

The Sieve of Eratosthenes – Expanded Version for Submitted Project

Griffin Lyons
CS 550
Fall 2023



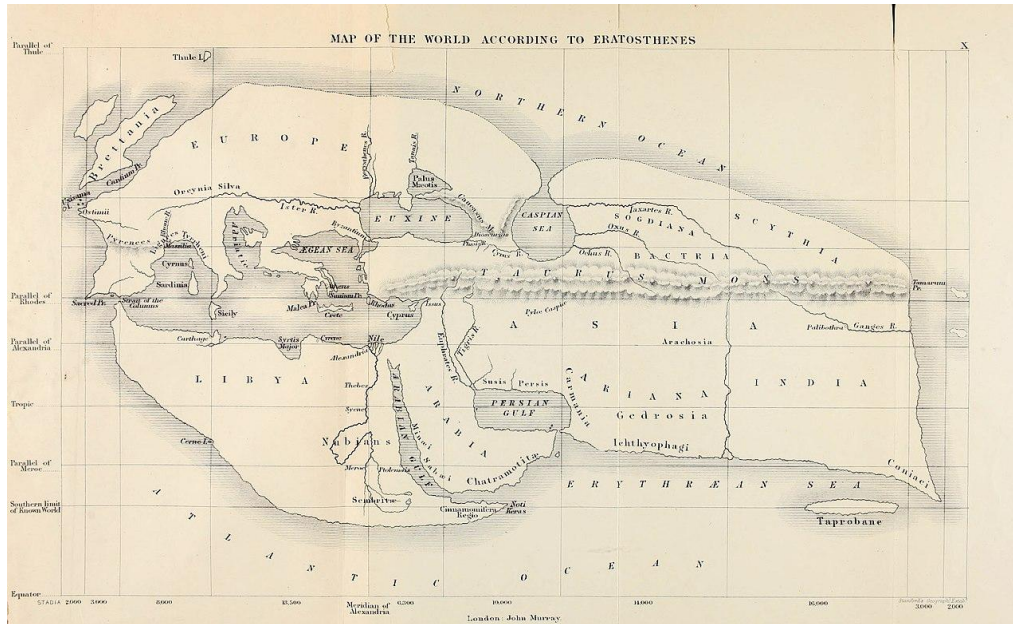
Who was Eratosthenes? (1)



"Eratosthenes Teaching in Alexandria", Bernardo Strozzi, c. 1635

- Greek scientist, philosopher, geographer, astronomer, poet
- Lived from approximately 276 BCE to 195/194 BCE
- Born in the Greek city of Cyrene on northern coast of what is now Libya
- Was chief librarian at Library of Alexandria

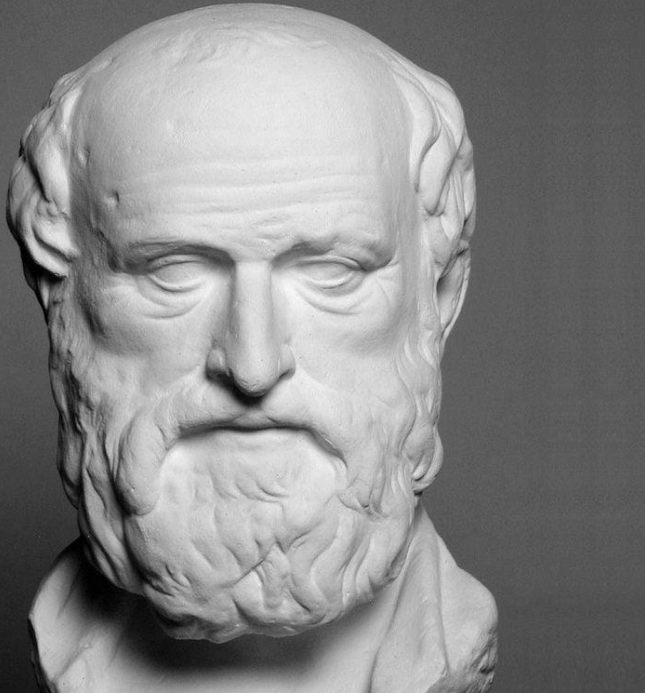
Who was Eratosthenes? (2)



- First known person to calculate circumference of Earth
- First to calculate axial tilt of Earth
- Created first global projection map of the world

A recreation of Eratosthenes's map, from 1883. Edward Bunbury, *A History of Ancient Geography among the Greeks and Romans from the Earliest Ages till the Fall of the Roman Empire*

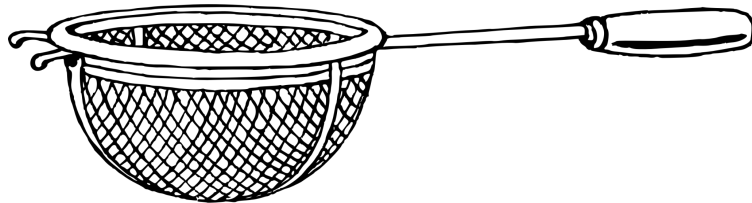
The life of Eratosthenes



- Founder of scientific chronology
- using multiple sources including historical records to estimate times and dates of events
- Renowned for his love of learning and wide knowledge of subjects
 - His admirers called him *Pentathlos*, after the versatility of Olympic athletes
 - Some of his critics said he came in second in every field, and thus called him *beta*

The Sieve of Eratosthenes

- A method of finding prime numbers
- Take a given range of numbers
- Start by removing every number that is a multiple of another number besides 1 (because it cannot be prime)
- Eventually, you have a list of primes



2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10

Applications of prime numbers

- The single most important use of prime numbers is in cryptography
- It is computationally very expensive and time consuming to factor a very large number back into prime numbers
- This means combinations of prime numbers can be used as cryptographic keys - the combination can be broken, but it would take an incredibly long time

Sieve of Eratosthenes – my implementation (1)

```
def naive_sieve_of_eratosthenes(upper_limit):  
    """This is a function that calculates primes based on the method  
    discovered by the Greek philosopher Eratosthenes. I call it  
    'naive' because I have not performed any optimizations on it or  
    tried to implement more sophisticated algorithmic choices as such."""  
    primes_list = []  
    for i in range(2, upper_limit):  
        prime_counter = 0  
        for j in primes_list:  
            if i % j == 0:  
                prime_counter += 1  
        if prime_counter == 0:  
            primes_list.append(i)  
        else:  
            pass  
    return primes_list
```

This is my function for implementing the Sieve of Eratosthenes. It's not perfect, and I realize that it doesn't implement the algorithm exactly as described - it does not remove every multiple of a prime in succession before moving to the next. It's also a little inelegant, particularly in resorting to a counter.

However, I opted to leave this code be, because I wanted to show that even a simple, unoptimized - that is, naive - implementation of the Sieve of Eratosthenes could be very powerful.

My implementation, continued (2)

```
#import the time module to time how fast our algorithm works
import time

time_0 = time.time()

primes_to_121_list = naive_sieve_of_eratosthenes(121)

time_1 = time.time()

total_time = time_1 - time_0
print("{:.100f}".format(time_0))
print("{:.100f}".format(time_1))
print("It took {:.500f} seconds to run the naive Sieve of Eratosthenes implementation".format(total_time))
```

[illegible]

Even in trying to time the run time of my implementation, my almost-ten-year-old laptop can't register a significant amount of time.

My implementation, continued (3)

```
#Repeat this, but for primes up to 25000 (as an arbitrary larger number)
import time
time_0 = time.time()

primes_to_25000_list = naive_sieve_of_eratosthenes(25000)

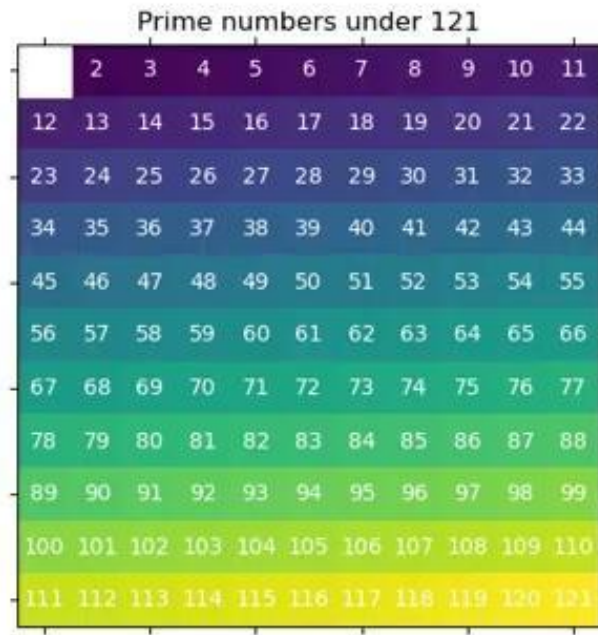
time_1 = time.time()

total_time = time_1 - time_0
print("It took {:.5f} seconds to run the naive Sieve of Eratosthenes implementation".format(total_time))

It took 3.25131 seconds to run the naive Sieve of Eratosthenes implementation
```

If we increase the upper limit to 25000 (as an arbitrary number greater than 121), it still takes a very small amount of time.

Animating the Sieve (1)



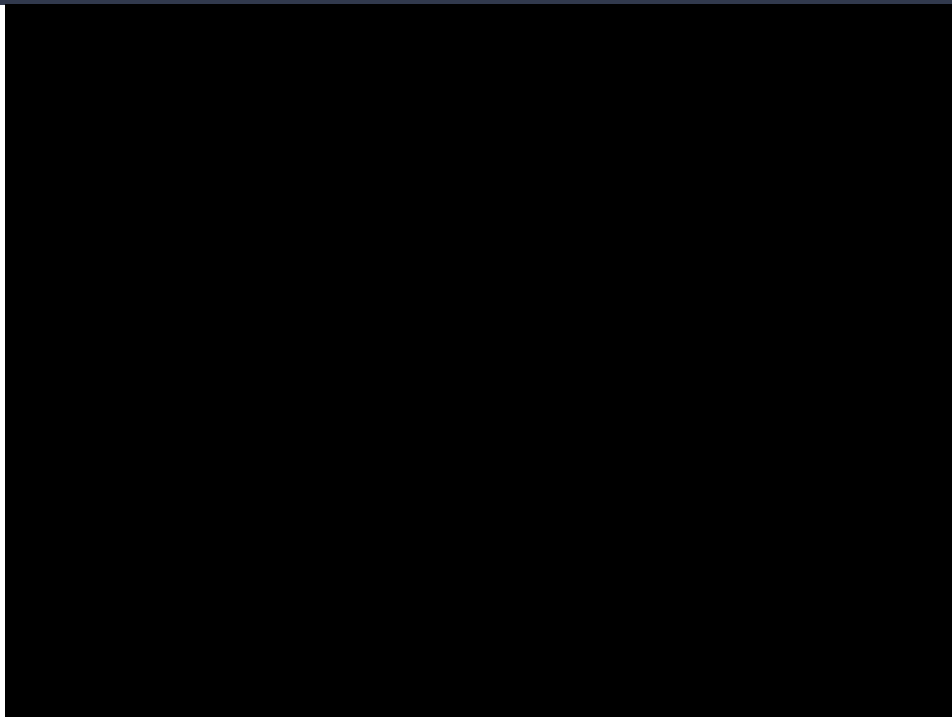
I decided to try plotting and animating the Sieve of Eratosthenes to visualize it in action.

(This is `_draft_4` in the attachments.)

This required learning and trying a lot of Python features I was previously unfamiliar with.

Here is the final animation. If I could change anything further, I'd implement a dictionary to contain both the prime number and its multiple, so that as each non-prime multiple of a prime was being removed, I could list at the bottom of the top which prime's multiples were being removed (e.g. "Removing multiples of 2", "Removing multiples of 3", and so on).

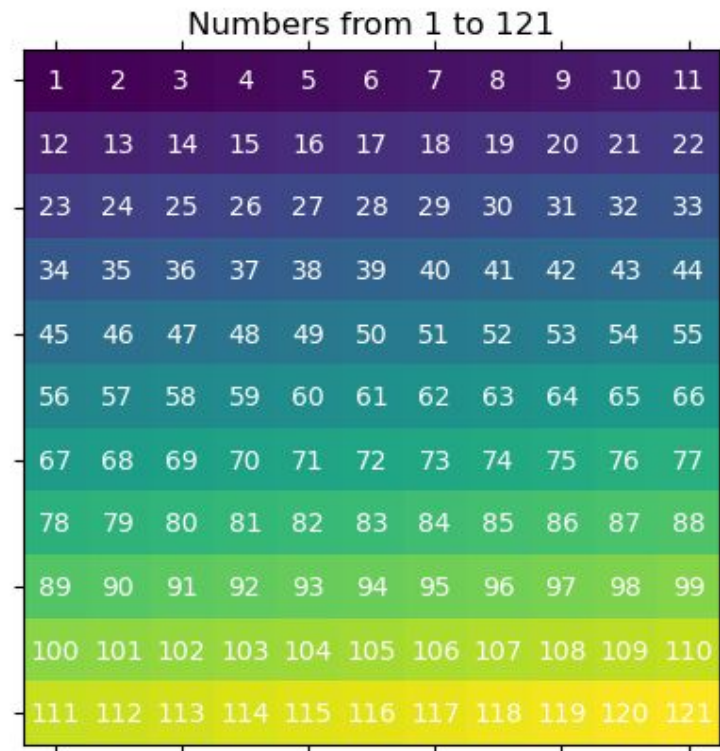
Animating the Sieve (3) – Early tries



What you're looking at is one of the very earliest versions of my animation (_original in my attachments). It's not very good. It just blanks out the non-primes in order without really effectively visualizing what the algorithm is doing.

I did this, and all animations, using Matplotlib's FuncAnimation feature.

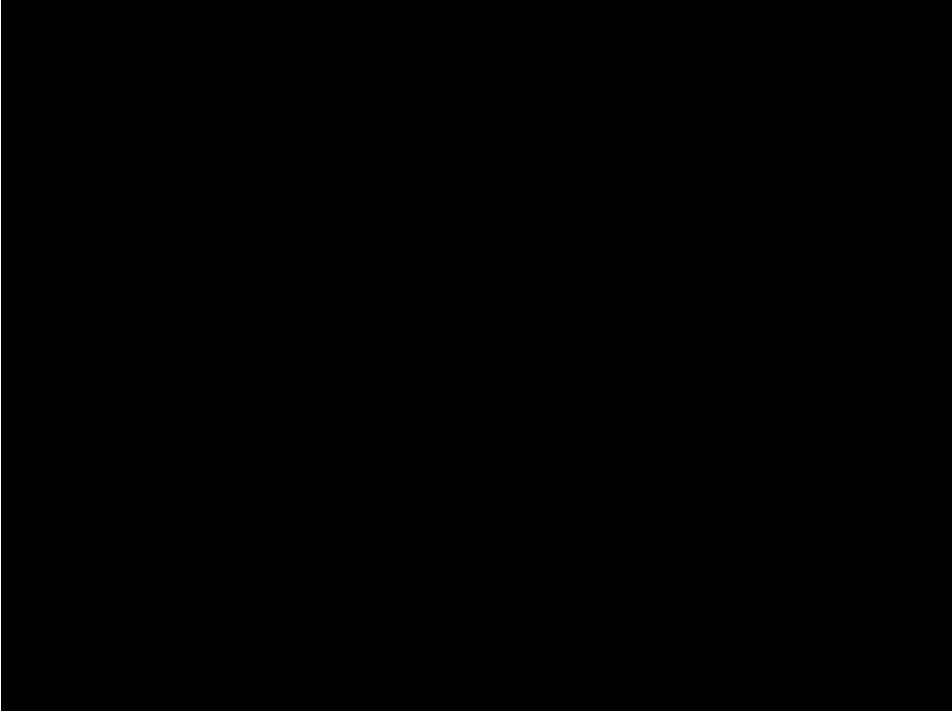
Animating the Sieve (2)



First, I had to learn how to plot a grid with my desired range. In my code, you can see that NumPy's `arange()` object and its `reshape` feature very helpfully make this possible.

After that, it was just a matter of finding a color palette. I opted for the visually pleasing gradient of the viridis palette.

Animating the Sieve (4) – Getting closer



This was my second try (`_draft_2` in my attachments). In all my animations, I plotted the grid, then wrote the custom function that Matplotlib's `FuncAnimate` would take and apply, using each frame of the animation as the value to be input into the function to create the new plot.

I tried to slow it down (by decrementing the frames per second argument in Matplotlib) to make it more obvious what was happening, but it's still confusing and hard to follow in my opinion.

Animating the Sieve (5) – Almost there



This one is a more incremental improvement (`_draft_3`). I had the wherewithal to remove 1, since it is non-composite rather than prime, and might be confusing to anyone paying attention.

Animating the Sieve (6) – The Final Result

Prime numbers under 121

	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	64	65	66
67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120	121

This is what you saw earlier (`_draft_4` in my attachments), and what I presented to the class.

I showed this to some peers, who gave me helpful feedback that inspired my use of the red ‘flash’ for each non-prime when it is removed (obscured with a white rectangle). I achieved this in code by doubling each non-prime number in my list to be crawled by my function, and using the odd and even nature of the list index to decide which animation would be used: the red rectangle (the “flash”) or the white rectangle (the “removal”, which would be permanent for the rest of the animation).

Conclusion

I'm very proud of what I achieved, even I may have not wholly observed some best practices in my haste to get working code for the animation (hard-coding a reference in my function to an object outside the function, etc).

The Sieve of Eratosthenes is a marvel of math, reasoning, and simplicity. I feel that as a choice of final project topic, it embodies the spirit of the computational mathematics topics we have learned - it is something with very old roots, yet with applications and relevance even today. (And in the spirit of Doctor Pinsky's lectures, the historical context is pretty cool, too. I was very amused to learn what Eratosthenes's detractors called him.)

As to what I might do differently, I think I would try to write cleaner code. I left it as it was when I finally got the animation I wanted, because it felt more honest that way as a reflection of my work on this project. In a professional setting I'd never leave it in that state.

In my attached Jupyter Notebook, you can see almost all of the states I left it in, in reverse order from the animation I used in my class presentation to the animation I saved as the "original" (that wasn't even my first, but it was the first where I realized I'd made a major leap with the next iteration and should save the now-deprecated code for posterity).