

symbex: search Python code for functions and classes, then pipe them into a LLM

18th June 2023

I just released a new Python CLI tool called [symbex](#). It's a search tool, loosely inspired by [ripgrep](#), which lets you search Python code for functions and classes by name or wildcard, then see just the source code of those matching entities.

Searching for functions and classes

Here's an example of what it can do. Running in my `datasette/` folder:

```
symbex inspect_hash
```

Output:

```
# File: datasette/inspect.py Line: 17
def inspect_hash(path):
    """Calculate the hash of a database, efficiently."""
    m = hashlib.sha256()
    with path.open("rb") as fp:
        while True:
            data = fp.read(HASH_BLOCK_SIZE)
            if not data:
                break
            m.update(data)

    return m.hexdigest()
```

I gave it the name of a function (classes work too) and it searched all subfolders of the current directory, found that function and output it to my terminal.

Why is this more useful than `ripgrep` or any of the many other tools that can do this?

I partly built this to have fun learning Python's [ast](#) module, but it's mainly designed to complement my [LLM](#) CLI tool for running large language model prompts.

Code explanations with a large language model

Check this out:

```
symbex inspect_hash | llm --system 'explain succinctly'
```

Output:

This function calculates the hash of a database file efficiently by reading the file in blocks and updating the hash object using SHA256 algorithm from the `hashlib` module. The resulting hash value is returned as a hexadecimal string.

This is pretty cool!

```
llm --system 'explain succinctly' runs a prompt against gpt-3.5-turbo using “explain succinctly” as the system prompt.
```

The system prompt says what to do, then the content piped to the tool is treated as the data that should be processed.

Using wildcards and guessing what a tool does from tests

That was a pretty basic example. Here’s something more fun:

```
sybex 'test*csv*' | llm --system 'based on these tests guess what this tool does'
```

I’m using the `test*csv*` wildcard here to find all of my test functions that mention CSV in the name. There are 18 of those, with names like `test_table_csv_with_labels` and `test_table_csv_cors_headers`.

I can count the number of tokens in the test output like this (see [llm, ttok and strip-tags—CLI tools for working with ChatGPT and other LLMs](#) for an intro to my `ttok` tool):

```
sybex 'test_*csv*' | ttok
```

```
2682
```

That’s significantly below the 4000 token limit for `gpt-3.5-turbo`.

Here’s what I get when I ask it “based on these tests guess what this tool does”:

This tool seems to be a web application that provides a user interface to query databases and export query results in various formats, including HTML, JSON, and CSV. The tests verify that the tool correctly generates HTML pages with links to export query results in JSON or CSV format, exports tables to CSV format with custom options (such as whether to include labels and nullable labels), and performs streaming of CSV data. The tests also check that the tool sets appropriate HTTP headers, allows CORS, and handles errors and large CSV files gracefully.

That’s a good summary of what Datasette does!

I was worried that it might be cheating since ChatGPT already knows about Datasette, but I tried this:

```
sybex 'test_*csv*' | grep datasette
```

And got no results, so at least that keyword wasn’t being leaked in the test details somehow.

Refactoring code

Let’s try something a whole lot more useful:

```
sybex Request | llm --system 'add type hints to this'
```

This locates the Request class in Datasette—[this one here](#), and starts adding Python type hints to it. The output started out like this (that code has no type hints at all at the moment):

```
from typing import Dict, Any, Awaitable
from http.cookies import SimpleCookie
from urllib.parse import urlunparse, parse_qs, parse_qsl

from .multidict import MultiParams

class Request:
    def __init__(self, scope: Dict[str, Any], receive: Awaitable) -> None:
        self.scope = scope
        self.receive = receive

    def __repr__(self) -> str:
        return '<asgi.Request method="{}" url="{}">'.format(self.method, self.url)

    @property
    def method(self) -> str:
        return self.scope["method"]

    @property
    def url(self) -> str:
        return urlunparse(
            (self.scheme, self.host, self.path, None, self.query_string, None)
        )

    @property
    def url_vars(self) -> Dict[str, str]:
        return (self.scope.get("url_route") or {}).get("kwargs") or {}

    # ...
```

Now this is getting impressive! Obviously I wouldn't just check code like this in without a comprehensive review and likely adjusting many of the decisions it's made, but this is a very good starting point—especially for the tiny amount of effort it takes to get started.

Picking a name for the tool

The most time-consuming part of this project ended up being picking the name!

Originally I planned to call it `py-grep`. I checked <https://pypi.org/project/py-grep/> and it was available, so I spun up the first version of the tool and attempted to upload it to PyPI.

PyPI gave me an error, because the name was too similar to the existing `pygrep` package. On the one hand that's totally fair, but it was annoying that I couldn't check for availability without attempting an upload.

I turned to ChatGPT to start brainstorming new names. I didn't use regular ChatGPT though: I fired up ChatGPT Browse, which could both read my README and, with some prompting, could learn to check if names were taken itself!



All of those names are taken, more ideas please - you can check if they are taken by checking for <https://pypi.org/project/pyseek/>



Finished browsing



- Clicked on: [pydex · PyPI](#)
- Clicked on: [pyscan · PyPI](#)
- Clicked on: [pylocator · PyPI](#)
- Clicked on: [pyspotter · PyPI](#)
- Clicked on: [pylocator · PyPI](#)
- Clicked on: [pyfinder · PyPI](#)
- Clicked on: [PyScanner · PyPI](#)
- Clicked on: [PyLookup · PyPI](#)
- Clicked on: [PySpot · PyPI](#)
- Clicked on: [PyLookup · PyPI](#)
- Clicked on: [pylocator · PyPI](#)
- Clicked on: [pyfinder · PyPI](#)
- Clicked on: [pylocator · PyPI](#)
- Clicked on: [pyfinder · PyPI](#)
- Click failed
- Finished browsing

It seems there was an error fetching the page. This might indicate that the name "pysourcefinder" is not taken, but it would be safer to verify manually.

I wrote up the full process for this in a TIL: [Using ChatGPT Browse to name a Python package.](#)

Posted [18th June 2023](#) at 10:11 pm · Follow me on [Mastodon](#) or [Twitter](#) or [subscribe to my newsletter](#)

More recent articles

- [How I make annotated presentations](#) - 6th August 2023
- [Weeknotes: Plugins for LLM, sqlite-utils and Datasette](#) - 5th August 2023
- [Catching up on the weird world of LLMs](#) - 3rd August 2023
- [Run Llama 2 on your own Mac using LLM and Homebrew](#) - 1st August 2023
- [sqlite-utils now supports plugins](#) - 24th July 2023
- [Accessing Llama 2 from the command-line with the llm-replicate plugin](#) - 18th July 2023
- [Weeknotes: Self-hosted language models with LLM plugins, a new Datasette tutorial, a dozen package releases, a dozen TILs](#) - 16th July 2023

- [My LLM CLI tool now supports self-hosted language models via plugins](#) - 12th July 2023
- [Weeknotes: symbex, LLM prompt templates, a bit of a break](#) - 27th June 2023

[python](#) 859

[generativeai](#) 257

[projects](#) 314

[chatgpt](#) 66

[ai](#) 276

[llms](#) 226

[symbex](#) 3

Next: [Weeknotes: symbex, LLM prompt templates, a bit of a break](#)

Previous: [Understanding GPT tokenizers](#)

Source code © 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
2016 2017 2018 2019 2020 2021 2022 2023