Data streams
- ▶ Python **iterators** vs **generators**
- ▶ Eager vs Lazy evaluation
- ▶ Data Analytics Project!
 - CSV Analysis with Python (Netflix Dataset)



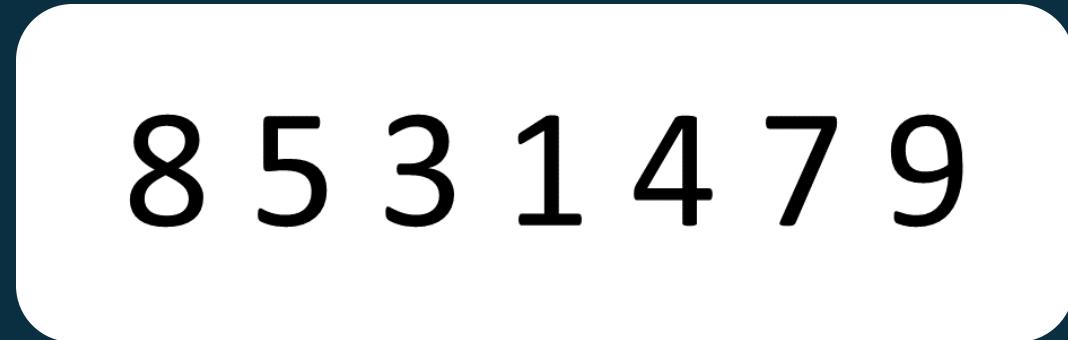
Sorting methods

Bubble sort

8 5 3 1 4 7 9

Quick quiz on Bubble sort!

- ▶ Worst case time complexity:
 - $n \cdot (n - 1) / 2 \rightarrow O(n^2)$
 - We focus only on the dominant term $\rightarrow n^2$
- ▶ Worst case space complexity:
 - $O(1)$
- ▶ Does Bubble Sort change the order of equal elements?
 - No! That's why we call it **stable algorithm**
- ▶ What does Bubble Sort do in each pass through the array?
 - ~~Finds the smallest element and moves it to the start~~
 - B) Finds the largest element and moves it to the end ← **Correct!**



What if?

- ▶ I want Bubble Sort to find the smallest element and move it to the beginning of the array.
 - Instead of:
 - `if arr[j] > arr[j + 1]`
 - Use:
 - `if arr[j] < arr[j + 1]`
 - Iterate from right to left in the inner loop, so small values move to the left.

Summary of worst-case time/space

► 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.
 - $(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} \approx O(n^2)$

► Space:

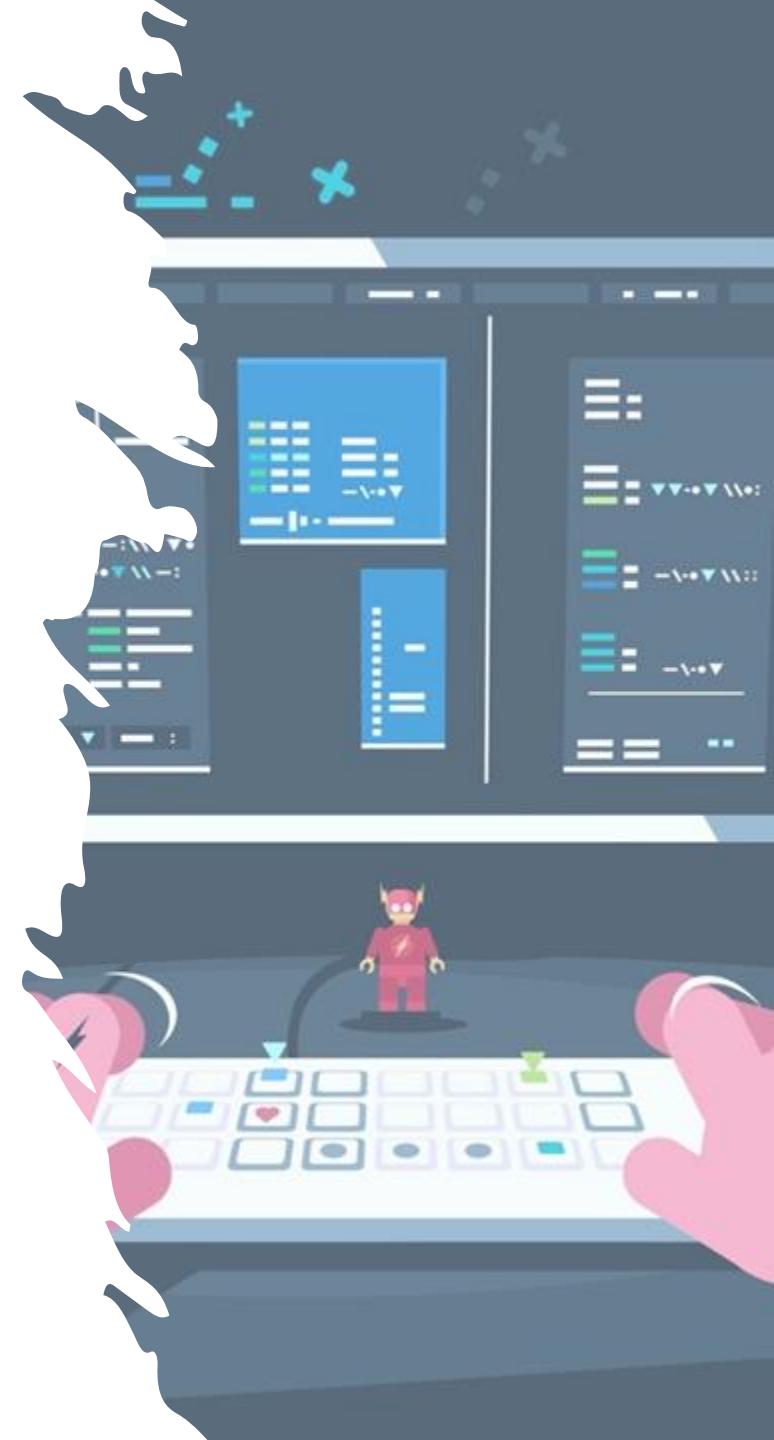
- **O(1)** – in-place swapping, no extra space is used ☺

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
- ▶ Worst-case time complexity: $O(n^2)$
- ▶ Worst-case space complexity: $O(1)$



Merge sort

6 5 3 1 8 7 2 4

Merge Sort

- ▶ Divides the array into two halves, sorts each half, and then merges the sorted halves.
 - Worst-case time complexity: $O(n \cdot \log n)$
- ▶ But!
 - During the merge step, it creates temporary arrays to hold left and right halves, which uses extra space.
 - Worst-case space complexity: $O(n)$



Why is $O(n \cdot \log n)$?

- ▶ Why $\log n$?
 - Because each time you divide the array in half:
 - First: n elements
 - Then: $n/2$
 - Then: $n/4$
 - until you reach 1 element
 - This takes $\log n$
- ▶ Why n at each level?
 - At each level, you merge all n elements back together:
 - ▶ Merging n elements takes linear time — $O(n)$
 - ▶ Finally:
$$O(n) \times O(\log n) = O(n \cdot \log n)$$

Pen & Paper: Order bottom (best) to top (worst)

$O(\log n)$

$O(n^2)$

$O(n!)$

$O(2^n)$

$O(n \log n)$

$O(1)$

$O(n)$

Pen & Paper: Order bottom (best) to top (worst)

$O(n!)$ - Factorial

$O(2^n)$ - Exponential

$O(n^2)$ - Quadratic

$O(n \log n)$ - Log-linear

$O(n)$ - Linear

$O(\log n)$ - Logarithmic

$O(1)$ - Constant

Let's do it together!

- ▶ Sort the following numbers using merge sort!

3 2 4 1 6 5

Step 1: Split the Array

3 2 4 1 6 5



3 2 4 1 6 5



3 2 4 1 6 5



3 2 4 1 6 5

Step 2: Merge and Sort

3 2 4 | 1 6 5



2 3 4 1 6 5



2 3 4 1 5 6

Step 3: Final merge

2 3 4

1 5 6



1 2 3 4 5 6

Insertion sort $O(n^2)$

Can you do it better?

6 5 3 1 8 7 2 4

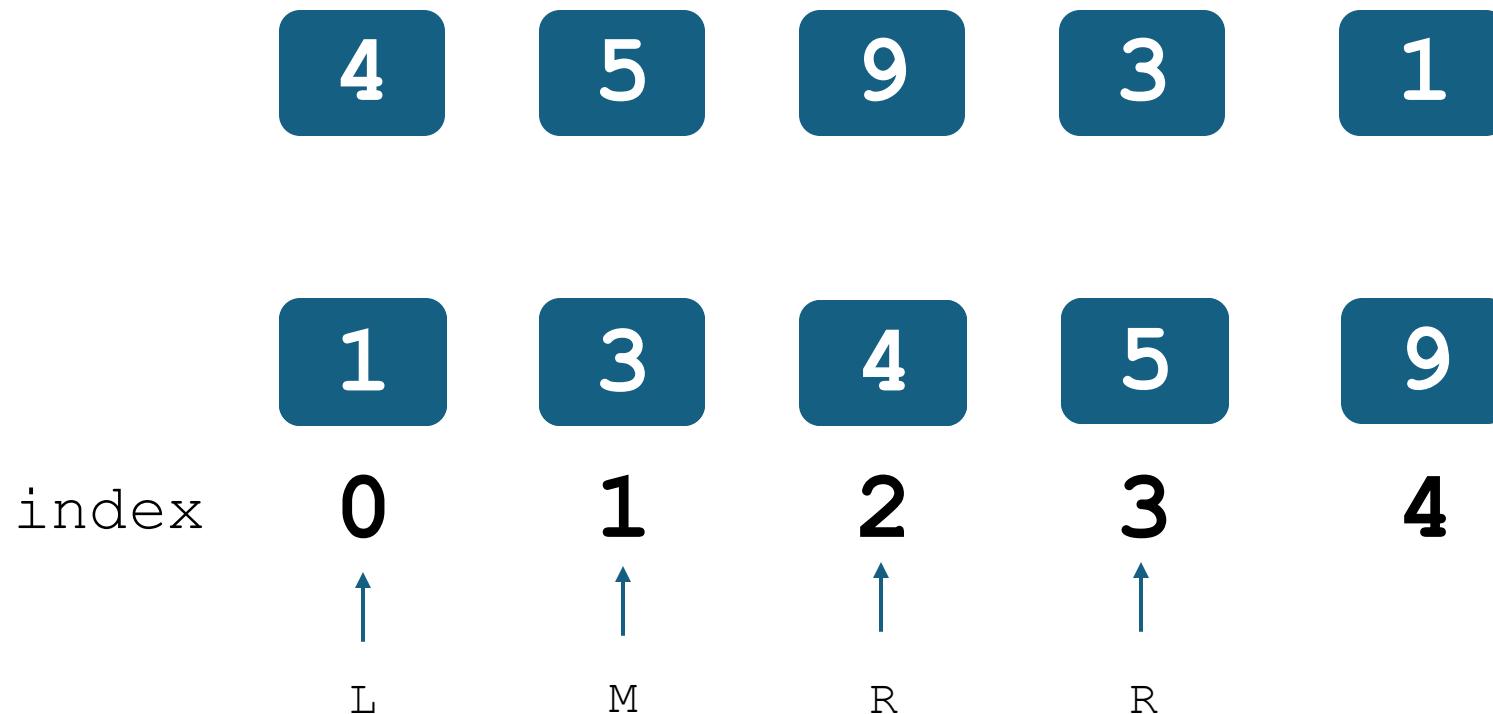
Yes! Use binary search to find the correct insertion point!

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:
 - $O(n \cdot \log(n))$
- ▶ 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.

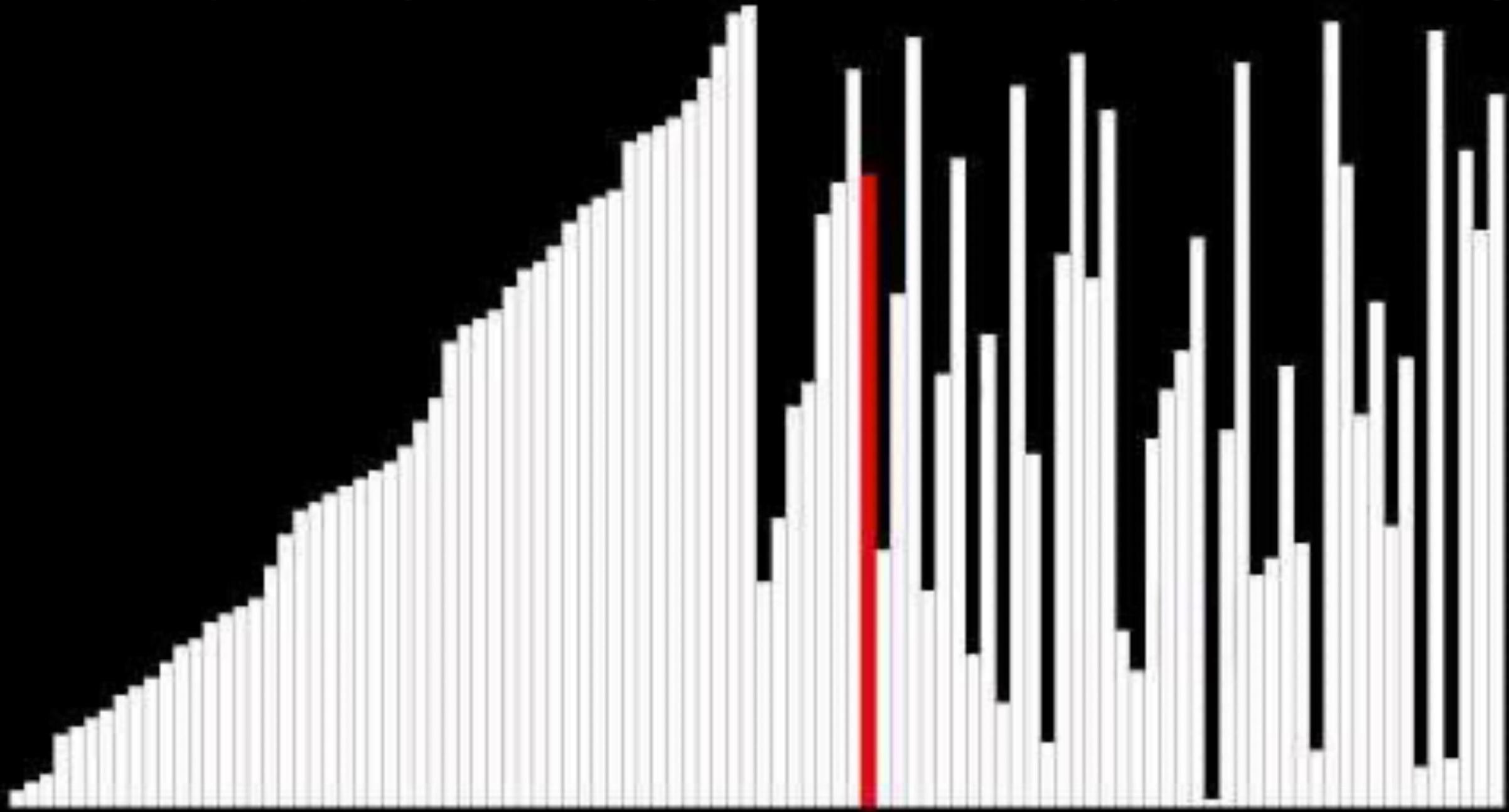


Tim sort hacks

- ▶ TimSort scans the array to detect already sorted sequences ("runs").
- ▶ Instead of regular insertion sort, it uses binary search to find the insert position.
- ▶ TimSort is a **stable algorithm**, preserving the original order of equal elements.
- ▶ In-Place Merge Buffers:
 - TimSort copies only the smaller run into a temp array, reads the other in place, and merges back into the original array.
- ▶ Galloping Mode:
 - When merging, if one run starts to dominate (many elements copied in a row), it switches to a fast copy mode.

Tim Sort - 236 comparisons, 723 array accesses, 34 ms delay

<http://pantheons.net/2013/sound-of-sorting>



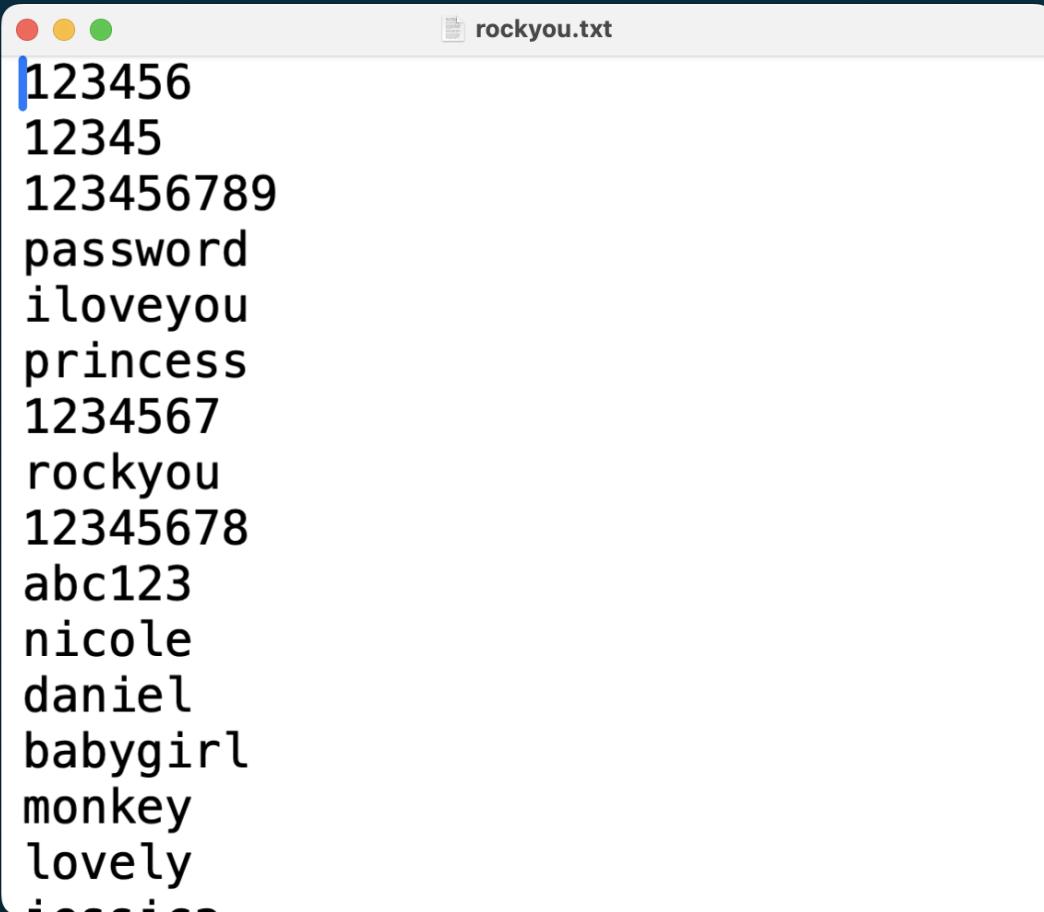
Let's compare them!

Algorithm	Time	Space
Linear search	$O(n)$	$O(1)$
Binary search	$O(\log n)$	$O(1)$
Bubble sort	$O(n^2)$	$O(1)$
Insertion sort	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n)$
Timsort	$O(n \log n)$	$O(n)$

Working with text files

Pen & Paper

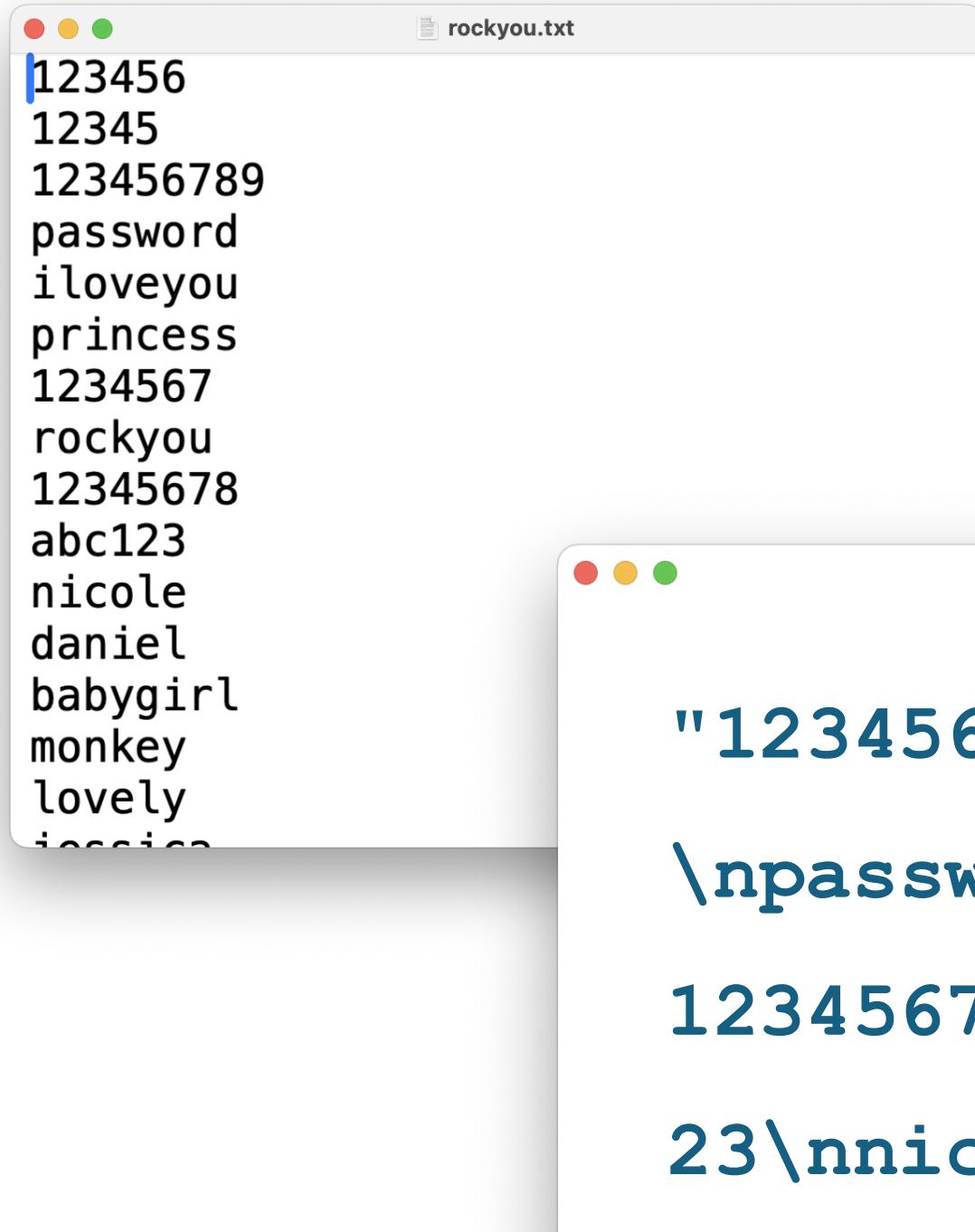
- ▶ Write the first 3 lines as Python sees them (raw input as a long String).



The screenshot shows a Mac OS X TextEdit window with a white background and a light gray border. The title bar at the top has three colored dots (red, yellow, green) on the left and the file name "rockyou.txt" on the right. The main area contains 17 lines of text, each starting with a blue vertical cursor bar:

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

"123456\n12345\n123456789\n"



The image shows a Mac OS X application window titled "rockyou.txt". The content of the file is a list of approximately 100 common passwords and names, each on a new line. The list includes "123456", "12345", "123456789", "password", "iloveyou", "princess", "1234567", "rockyou", "12345678", "abc123", "nicole", "daniel", "babygirl", "monkey", "lovely", and "jessica".

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

How Python reads txt data!



The image shows a Mac OS X application window titled "script.py.md". The content of the file is a single string of text representing the entire "rockyou.txt" file, where each line is separated by a backslash followed by the letter "n". The text is displayed in a large, bold, blue font.

```
"123456\n12345\n123456789\n\npassword\niloveyou\nprincess\n\n1234567\nrockyou\n12345678\nabc1\n23\nnicole\nndaniel\nnbabygirl\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!

CSV → Comma Separated Values

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\nD  
ave,M,39,72,167\nElly,F,30,66,1  
24\nFran,F,33,66,115\nGwen,F,26  
,64,121"
```

CSV is just one type of delimited text format

Format	Separator	Example
CSV	,	(comma) name,age,city
TSV	\t	(tab) name\tage\tcity
SSV	;	(semicolon) name;age;city
Pipe-delimited	`	(pipe) ` (pipe)
Colon-separated	:	(colon) name:age:city



A screenshot of a terminal window titled "script.py.md". The window shows Python code for reading a TSV file named "data.tsv". The code uses the csv module to create a reader object with a tab delimiter.

```
import csv
with open('data.tsv', newline='') as file:
    reader = csv.reader(file, delimiter='\t')
    ...

```

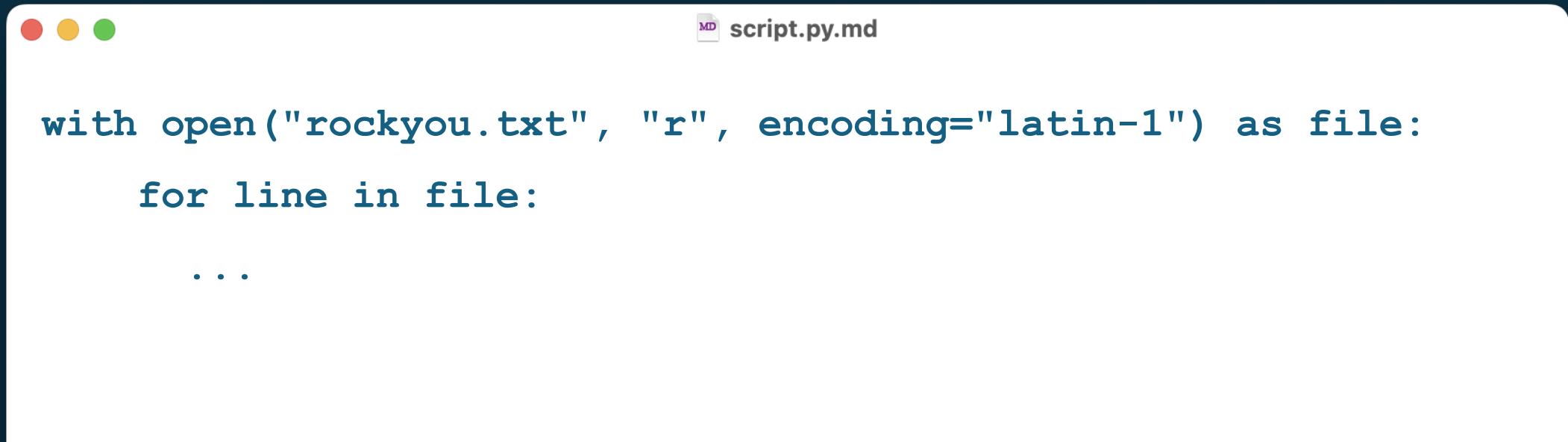
Data streams

What are examples of data streams?

- ▶ Twitter feed (tweets coming in live) - Social Media
- ▶ Temperature readings from smart thermostats - Sensor Data (IoT)
- ▶ Real-time stock prices or crypto price feeds - Financial Markets
- ▶ Live video from a webcam or streaming audio - Video/Audio
- ▶ Real-time user activity (clicks, page views) - Web Analytics
- ▶ Location updates from delivery vehicles or phones - GPS/Tracking
- ▶ Searching in a huge file!
 - File doesn't fit to computer memory!
 - Read it line by line or in chunks

Pen & paper!

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



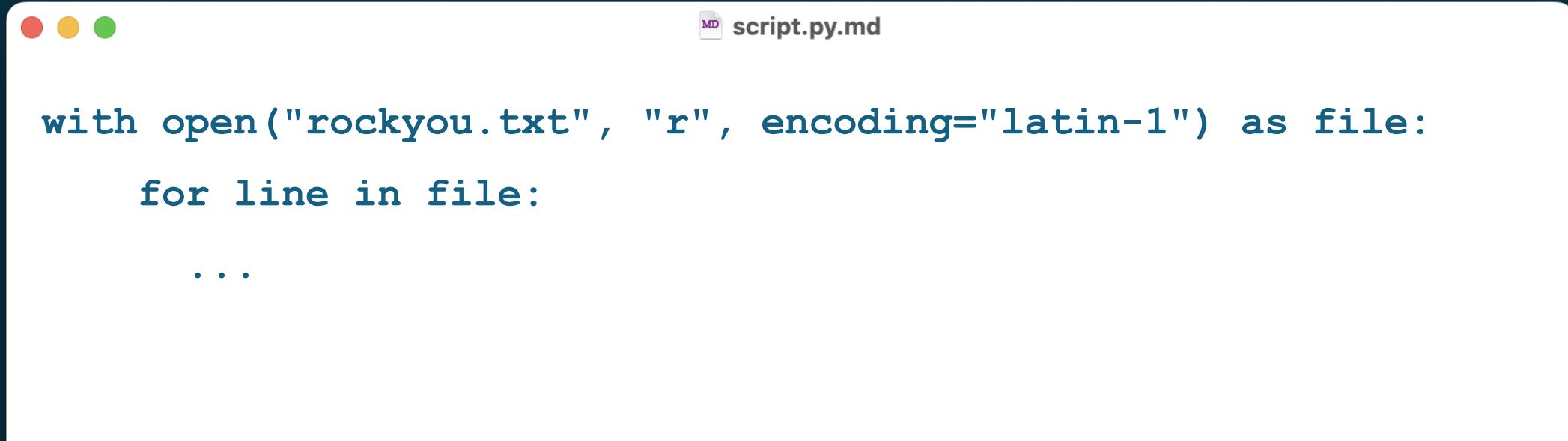
A screenshot of a Mac OS X desktop environment. In the top right corner, there is a file icon labeled "script.py.md". The main area shows a terminal window with the following code:

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

02:00

Pen & paper!

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



A screenshot of a Mac OS X desktop environment. In the top right corner, there's a white rounded rectangle containing the text "02:00". Below it, on the desktop, is a small icon of three colored dots (red, yellow, green). To the right of the desktop area is a white terminal window. At the top of the terminal, there are three colored window control buttons (red, yellow, green) on the left and the file name "script.py.md" with a ".md" icon on the right. Inside the terminal window, the following Python code is displayed:

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



```
found = False

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

    for line in file:

        if line.strip() == "ilovepizza22":

            found = True

            break

print(found)
```

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

We always need better...

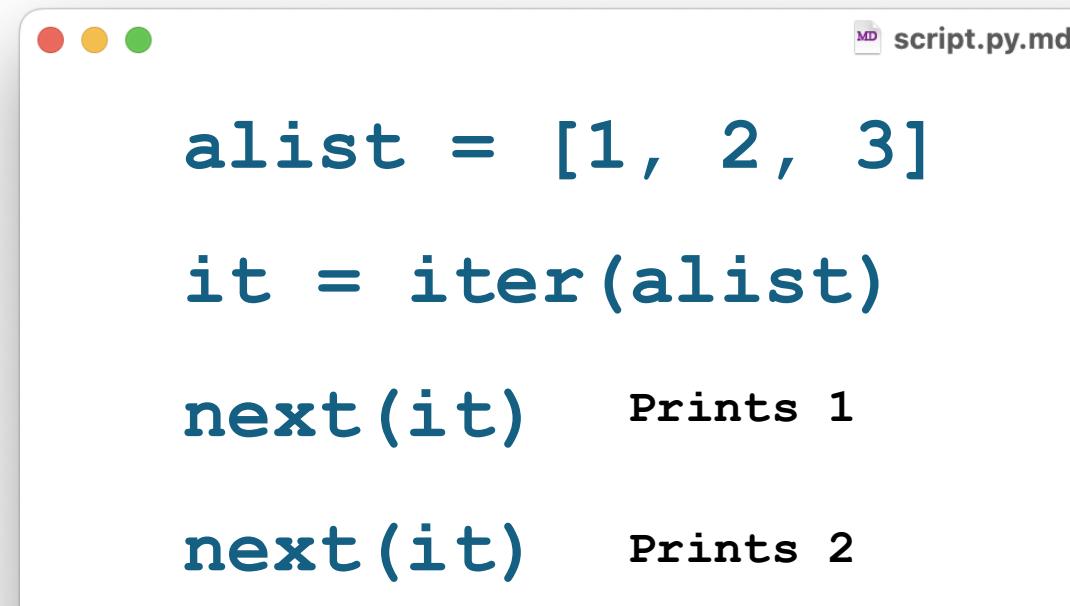
- ▶ Continuous goal to reduce the cost of:
 - Processing data → improve time complexity
 - Storing data → reduce space complexity

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

Iterators: How does it work?

- `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.

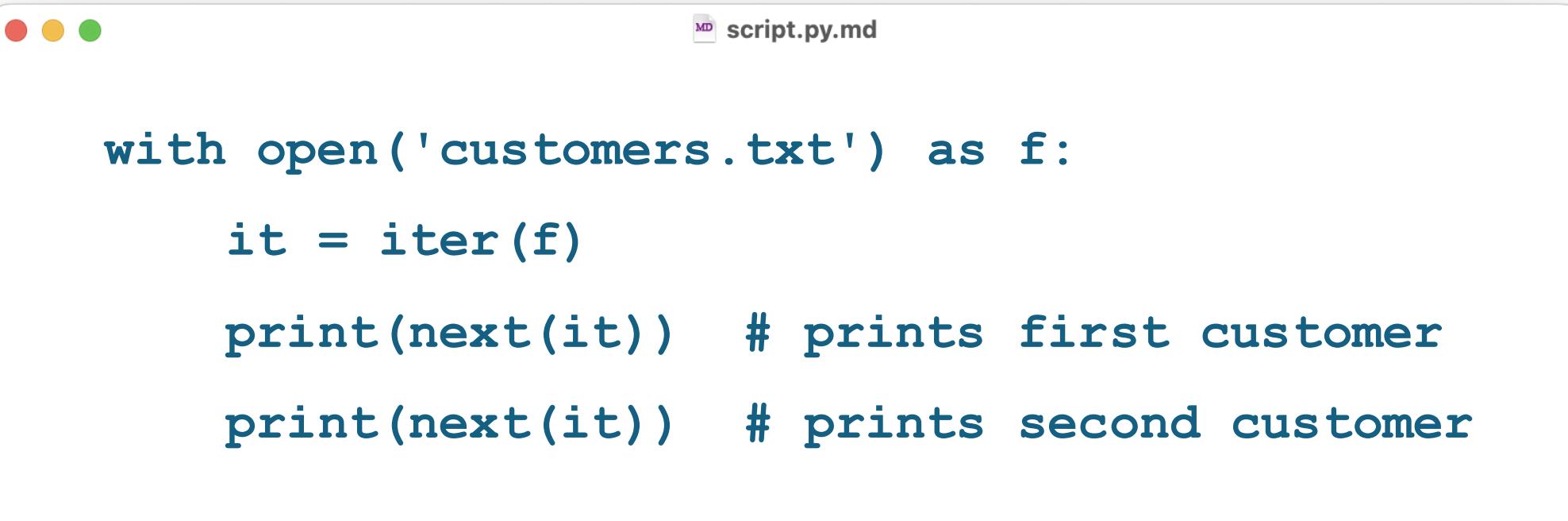


The image shows a screenshot of a Mac OS X application window titled "script.py.md". The window contains the following Python code:

```
alist = [1, 2, 3]
it = iter(alist)
next(it)    Prints 1
next(it)    Prints 2
```

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.



The image shows a screenshot of a Mac OS X desktop environment. In the top right corner, there is a window titled "script.py.md". The window has the standard red, yellow, and green close buttons. Inside the window, there is a code editor displaying the following Python script:

```
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

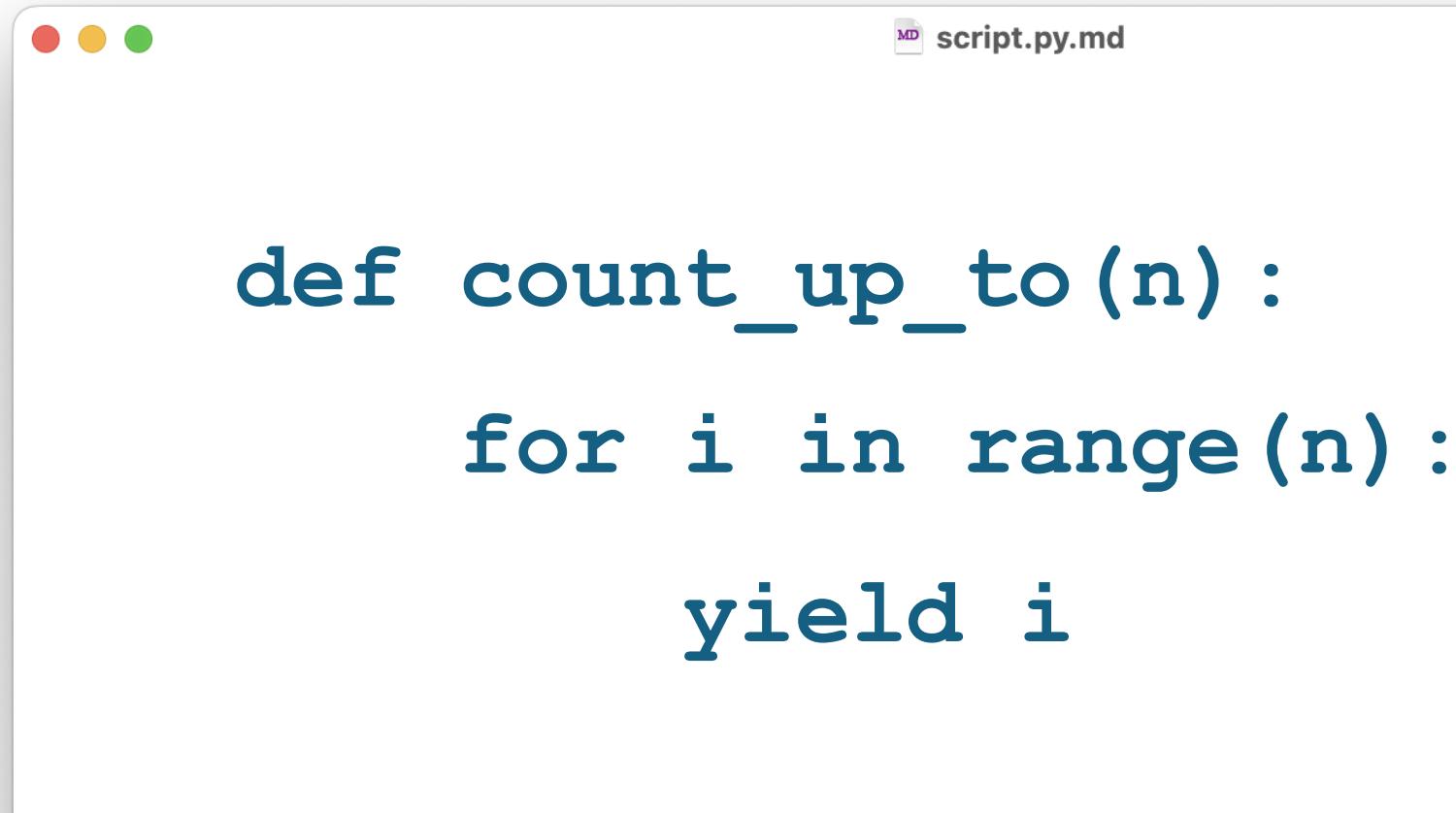
Space complexity analysis

- ▶ 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**

Python generators

Using generators

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



The image shows a screenshot of a Mac OS X application window. The title bar reads "script.py.md". The main content area contains the following Python code:

```
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 $O(n \cdot m)$

```
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

	<i>return (load all rows)</i>	<i>yield (stream one row at a time)</i>
What it does	Loads and returns the full list	Yields one row at a time (generator)
Time Complexity	$O(n \cdot m)$ (<i>read all n rows with m fields</i>)	$O(n \cdot m)$ (<i>same total work, row-by-row</i>)
Space Complexity	$O(n \cdot m)$ (<i>stores all rows in memory</i>)	$O(m)$ (<i>just one row at a time</i>)
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

L22 In the late 1970s, an accused serial rapist claims multiple personalities control his behavior, setting off a legal odyssey that captivates America.

	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 2020	2020	PG-13	90 min	Documentaries	As her father nears
3	s2	TV Show	Blood & Water		Ama Qamata	South Africa	September 2021	2021	TV-MA	2 Seasons	International	After crossing paths
4	s3	TV Show	Ganglands	Julien Leclercq	Sami Bouajila, Tracy Goto	September 2021	2021	TV-MA	1 Season	Crime TV Shows	To protect his family	
5	s4	TV Show	Jailbirds	New Orleans		September 2021	2021	TV-MA	1 Season	Docuseries, Feuds,	Flirtations and	
6	s5	TV Show	Kota Factory		Mayur More, India	September 2021	2021	TV-MA	2 Seasons	International	In a city of coaching	
7	s6	TV Show	Midnight Mass	Mike Flanagan	Kate Siegel, Zach Gilford, Jonathan Groff	September 2021	2021	TV-MA	1 Season	TV Dramas, Thrillers	The arrival of a char	
8	s7	Movie	My Little Pony:	Robert Culler	Vanessa Hudgens, Kimiko Glenn, Dove Cameron	September 2021	2021	PG	91 min	Children & Family	Equestria's divided.	
9	s8	Movie	Sankofa	Haile Gerima	Kofi Ghanaba	United States	September 2021	1993	TV-MA	125 min	Dramas, Indie	On a photo shoot in
10	s9	TV Show	The Great British	Andy Devons	Mel Giedroyc	United Kingdom	September 2021	2021	TV-14	9 Seasons	British TV Shows	A talented batch of
11	s10	Movie	The Starling	Theodore Melfi	Melissa McCarthy	United States	September 2021	2021	PG-13	104 min	Comedies, Drama	A woman adjusting
12	s11	TV Show	Vendetta: Truth, Lies and The Mafia			September 2021	2021	TV-MA	1 Season	Crime TV Shows	Sicily boasts a bold	
13	s12	TV Show	Bangkok Bloodbath	Kongkiat Komorn	Sukollawat Kanarot, Sushant Singh Rajput	September 2021	2021	TV-MA	1 Season	Crime TV Shows	Struggling to earn a	
14	s13	Movie	Je Suis Karl	Christian Schwochow	Luna Wedler, Germany, Czech Republic	September 2021	2021	TV-MA	127 min	Dramas, International	After most of her fa	
15	s14	Movie	Confessions	Bruno Garotti	Klara Castanho, Lucca Picoli	September 2021	2021	TV-PG	91 min	Children & Family	When the clever bu	
16	s15	TV Show	Crime Stories: India Detectives			September 2021	2021	TV-MA	1 Season	British TV Shows	Cameras following	
17	s16	TV Show	Dear White People		Logan Browning	United States	September 2021	2021	TV-MA	4 Seasons	TV Comedies	Students of color na
18	s17	Movie	Europe's Most Wanted	Pedro de Echave Garcí	Á, Pablo Azorín, Iván Marí	September 2020	2020	TV-MA	67 min	Documentaries	Declassified docum	
19	s18	TV Show	Falsa identidad		Luis Ernesto	Mexico	September 2020	2020	TV-MA	2 Seasons	Crime TV Shows	Strangers Diego and
20	s19	Movie	Intrusion	Adam Salter	Freida Pinto, Logan Marshall-Green	September 2021	2021	TV-14	94 min			
21	s20	TV Show	Jaguar		Blanca Suárez, Iván Martínez	September 2021	2021	TV-MA	1 Season			
22	s21	TV Show	Monsters Inside	Olivier Megaton		September 2021	2021	TV-14	1 Season			
23	s22	TV Show	Resurrection: Ertugrul	Engin Altan Durmus	Turkey	September 2021	2018	TV-14	5 Seasons			
24	s23	Movie	Awai Shanmugam	K.S. Ravikumar	Kamal Hassan, Meena, Gautham Karthik	September 2021	1996	TV-PG	161 min	Comedies, Indian	Newly divorced and	
25	s24	Movie	Cal	Alex Wegerif	Stéphane Papon, Paul Kilkenny	September 2021	2021	TV-Y	61 min	Children & Family	From a grade school	

Netflix Dataset

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)

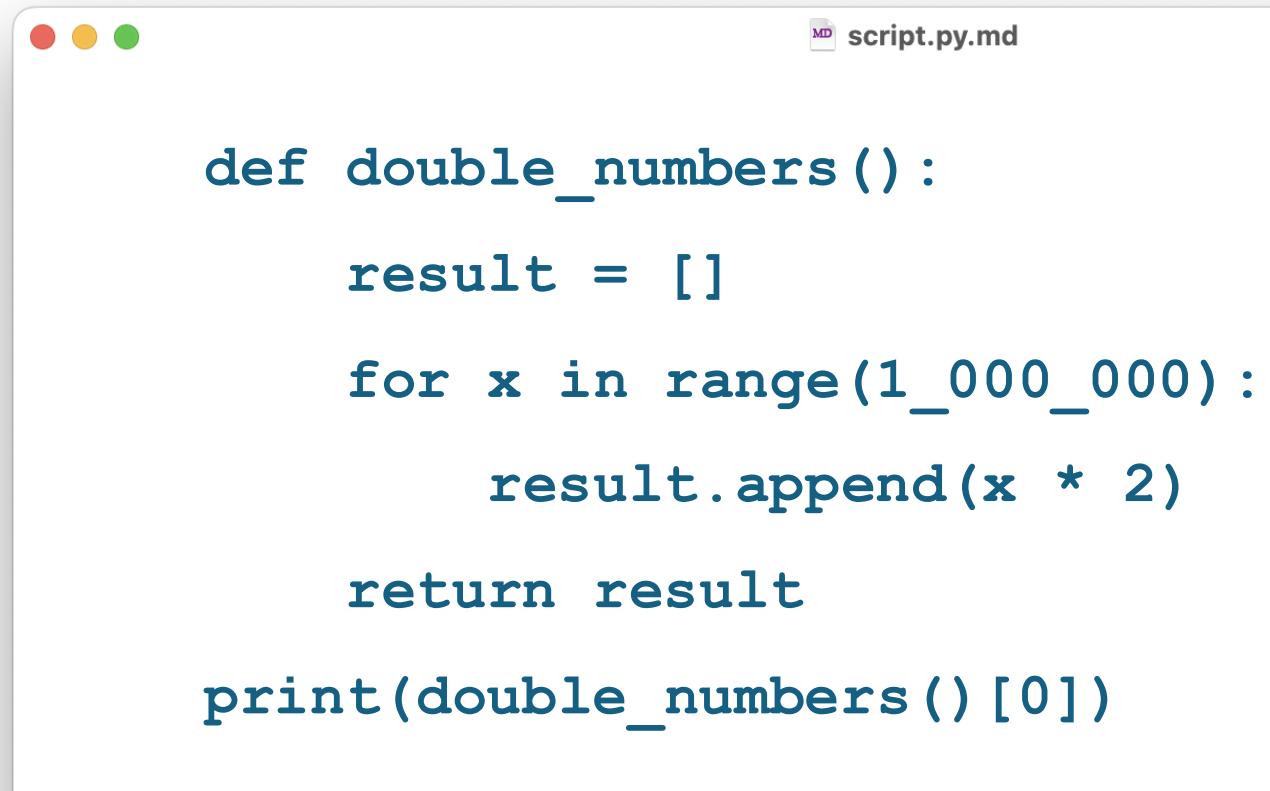
What is a lazy evaluation?

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!



A screenshot of a Mac OS X terminal window. The window title is "script.py.md". The code inside the window is:

```
def double_numbers():

    result = []

    for x in range(1_000_000):

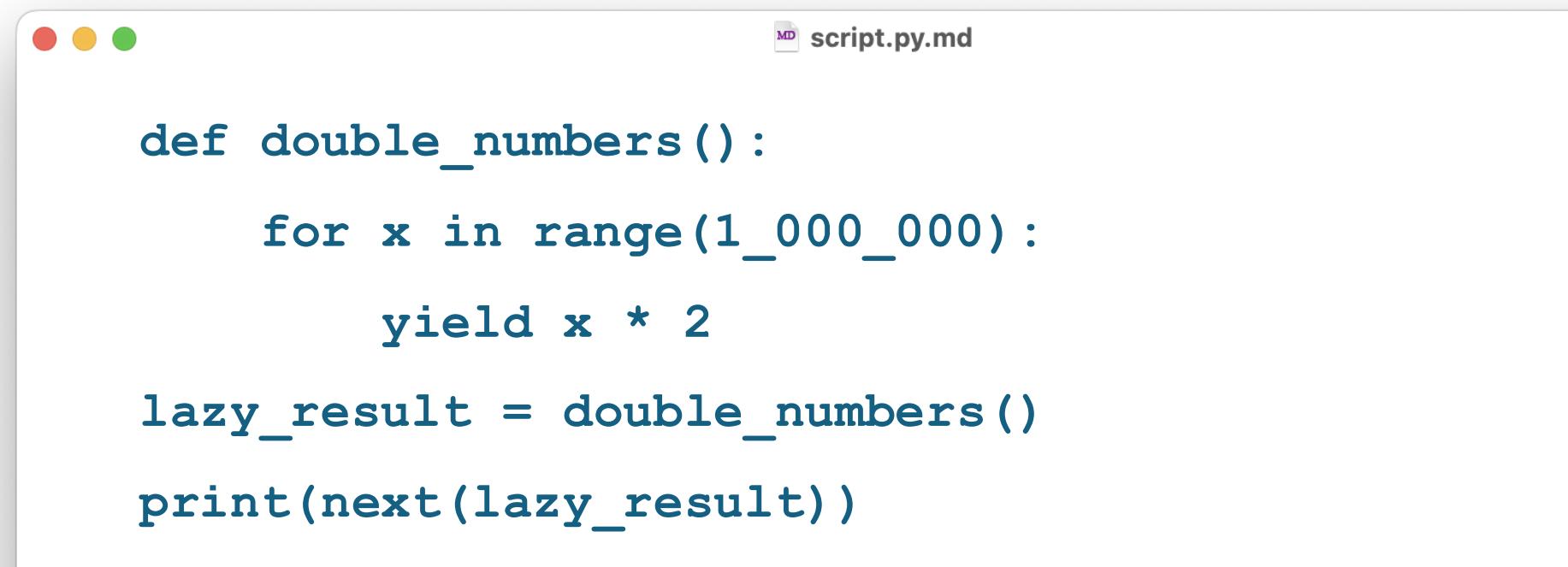
        result.append(x * 2)

    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



A screenshot of a terminal window titled "script.py.md". The window contains the following Python code:

```
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.