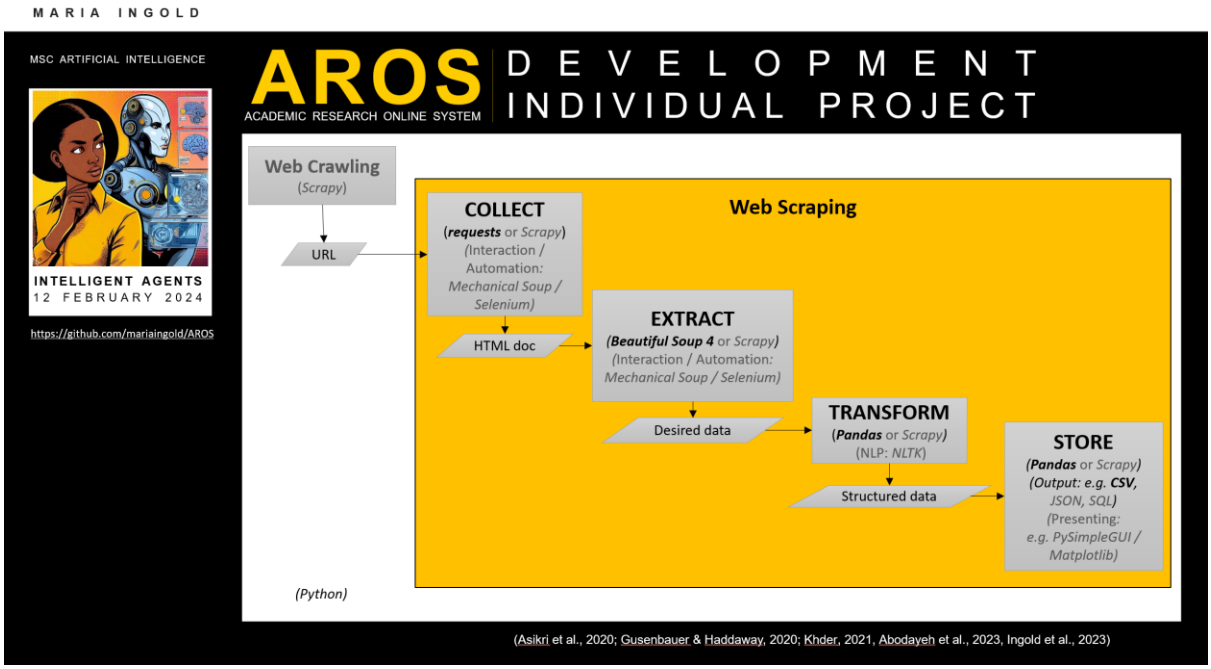
Academic Research Online System (AROS) Development Individual Project: Presentation Transcript

by Maria Ingold

12 February 2024

INTRODUCTION



1 | SLIDE 1

[SLIDE 1]

Academic research requires what is known as ‘robust evidence synthesis’—which means comprehensive evidence been gathered. But that data is spread across at least 28 academic search systems (Gusenbauer & Haddaway, 2020). ‘Comprehensive’ requires searching *all* the research.

Using a simple reflex agent, Academic Research Online System (AROS) is a first step towards that. This project is based on our AROS team report (Ingold et al., 2024).

At this stage, AROS is very simple. It only focuses on Web Scraping (Khder, 2021).

It is built in Python because Python is well supported and readable, with a range of libraries.

To evaluate libraries, I wrote two very basic scripts during the team report—covering all four stages—collect, extract, transform, and store. (These are in my e-Portfolio.) I

decided not to proceed with Scrapy, even though it is one library, faster and for medium-scale projects. I selected *requests*, *Beautiful Soup 4,* and *Pandas* as they are more suitable for beginners—which I am—and small projects—which this is (Abodayeh et al., 2023).

AROS V0.1

I refer to that early prototype script as V0.1 (Nigam & Biswas, 2021; Amos, 2022).

I started with code from a Nigam & Biswas (2021) research paper. The first thing I discovered was that it did not work. I used Beautiful Soup to print the page text. While the code in their paper obviously once worked, Yahoo now returns an error page.

'We'll be right back…,' it says. (Nothing Terminator about that.)

Other sites behaved similarly.

[SLIDE 2]

I finally found that arXiv.org returned its web page. This specific page is one of my test pages. Note, that it is on 'web scraping'.

[SLIDE 2-1]

By looking at the source code I was able to identify citation data.

[SLIDE 2-2]

This seemed like a simple, useful starting point.

[SLIDE 2-3]

Version 0.1 simply parses a hardcoded page. It takes the citation metadata within the page—title, multiple authors, and date…

[SLIDE 2-4]

…and writes it to a CSV (Comma Separated Values) file.

It works. But it is very much a prototype.

AROS V0.2

[SLIDE 3]

I am a strategy and innovation CTO (Chief Technology Officer). I last programmed 20 years ago. So, I have kept this simple. But there were several areas I wanted to address in my second version.

Firstly, to increase modularity—in other words, logically separate each task and simplify maintenance—I moved from a script to a class. I was initially going to create four classes. However, I discovered that a primary tenet of Python is chose simplicity over complexity, especially with classes with a single method (Ramos, 2023). So, I

created an arXiv scraper class with four methods instead—collect, extract, transform and store.

[SLIDE 3-1]

Secondly, to improve flexibility, I eliminated my hardcoded URL (Uniform Resource Locator) and passed it as a parameter. This is used by Requests to get the page content (PyPi, N.D.; Nigam & Biswas, 2021; Amos, 2022; Abodayeh et al., 2023).

[SLIDE 3-2]

Thirdly, to better facilitate search in my next version, I also extracted abstract. This is in addition to the title, authors, and date—all extracted by Beautiful Soup 4 (Khder, 2021; Amos, 2022; Abodayeh et al., 2023).

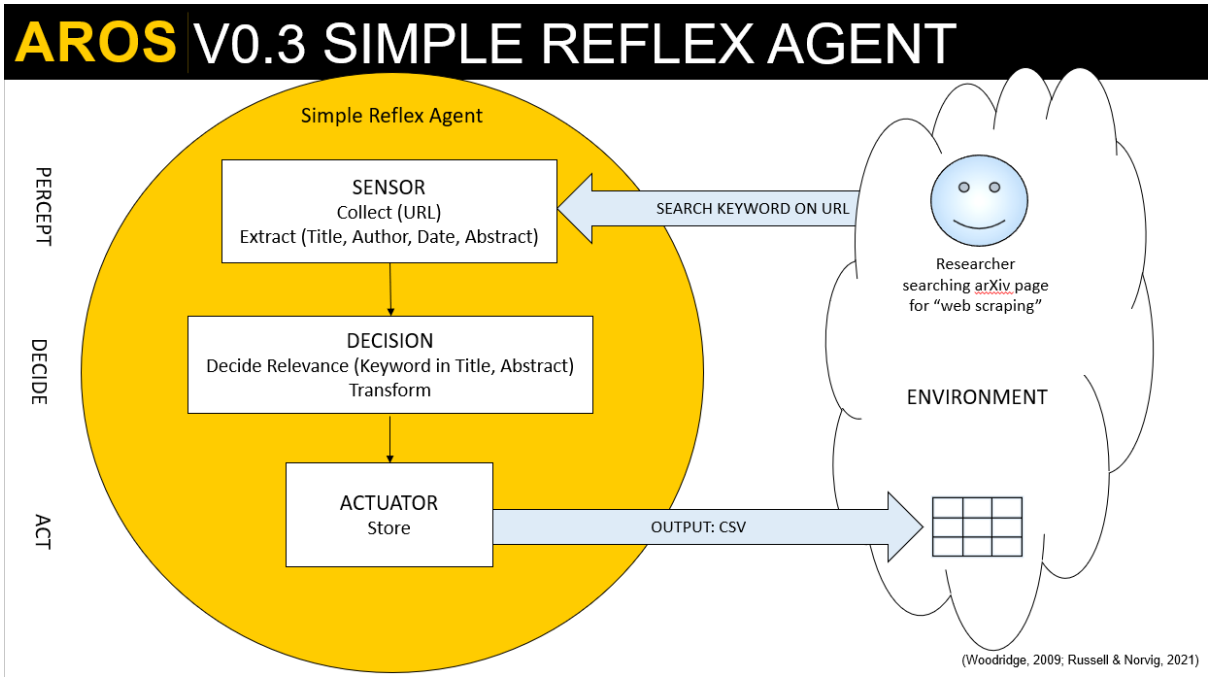
[SLIDE 3-3]

It transforms the data into a Python dictionary, converts it to a Pandas DataFrame and uses Pandas to store it as a CSV file (Nigam & Biswas, 2021; Pandas, 2024).

[SLIDE 3-4]

I tested version 0.2's core functionality along the way with print statements to verify output and by successfully outputting the data to a CSV file.

AROS V0.3



2 | SLIDE 4

[SLIDE 4]

To make the class a simple reflex agent, it needs to receive sensory data from an environment, decide what to do, then act on the environment (Wooldridge, 2009; Russell & Norvig, 2021).

The third version of AROS receives a URL and a keyword (or phrase), collects the URL, and extracts the required data. That is the *sensor*.

It then *decides* if that keyword is present in the title or abstract, and if so, transforms the data.

It *acts* on the environment by storing the result in a CSV file (Abodayeh et al., 2023).

[SLIDE 5]

The main changes to the code are adding a decide_relevance method and a run method.

decide_relevance uses the delightfully simple membership operator 'in' to find a substring—that is it finds the keyword in either the title or abstract that was just added in version 0.2. (Stack Overflow, 2016; Breuss, 2022).

run then collects and extracts the citation data, tries to find the keyword and…

[SLIDE 5-1]

…if the paper is relevant, notifies the researcher and writes the citation data to a CSV file.

[SLIDE 5-2]

Otherwise, it states that it is not relevant and does not store the file. In version 0.3, this occurs for both if it is not relevant and if the page is not found. Version 0.4 addresses this.

TESTING

AROS TESTING

TEST TYPE	WHAT IT CHECKS	HOW I USED IT
Unit Tests	Do individual components (methods) work correctly?	Manual print statements Automated pytest
Integration Tests	Do modules work together as expected? Verifies they can.	Manual print statements and CSV
Functional Tests	Does it deliver the requirements? Verifies the result.	Manual print statements and CSV
System (End-to-End) Tests	Does it deliver expected holistic user flows?	Very simple so covered above.
User Acceptance Tests	Does it formally pass user acceptance?	Very simple so covered above.
Performance (Stress) Tests	Is the system, responsive, stable and scalable?	Only tested on local PC. Effective for exercise. Libraries meant for small systems.
Security Tests	Is the system secure?	Not done.
Compatibility Tests	Does the system work on different platforms?	Only tested on Windows 10 PC
Smoke Tests	Have changes affected stability?	Manual print statements and CSV
Regression Tests	Have changes affected functionality?	Manual print statements and CSV

(Pittet, N.D. b; Aqab & Tariq, 2020)

3 | SLIDE 6

[SLIDE 6]

Before I get on to the final version, I want to talk about testing.

I used a PRINCE2 Agile approach—I thought through the big picture in a planning stage for our team report—and implemented it iteratively for this individual project. I got each component to work, and I slowly built the functionality.

As you have seen, I first performed basic unit testing—does each component work correctly? I tested every step with print statements and ran the code after every change (Pittet, N.D. b; Aqab & Tariq, 2020).

Similarly, with integration testing—do the modules work together as expected?

And functional testing—does it deliver my requirements—my print statements and the CSV file?

At each version I had to perform a basic smoke and regression test—did it still work as expected?

But this was all manual. This was decidedly not DevOps CI/CD—continuous integration, delivery, and deployment (Pittet, N.D. a). To get closer to that I needed two key things: version control—I am using GitHub—and automation (Atlassian, N.D.).

V0.4 PYTEST UNIT TESTING

[SLIDE 7]

Automated unit testing in Python is provided by unittest and pytest (Microsoft, N.D.). In version 0.4, I selected pytest, because it worked, and it is more succinct (Pytest, N.D.).

This is the pytest code using a good arXiv URL.

Each test tests a method in the ArxivScraper class. These are basic unit tests. Does the method work correctly, rather than a comprehensive functional test.

These tests can be run all at once, or one at a time in any order. As you can see by the 100% and green tick boxes, they all passed.

However, pytest raises a deprecation warning in one of the libraries. I have updated all the libraries to the latest versions, and it still appears. My guess is the library—possibly Pandas—has not been updated yet.

One other thing you can just see here is that I have created the potential to pass multiple URLs in Version 0.4, not just a single URL.

AROS V0.4

[Start demo]

This is AROS version 0.4. I have added the ability to input multiple URLs. You can add more here.

In the run command, you can add the keyword (or phrase) and the output filename to store any relevant search results.

I have also added some basic error handling around the URL (W3 Schools, no date).

If I run AROS…

[run]

…the first URL will fail because it is not an arXiv URL.

The second will fail because it is incomplete.

The third and fourth will either find the provided keyword relevant or not relevant.

If the keyword is relevant, the citation—title, authors, date and abstract—are displayed and written to a CSV file.

The third URL will find ‘artificial intelligence’ and the fourth will find ‘web scraping’.

Errors and explanations are provided if the phrases are not found.

[dir]

[Get-Date]

You can see the CSV files have been written to the current folder.

[more citation_data_v0.4-1.csv]

[more citation_data_v0.4-2.csv]

They are in the same format you have seen before.

[Close panel]

[Scroll up]

This is the code:

- The libraries

[Scroll down…]

- The ArxivScraper class
- Initialisation
- Collecting—get the page content or return an error—using requests
- Extract the citation title, authors, date, and abstract—using Beautiful Soup 4—and store it in a Python dictionary
- Transform the dictionary to a Pandas DataFrame

- And use Pandas to store the DataFrame in a CSV file
- Relevance is decided by finding the keyword in the title or abstract
- And if it is a valid URL, run prints the results, or notes if it is not relevant.

There we are…AROS version 0.4!

FUTURE DIRECTIONS

[SLIDE 9]

Versions 0.1 through 0.4 look like this. To move towards a production-ready version 1.0 requires a range of other actions.

Technically…

[SLIDE 9-1]

For scalability, performance, and medium-sized projects, Scrapy is better. It can also handle web-crawling pages instead of a URL list and can schedule crawling during non-peak hours or rate limit. By default, the latest versions of Scrapy respect robots.txt policies (Scrapy, N.D.; Asikri, Krit & Chaib, 2020).

[SLIDE 9-2]

Mechanical Soup and Selenium can provide interaction and automated browsing. Beautiful Soup handles small static HTML (HyperText Markup Language) sites, but Selenium is required for dynamic web pages with JavaScript (Morina & Sejdiu, 2022).

[SLIDE 9-3]

To become a more sophisticated intelligent agent, search relevance can be increased with natural language processing (NLP) using the *Natural Language Toolkit (NLTK)* (Van Dijk, et al., 2019).

[SLIDE 9-4]

Storing it in a database instead of a CSV file would enable persistent storage and permit more complex queries and analysis.

[SLIDE 9-5]

Of course, a graphical user interface to enter search terms and view results would improve the experience.

As the service extends, more error handling will be required. For robustness, more tests for performance, security and compatibility will also be needed. Automated testing, where possible, will help create a CI/CD workflow.

For ethical and legal compliance, Scrapy has more built-in capabilities, like RobotsTxtMiddleware to respect robots.txt. However, it does not address all the issues (Scrapy, N.D.).

It might be worth exploring Common Crawl, a free open web crawl data repository containing over 3 billion compliant pages. It means pages would not be hit unnecessarily, which would also improve sustainability (Turski et al., 2023).

CONCLUSION AND CODE

In conclusion, we have discussed how—over four versions—I have built up from a simple web scraping script to a modular simple reflex agent which finds a keyword (or phrase) in arXiv citation data.

Several URLs were tested. Errors were caught and reported back to the user. Title and abstract citation data from valid arXiv URLs were checked against the keyword(s) for relevance. And relevant citations were displayed and written to a CSV file.

As a step towards automated testing, pytest was used for unit testing.

Moving towards a 1.0 product would require moving to Scrapy, evolving with NLTK, and considering ethical, legal, and sustainable approaches through Scrapy and perhaps Common Crawl.

The code is available in GitHub. Please check it out! Thank you.

https://github.com/mariaingold/AROS

REFERENCES

Abodayeh, A. et al. (2023) Web Scraping for Data Analytics: A BeautifulSoup Implementation, *Proceedings - 2023 6th International Conference of Women in Data Science at Prince Sultan University, WiDS-PSU 2023* 65–69. DOI: https://doi.org/10.1109/WIDS-PSU57071.2023.00025.

Amos, D. (2022) *A Practical Introduction to Web Scraping in Python – Real Python*. Available from: https://realpython.com/python-web-scraping-practical-introduction/ [Accessed 10 December 2023].

Aqab, S. and Tariq, M.U. (2020) Handwriting Recognition using Artificial Intelligence Neural Network and Image Processing, *IJACSA) International Journal of Advanced Computer Science and Applications* 11(7). DOI: https://doi.org/10.14569/IJACSA.2020.0110719.

Asikri, M.E., Krit, S. & Chaib, H. (2020) Using Web Scraping In A Knowledge Environment To Build Ontologies Using Python And Scrapy, *European Journal of Translational and Clinical Medicine* 7(3): 433–442. Available from: https://www.researchgate.net/publication/346215371_Using_Web_Scraping_In_A_Knowledge_Environment_To_Build_Ontologies_Using_Python_And_Scrapy [Accessed 10 December 2023].

Atlassian (N.D.) *DevOps CI CD Tutorials*, *Atlassian*. Available from: https://www.atlassian.com/devops/continuous-delivery-tutorials [Accessed 6 February 2024].

Breuss, M. (2022) *How to Check if a Python String Contains a Substring*. Available from: https://realpython.com/python-string-contains-substring/ [Accessed 5 February 2024].

Van Dijk, R.H.W., Staut, N. & Wolfs, C.J.A. (2019) Automated Sentiment Analysis of Text Data with NLTK, *Journal of Physics: Conference Series* 1187(5): 052020. DOI: https://doi.org/10.1088/1742-6596/1187/5/052020.

Gusenbauer, M. & Haddaway, N.R. (2020) Which academic search systems are suitable for systematic reviews or meta-analyses? Evaluating retrieval qualities of Google Scholar, PubMed, and 26 other resources, *Research Synthesis Methods* 11(2): 181–217. DOI: https://doi.org/10.1002/JRSM.1378.

Ingold, M., Grimal, C. and Alaskar, A. (2023) *Intelligent Agents: Development Team Project Report*. Available from: https://mariaingold.github.io/artefacts/2_IA_Team1_DevelopmentTeamProjectReport.pdf [Accessed 3 February 2024].

Khder, M.A. (2021) Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application, *Int J Advance Soft Compu Appl* 13(3). DOI: https://doi.org/10.15849/IJASCA.211128.11.

Microsoft (N.D.) *Python Testing in Visual Studio Code*. Available from: https://code.visualstudio.com/docs/python/testing#_tests-in-pytest [Accessed 6 February 2024].

Morina, V. & Sejdiu, S. (2022) Evaluating and comparing web scraping tools and techniques for data collection., *UBT Knowledge Center* 53–57. Available from: https://www.researchgate.net/publication/369114323_Evaluating_and_comparing_web_scraping_tools_and_techniques_for_data_collection [Accessed 29 November 2023].

Nigam, H. & Biswas, P. (2021) Web Scraping: From Tools to Related Legislation and Implementation Using Python, *Innovative Data Communication Technologies and Application* 149–164. DOI: https://doi.org/10.1007/978-981-15-9651-3_13.

Pandas (2024) *Pandas Documentation*. Available from: https://pandas.pydata.org/docs/index.html [Accessed 4 February 2024].

Pittet, S. (N.D. a) *Continuous integration vs. delivery vs. deployment*, *Atlassian*. Available from: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment [Accessed 6 February 2024].

Pittet, S. (N.D. b) *The different types of testing in software*, *Atlassian*. Available from: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing [Accessed 6 February 2024].

PyPi (N.D.) *Requests 2.31.0*. Available from: https://pypi.org/project/requests/ [Accessed 4 February 2024].

Pytest (N.D.) *How to use temporary directories and files in tests*. Available from: https://docs.pytest.org/en/7.1.x/how-to/tmp_path.html [Accessed 6 February 2024].

W3 Schools (N.D.) *Python Try Except*. Available from: https://www.w3schools.com/python/python_try_except.asp [Accessed 7 February 2024].

Ramos, L.P. (2023) *Python Classes: The Power of Object-Oriented Programming – Real Python*. Available from: https://realpython.com/python-classes/#getting-started-with-python-classes [Accessed 4 February 2024].

Russell, S. & Norvig, P. (2021) *Artificial Intelligence: A Modern Approach, Global Edition*. 4th ed. Pearson Education, Limited.

Scrapy (N.D.) *Scrapy 2.11 Documentation*. Available from: https://docs.scrapy.org/en/latest/index.html [Accessed 7 February 2024].

Stack Overflow (2016) *Python : Web Scraping Specific Keywords*. Available from: https://stackoverflow.com/questions/40121232/python-web-scraping-specific-keywords [Accessed 5 February 2024].

Turski, M. et al. (2023) CCpdf: Building a High Quality Corpus for Visually Rich Documents from Web Crawl Data. Available from: https://arxiv.org/abs/2304.14953 [Accessed 8 February 2024].

Wooldridge, M. (2009) *An Introduction to MultiAgent Systems*. Chichester, UK: John Wiley & Sons.