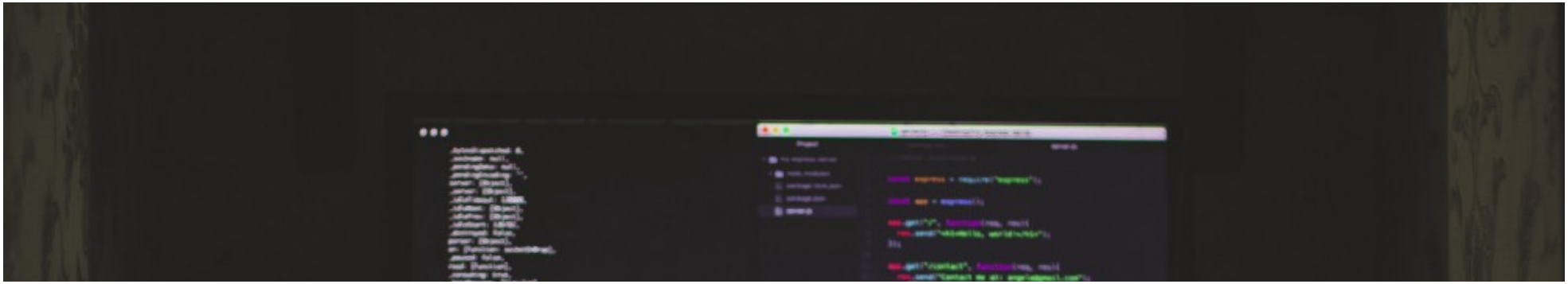
# The Most Efficient Way to Read Code Written by Someone Else

How to speed up the struggle of reading others' code in 4 steps?

Sara A. Metwalli Sep 15, 2020 · 5 min read ★



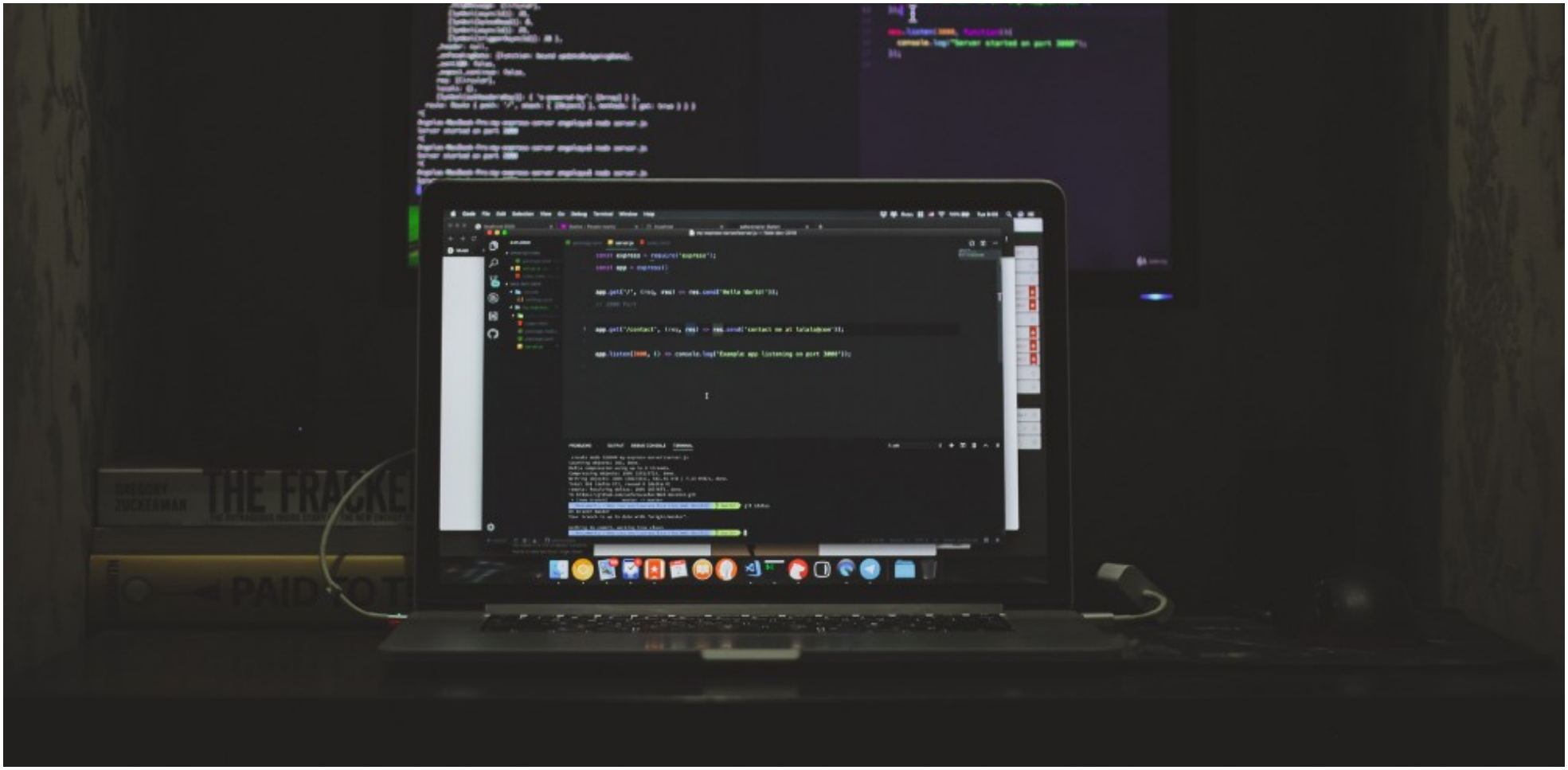


Photo by Safar Safarov on Unsplash

As developers, regardless of our specialty, whether it being data science, front end, or back end, we spend more than 75% of our time reading code written by others. That task can be such a demanding task.

That being said, the ability to read others' code efficiently is one of the skills that could make one's job in software engineering much more pleasant. Unfortunately, it is also a skill that is widely overlooked by schools, bootcamps, and companies. It is one of those skills that everyone assumes you know, or be good at, just because you know how to write code.

The thing about writing code is, everyone has there own coding style. Reading code is not like reading a novel or a story; it's not just about reading instructions on the screen. Instead, when you read code written by someone else, you're not just reading their code. You're trying to get into their thoughts process and what they were thinking when they wrote that code.

Needless to say, that is an extremely challenging task. But, it's a task that you can make exponentially easier. In this article, I will walk you through my 4 steps process of reading and understanding other people's code.

To explain the different steps, I will go through a code I wrote for a web scraping tutorial.

```
1  #Import needed libraries
2  from bs4 import BeautifulSoup as bs
3  import requests as rq
4  import pygal
```

```
import time
import pygal
from IPython.display import display, HTML

base_html = """
<!DOCTYPE html>
<html>
  <head>
  <script type="text/javascript" src="http://kozea.github.com/pygal.js/javascripts/svg.jquery.j
  <script type="text/javascript" src="https://kozea.github.io/pygal.js/2.0.x/pygal-tooltips.min
  </head>
  <body>
    <figure>
      {rendered_chart}
    </figure>
  </body>
</html>
"""
#define functions for data collection
def find_book_name(table):
    if table.find('caption'):
        name = table.find('caption')
    return name.text

def get_author(table):
    author_name = table.find(text='Author').next.text
    return author_name

def get_genre(table):
    if table.find(text='Genre'):
        genre = table.find(text='Genre').next.text
    else:
        genre = table.find(text='Subject').next.next.next.text
    return genre
```

```
    return genre

def get_publishing_date(table):
    if table.find(text='Publication date'):
        date = table.find(text='Publication date').next.text
    else:
        date = table.find(text='Published').next.text
    pattern = re.compile(r'\d{4}')
    year = re.findall(pattern, date)[0]
    return int(year)

def get_pages_count(table):
    pages = table.find(text='Pages').next.text
    return int(pages)

def parse_wiki_page(url):
    page = rq.get(url).text
    return bs(page)

def get_book_info_robust(book_url):
    #To avoid breaking the code
    try:
        book_soup = parse_wiki_page(book_url)
        book_table = book_soup.find('table',class_="infobox vcard")
    except:
        print(f"Cannot parse table: {book_url}")
        return None

    book_info = {}
    #get info with custom functions
    values = ['Author', 'Book Name', 'Genre',
            'Publication Year', 'Page Count']
    functions = [get_author, find_book_name, get_genre,
                get_publishing_date, get_pages_count]
```

```
72
73      for val, func in zip(values, functions):
74          try:
75              book_info[val] = func(book_table)
76          except:
77              book_info[val] = None
78
79      return book_info
80  #Get books
81  url = 'https://en.wikipedia.org/wiki/Time%27s_List_of_the_100_Best_Novels'
82  page = rq.get(url).text
83  soup = bs(page)
84  rows = soup.find('table', class_="wikitable sortable").find_all('tr')[1:]
85  books_links = [row.find('a')['href'] for row in rows]
86  base_url = 'https://en.wikipedia.org'
87  books_urls = [base_url + link for link in books_links]
88  #to store books info
89  book_info_list = []
90  #loop first books
91  for link in books_urls:
92    #get book info
93    book_info = get_book_info_robust(link)
94    #if everything is correct and no error occurs
95    if book_info:
96      book_info_list.append(book_info)
97    #puase a second between each book
98    time.sleep(1)
99
100 #Collect different genres
101 genres = {}
102 for book in book_info_list:
103     book_gen = book['Genre']
104     if book_gen:
105         if 'fiction' in book_gen or 'Fiction' in book_gen:
```

```
            book_gen = 'fiction'
        if book_gen not in genres: #count books in each genre
            genres[book_gen] = 1
        else:
            genres[book_gen] += 1

print(genres)
#Plot results
bar_chart = pygal.Bar(height=400)
[bar_chart.add(k,v) for k,v in genres.items()]
display(HTML(base_html.format(rendered_chart=bar_chart.render(is_unicode=True))))

```

Top100Books.py hosted with ♡ by GitHub view raw

## Step #1: Run the code and see what it does

Whenever you get a code that you need to read and understand, the first thing — and most apparent, I may argue — to do is to run the code and see what it does. What does it take as inputs? What are the outputs?

So, let's go ahead and run the code above and explore the results.

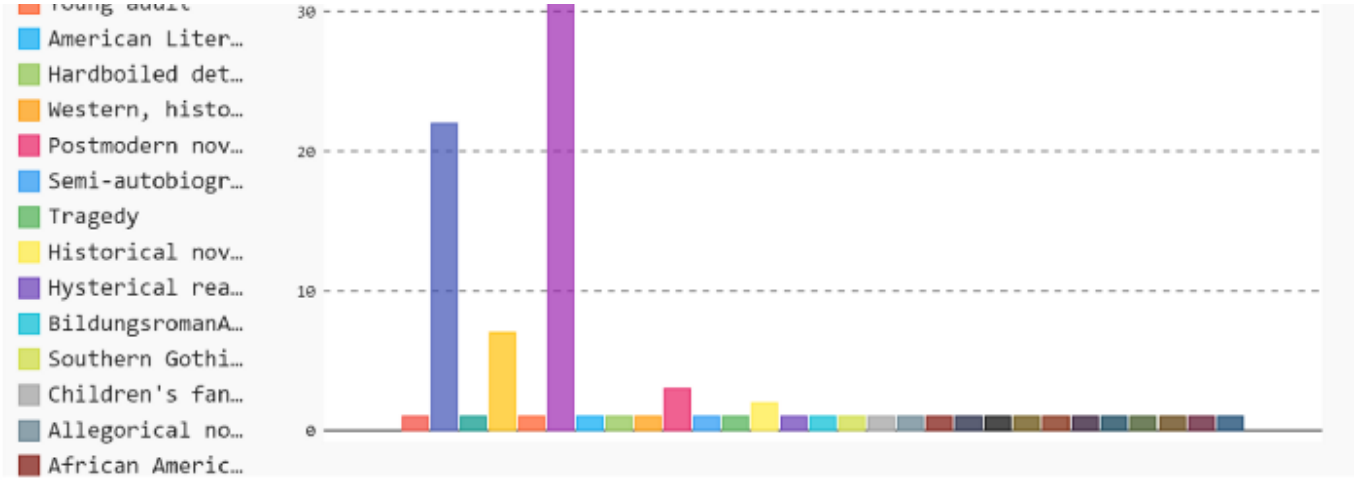



Image by the Author (The code's results screenshot)

Not all codes will result in a chart; some codes' output will be a text output. Regardless of the output type, we can explore it to guess — if you don't already know — the general idea of what the code was written to do.

However, this might not give you a lot of details about the details of the code, or if this code was a part of a bigger project, you might not know how it all connects, but you will learn how to build it and run it. Also, you will get to know about the libraries it uses and the frameworks it depends on.

Nevertheless, running the code will give you the information you need to understand it better, and to make sure you got every dependency, you need to start working and expanding it.

## Step #2: Find the main function or the start point

Now that we know what the code does and what the output is/are, we can start looking deeper into its details. To do that, we need to pinpoint the start of the code. If you're using a programming language with a must `main` function, like C, C++, or Java, start from there and walk your way through to the other functions.

If you're using Python, however, not all codes will have a `main` function, but you can use the indentations to know where the code starts. For example, in the code above, we have multiple functions, and then the starting point of the code is on ***line 64***.

So, we can start going through the comments first — if there is any — and go through the entire main section of the code without going into details of the subfunctions.

Going through the `main` function gives you the general flow of the code, what each subfunction does, not how they work, but what they do.

## Step #3: Run the code on debugging mode

Once you're done carefully reading the main part of the code, you may find it useful to run the code in debugging mode. The reason for that is, when you run code in the debugger, it allows you to observe how your code interacts with the memory.

It will show you how each variable changes with every step in the code. Doing so will give you a more in-depth understanding of the inner functionality of the code and its different functions.

Once you see how each variable in the code changes with every line, you can start adding your own comments in the code, explaining to yourself what every line of the code does.

## Step #4: Build a mindmap of the connections between different parts of the code

One thing I found to help tremendously is to build a mindmap of the code connections while I am running it in the debugger mode. The debugger mode shows you a clear connection between the different code items.

Start with the name of the code file in the middle of your mindmap and then branch out with the different functions and how they connect. Try ti

encourage the variables in the code into the mindmap as well, maybe not all of them, but the ones that have the most effect on the overall results.

Moreover, try to include the inputs and outputs of the code as well as their types or expected types. Here's the mindmap for the code above.

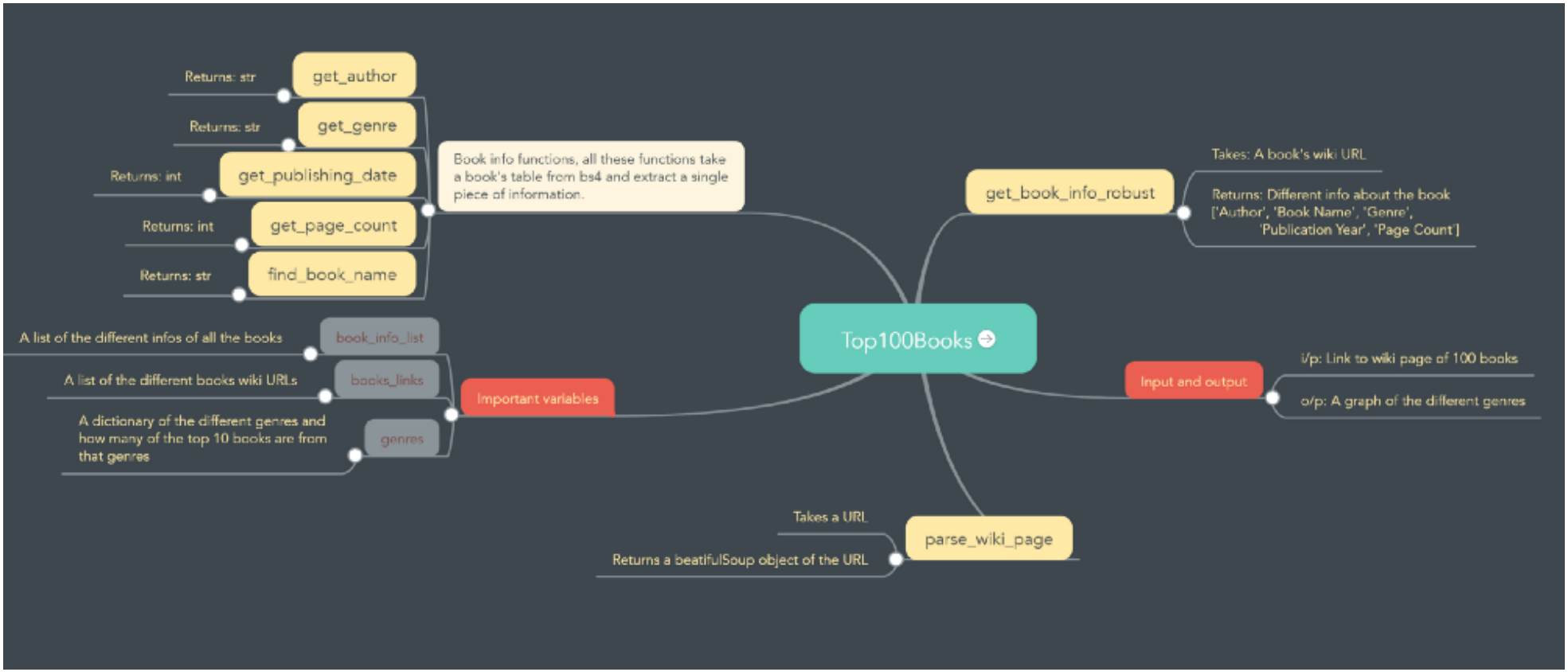



Image by the author (made using Mindmeister) A mindmap of the code above

Building the mindmap will save you so much time whenever you interact

with this codebase. It will help you know the connection in case you want to add or remove any parts of the code.

## Takeaway

Reading code written by other developers can be such a challenging task; you need to understand their logic, their style, and their specific choices. I have read so many codes written by programmers of different levels and ages. Having experienced that, I came with my own 4 steps process to ease up reading, exploring, and understanding codes written by basically someone who is not me.

This 4 step process is simple and will save you a lot of time and effort; all you need to do is:

1. Run the code and explore the results.
2. Find the main function or the start point of the code.
3. Run the code under the debugger and fully understand the code's mechanics.
4. Build a mindmap of the connections between the different code elements and use it at any time you interact with the code.

I hope you find these steps useful in saving you much time and effort in your next code exploration adventure.

> “Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”
>
> — Robert C. Martin,