JS Proposals Project

Classifying EcmaScript proposals

Kai Waløen
21 May 2025

Supervised by Mikhail Barash

Aims

- Collect proposal data
  - Snapshot from Feb 2025
  - From TC39/proposals repositories
- Classify proposals
  - Stage
  - Type of change
  - Topics and keywords
  - Present on website
- Analyse evolution of proposals
  - Pattern in stage bumps?
  - Pattern in stage duration?
  - Topic dependent?
  - Time dependent?

Stage 4
Classification: Syntactic Change Semantic Change
Human Validated: KW
Title: RegExp v flag with set notation + properties of strings
Authors: Markus Scherer, Mathias Bynens
Champions: Mathias Bynens
Last Presented: May 2023
Stage Upgrades:
Stage 1: 2021-01-28
Stage 2: 2021-05-27
Stage 2.7: NA
Stage 3: 2022-03-29
Stage 4: 2023-05-16
Last Commit: 2023-09-22
Topics: #regex #others #collections
Keywords: #regex #flag #string #set
GitHub Link: https://github.com/tc39/proposal-regexp-v-flag
GitHub Note Link: https://github.com/tc39/notes/blob/HEAD/meetings/2023-05/may-16.md#regexp-v-flag-for-stage-4

https://js-proposals.vercel.app/

Planning

- Requirements:
  - Present text from proposals
  - Proposal details need to be fetchable from Github
  - Visualize links between proposals
  - Use ChatGPT to assist
  - Create plots
  - Statistical analysis
  - Easily and programatically editable text
  - Easily (Read: quickly) built website

Right tools for the job

- Full stack website?
  - React
  - Django
  - DB
  - D3.js
  - Cytoscape.js
  - Server - nginx
- Pros:
  - Professional
  - Learning
  - Exciting
- Cons:
  - Time consuming
  - Skill issues?

Simpler solution?

Chosen tools

- Python
  - Easy and familiar
  - OpenAI API
- Obsidian
  - .md documents
  - Links between md
  - Graph
  - Free
  - Large community
  - Static page generators
    - Backup Obsidian Publish
    - Many open source solutions

- Requirements:
  - ✓ Present text from proposals
  - Proposal details need to be fetchable from Github
  - ✓ Visualize links between proposals
  - ✓ Use ChatGPT to assist
  - Create plots
  - Statistical analysis
  - ✓ Easily (Read: quickly) built website
  - Easily and programatically editable text

All tools have downsides

Step 1: Collect Data

"Collect the data and then you can do whatever you want with it"

- Web scraper? – Experimented with it but not ideal
- Github API – fetch Readme.md
  - Well documented
- Then I realized that I can just create .md files using Python and open in Obsidian!

Stage_4 > apiCall.py

```python
import requests
import base64

url = f"https://api.github.com/repos/tc39/proposals/contents/finished-proposals.md"

response = requests.get(url)
data = response.json()
file_content = base64.b64decode(data["content"]).decode("utf-8")

with open("Stage_4/outputMD/apiResponse.md","w") as contents:
    contents.write(file_content)

```

Chosen tools

- Python
  - Easy and familiar
  - OpenAI API
- Obsidian
  - .md documents
  - Links between md
  - Graph
  - Free
  - Static page generators
    - Backup Obsidian Publish
    - Open source Quartz

- Requirements:
  - ✓ Present text from proposals
  - ✓ Proposal details need to be fetchable from Github
  - ✓ Visualize links between proposals
  - ✓ Use ChatGPT to assist
  - Create plots
  - Statistical analysis
  - ✓ Easily (Read: quickly) built website
  - ✓ Easily and programatically editable text

Step 2: Parse API Response

- Plan:
  - Iterate through apiResp
  - Use string methods to e
  - All data is between “|”

Finished Proposals

Finished proposals are proposals that have reached stage 4, and are included in the latest draft of the specification.

Stage_4 > outputMD > apiResults.md


```
1  {'Title': 'Promise.try', 'Author(s)': 'Jordan Harband', 'Champion(s)': 'Jordan Harband', 'Date': 'October 2024', 'Link Titles': '[[try]]', 'GitHub Link': 'https://
   github.com/tc39/proposal-promise-try', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2024-10/october-09.md#promisetry-for-stage-4'}
2  {'Title': 'Sync Iterator helpers', 'Author(s)': 'Gus Caplan', 'Champion(s)': 'Michael Ficarra, Jonathan Keslin, Kevin Gibbons', 'Date': 'October 2024', 'Link
   Titles': '[[iterator-helpers]]', 'GitHub Link': 'https://github.com/tc39/proposal-iterator-helpers', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/
   meetings/2024-10/october-08.md#iterator-helpers-for-stage-4'}
3  {'Title': 'JSON Modules', 'Author(s)': 'Myles Borins, Sven Sauleau, Dan Clark, Daniel Ehrenberg', 'Champion(s)': 'Myles Borins, Sven Sauleau, Dan Clark, Daniel
   Ehrenberg', 'Date': 'October 2024', 'Link Titles': '[[json-modules]]', 'GitHub Link': 'https://github.com/tc39/proposal-json-modules', 'GitHub Note Link': 'https://
   github.com/tc39/notes/blob/HEAD/meetings/2024-10/october-08.md#import-attributes-and-json-modules-for-stage-4'}
4  {'Title': 'Import Attributes', 'Author(s)': 'Myles Borins, Sven Sauleau, Dan Clark, Daniel Ehrenberg', 'Champion(s)': 'Sven Sauleau, Dan Clark, Daniel Ehrenberg,
   Nicolò Ribaudo', 'Date': 'October 2024', 'Link Titles': '[[import-attributes]]', 'GitHub Link': 'https://github.com/tc39/proposal-import-attributes', 'GitHub Note
   Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2024-10/october-08.md#import-attributes-and-json-modules-for-stage-4'}
5  {'Title': 'RegExp Modifiers', 'Author(s)': 'Ron Buckton', 'Champion(s)': 'Ron Buckton', 'Date': 'October 2024', 'Link Titles': '[[regexp-modifiers]]', 'GitHub
   Link': 'https://github.com/tc39/proposal-regexp-modifiers', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2024-10/october-08.
   md#regexp-modifiers-for-stage-4'}
6  {'Title': 'New Set methods', 'Author(s)': 'Michał Wadas, Sathya Gunasekaran, Kevin Gibbons', 'Champion(s)': 'Kevin Gibbons', 'Date': 'April 2024', 'Link Titles': '
   [[set-methods]]', 'GitHub Link': 'https://github.com/tc39/proposal-set-methods', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2024-04/
   april-08.md#set-methods-for-stage-4'}
7  {'Title': 'Duplicate named capture groups', 'Author(s)': 'Kevin Gibbons', 'Champion(s)': 'Kevin Gibbons', 'Date': 'April 2024', 'Link Titles': '
   [[named-capture-groups]]', 'GitHub Link': 'https://github.com/tc39/proposal-duplicate-named-capturing-groups', 'GitHub Note Link': 'https://github.com/tc39/notes/
   blob/HEAD/meetings/2024-04/april-08.md#duplicate-named-capture-groups-for-stage-4'}
8  {'Title': 'ArrayBuffer transfer', 'Author(s)': 'Shu-yu Guo, Jordan Harband, Yagiz Nizipli', 'Champion(s)': 'Shu-yu Guo, Jordan Harband, Yagiz Nizipli', 'Date':
   'February 2024', 'Link Titles': '[[arraybuffer-transfer]]', 'GitHub Link': 'https://github.com/tc39/proposal-arraybuffer-transfer', 'GitHub Note Link': 'https://
   github.com/tc39/notes/blob/HEAD/meetings/2024-02/feb-6.md#arraybuffer-transfer-for-stage-4'}
9  {'Title': 'Promise.withResolvers', 'Author(s)': 'Peter Klecha', 'Champion(s)': 'Peter Klecha', 'Date': 'sub> - [2023-11', 'Link Titles': '[[promise-defer]]',
   'GitHub Link': 'https://github.com/tc39/proposal-promise-with-resolvers', 'GitHub Note Link': None}
10 {'Title': 'Array Grouping', 'Author(s)': 'Justin Ridgewell', 'Champion(s)': 'Justin Ridgewell, Jordan Harband', 'Date': 'November 2023', 'Link Titles': '
   [[array-grouping]]', 'GitHub Link': 'https://github.com/tc39/proposal-array-grouping', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2023-11/
   november-27.md#array-grouping-for-stage-4'}
11 {'Title': 'Resizable and growable ArrayBuffers', 'Author(s)': 'Shu-yu Guo', 'Champion(s)': 'Shu-yu Guo', 'Date': 'September 2023', 'Link Titles': '[[resizable]]',
   'GitHub Link': 'https://github.com/tc39/proposal-resizablearraybuffer', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2023-09/september-26.
   md#resizable-buffers-for-stage-4'}
12 {'Title': 'RegExp v flag with set notation + properties of strings', 'Author(s)': 'Markus Scherer, Mathias Bynens', 'Champion(s)': 'Mathias Bynens', 'Date': 'May
   2023', 'Link Titles': '[[regexp-v-flag]]', 'GitHub Link': 'https://github.com/tc39/proposal-regexp-v-flag', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/
   HEAD/meetings/2023-05/may-16.md#regexp-v-flag-for-stage-4'}
13 {'Title': 'Atomics.waitAsync', 'Author(s)': 'Lars Hansen', 'Champion(s)': 'Shu-yu Guo, Lars Hansen', 'Date': 'May 2023', 'Link Titles': '[[nonblocking]]', 'GitHub
   Link': 'https://github.com/tc39/proposal-atomics-wait-async', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2023-05/may-15.
   md#atomicswaitasync-for-stage-4'}
14 {'Title': 'Well-Formed Unicode Strings', 'Author(s)': 'Guy Bedford, Bradley Farias', 'Champion(s)': 'Guy Bedford, Bradley Farias, Michael Ficarra', 'Date': 'May
   2023', 'Link Titles': '[[usv-string]]', 'GitHub Link': 'https://github.com/tc39/proposal-is-usv-string', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/
   HEAD/meetings/2023-05/may-15.md#well-formed-unicode-strings-for-stage-4'}
```


Stage_4 > extractFromApiResponse.py

```python
# Import extractTitle from processApiResponse
from sharedMethods.processApiResponse import prepTe

# Open and read the file content
with open("Stage_4/outputMD/apiResponse.md", "r") as
    fileContent = file.read()

rawText = prepText(fileContent)

extractResults = extractDetails(rawText, fileContent)

with open("Stage_4/outputMD/apiResults.md", "w") as results:
    for each in extractResults:
        if "error with this link:" not in each:
            results.write(str(each) + "\n")
```

```python
text = textRaw

return text
```

```python
    proposals.append({
        "Title": titles[i],
        "Author(s)": authors[i],
        "Champion(s)": champions[i],
        "Date": dates[i],
        "Link Titles": "[["+linkTitles[i]+"]]",
        "GitHub Link": links[i],
        "GitHub Note Link": proposalNoteLinks[i]})

#first line contains table title and line. remove this

return proposals
```

Stage 2, 2.7, and 3

- All in same Readme.md
- Same structure with extra step
- Inconsistencies
- Debugging and logging

ECMAScript proposals

- Stage 1 Proposals
- Stage 0 Proposals
- Finished Proposals
- Inactive Proposals

ECMAScript Internationalization API Specification proposals

Contributing to proposals

Active proposals

Proposals follow this process document. This list contains only stage 2 proposals and higher that have not yet
expects these features to

ature Flag	Last Presented
	May 2017

```python
def delegateDetails(fileContent):

    global stage3Extract
    global stage2_7Extract
    global stage2Extract

    stage3Title = fileContent.index("### Stage 3\n")
    stage2_7Title = fileContent.index("### Stage 2.7\n")
    stage2Title = fileContent.index("### Stage 2\n")
    nextTitle = fileContent.index("## Contributing to proposals\n")

    links = fileContent[nextTitle:]

    stage3 = fileContent[stage3Title:stage2_7Title]
    stage3Extract.append(stage3)
    stage3Extract.append(links)

    stage2_7 = fileContent[stage2_7Title:stage2Title]
    stage2_7Extract.append(stage2_7)
    stage2_7Extract.append(links)

    stage2 = fileContent[stage2Title:nextTitle]
    stage2Extract.append(stage2)
    stage2Extract.append(links)

    with open("Stage_2_2_7_3/Stage_3/outputMD/delegatedApiResponse.md", "w") as results:
        for each in stage3Extract:
            if "error with this link:" not in each:
                results.write(str(each) + "\n")

    with open("Stage_2_2_7_3/Stage_2_7/outputMD/delegatedApiResponse.md", "w") as results:
        for each in stage2_7Extract:
            if "error with this link:" not in each:
                results.write(str(each) + "\n")

    with open("Stage_2_2_7_3/Stage_2/outputMD/delegatedApiResponse.md", "w") as results:
        for each in stage2Extract:
            if "error with this link:" not in each:
                results.write(str(each) + "\n")
```

Status

- Able to extract proposal data from Readme.md’s from https://github.com/tc39/proposals for all stages
- API response is processed and saved in Stage_X/apiResults.md
  - {'Title': 'Promise.try', 'Author(s)': 'Jordan Harband', 'Champion(s)': 'Jordan Harband', 'Date': 'October 2024', 'Link Titles': '[[try]]', 'GitHub Link': 'https://github.com/tc39/proposal-promise-try', 'GitHub Note Link': 'https://github.com/tc39/notes/blob/HEAD/meetings/2024-10/october-09.md#promisetry-for-stage-4'}
- Next step is to create md files per proposal in Obsidian

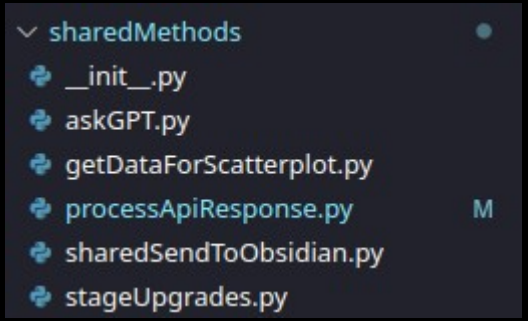
Lessons

- Using Github API
- Importance of logging to debug
- Be careful with Try-catch
- Remember Inf101
- I don’t know regex
- Lots of repeated code

Refactoring

- Code is beginning to be unmaintainable
- DRY principle
- “sharedMethods”
  - Refactor code to use shared utility methods
  - Import methods

```
# Import extractTitle from processApiResponse
from sharedMethods.processApiResponse import prepText, extractDetails
```



Lessons

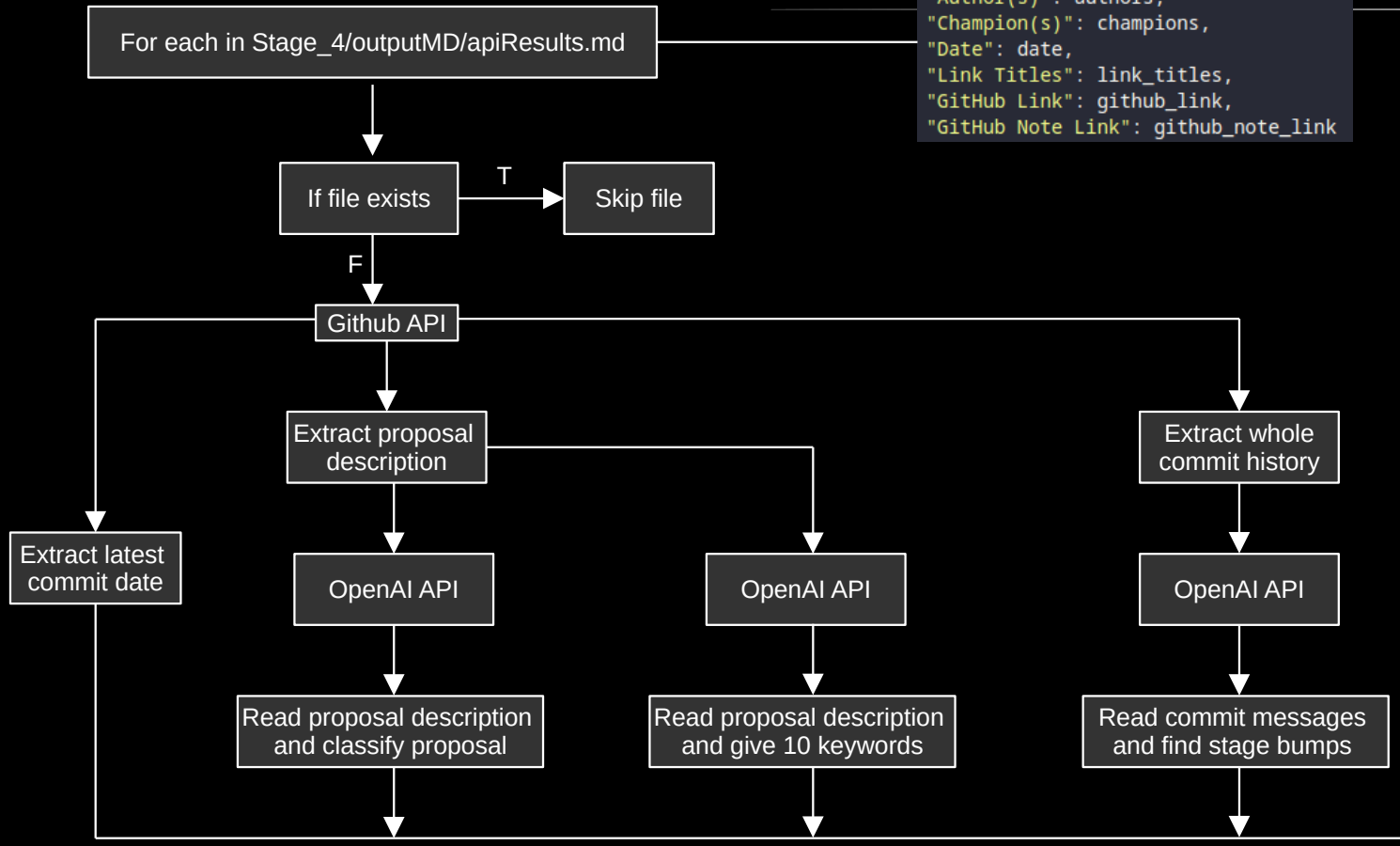
- Using Github API
- Importance of logging to debug
- Be careful with Try-catch
- String handling practice
- Remember Inf101
- I don’t know regex
- ~~Lots of repeated code~~
- DRY principle

Step 3: Create files in Obsidian

- Iterate through and create .md per entry
  - Extract fields from apiResults.md
  - For each link per stage
    - Step 3.1: Extract proposal description
    - Step 3.2: Extract last commit date
    - Step 3.3: Extract stage bumps
    - Step 3.4: Generate 10 keywords
    - Step 3.5: Create md document for each proposal

sendToObsidian('Stage 4', 'Stage_4')

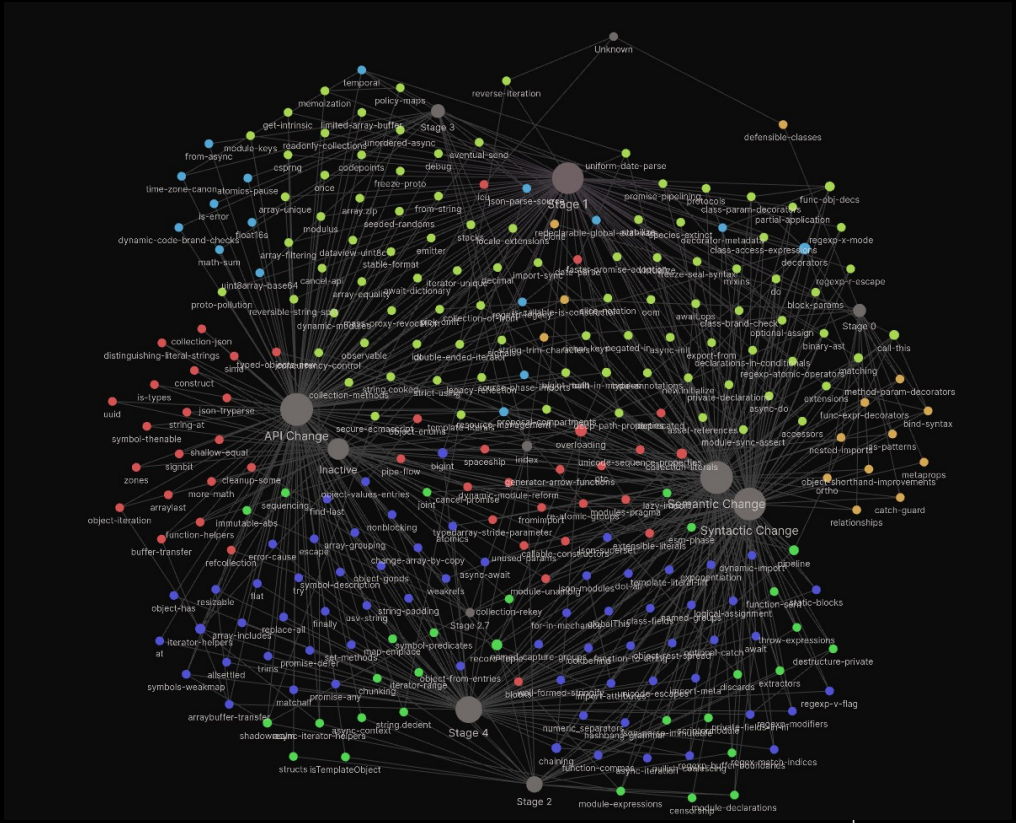
```
"Title": title,
"Author(s)": authors,
"Champion(s)": champions,
"Date": date,
"Link Titles": link_titles,
"GitHub Link": github_link,
"GitHub Note Link": github_note_link
```




```
f"[[{obsidianFile}]]<br>"
f"Classification: {classification}<br>"
f"Human Validated: No<br>"
f"Title: {title}<br>"
f"Authors: {authors}<br>"
f"Champions: {champions}<br>"
f"Last Presented: {date}<br>"
f"Stage Upgrades:<br>{stageUpgrades}<br>"
f"Last Commit: {returnDate}<br>"
f"Keywords: {keywords}<br>"
f"GitHub Link: {github_link} <br>"
f"GitHub Note Link: {github_note_link}\n"
f"# Proposal Description:\n{file_content}<br>"
```

Status

- DRY and maintainable code
- Successfully collected data
  - Classification
  - Title
  - Authors
  - Champions
  - Last presented date
  - Stage bump dates
  - Last Commit date
  - Keywords
  - Github Link
  - Last meeting link
  - Project description
- Md files that are editable in Obsidian with interactive graph
- TODO: ChatGPT assisted but need to verify



Lessons

- Importance
- Github API
- Pagination
- Extracting proposals)
- OpenAI AP
- Usefulness
- Frustrating
  - Main v
  - READI
- Terrible co

```python
try:
    apiProposalName = github_link.split("/")[-1]
    if "#" in apiProposalName:
        apiProposalName = apiProposalName.split("#")[0]
except:
    print("Error with link:", link_title)

try:
    # default branch called main
    try:
        commitDate = f"https://api.github.com/repos/tc39/{apiProposalName}/branches/main"

        commitDateResponse = requests.get(commitDate, auth=(os.getenv("USERNAME"), os.getenv("API_KEY")))
        commitDate = commitDateResponse.json()
        commitDateIso = commitDate["commit"]["commit"]["author"]["date"]
        commitDate = commitDateIso.split("T")
        returnDate = commitDate[0]

    # default branch called master
    except:
        commitDate = f"https://api.github.com/repos/tc39/{apiProposalName}/branches/master"

        commitDateResponse = requests.get(commitDate, auth=(os.getenv("USERNAME"), os.getenv("API_KEY")))
        commitDate = commitDateResponse.json()
        commitDateIso = commitDate["commit"]["commit"]["author"]["date"]
        commitDate = commitDateIso.split("T")
        returnDate = commitDate[0]
except:
    returnDate = None

try:
    stageUpgrades = getStageUpgrades(github_link).strip()
    print(stageUpgrades)
except:
    stageUpgrades = None
    print("error with getStageUpgrade for", github_link)
```

Step 4: Verify ChatGPT’s work

- Including
  - Classification
  - Stage bump dates
  - Keywords

- Painstaking manual labor
  - ca. 4-5 days to go through everything
  - Revised about 5 times

Classifications

The changes we are classifying are the following:

- API Change: Modifies or introduces new functions, objects, or methods in the standard library. These changes do not affect the syntax of the language but add new functionality to existing features.
- Semantic Change: Changes the meaning of the JavaScript code even if the syntax remains the same. These changes can alter the behavior of existing JavaScript programs in subtle or breaking ways. Usually involves modifying execution rules rather than introducing new syntax.
- Syntactic Change: Introduces new syntax or modifies existing syntax rules. Usually involves new keywords, operators, or expressions. These changes often require updates to parsers and affect how JavaScript code is written.

- Any combination of the three

Verifying classifications

```
systemPrompt = """
    I am conducting research on proposals for ECMAScript and have exported proposal descriptions from the GitHub repositories for each proposal.

    I am sending you the proposal description and I want you to look at it and classify the type of proposal it is. I want to classify them as API changes, Semantic changes, or Syntactic changes.

    ### Change Definitions:

    - **API Change**: Modifies or introduces new built-in functions, objects, or methods in the standard library. These changes do not affect the syntax of the language but add new functionality to existing features.

    - **Semantic Change**: Changes the meaning of the JavaScript code even if the syntax remains the same. These changes can alter the behavior of existing JavaScript programs in subtle or breaking ways. Usually involves modifying execution rules rather than introducing new syntax.

    - **Syntactic Change**: Introduces new syntax or modifies existing syntax rules. Usually involves new keywords, operators, or expressions. These changes often require updates to parsers and affect how JavaScript code is written.

    ### CRITICAL: RESPONSE FORMAT ###

    After classifying the proposal, return **ONLY** one of the following strings: - [[API Change]] - [[Semantic Change]] - [[Syntactic Change]] If multiple classifications apply, return a **space-separated string** like:

    - [[API Change]] [[Syntactic Change]]

    DO NOT return any explanations, markdown, or extra text. Just the classification string.
    """
```

Verifying classifications

Challenges:

- Reading and understanding proposals
  - Main purpose?
- Reading spec-texts
  - Strictly judge based on spec-text?
- Overlapping classifications

Lesson:

- Proposals and spec-texts are complex
- Almost all syntactic changes need to be backed up by semantic change
- Most proposals are a combination of synactic/semantic/API change
- Open to interpretation
- We agreed to focus on the main purpose otherwise all proposals will be syntactic + semantic

Verifying stage bumps

systemPrompt = """

I am conducting research into ECMAScript proposals and I want to look at the timeline of the commits for each proposal. I have extracted the commit history for each proposal and have filtered it down to commit messages, authors, and dates for when the proposal upgraded stage.

I am sending you this data and I want you to take a look at the commit history and return to me a list of when the stage upgrade happened in this format:

Stage 1: *insert date*

Stage 2: *insert date*

Stage 2.7: *insert date*

Stage 3: *insert date*

Stage 4: *insert date*

Since I will be inserting these dates into a md file, please only inlcude the information I have asked for above.

The date for stage 1 should be the earliest commit in the history unless explicitly stated.

If there is no mention of the specific proposal being upgraded, please fill in the date as NA instead. If there is more than one mention of a upgrade for a specific stage, use the earliest date.

Please and thank you in advance!

"""

Verifying stage bumps

- Idea: Stage bumps are a big deal so assume they would announce it in commit messages
- Turns out, ca. less than half the time especially at lower stages
- Went through all commit history
- GPT was correct if the stage bump was announced

Verifying keywords

```
systemPrompt = """

    I am conducting research on proposals for ECMAScript and have exported proposal descriptions from the GitHub repositories for each proposal.

    I am sending you the proposal description and I want you to look at it and and return to me 10 keywords that are relevant for this proposal.

    Please do not include generic, surface level keywords such as "TC39" or "ECMAScript" or "JavaScript".

    Please instead include keywords that are directly related to the contents of this proposal.

    Please keep the keywords lowercase.

    If the proposal is related to a data structure, please include the data structure amongst the keywords.

    Please return the keywords in this format with an # in front of the keyword without space: #keyword1 #keyword2 #keyword3 etc...

    The correct formatting is CRITICAL. Keep the keywords one word and NEVER EVER use short hand notation. Instead use the full word: for example, use asynchronous instead of async and synchronous instead of sync. If you must to use more than one word per keyword, use an underscore between the words like such: #this_keyword.

    CRITICAL: Do NOT, I REPEAT, NEVER include the name OR PART OF THE NAME of the proposal or use the same words as is in the proposal as one of the keywords!

    Please and thank you!

    """
```

Verifying keywords

- GPT was not very good at this task
  - Didnt follow prompt instructions
  - Words were not uniform:
    - Eg: “async” vs “Asynchronous” vs “Async”
  - GPT used words like “ECMAScript”, “Proposal”, “JavaScript”
- Task was to make the keywords uniform as much as possible
- Task was to understand proposals at a deeper level and make connections to language constructs

326 Keywords

Rank	Keywords	n
1	#performance	32
2	#iterator	26
3	#asynchronous	22
4	#promise	22
5	#module	21
6	#regex	20
7	#array	19
8	#property	19
9	#class	18
10	#security	18
11	#string	17
12	#error_handling	15
13	#memory_management	15
14	#typedarray	12
15	#concurrency	10

Rank	Keywords	n
16	#arithmetic	9
17	#destructuring	8
18	#map	8
19	#numeric	8
20	#arraybuffer	7
21	#decorator	7
22	#json	7
23	#math	7
24	#unicode	7
25	#bigint	6
26	#generator	6
27	#global	6
28	#iterable	6
29	#key_value_pairs	6
30	#parse	6

Rank	Keywords	n
31	#realm	6
32	#symbol	6
33	#date_time	5
34	#encapsulation	5
35	#grammar	5
36	#metadata	5
37	#operator	5
38	#pattern_matching	5
39	#readability	5
40	#resource_management	5
41	#set	5
42	#string_manipulation	5
43	#synchronous	5
44	#wait	5
45	#accessor	4

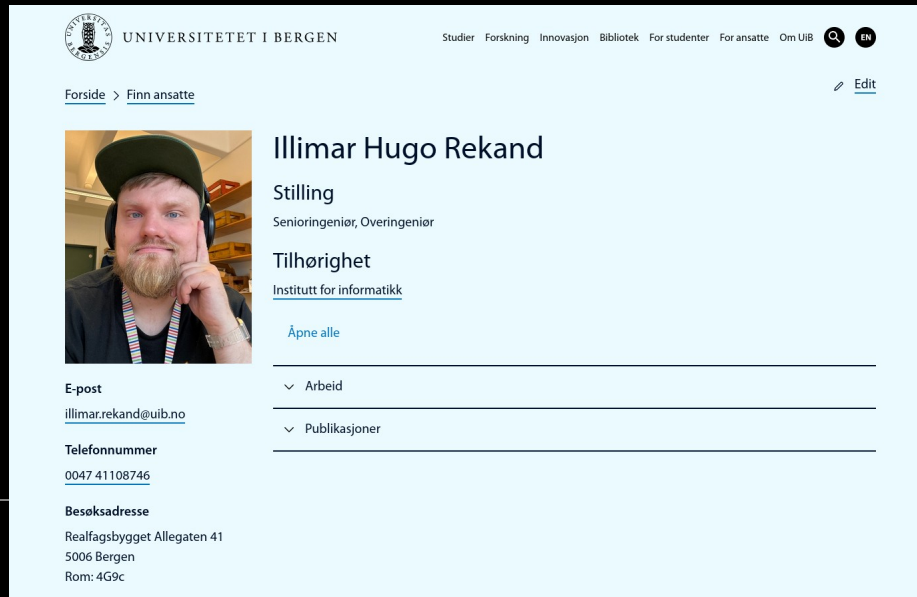
Topics

- Reduced to 20 Topics from 326 Keywords

Rank	Topics	Count
1	#others	281
2	#objects	131
3	#async	51
4	#arrays	47
5	#iterators	45
6	#modules	37
7	#numbers	36
8	#performance	32
9	#concurrency	31
10	#collections	25
11	#regex	25
12	#security	23
13	#memory	22
14	#intl	21
15	#functions	12
16	#types	11
17	#realms	9
18	#ergonomics	8
19	#json	6
20	#webassembly	2

Data Analysis

- R and Rstudio
- Assistance from Illimar Rekand
  - Crash course in R
    - Software carpentry
  - Advised to use GGPLOT2
    - Fantastic documention



https://swcarpentry.github.io/r-novice-gapminder/

https://r-graph-gallery.com/scatterplot.html

Chosen tools

- Python
  - Easy and familiar
  - OpenAI API
- Obsidian
  - .md documents
  - Links between md
  - Graph
  - Free
  - Static page generators
    - Backup Obsidian Publish
    - Open source Quartz

- Requirements:
  - ✓ Present text from proposals
  - ✓ Proposal details need to be fetchable from Github
  - ✓ Visualize links between proposals
  - ✓ Use ChatGPT to assist
  - ✓ Create plots
  - ✓ Statistical analysis
  - ✓ Easily (Read: quickly) built website
  - ✓ Easily and programatically editable text

Notes on Data Analysis

- Lack of database
- Rushed

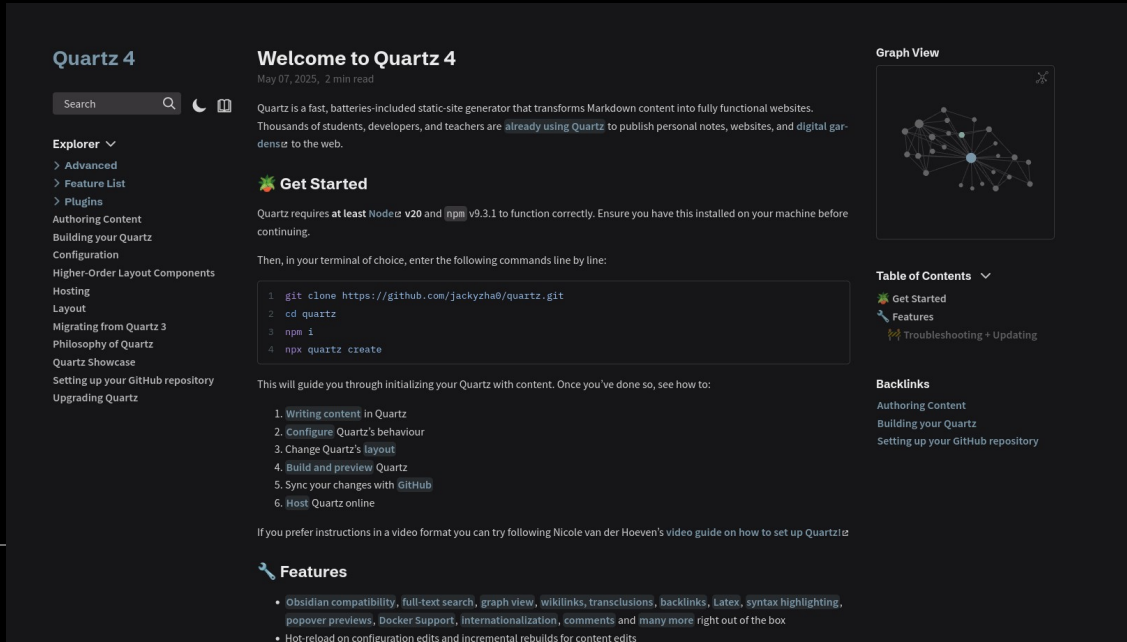
```
#
# Run this script like
#
# For extracting stage
# python sharedMethods
# python sharedMethods
# python sharedMethods
#
#
# Extract tags by:
# python sharedMethods/getDa
#
```

```
if __name__ == "__main__":

    if sys.argv[1] == "tags":
        getTags()
    elif sys.argv[1] == "updateTags":
        updateTags()
    elif len(sys.argv) == 3:
        if sys.argv[1] == "stage":
            stage = sys.argv[2]
            getStageBumpsAndLastCommit(stage)
        elif sys.argv[1] == "change":
            change = sys.argv[2]
            getClassifiedChanges(change)
    elif len(sys.argv) == 4:
        if sys.argv[1] == "noncrossover" and sys.argv[2] == "stage":
            stage = sys.argv[3]
            getNonCrossoverClassifiedChanges(stage)
    elif len(sys.argv) == 5:
        if sys.argv[1] == "change" and sys.argv[3] == "stage":
            change = sys.argv[2]
            stage = sys.argv[4]
            getStageSpecificClassifiedChanges(change,stage)

#
```

```
acky but on a time crunch!
t:
ch:
each:
ist.append(each)

semanic AND syntactic):
```

Website

- Tried several open source solutions
- Decided on Quartz: https://quartz.jzhao.xyz/
- Excellent documentation
- Hosted on Vercel



My website

- Demonstration

https://js-proposals.vercel.app/

Proposed change to process documents

- Presented results to TC39-TG5 on April 30
- Proposed change:

```
144            <li>Prose outlining the problem/need and the general shape of a solution
145            <li>Discussion of key algorithms, abstractions, and semantics
146            <li>Identification of potential cross-cutting concerns and implementation
    challenges/complexity
147  +         <li>Identification of topics that characterize the proposal, to support categorization,
    discoverability, and alignment with related proposals and areas of the language
148            <li>A publicly available repository for the proposal that captures the above
    requirements
149          </ul>
150        </td>
```

My Take Home Lessons

- Learnt a lot about JavaScript
  - “The more you know, the more you realize you don’t know”
- Logging
- DRY code
- Good commit messages
- Good documentation
- Limit refactoring
- Programming is a muscle

About TC39

- Dedicated and competent people
- Organized
- Cooperative