APIs, web scraping, and more packaging

Practical Programming in Chemistry

Prof. Philippe Schwaller

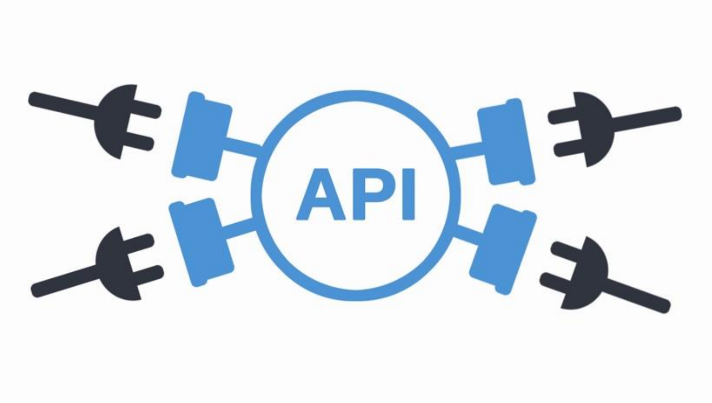
This lecture

- Useful chemistry tools
- APIs (not active pharmaceutical ingredients, but application programming interfaces)
- Web scraping
- More packaging

- For those who emailed me about the project, could you send me a friendly reminder, and could you add “[CH200]” to the subject?

APIs

- APIs, or **Application Programming Interfaces**, are **a set of rules and protocols** that allow different software applications to communicate with each other.
- Instead of communicating with a database/webpage manually, an **API can provide a set of functions that facilitate communications**.



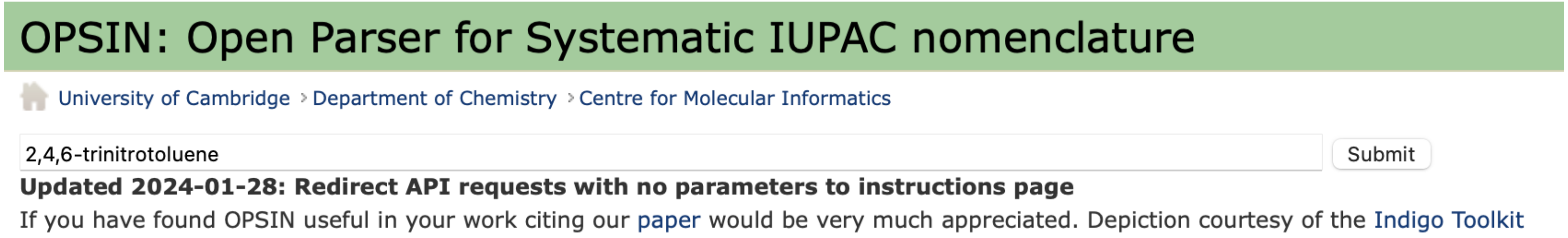
A function with well-defined inputs and outputs can be an API.

https://www.danielleskosky.com/wp-content/uploads/media-uploads/what-is-an-api/api-with-outlets.jpg

Let’s start with some useful chemistry tools, and their Python API.

OPSIN - Open Parser for Systematic IUPAC nomenclature

- IUPAC name to structure converter (https://opsin.ch.cam.ac.uk) made by Daniel Lowe

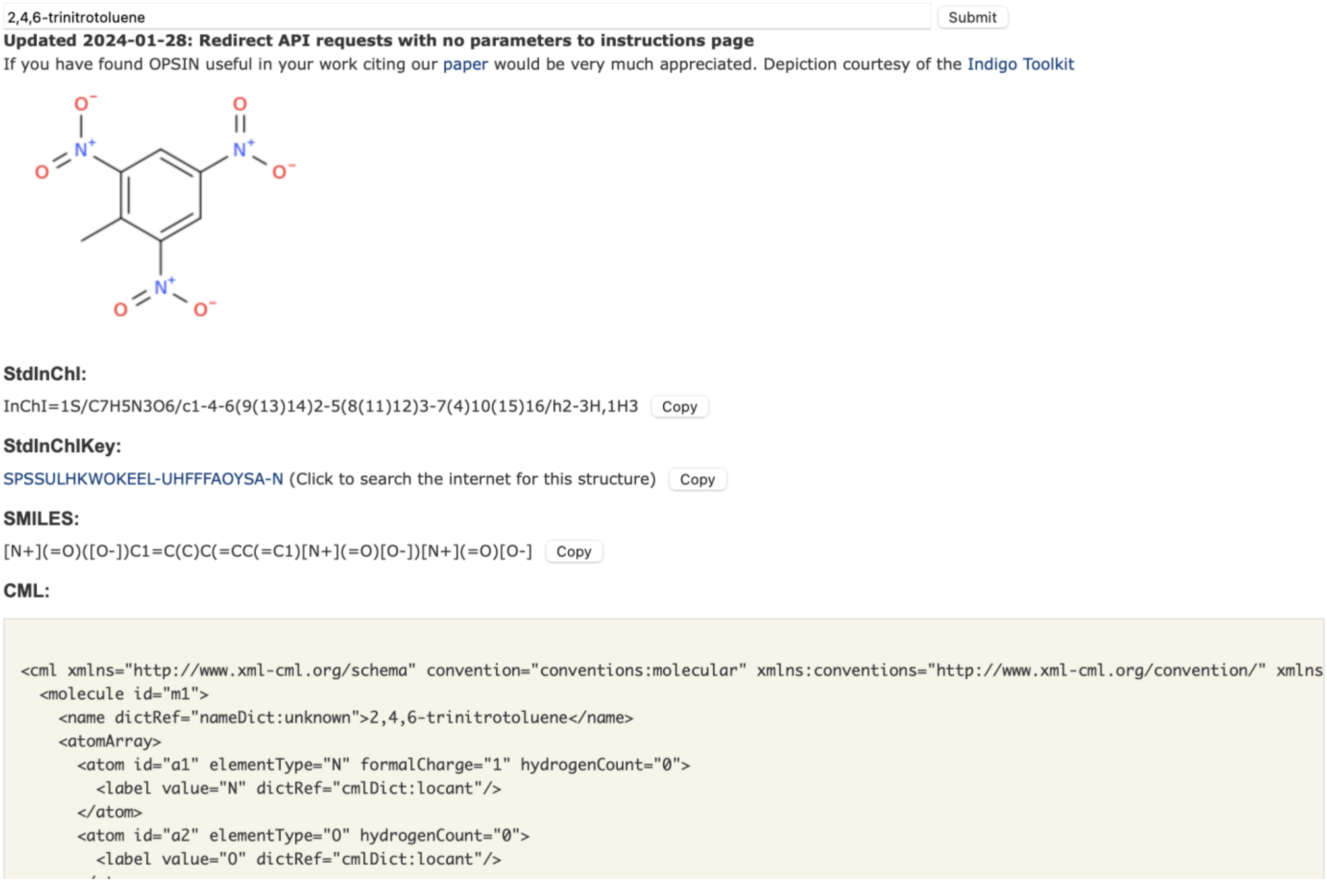

OPSIN: Open Parser for Systematic IUPAC nomenclature

2,4,6-trinitrotoluene

Updated 2024-01-28: Redirect API requests with no parameters to instructions page

If you have found OPSIN useful in your work citing our paper would be very much appreciated. Depiction courtesy of the Indigo Toolkit

OPSIN output



Updated 2024-01-28: Redirect API requests with no parameters to instructions page

If you have found OPSIN useful in your work citing our paper would be very much appreciated. Depiction courtesy of the Indigo Toolkit

StdInChI:

InChI=1S/C7H5N3O6/c1-4-6(9(13)14)2-5(8(11)12)3-7(4)10(15)16/h2-3H,1H3

StdInChIKey:

SPSSULHKWOKEEL-UHFFFAOYSA-N (Click to search the internet for this structure)

SMILES:

[N+](=O)([O-])C1=C(C)C(=CC(=C1)[N+](=O)[O-])[N+](=O)[O-]

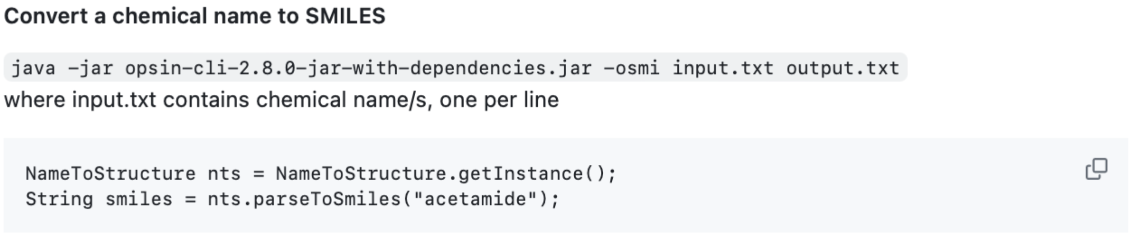
CML:

```
<cml xmlns="http://www.xml-cml.org/schema" convention="conventions:molecular" xmlns:conventions="http://www.xml-cml.org/convention/" xmlns
  <molecule id="m1">
    <name dictRef="nameDict:unknown">2,4,6-trinitrotoluene</name>
    <atomArray>
      <atom id="a1" elementType="N" formalCharge="1" hydrogenCount="0">
        <label value="N" dictRef="cmlDict:locant"/>
      </atom>
      <atom id="a2" elementType="O" hydrogenCount="0">
        <label value="O" dictRef="cmlDict:locant"/>
```

It’s ok to query the webpage manually if you want to convert a few names, but what if you had 1M names?

Let's get the code..

- https://github.com/dan2097/opsin, found it! And it's MIT-licensed :D
- But it's in Java … I don't want to learn another programming language.



Convert a chemical name to SMILES

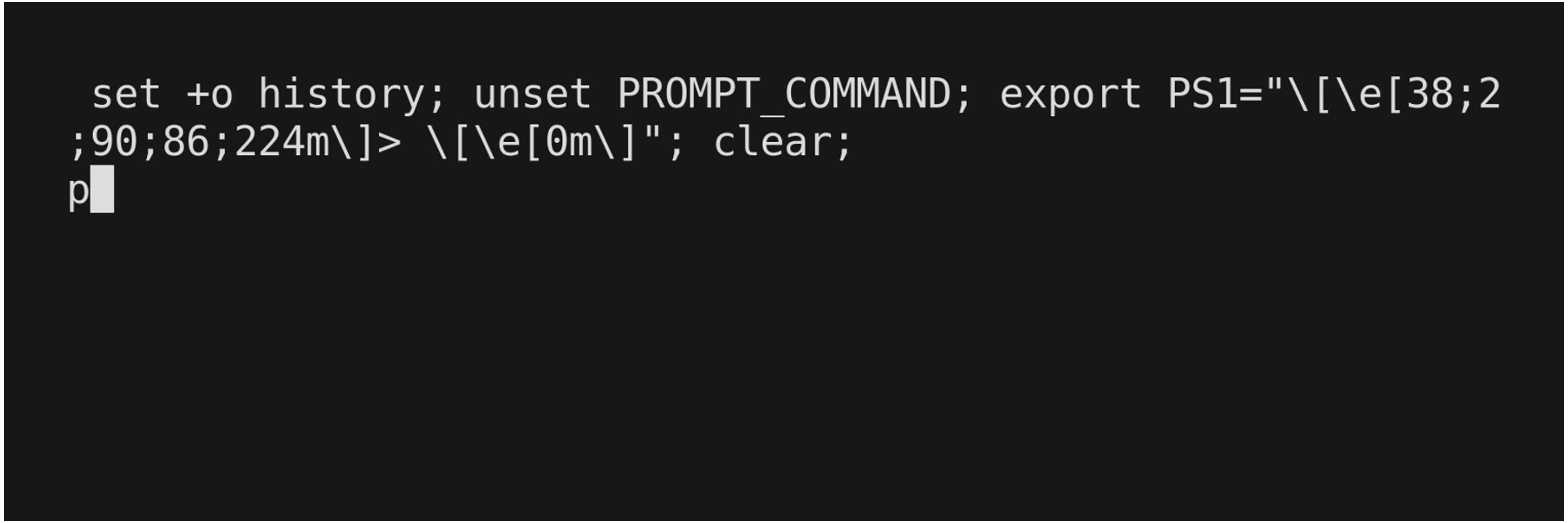
```
java -jar opsin-cli-2.8.0-jar-with-dependencies.jar -osmi input.txt output.txt
```

where input.txt contains chemical name/s, one per line

```
NameToStructure nts = NameToStructure.getInstance();
String smiles = nts.parseToSmiles("acetamide");
```

- Luckily, there is there is Jackson Burns, who faced the same problem: https://github.com/JacksonBurns/py2opsin/tree/main

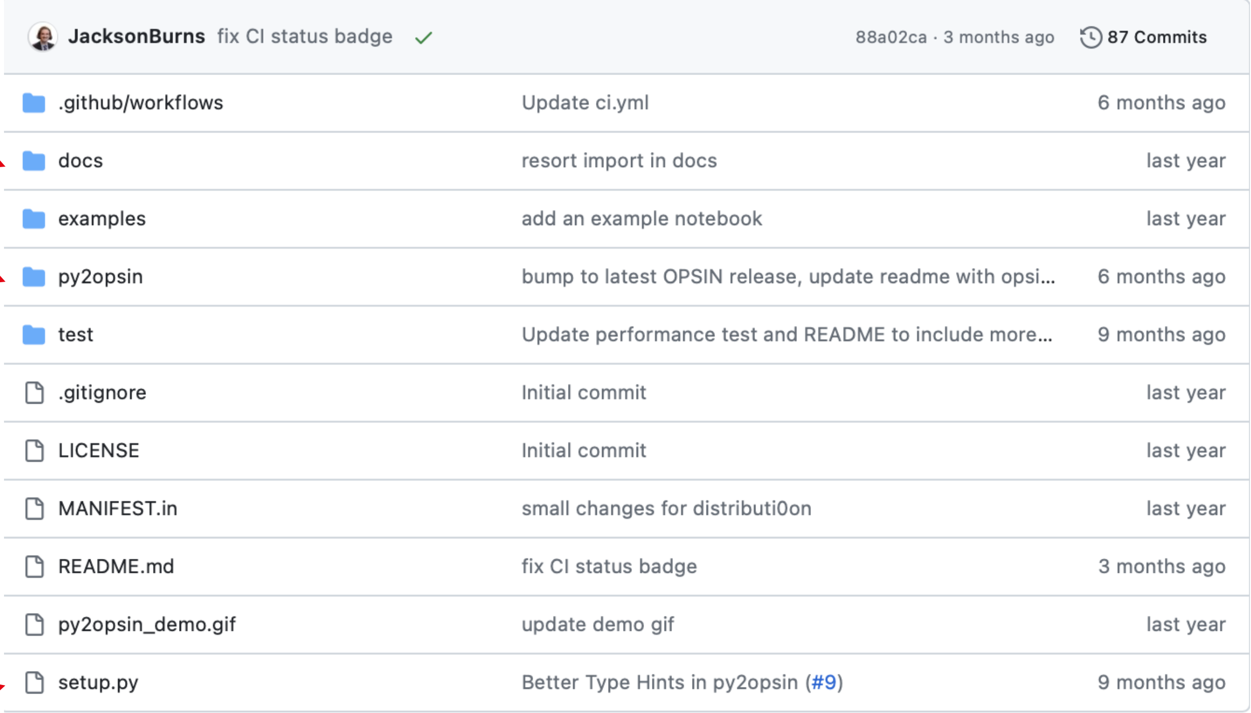
py2opsin - Simple Python interface to OPSIN: Open Parser for Systematic IUPAC nomenclature



```
 set +o history; unset PROMPT_COMMAND; export PS1="\[\e[38;2
;90;86;224m\]> \[\e[0m\]"; clear;
p
```

https://github.com/JacksonBurns/py2opsin/tree/main

The Python package



JacksonBurns fix CI status badge ✓ 88a02ca · 3 months ago 87 Commits

.github/workflows	Update ci.yml	6 months ago
docs	resort import in docs	last year
examples	add an example notebook	last year
py2opsin	bump to latest OPSIN release, update readme with opsi...	6 months ago
test	Update performance test and README to include more...	9 months ago
.gitignore	Initial commit	last year
LICENSE	Initial commit	last year
MANIFEST.in	small changes for distributi0on	last year
README.md	fix CI status badge	3 months ago
py2opsin_demo.gif	update demo gif	last year
setup.py	Better Type Hints in py2opsin (#9)	9 months ago

Documentation

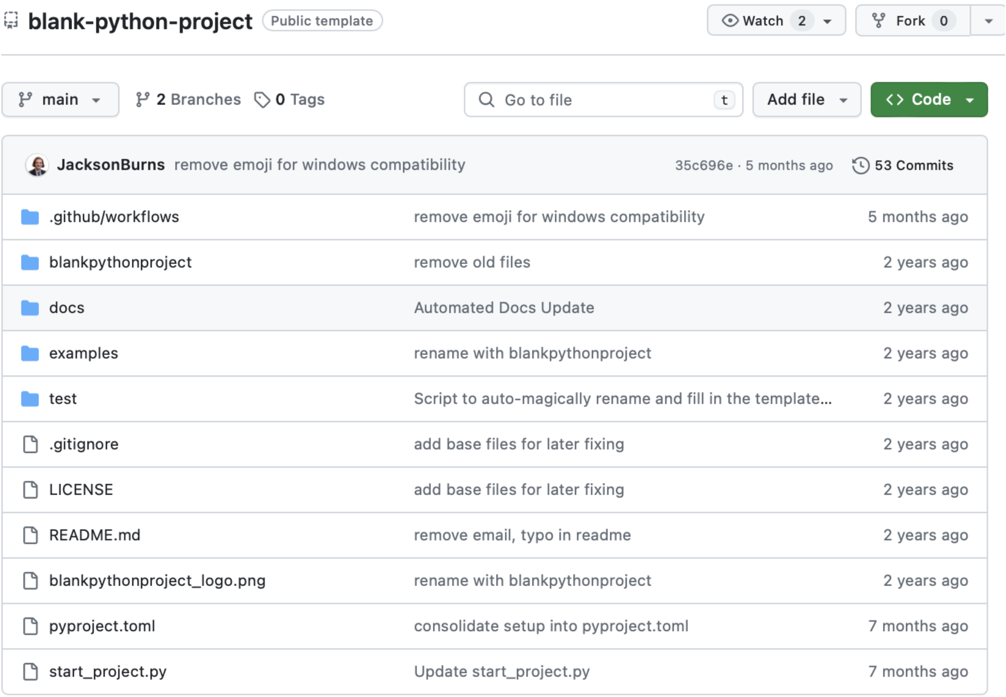
The source code

Tests

File to tell
pip how to install the package

Fun fact – he also changed pyproject.toml

Jackson Burns template for python projects



blank-python-project Public template

JacksonBurns remove emoji for windows compatibility 35c696e · 5 months ago 53 Commits

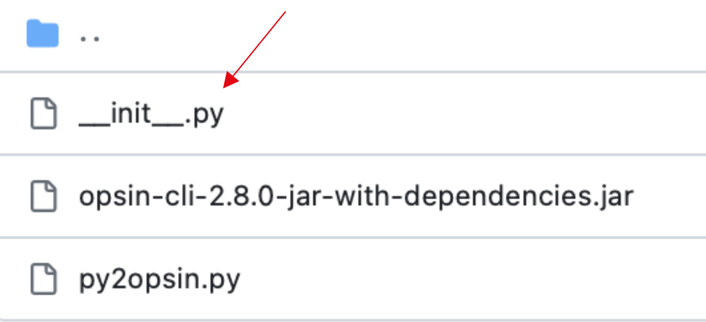
.github/workflows	remove emoji for windows compatibility	5 months ago
blankpythonproject	remove old files	2 years ago
docs	Automated Docs Update	2 years ago
examples	rename with blankpythonproject	2 years ago
test	Script to auto-magically rename and fill in the template...	2 years ago
.gitignore	add base files for later fixing	2 years ago
LICENSE	add base files for later fixing	2 years ago
README.md	remove email, typo in readme	2 years ago
blankpythonproject_logo.png	rename with blankpythonproject	2 years ago
pyproject.toml	consolidate setup into pyproject.toml	7 months ago
start_project.py	Update start_project.py	7 months ago

pyproject.toml,
not setup.py

https://github.com/JacksonBurns/blank-python-project

Let’s check out the package source of pyopsin

The file that says:
“This is part of the package”

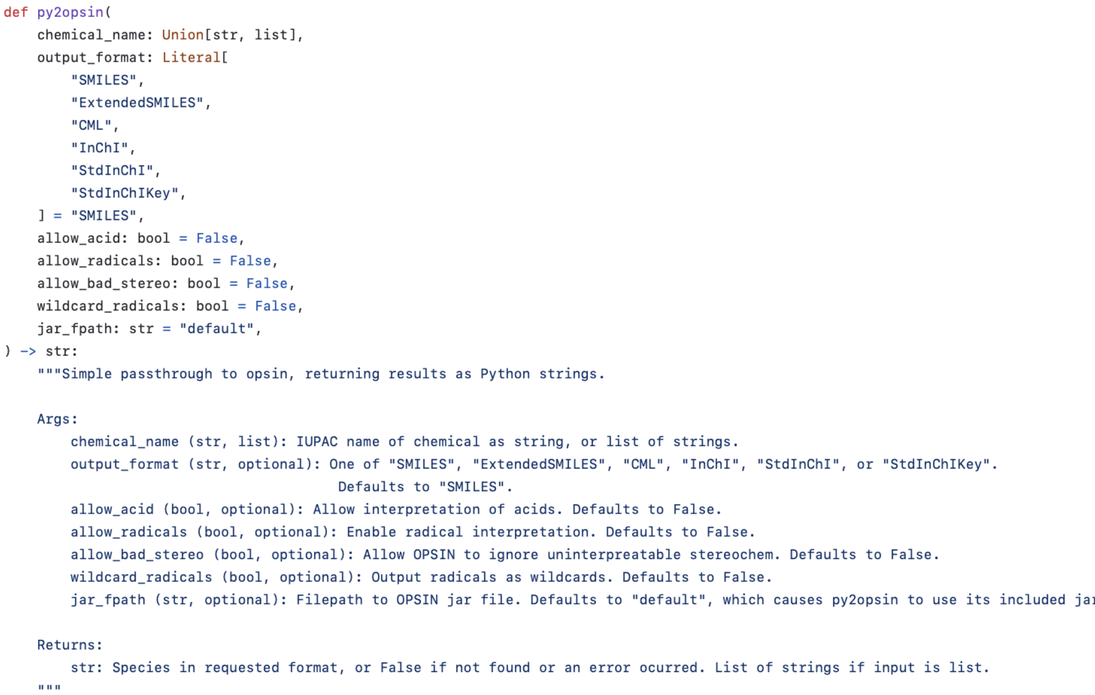


..

__init__.py

opsin-cli-2.8.0-jar-with-dependencies.jar

py2opsin.py



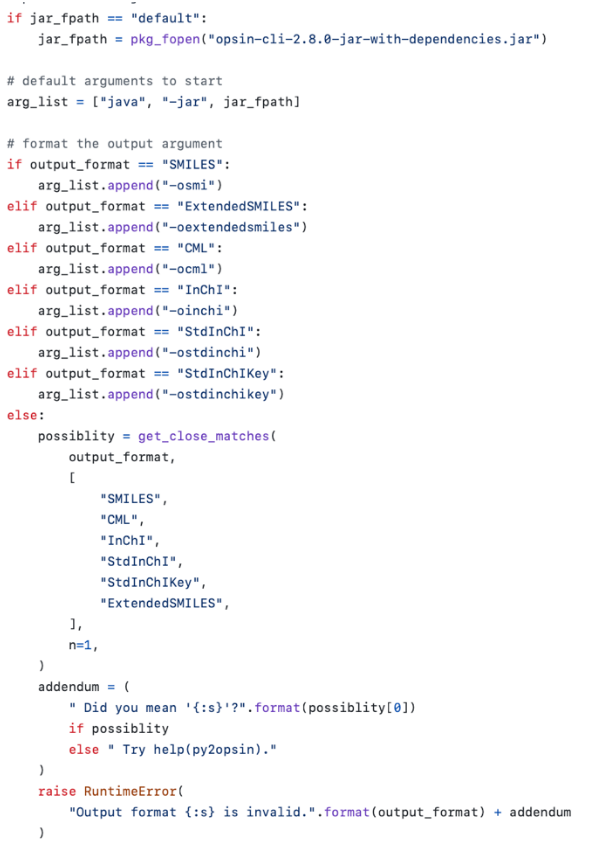
```
def py2opsin(
    chemical_name: Union[str, list],
    output_format: Literal[
        "SMILES",
        "ExtendedSMILES",
        "CML",
        "InChI",
        "StdInChI",
        "StdInChIKey",
    ] = "SMILES",
    allow_acid: bool = False,
    allow_radicals: bool = False,
    allow_bad_stereo: bool = False,
    wildcard_radicals: bool = False,
    jar_fpath: str = "default",
) -> str:
    """Simple passthrough to opsin, returning results as Python strings.

    Args:
        chemical_name (str, list): IUPAC name of chemical as string, or list of strings.
        output_format (str, optional): One of "SMILES", "ExtendedSMILES", "CML", "InChI", "StdInChI", or "StdInChIKey".
                                Defaults to "SMILES".
        allow_acid (bool, optional): Allow interpretation of acids. Defaults to False.
        allow_radicals (bool, optional): Enable radical interpretation. Defaults to False.
        allow_bad_stereo (bool, optional): Allow OPSIN to ignore uninterpreatable stereochem. Defaults to False.
        wildcard_radicals (bool, optional): Output radicals as wildcards. Defaults to False.
        jar_fpath (str, optional): Filepath to OPSIN jar file. Defaults to "default", which causes py2opsin to use its included ja

    Returns:
        str: Species in requested format, or False if not found or an error ocurred. List of strings if input is list.
    """
```

1 single function that wraps the java-based OPSIN, and provides a clear interface to use it in Python.

→ local API

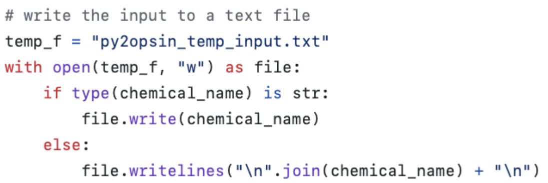
- https://github.com/JacksonBurns/py2opsin/blob/main/py2opsin/py2opsin.py

How it is done.



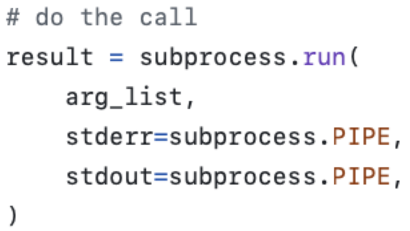
```
if jar_fpath == "default":
    jar_fpath = pkg_fopen("opsin-cli-2.8.0-jar-with-dependencies.jar")

# default arguments to start
arg_list = ["java", "-jar", jar_fpath]

# format the output argument
if output_format == "SMILES":
    arg_list.append("-osmi")
elif output_format == "ExtendedSMILES":
    arg_list.append("-oextendedsmiles")
elif output_format == "CML":
    arg_list.append("-ocml")
elif output_format == "InChI":
    arg_list.append("-oinchi")
elif output_format == "StdInChI":
    arg_list.append("-ostdinchi")
elif output_format == "StdInChIKey":
    arg_list.append("-ostdinchikey")
else:
    possiblity = get_close_matches(
        output_format,
        [
            "SMILES",
            "CML",
            "InChI",
            "StdInChI",
            "StdInChIKey",
            "ExtendedSMILES",
        ],
        n=1,
    )
    addendum = (
        " Did you mean '{:s}'?".format(possiblity[0])
        if possiblity
        else " Try help(py2opsin)."
    )
    raise RuntimeError(
        "Output format {:s} is invalid.".format(output_format) + addendum
    )
```

Defining the input to the java program



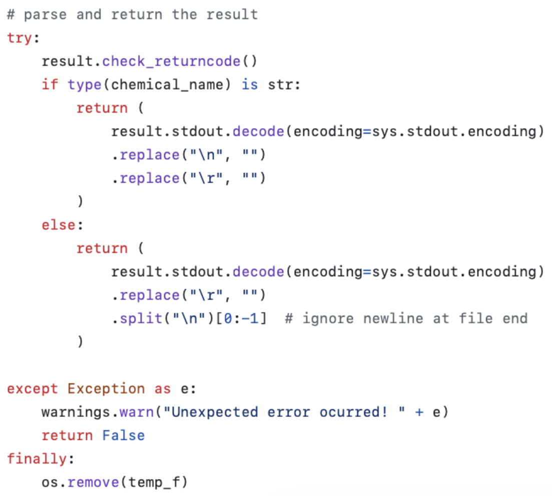
```
# write the input to a text file
temp_f = "py2opsin_temp_input.txt"
with open(temp_f, "w") as file:
    if type(chemical_name) is str:
        file.write(chemical_name)
    else:
        file.writelines("\n".join(chemical_name) + "\n")
```

Creating a temporary input file



```
# do the call
result = subprocess.run(
    arg_list,
    stderr=subprocess.PIPE,
    stdout=subprocess.PIPE,
)
```

Running the java program on the file, saving the output in ”result”

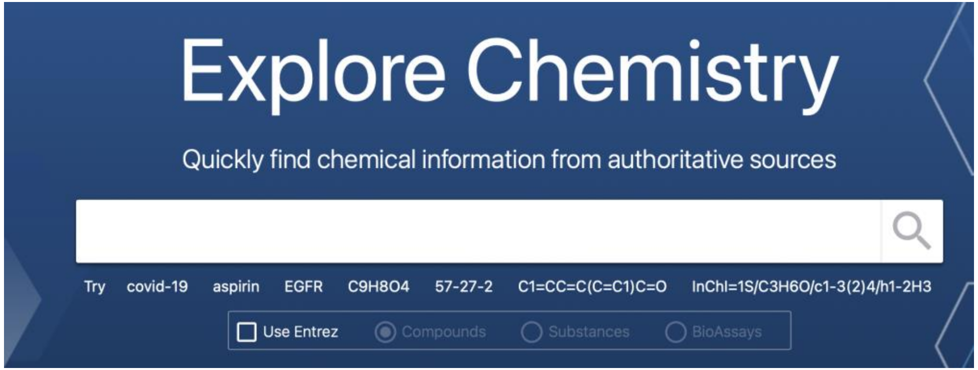


```
# parse and return the result
try:
    result.check_returncode()
    if type(chemical_name) is str:
        return (
            result.stdout.decode(encoding=sys.stdout.encoding)
            .replace("\n", "")
            .replace("\r", "")
        )
    else:
        return (
            result.stdout.decode(encoding=sys.stdout.encoding)
            .replace("\r", "")
            .split("\n")[0:-1]  # ignore newline at file end
        )

except Exception as e:
    warnings.warn("Unexpected error ocurred! " + e)
    return False
finally:
    os.remove(temp_f)
```

Parsing the result and deleting the temporary file.

PubChem

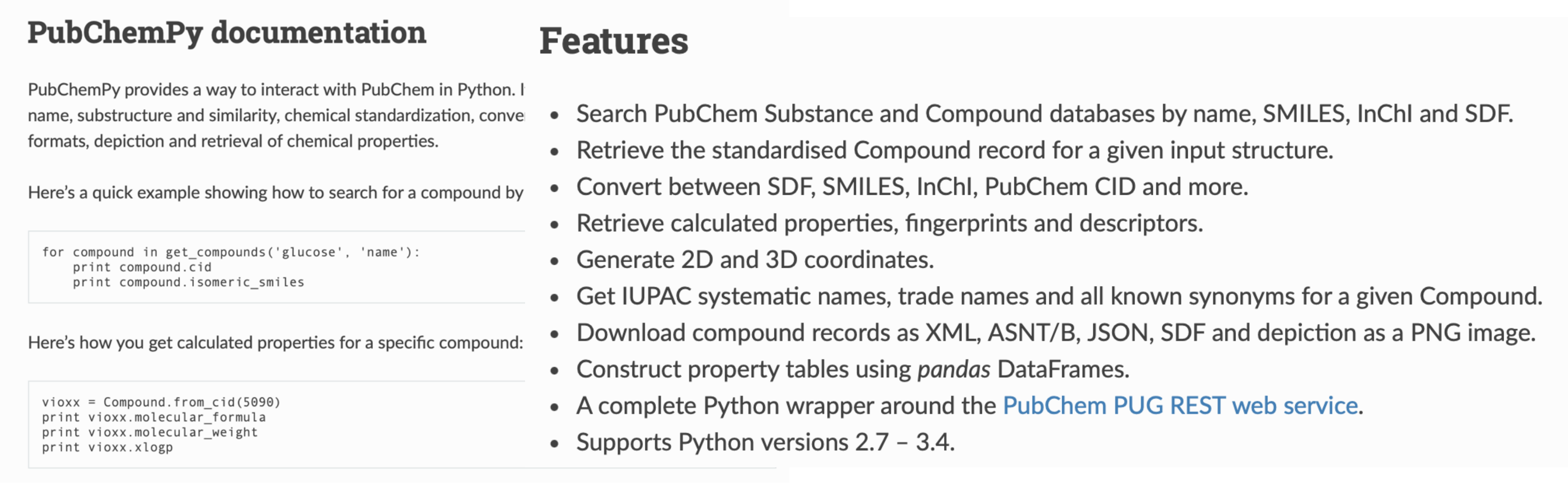
- PubChem is a **public chemical database** aggregating various information on mostly small molecules from multiple sources.
- PubChem provides broad annotations from more than 870 sources grouped into data collections containing information about substances, bioassays, protein targets, genes, pathways, cell lines, taxonomy, and patents.
- Currently, more than **111 million unique molecules** are deposited in PubChem.
- https://pubchem.ncbi.nlm.nih.gov


Explore Chemistry

Quickly find chemical information from authoritative sources

Manual search is fine for a few compounds, but again what if you want to query hundreds.

PubChemAPI



PubChemPy documentation

PubChemPy provides a way to interact with PubChem in Python. I
name, substructure and similarity, chemical standardization, conve
formats, depiction and retrieval of chemical properties.

Here's a quick example showing how to search for a compound by

```
for compound in get_compounds('glucose', 'name'):
    print compound.cid
    print compound.isomeric_smiles
```

Here's how you get calculated properties for a specific compound:

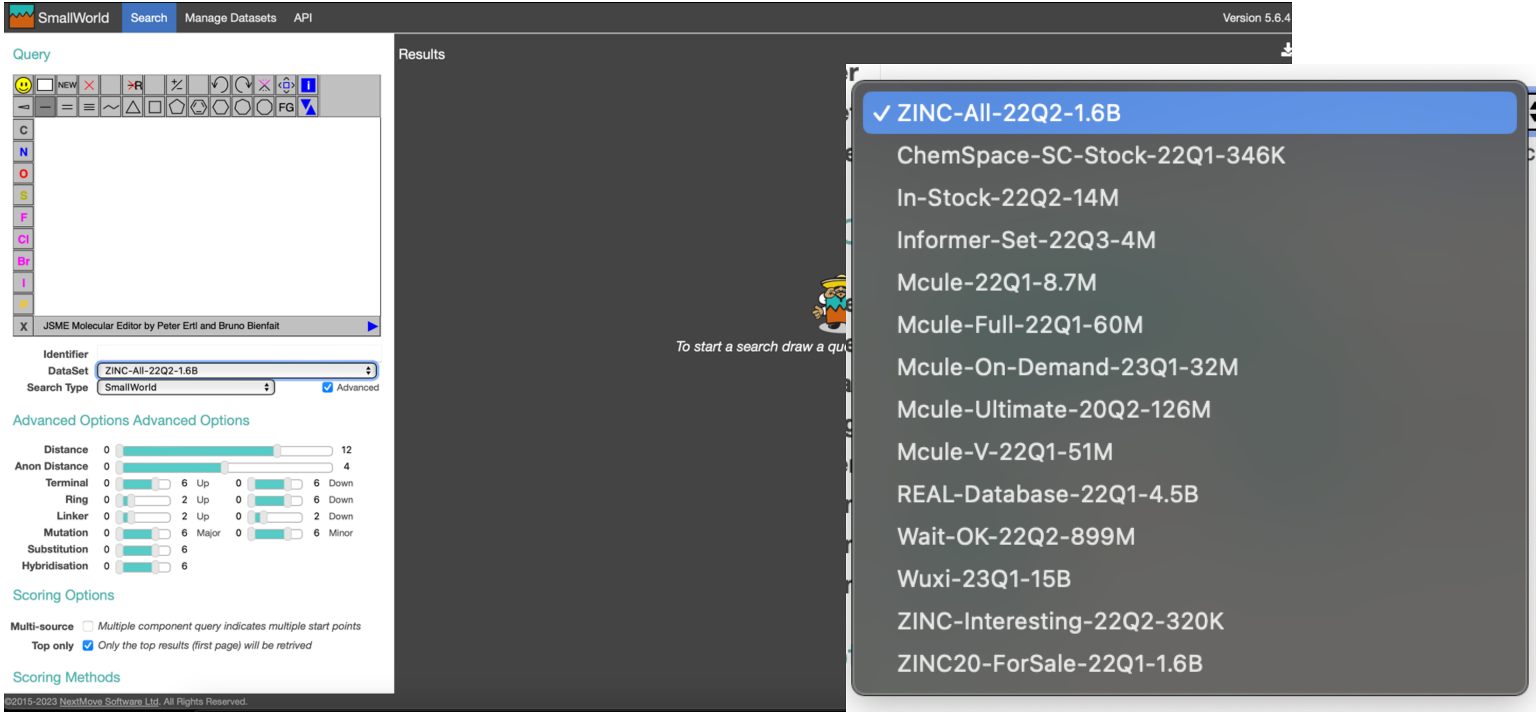
```
vioxx = Compound.from_cid(5090)
print vioxx.molecular_formula
print vioxx.molecular_weight
print vioxx.xlogp
```

Features

- Search PubChem Substance and Compound databases by name, SMILES, InChI and SDF.
- Retrieve the standardised Compound record for a given input structure.
- Convert between SDF, SMILES, InChI, PubChem CID and more.
- Retrieve calculated properties, fingerprints and descriptors.
- Generate 2D and 3D coordinates.
- Get IUPAC systematic names, trade names and all known synonyms for a given Compound.
- Download compound records as XML, ASNT/B, JSON, SDF and depiction as a PNG image.
- Construct property tables using *pandas* DataFrames.
- A complete Python wrapper around the PubChem PUG REST web service.
- Supports Python versions 2.7 – 3.4.

https://github.com/mcs07/PubChemPy

SmallWorld - https://sw.docking.org/search.html



Identifier

DataSet ZINC-All-22Q2-1.6B

Search Type SmallWorld ☑ Advanced

Advanced Options Advanced Options

Distance 0 12

Anon Distance 0 4

Terminal 0 6 Up 0 6 Down

Ring 0 2 Up 0 6 Down

Linker 0 2 Up 0 2 Down

Mutation 0 6 Major 0 6 Minor

Substitution 0 6

Hybridisation 0 6

Scoring Options

Multi-source ☐ *Multiple component query indicates multiple start points*

Top only ☑ *Only the top results (first page) will be retrived*

Scoring Methods

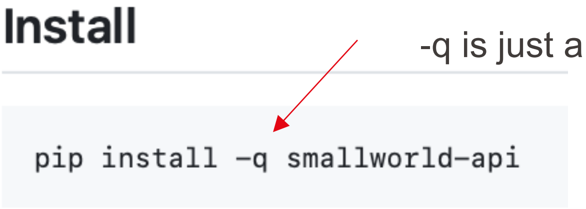
©2015-2023 NextMove Software Ltd. All Rights Reserved.

Blazing fast molecular similarity search in some of the world's most extensive molecule databases.

Again, you find a inofficial Python API

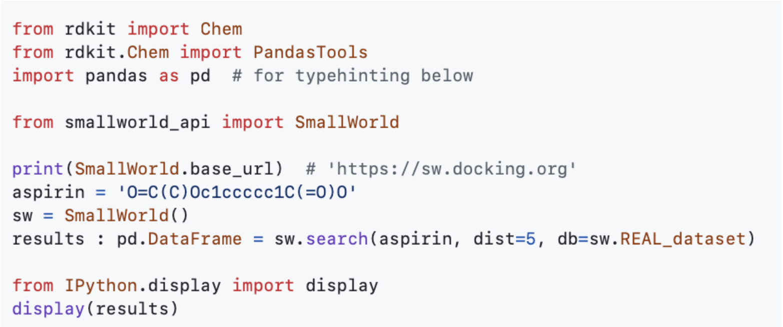
- https://github.com/matteoferla/Python_SmallWorld_API (MIT)

-q is just an option for “quite”, less output in the terminal while installing

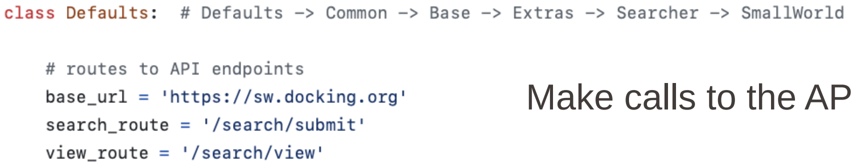


Install

```
pip install -q smallworld-api
```



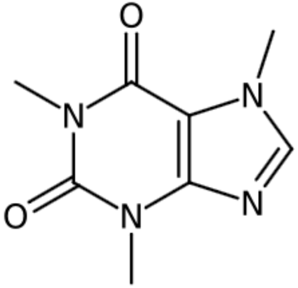
```
from rdkit import Chem
from rdkit.Chem import PandasTools
import pandas as pd  # for typehinting below

from smallworld_api import SmallWorld

print(SmallWorld.base_url)  # 'https://sw.docking.org'
aspirin = 'O=C(C)Oc1ccccc1C(=O)O'
sw = SmallWorld()
results : pd.DataFrame = sw.search(aspirin, dist=5, db=sw.REAL_dataset)

from IPython.display import display
display(results)
```



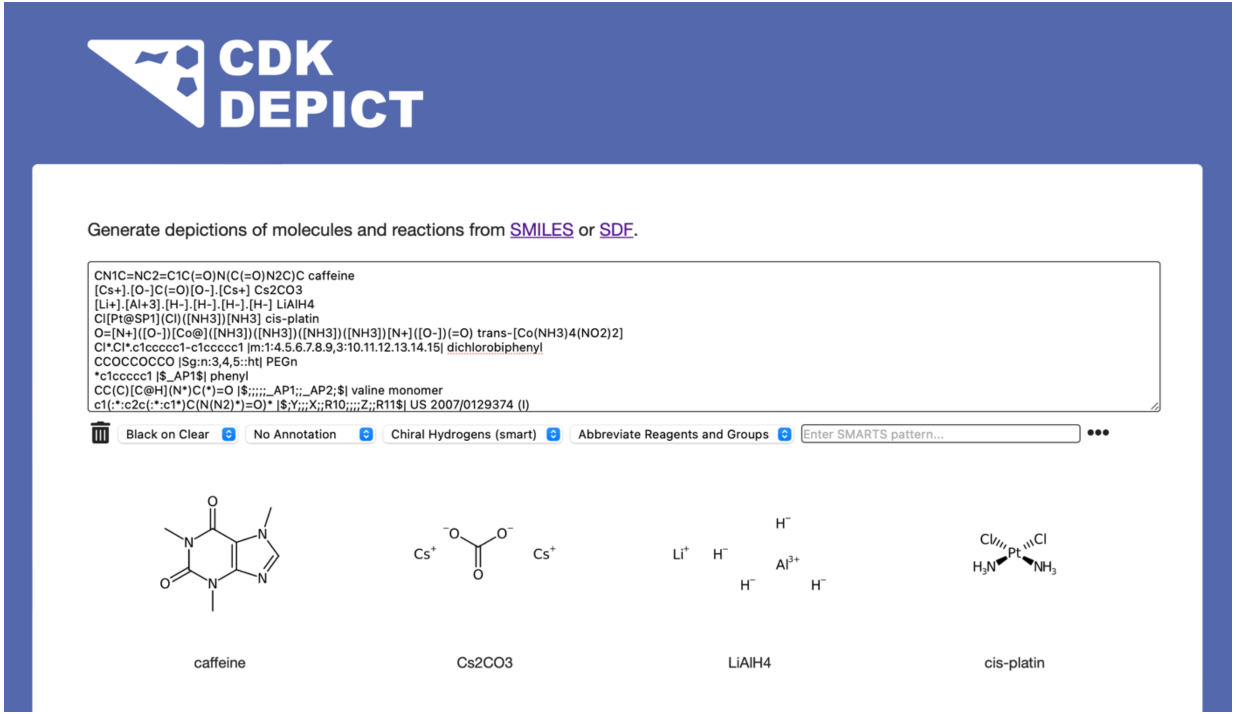
```
class Defaults:  # Defaults -> Common -> Base -> Extras -> Searcher -> SmallWorld

    # routes to API endpoints
    base_url = 'https://sw.docking.org'
    search_route = '/search/submit'
    view_route = '/search/view'
```

Make calls to the API endpoints

CDK Depict



If we click on a molecule,
we get back the following URL.



Generate depictions of molecules and reactions from SMILES or SDF.

```
CN1C=NC2=C1C(=O)N(C(=O)N2C)C caffeine
[Cs+].[O-]C(=O)[O-].[Cs+] Cs2CO3
[Li+].[Al+3].[H-].[H-].[H-].[H-] LiAlH4
Cl[Pt@SP1](Cl)([NH3])[NH3] cis-platin
O=[N+]([O-])[Co@]([NH3])([NH3])([NH3])([NH3])[N+]([O-])(=O) trans-[Co(NH3)4(NO2)2]
Cl*.Cl*.c1ccccc1-c1ccccc1 |m:1:4.5.6.7.8.9,3:10.11.12.13.14.15| dichlorobiphenyl
CCOCCOCCO |Sg:n:3,4,5::ht| PEGn
*c1ccccc1 |$_AP1$| phenyl
CC(C)[C@H](N*)C(*)=O |$;;;;;_AP1;;_AP2;$| valine monomer
c1(:*:c2c(:*:c1*)C(N(N2)*)=O)* |$;Y;;;X;;R10;;;;Z;;R11$| US 2007/0129374 (I)
```

Black on Clear | No Annotation | Chiral Hydrogens (smart) | Abbreviate Reagents and Groups | Enter SMARTS pattern...

https://www.simolecule.com/cdkdepict/depict/bot/svg?smi=CN1C%3DNC2%3DC1C(%3DO)N(C(%3DO)N2C)C%20caffeine&w=-1&h=-1&abbr=on&hdisp=bridgehead&zoom=1.3&annotate=none&r=0

How would we build an API to be able to request the CDK depict SVG in a Jupyter notebook?

https://www.simolecule.com/cdkdepict/depict/bot/svg?smi=CN1C%3DNC2%3DC1C(%3DO)N(C(%3DO)N2C)C%20caffeine&w=-1&h=-1&abbr=on&hdisp=bridgehead&zoom=1.3&annotate=none&r=0

- Define: base_url → https://www.simolecule.com/cdkdepict/depict/
- Make a function that takes the following parameters
  - style → default: bot
  - format → default: svg
  - smiles_string + description
  - abbreviation → default: on
  - hydrogens → default: bridgehead
  - annotations → default: none
  - rotation → default: 0
- Make a dictionary out of the parameters
- Encode them with params_str = urllib.parse.urlencode(params)
- Create the URL: `f'{base_url}/{style}/svg?{params_str}'`
- Call that URL and return the SVG

Getting data/information from webpages using Python

How do we connect from Python to webpages?

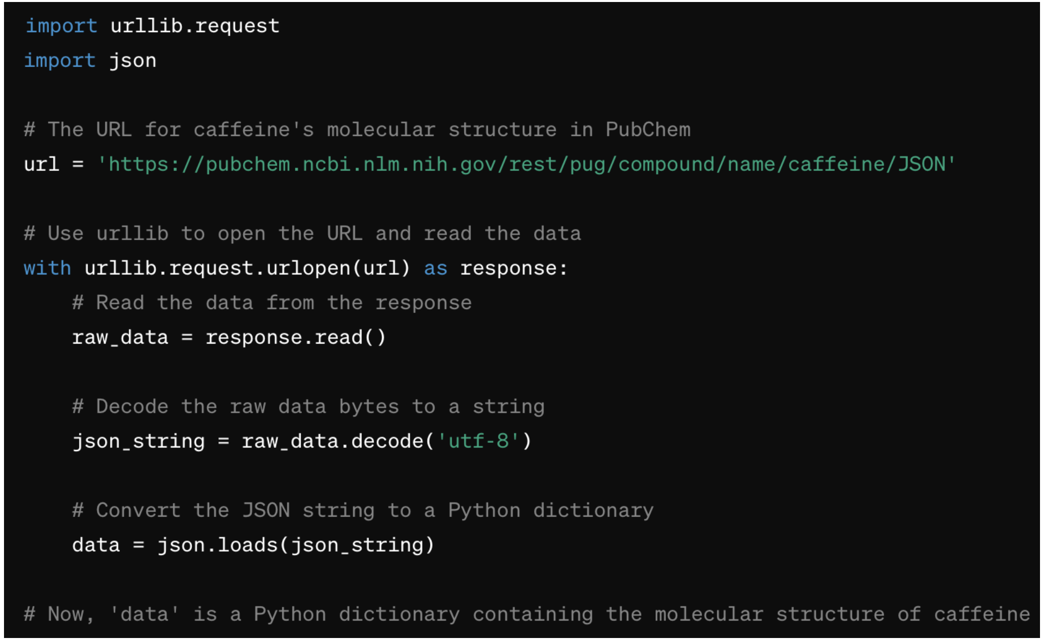
- **urllib** (standard Python library), **requests** (community-driven library)
- **urllib** is a **Python module that allows you to interact with websites** by sending requests and handling responses. Think of it as a tool that lets your Python script **browse the internet, access web resources, and even download data**, much like you would with a web browser, but programmatically.
- **requests** is a highly popular Python library designed to simplify the process of making HTTP requests. It offers a **more user-friendly** and intuitive approach compared to Python's standard library modules like urllib.

Basic concepts

- **URLs (uniform resource locators)**: Just like web addresses you enter in a browser to visit a webpage, urllib/requests use URLs to **locate and access web resources**.
- **Requests**: This is how you **ask for data from a website**. For example, requesting the webpage’s source code or data for a molecule.
- **Responses**: After you make a request, the **website sends back a response**, which includes the data you asked for, like the HTML of a webpage or a file to download.

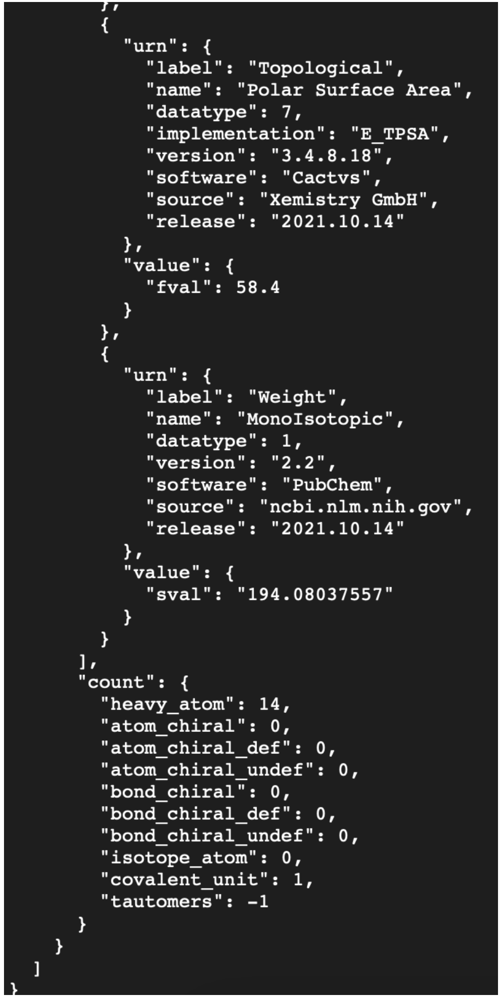
Basic example → PubChem

- https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/caffeine/JSON



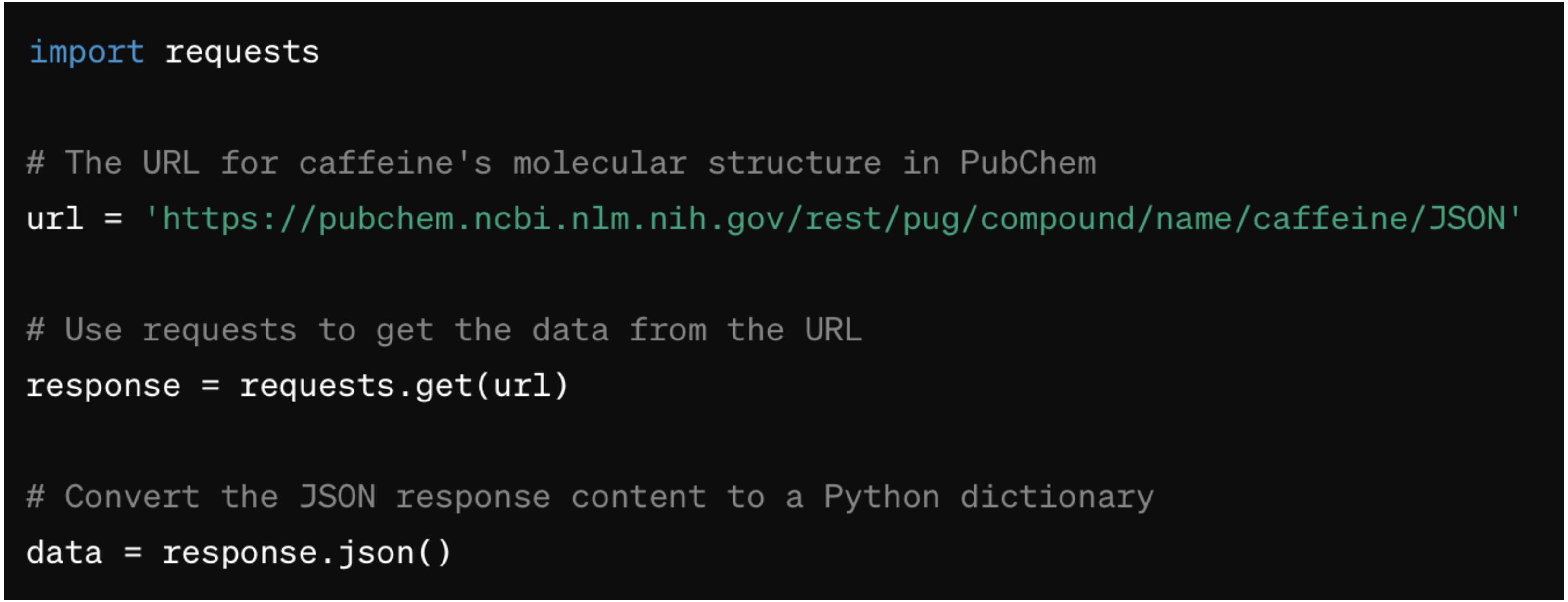
```python
import urllib.request
import json

# The URL for caffeine's molecular structure in PubChem
url = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/caffeine/JSON'

# Use urllib to open the URL and read the data
with urllib.request.urlopen(url) as response:
    # Read the data from the response
    raw_data = response.read()

    # Decode the raw data bytes to a string
    json_string = raw_data.decode('utf-8')

    # Convert the JSON string to a Python dictionary
    data = json.loads(json_string)

# Now, 'data' is a Python dictionary containing the molecular structure of caffeine
```

This would give you the JSON dictionary in *data*



```
        },
        {
          "urn": {
            "label": "Topological",
            "name": "Polar Surface Area",
            "datatype": 7,
            "implementation": "E_TPSA",
            "version": "3.4.8.18",
            "software": "Cactvs",
            "source": "Xemistry GmbH",
            "release": "2021.10.14"
          },
          "value": {
            "fval": 58.4
          }
        },
        {
          "urn": {
            "label": "Weight",
            "name": "MonoIsotopic",
            "datatype": 1,
            "version": "2.2",
            "software": "PubChem",
            "source": "ncbi.nlm.nih.gov",
            "release": "2021.10.14"
          },
          "value": {
            "sval": "194.08037557"
          }
        }
      ],
      "count": {
        "heavy_atom": 14,
        "atom_chiral": 0,
        "atom_chiral_def": 0,
        "atom_chiral_undef": 0,
        "bond_chiral": 0,
        "bond_chiral_def": 0,
        "bond_chiral_undef": 0,
        "isotope_atom": 0,
        "covalent_unit": 1,
        "tautomers": -1
      }
    }
  ]
}
```

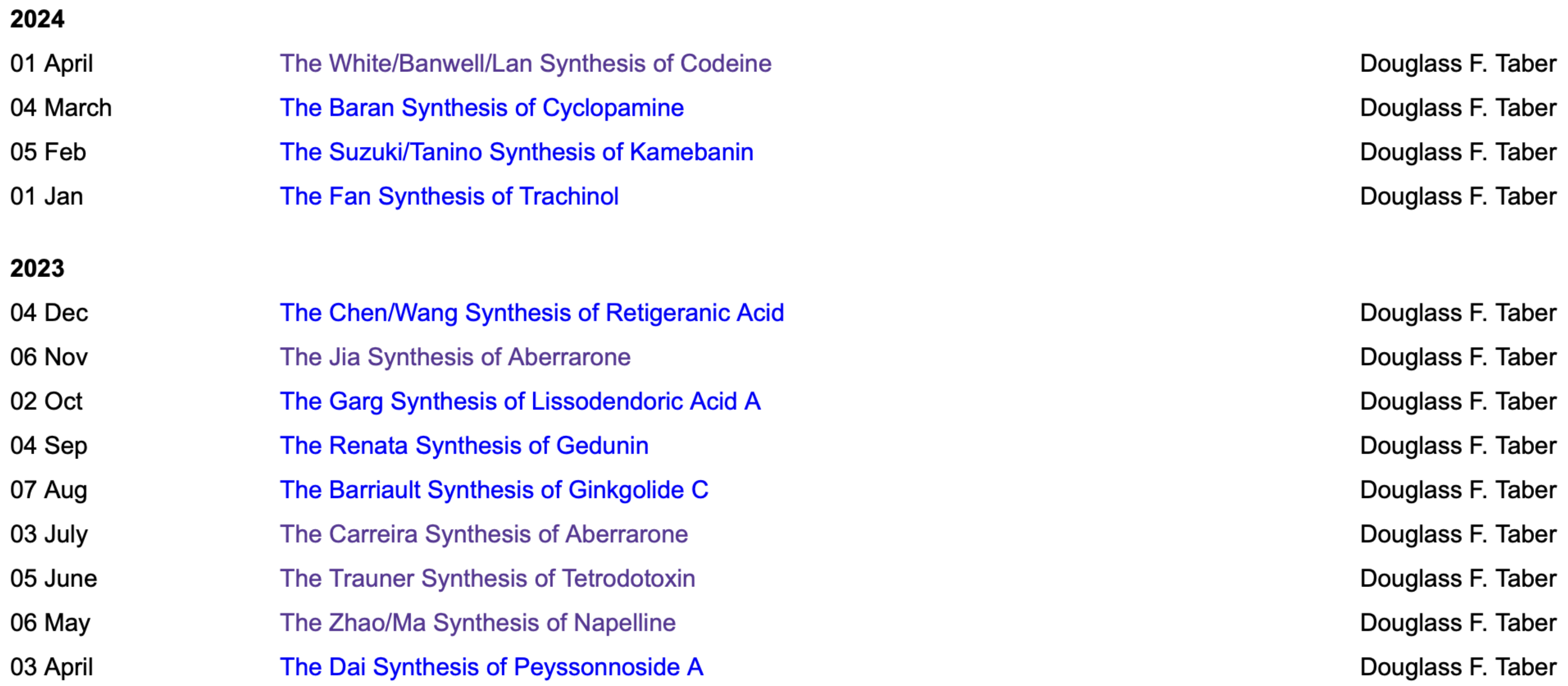
Same with requests



```python
import requests

# The URL for caffeine's molecular structure in PubChem
url = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/caffeine/JSON'

# Use requests to get the data from the URL
response = requests.get(url)

# Convert the JSON response content to a Python dictionary
data = response.json()
```

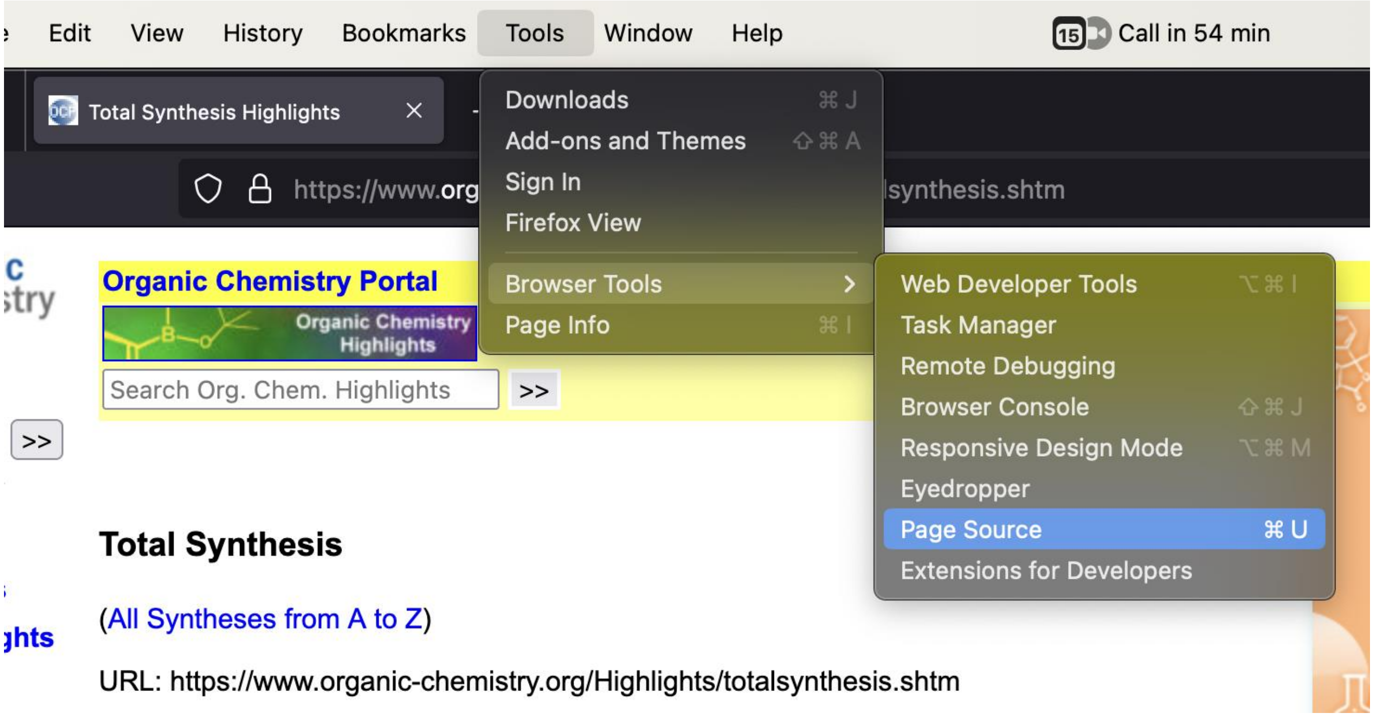
Web scraping – an example

- https://www.organic-chemistry.org/Highlights/totalsynthesis.shtm



2024		
01 April	The White/Banwell/Lan Synthesis of Codeine	Douglass F. Taber
04 March	The Baran Synthesis of Cyclopamine	Douglass F. Taber
05 Feb	The Suzuki/Tanino Synthesis of Kamebanin	Douglass F. Taber
01 Jan	The Fan Synthesis of Trachinol	Douglass F. Taber
2023		
04 Dec	The Chen/Wang Synthesis of Retigeranic Acid	Douglass F. Taber
06 Nov	The Jia Synthesis of Aberrarone	Douglass F. Taber
02 Oct	The Garg Synthesis of Lissodendoric Acid A	Douglass F. Taber
04 Sep	The Renata Synthesis of Gedunin	Douglass F. Taber
07 Aug	The Barriault Synthesis of Ginkgolide C	Douglass F. Taber
03 July	The Carreira Synthesis of Aberrarone	Douglass F. Taber
05 June	The Trauner Synthesis of Tetrodotoxin	Douglass F. Taber
06 May	The Zhao/Ma Synthesis of Napelline	Douglass F. Taber
03 April	The Dai Synthesis of Peyssonnoside A	Douglass F. Taber

First step – check out page source



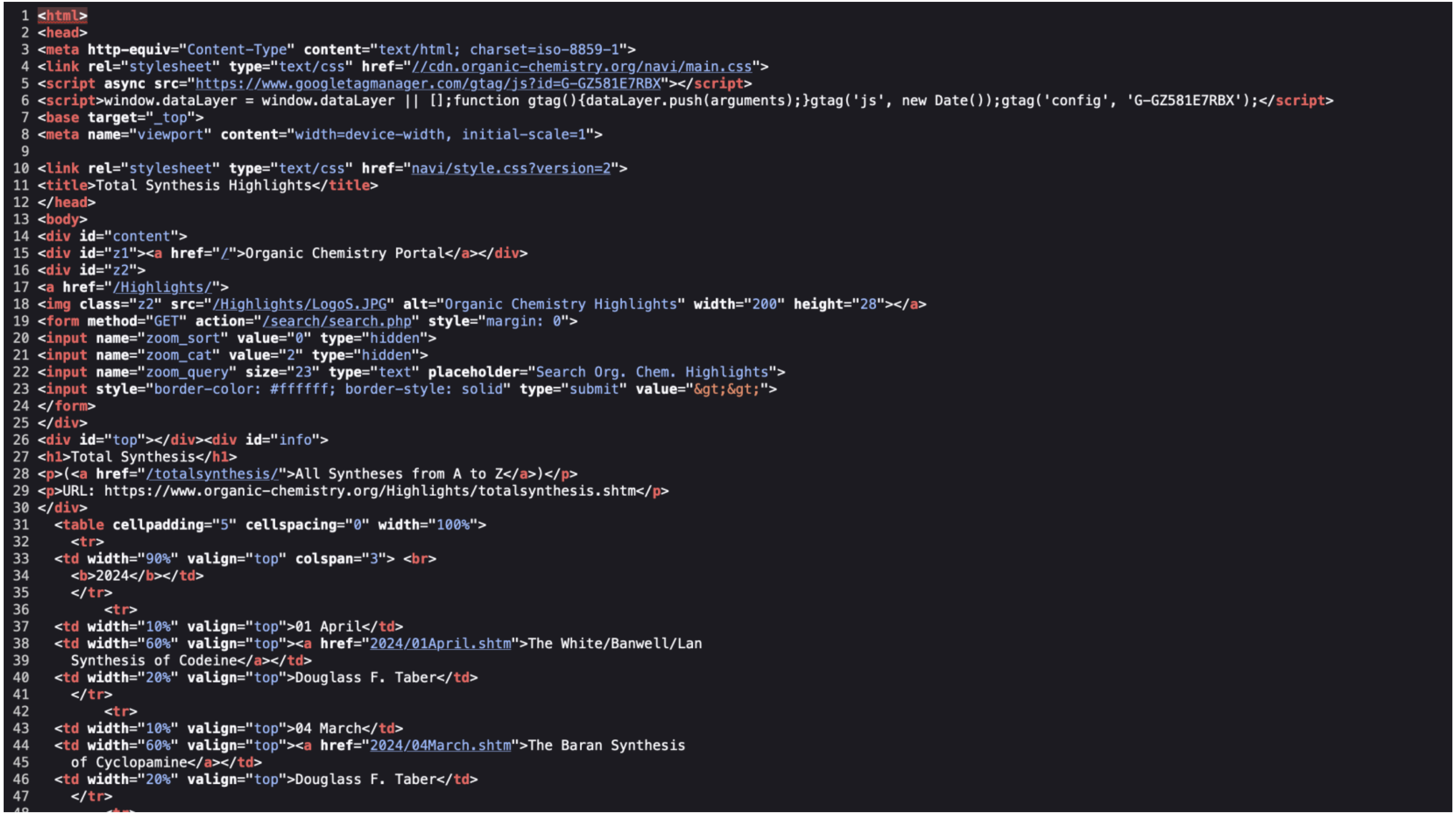
Total Synthesis

(All Syntheses from A to Z)

URL: https://www.organic-chemistry.org/Highlights/totalsynthesis.shtm

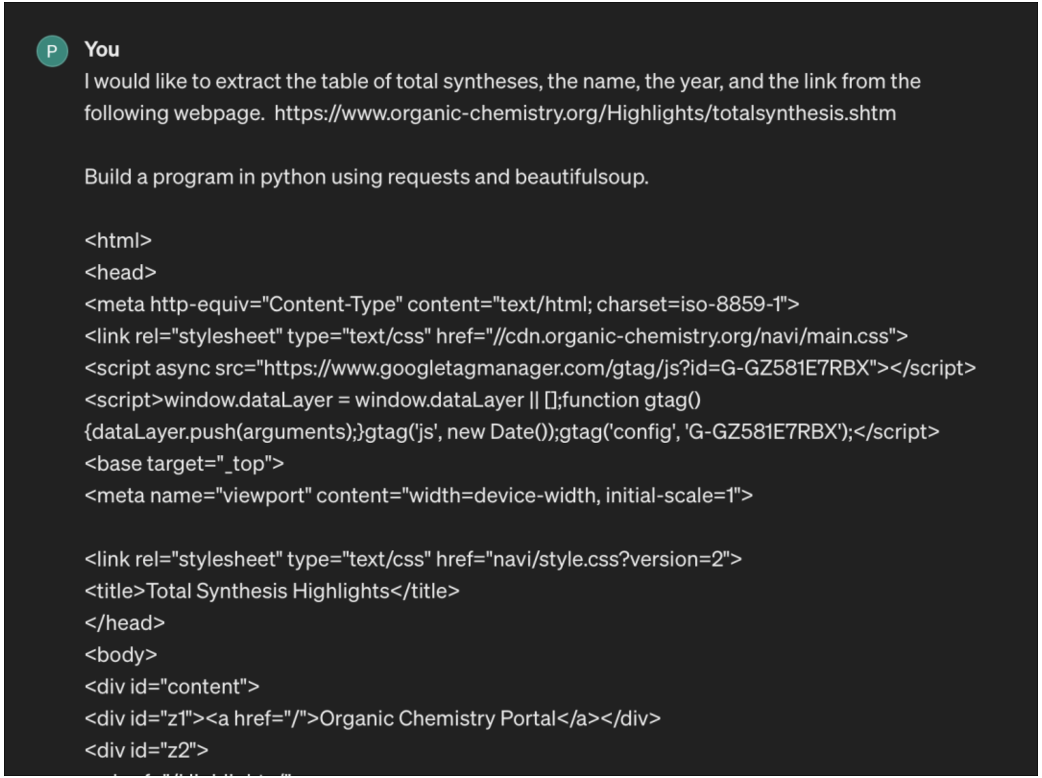
Alternatively, you could also just use requests to get the page source.
(the example above is with Firefox)

The source code



```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="//cdn.organic-chemistry.org/navi/main.css">
<script async src="https://www.googletagmanager.com/gtag/js?id=G-GZ581E7RBX"></script>
<script>window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}gtag('js', new Date());gtag('config', 'G-GZ581E7RBX');</script>
<base target="_top">
<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" type="text/css" href="navi/style.css?version=2">
<title>Total Synthesis Highlights</title>
</head>
<body>
<div id="content">
<div id="z1"><a href="/">Organic Chemistry Portal</a></div>
<div id="z2">
<a href="/Highlights/">
<img class="z2" src="/Highlights/LogoS.JPG" alt="Organic Chemistry Highlights" width="200" height="28"></a>
<form method="GET" action="/search/search.php" style="margin: 0">
<input name="zoom_sort" value="0" type="hidden">
<input name="zoom_cat" value="2" type="hidden">
<input name="zoom_query" size="23" type="text" placeholder="Search Org. Chem. Highlights">
<input style="border-color: #ffffff; border-style: solid" type="submit" value="&gt;&gt;">
</form>
</div>
<div id="top"></div><div id="info">
<h1>Total Synthesis</h1>
<p>(<a href="/totalsynthesis/">All Syntheses from A to Z</a>)</p>
<p>URL: https://www.organic-chemistry.org/Highlights/totalsynthesis.shtm</p>
</div>
  <table cellpadding="5" cellspacing="0" width="100%">
    <tr>
  <td width="90%" valign="top" colspan="3"> <br>
     <b>2024</b></td>
    </tr>
        <tr>
  <td width="10%" valign="top">01 April</td>
  <td width="60%" valign="top"><a href="2024/01April.shtm">The White/Banwell/Lan
    Synthesis of Codeine</a></td>
  <td width="20%" valign="top">Douglass F. Taber</td>
    </tr>
        <tr>
  <td width="10%" valign="top">04 March</td>
  <td width="60%" valign="top"><a href="2024/04March.shtm">The Baran Synthesis
    of Cyclopamine</a></td>
  <td width="20%" valign="top">Douglass F. Taber</td>
    </tr>
```

Ask ChatGPT for initial code (might not be perfect in the first try, but a good way to start)

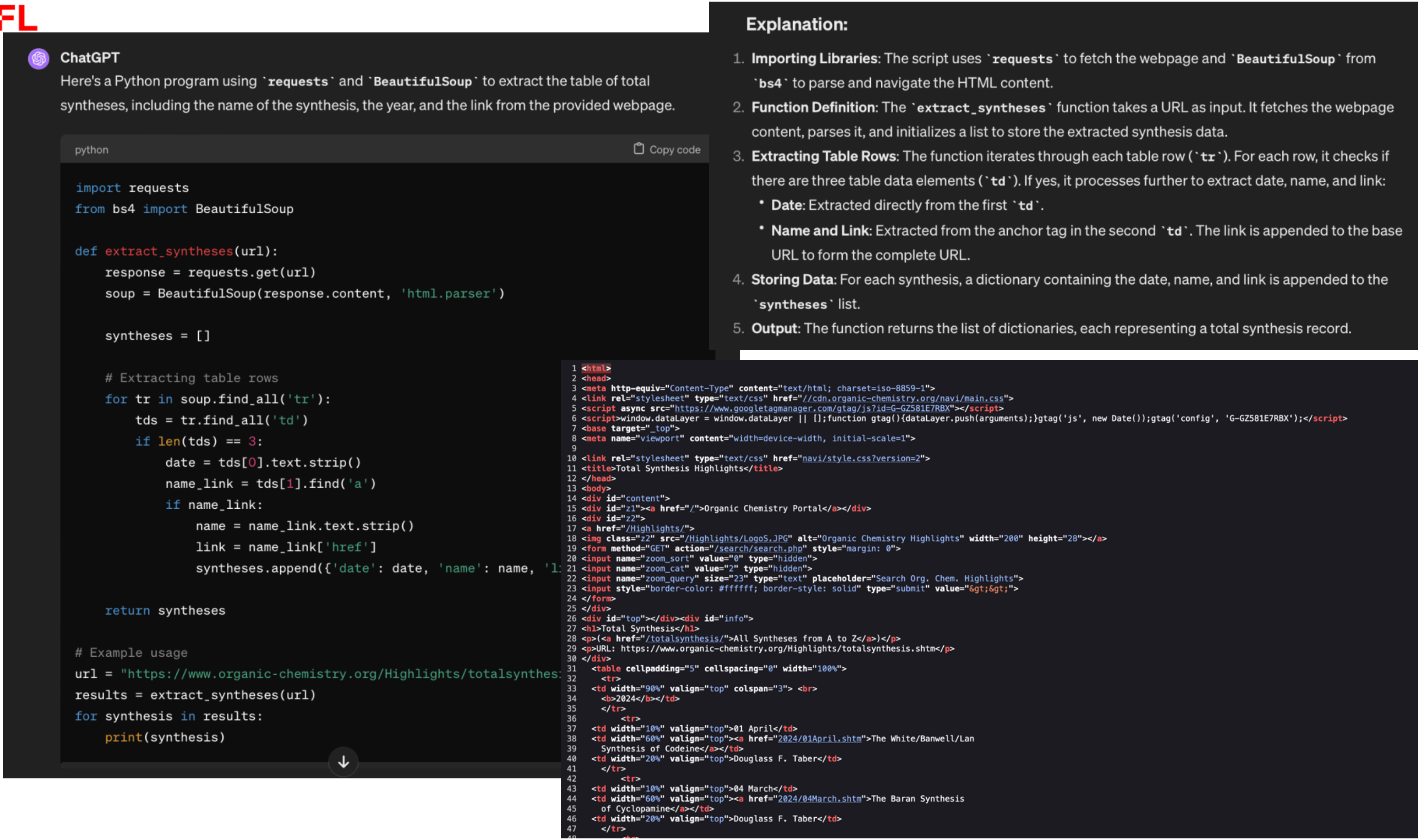


You

I would like to extract the table of total syntheses, the name, the year, and the link from the following webpage. https://www.organic-chemistry.org/Highlights/totalsynthesis.shtm

Build a program in python using requests and beautifulsoup.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="//cdn.organic-chemistry.org/navi/main.css">
<script async src="https://www.googletagmanager.com/gtag/js?id=G-GZ581E7RBX"></script>
<script>window.dataLayer = window.dataLayer || [];function gtag()
{dataLayer.push(arguments);}gtag('js', new Date());gtag('config', 'G-GZ581E7RBX');</script>
<base target="_top">
<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" type="text/css" href="navi/style.css?version=2">
<title>Total Synthesis Highlights</title>
</head>
<body>
<div id="content">
<div id="z1"><a href="/">Organic Chemistry Portal</a></div>
<div id="z2">
```



ChatGPT

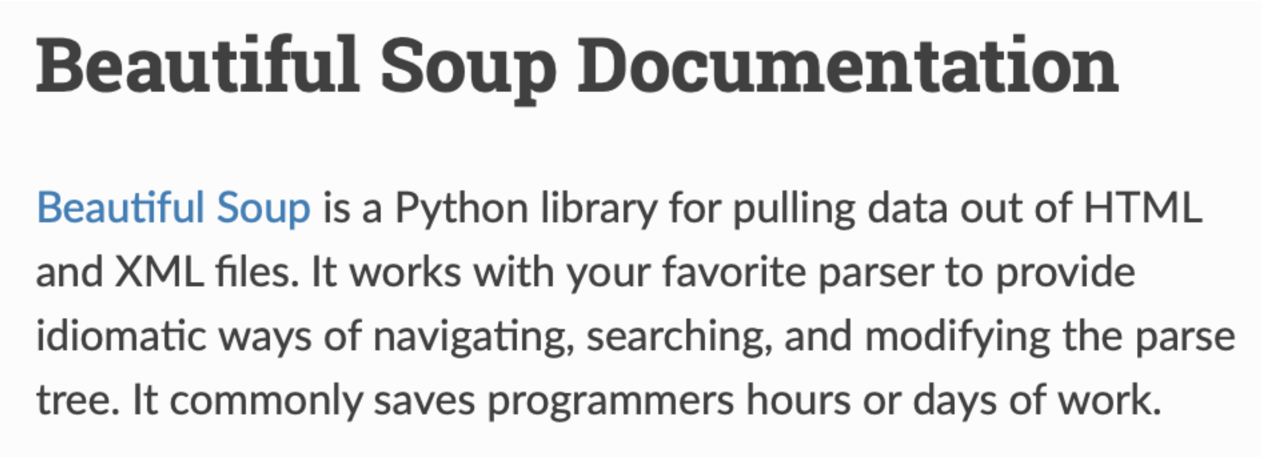
Here's a Python program using `requests` and `BeautifulSoup` to extract the table of total syntheses, including the name of the synthesis, the year, and the link from the provided webpage.

```python
import requests
from bs4 import BeautifulSoup

def extract_syntheses(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    syntheses = []

    # Extracting table rows
    for tr in soup.find_all('tr'):
        tds = tr.find_all('td')
        if len(tds) == 3:
            date = tds[0].text.strip()
            name_link = tds[1].find('a')
            if name_link:
                name = name_link.text.strip()
                link = name_link['href']
                syntheses.append({'date': date, 'name': name, 'l

    return syntheses

# Example usage
url = "https://www.organic-chemistry.org/Highlights/totalsynthes
results = extract_syntheses(url)
for synthesis in results:
    print(synthesis)
```

Explanation:

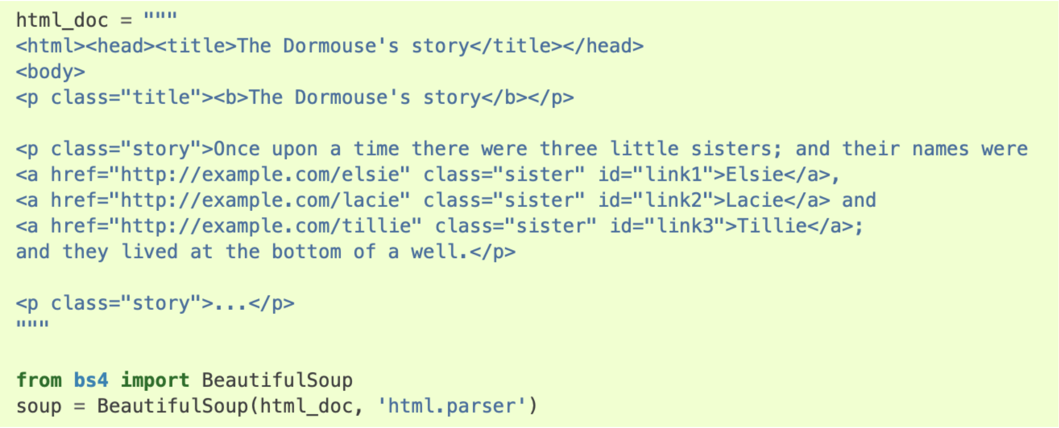
1. **Importing Libraries**: The script uses `requests` to fetch the webpage and `BeautifulSoup` from `bs4` to parse and navigate the HTML content.
2. **Function Definition**: The `extract_syntheses` function takes a URL as input. It fetches the webpage content, parses it, and initializes a list to store the extracted synthesis data.
3. **Extracting Table Rows**: The function iterates through each table row (`tr`). For each row, it checks if there are three table data elements (`td`). If yes, it processes further to extract date, name, and link:
   - **Date**: Extracted directly from the first `td`.
   - **Name and Link**: Extracted from the anchor tag in the second `td`. The link is appended to the base URL to form the complete URL.
4. **Storing Data**: For each synthesis, a dictionary containing the date, name, and link is appended to the `syntheses` list.
5. **Output**: The function returns the list of dictionaries, each representing a total synthesis record.

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="//cdn.organic-chemistry.org/navi/main.css">
<script async src="https://www.googletagmanager.com/gtag/js?id=G-GZ581E7RBX"></script>
<script>window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}gtag('js', new Date());gtag('config', 'G-GZ581E7RBX');</script>
<base target="_top">
<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" type="text/css" href="navi/style.css?version=2">
<title>Total Synthesis Highlights</title>
</head>
<body>
<div id="content">
<div id="z1"><a href="/">Organic Chemistry Portal</a></div>
<div id="z2">
<a href="/Highlights/">
<img class="z2" src="/Highlights/LogoS.JPG" alt="Organic Chemistry Highlights" width="200" height="28"></a>
<form method="GET" action="/search/search.php" style="margin: 0">
<input name="zoom_sort" value="0" type="hidden">
<input name="zoom_cat" value="2" type="hidden">
<input name="zoom_query" size="23" type="text" placeholder="Search Org. Chem. Highlights">
<input style="border-color: #ffffff; border-style: solid" type="submit" value="&gt;&gt;">
</form>
</div>
<div id="top"></div><div id="info">
<h1>Total Synthesis</h1>
<p>(<a href="/totalsynthesis/">All Syntheses from A to Z</a>)</p>
<p>URL: https://www.organic-chemistry.org/Highlights/totalsynthesis.shtm</p>
</div>
  <table cellpadding="5" cellspacing="0" width="100%">
    <tr>
  <td width="90%" valign="top" colspan="3"> <br>
    <b>2024</b></td>
    </tr>
        <tr>
  <td width="10%" valign="top">01 April</td>
  <td width="60%" valign="top"><a href="2024/01April.shtm">The White/Banwell/Lan
    Synthesis of Codeine</a></td>
  <td width="20%" valign="top">Douglass F. Taber</td>
    </tr>
        <tr>
  <td width="10%" valign="top">04 March</td>
  <td width="60%" valign="top"><a href="2024/04March.shtm">The Baran Synthesis
    of Cyclopamine</a></td>
  <td width="20%" valign="top">Douglass F. Taber</td>
    </tr>
```



Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.



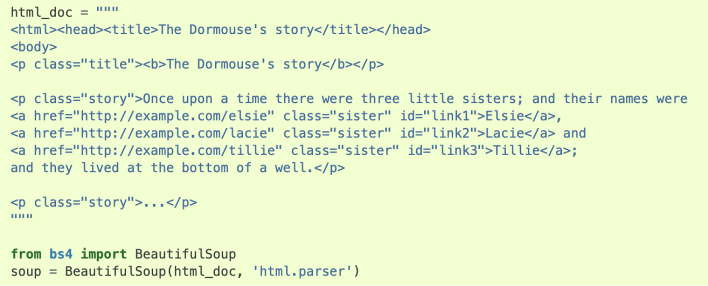
```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""


from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```

But the html document could be the output of a request to a webpage.

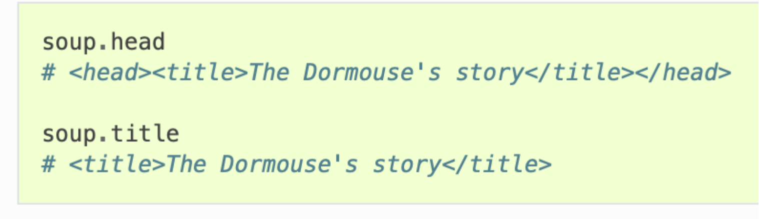
https://beautiful-soup-4.readthedocs.io/en/latest/

pip install beautifulsoup4

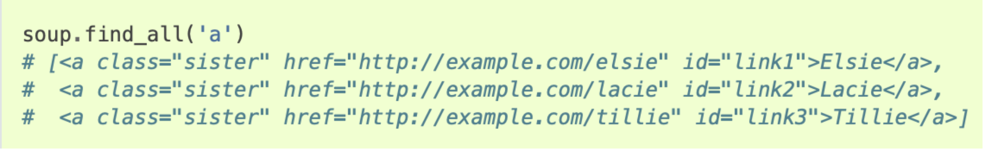
Going down the tree



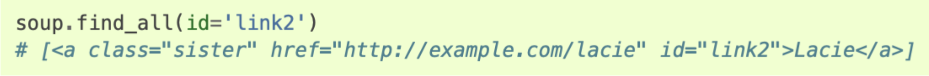
```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```



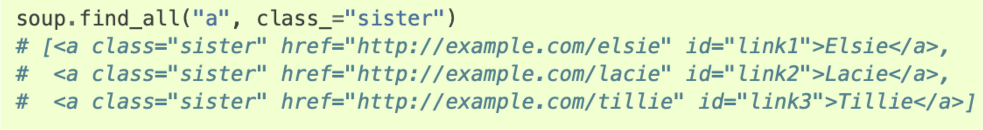
```
soup.head
# <head><title>The Dormouse's story</title></head>

soup.title
# <title>The Dormouse's story</title>
```



```
soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```



```
soup.find_all(id='link2')
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```



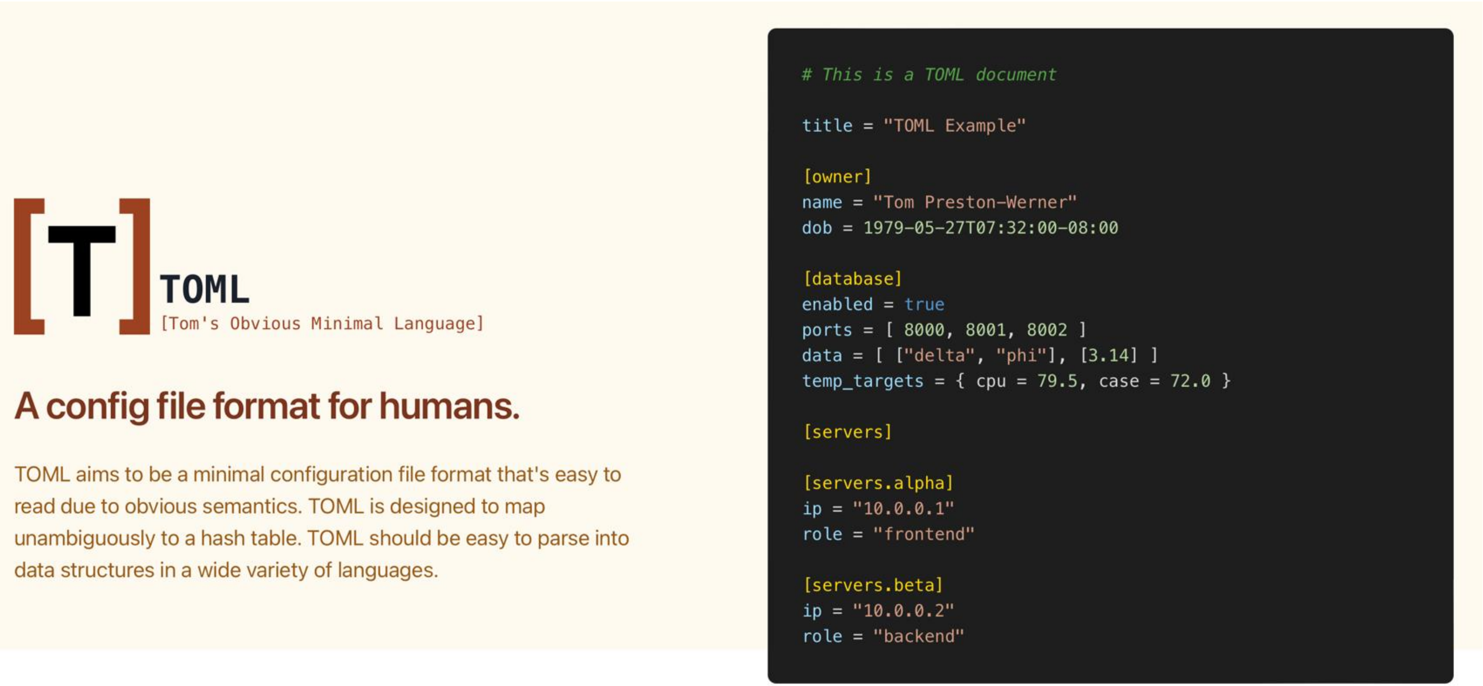
```
soup.find_all("a", class_="sister")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

And plenty of other methods,
that lets you navigate and
identify specific content of a webpage.

Back to Python packages

pyproject.toml - What does TOML mean?

https://toml.io/en/



TOML

[Tom's Obvious Minimal Language]

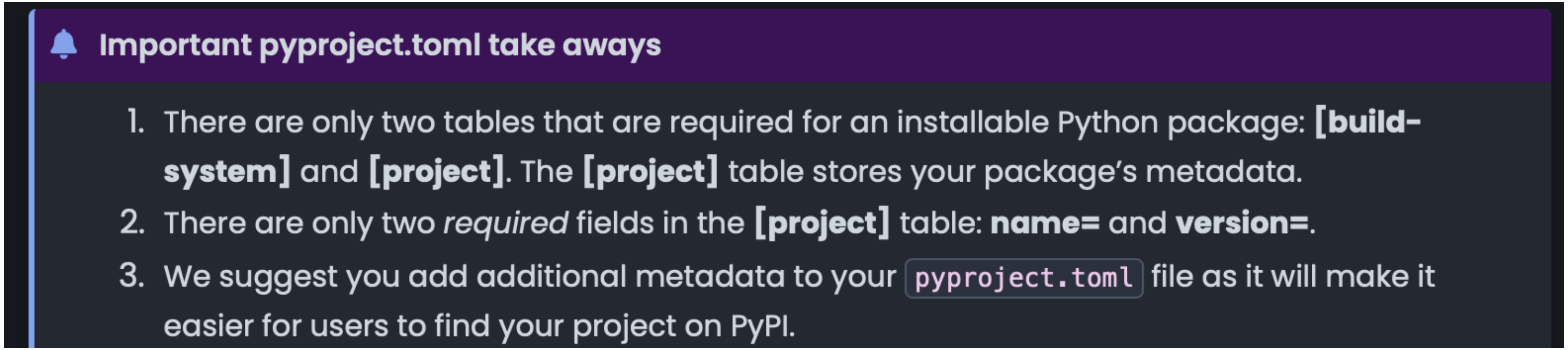
A config file format for humans.

TOML aims to be a minimal configuration file format that's easy to read due to obvious semantics. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages.

```toml
# This is a TOML document

title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3.14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"

[servers.beta]
ip = "10.0.0.2"
role = "backend"
```

pyproject.toml

- https://www.pyopensci.org/python-package-guide/package-structure-code/pyproject-toml-python-package-metadata.html

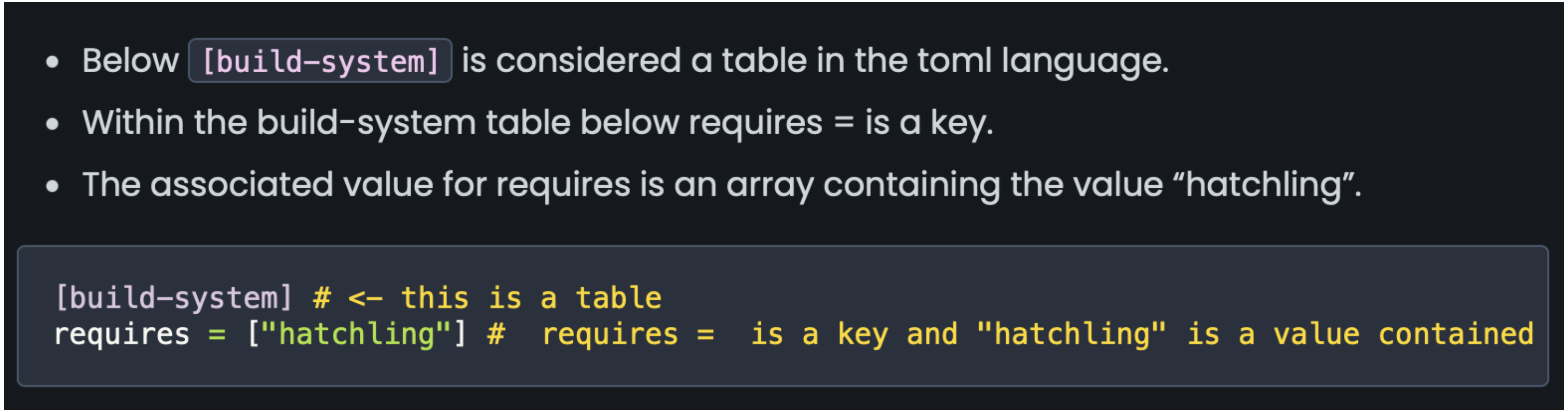


Important pyproject.toml take aways

1. There are only two tables that are required for an installable Python package: **[build-system]** and **[project]**. The **[project]** table stores your package's metadata.
2. There are only two *required* fields in the **[project]** table: **name=** and **version=**.
3. We suggest you add additional metadata to your `pyproject.toml` file as it will make it easier for users to find your project on PyPI.

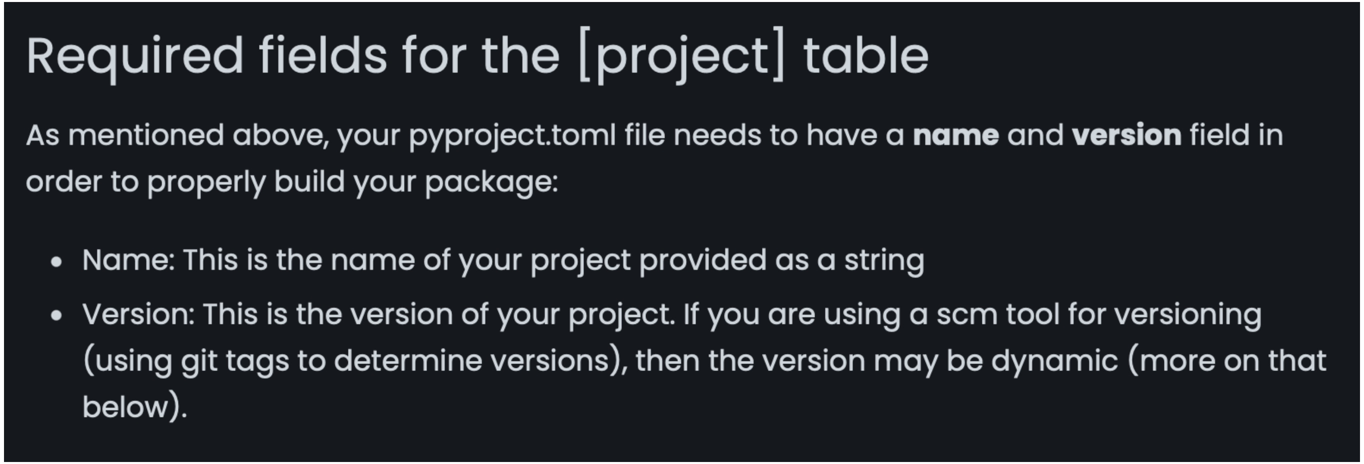
2 "tables" required [build-system] and [project]

pyproject.toml – tables



- Below `[build-system]` is considered a table in the toml language.
- Within the build-system table below requires = is a key.
- The associated value for requires is an array containing the value “hatchling”.

```
[build-system] # <- this is a table
requires = ["hatchling"] #  requires =  is a key and "hatchling" is a value contained
```



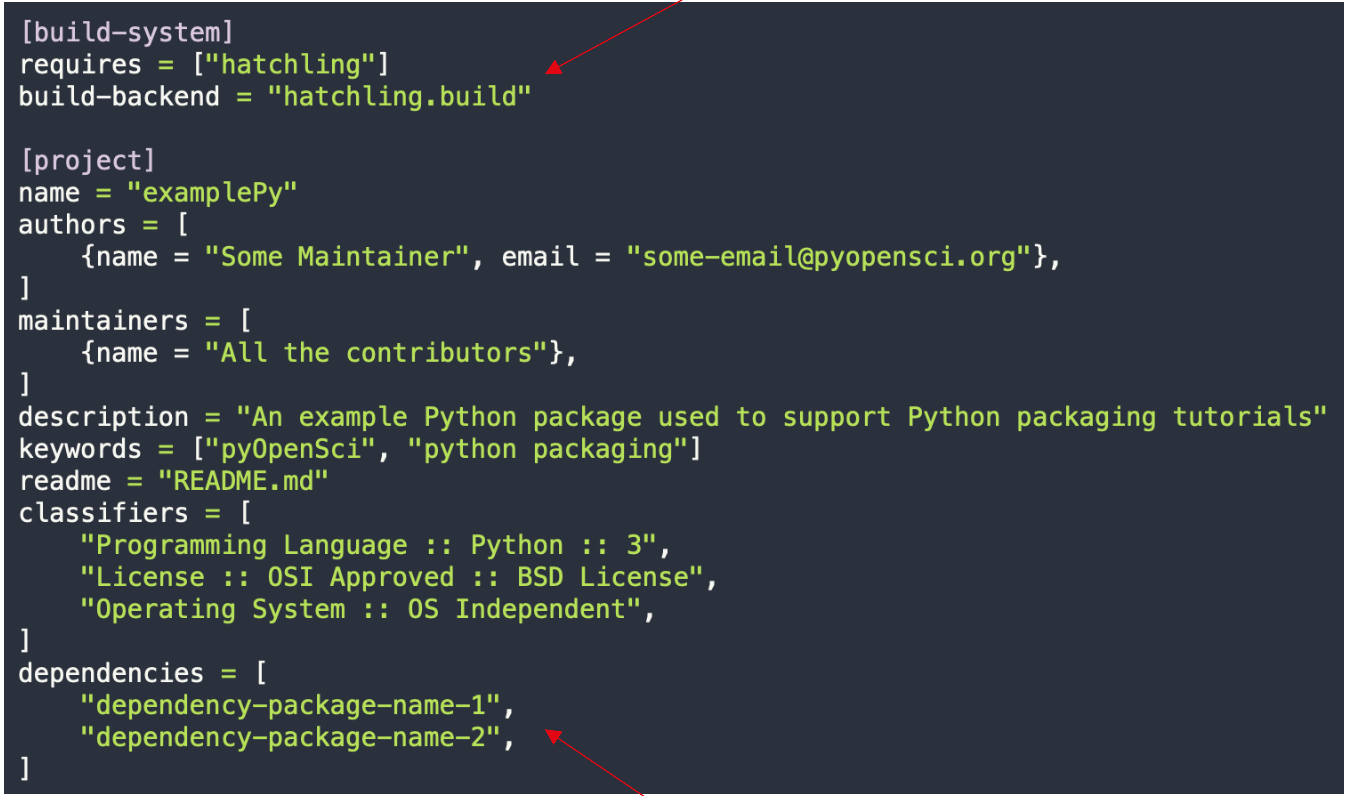
Required fields for the [project] table

As mentioned above, your pyproject.toml file needs to have a **name** and **version** field in order to properly build your package:

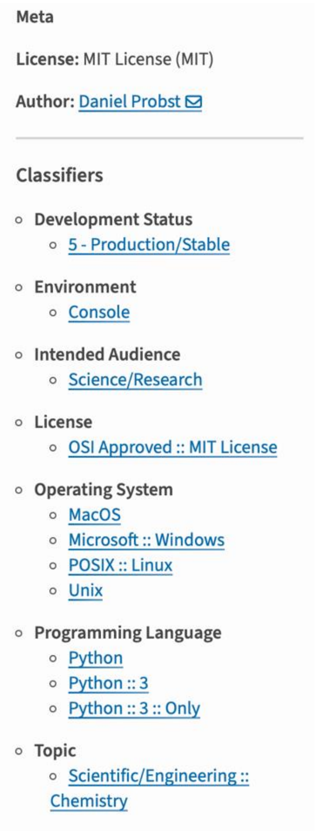
- Name: This is the name of your project provided as a string
- Version: This is the version of your project. If you are using a scm tool for versioning (using git tags to determine versions), then the version may be dynamic (more on that below).

Recommended build system
→ Comes with the liac copier template

Additional meta data:



```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "examplePy"
authors = [
    {name = "Some Maintainer", email = "some-email@pyopensci.org"},
]
maintainers = [
    {name = "All the contributors"},
]
description = "An example Python package used to support Python packaging tutorials"
keywords = ["pyOpenSci", "python packaging"]
readme = "README.md"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: BSD License",
    "Operating System :: OS Independent",
]
dependencies = [
    "dependency-package-name-1",
    "dependency-package-name-2",
]
```



Meta

License: MIT License (MIT)

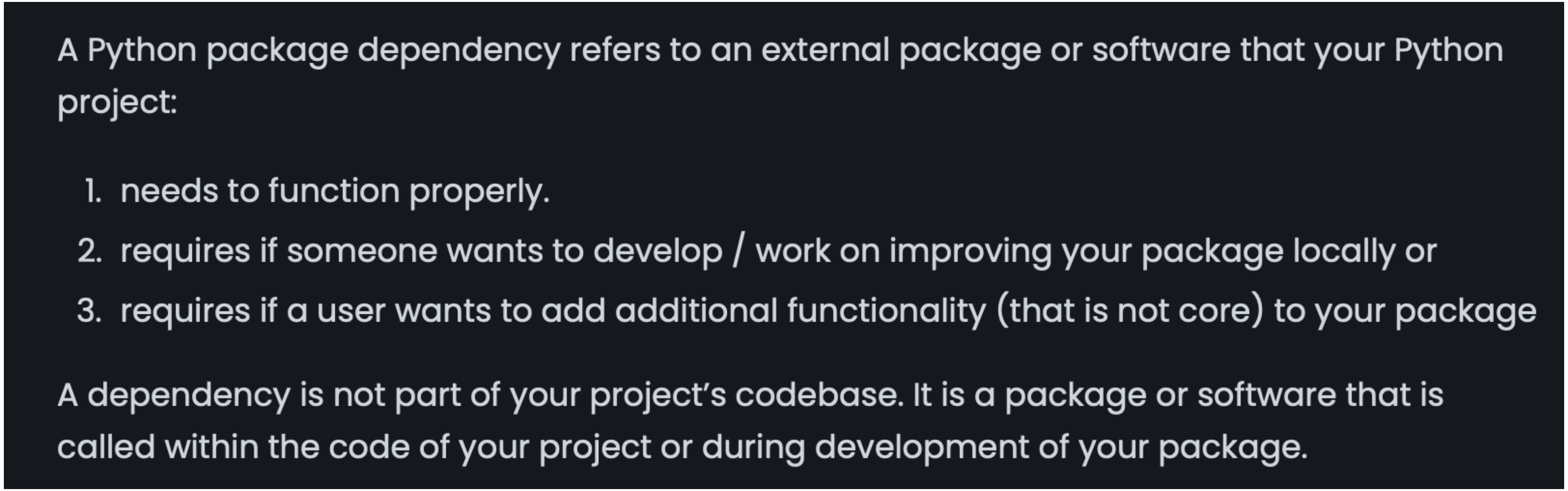
Author: Daniel Probst

Classifiers

- **Development Status**
  - 5 - Production/Stable
- **Environment**
  - Console
- **Intended Audience**
  - Science/Research
- **License**
  - OSI Approved :: MIT License
- **Operating System**
  - MacOS
  - Microsoft :: Windows
  - POSIX :: Linux
  - Unix
- **Programming Language**
  - Python
  - Python :: 3
  - Python :: 3 :: Only
- **Topic**
  - Scientific/Engineering :: Chemistry

For example, numpy==1.24.

What is a package dependency?



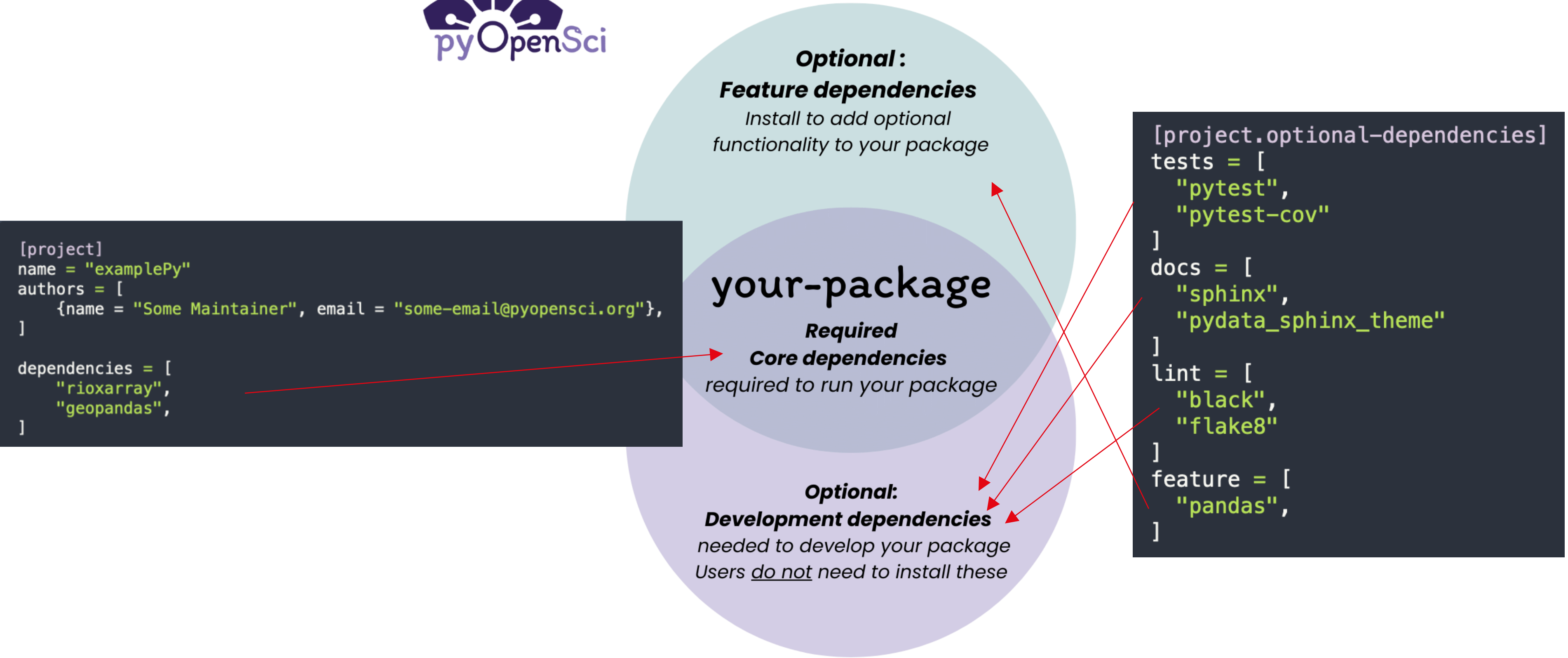
A Python package dependency refers to an external package or software that your Python project:

1. needs to function properly.
2. requires if someone wants to develop / work on improving your package locally or
3. requires if a user wants to add additional functionality (that is not core) to your package

A dependency is not part of your project's codebase. It is a package or software that is called within the code of your project or during development of your package.

https://www.pyopensci.org/python-package-guide/package-structure-code/declare-dependencies.html

Python Package Dependency Types



```
[project]
name = "examplePy"
authors = [
    {name = "Some Maintainer", email = "some-email@pyopensci.org"},
]

dependencies = [
    "rioxarray",
    "geopandas",
]
```

Optional :
Feature dependencies
Install to add optional functionality to your package

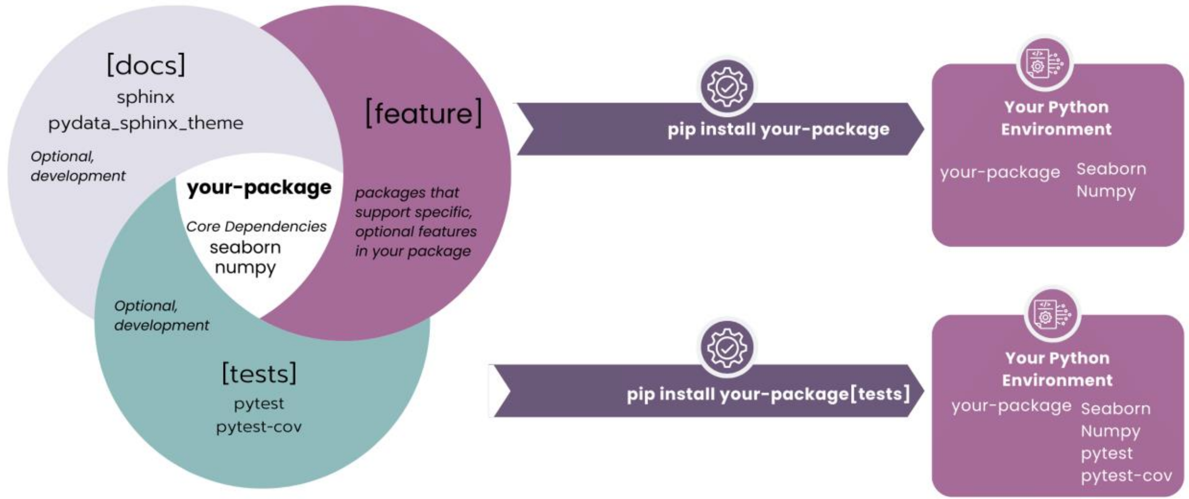
your-package

Required
Core dependencies
required to run your package

Optional:
Development dependencies
needed to develop your package
Users do not need to install these

```
[project.optional-dependencies]
tests = [
  "pytest",
  "pytest-cov"
]
docs = [
  "sphinx",
  "pydata_sphinx_theme"
]
lint = [
  "black",
  "flake8"
]
feature = [
  "pandas",
]
```

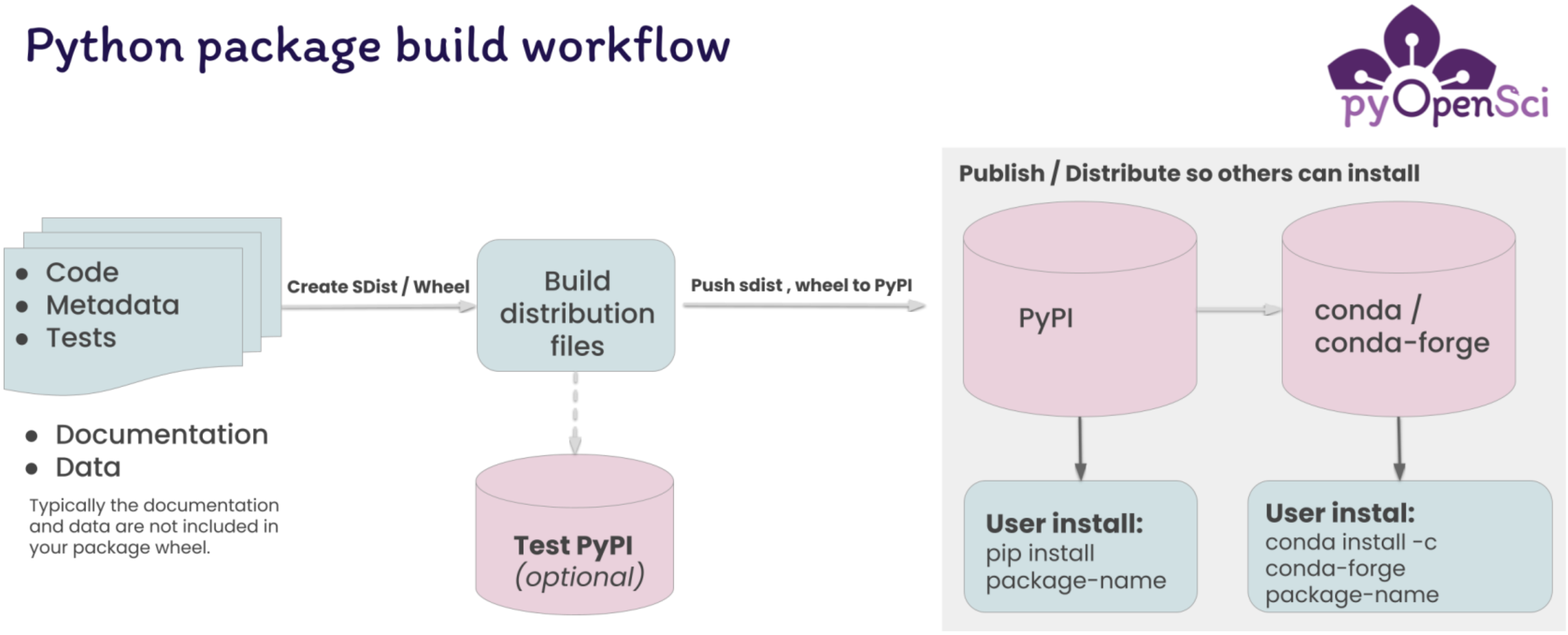
Optional vs. Required Python Package Dependencies



Development Dependencies	Feature Dependencies	Core Dependencies
Optional packages / code needed develop your package (run tests, format code, etc)	**Optional packages that a user can install to support additional functionality in your package**	**Packages / code <u>required</u> to run your code that need to be installed into a users environment when your package is installed**

- https://www.pyopensci.org/python-package-guide/package-structure-code/declare-dependencies.html#install-dependency-groups

You could also publish your package on pypi/conda-forge (advanced!)



Python package build workflow

pyOpenSci

- Code
- Metadata
- Tests

Create SDist / Wheel

Build distribution files

Push sdist , wheel to PyPI

- Documentation
- Data

Typically the documentation and data are not included in your package wheel.

Test PyPI *(optional)*

Publish / Distribute so others can install

PyPI

conda / conda-forge

User install: pip install package-name

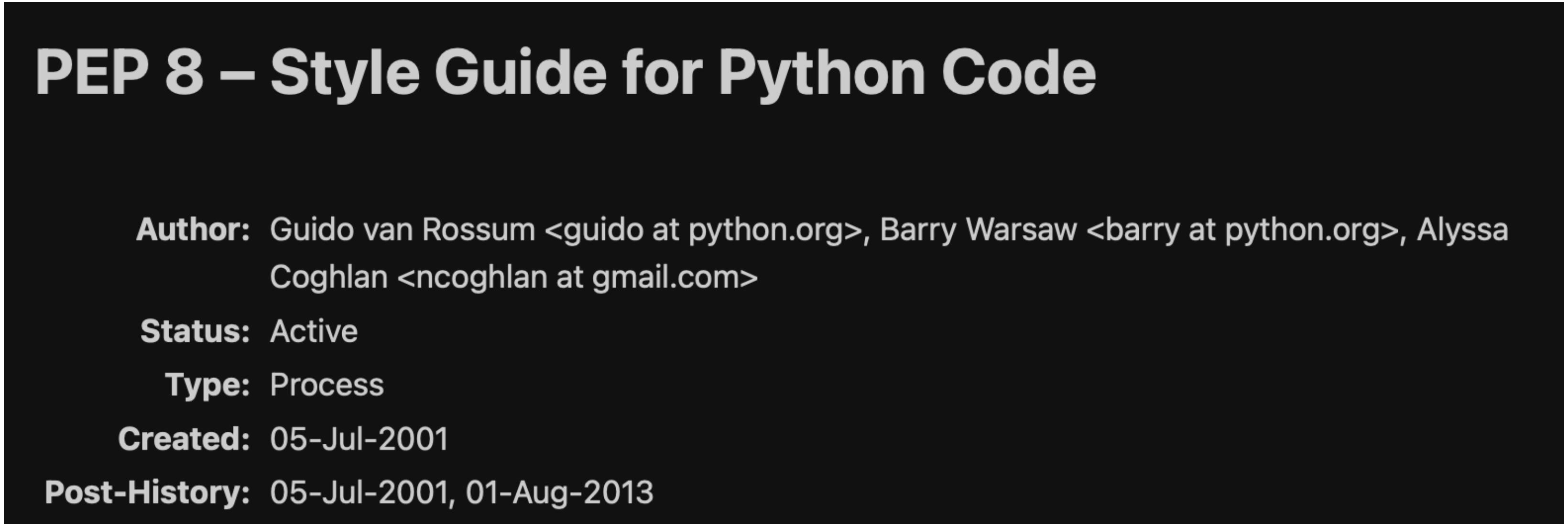
User instal: conda install -c conda-forge package-name

https://www.pyopensci.org/python-package-guide/package-structure-code/publish-python-package-pypi-conda.html

Having a consistent code format - helper tools

Code style

- https://peps.python.org/pep-0008/



PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>

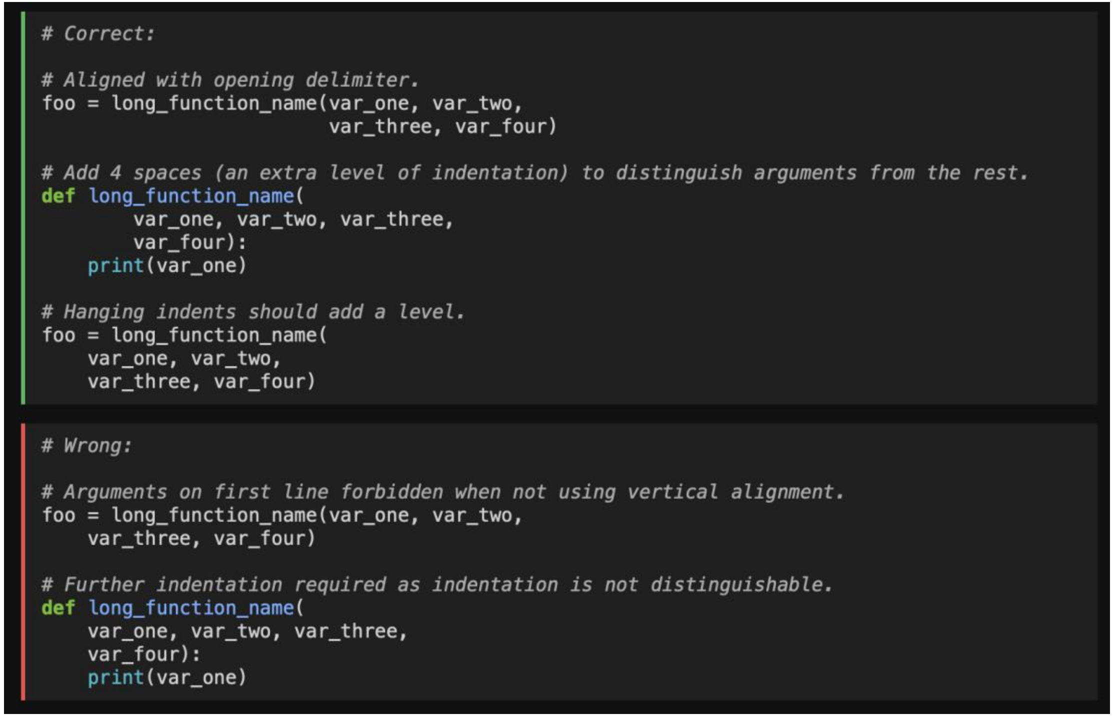
Status: Active

Type: Process

Created: 05-Jul-2001

Post-History: 05-Jul-2001, 01-Aug-2013

Lot’s of correct vs wrong examples.

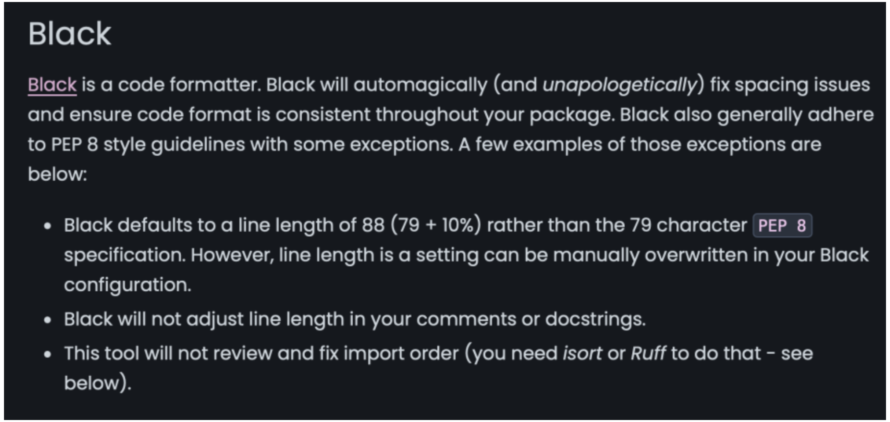


```
# Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

```
# Wrong:

# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Both code examples work, but one is much more readable than the other.

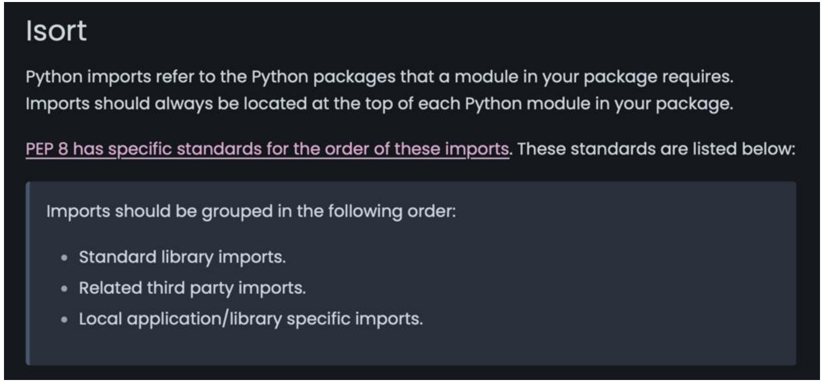
How to check style? → Linters



Black

Black is a code formatter. Black will automagically (and *unapologetically*) fix spacing issues and ensure code format is consistent throughout your package. Black also generally adhere to PEP 8 style guidelines with some exceptions. A few examples of those exceptions are below:

- Black defaults to a line length of 88 (79 + 10%) rather than the 79 character `PEP 8` specification. However, line length is a setting can be manually overwritten in your Black configuration.
- Black will not adjust line length in your comments or docstrings.
- This tool will not review and fix import order (you need *isort* or *Ruff* to do that - see below).

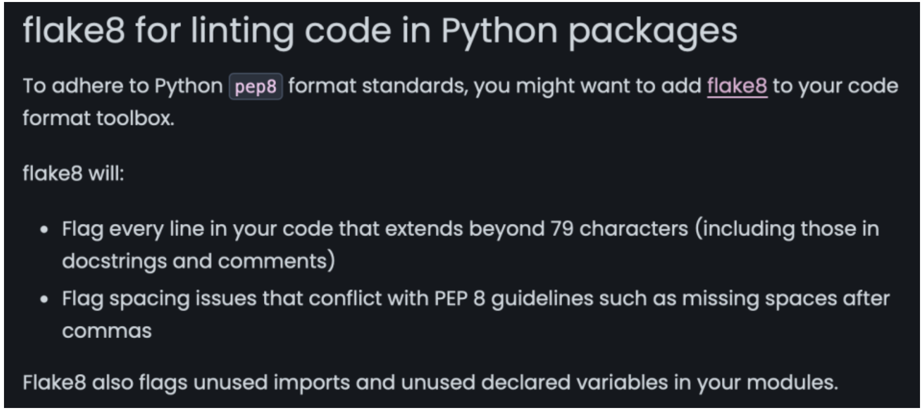


Isort

Python imports refer to the Python packages that a module in your package requires. Imports should always be located at the top of each Python module in your package.

PEP 8 has specific standards for the order of these imports. These standards are listed below:

> Imports should be grouped in the following order:
>
> - Standard library imports.
> - Related third party imports.
> - Local application/library specific imports.



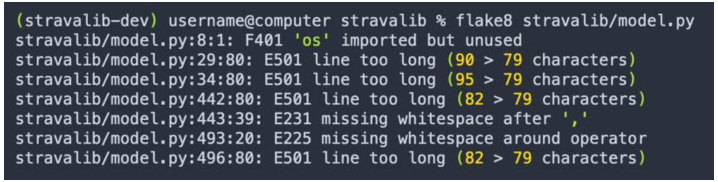
flake8 for linting code in Python packages

To adhere to Python `pep8` format standards, you might want to add flake8 to your code format toolbox.

flake8 will:

- Flag every line in your code that extends beyond 79 characters (including those in docstrings and comments)
- Flag spacing issues that conflict with PEP 8 guidelines such as missing spaces after commas

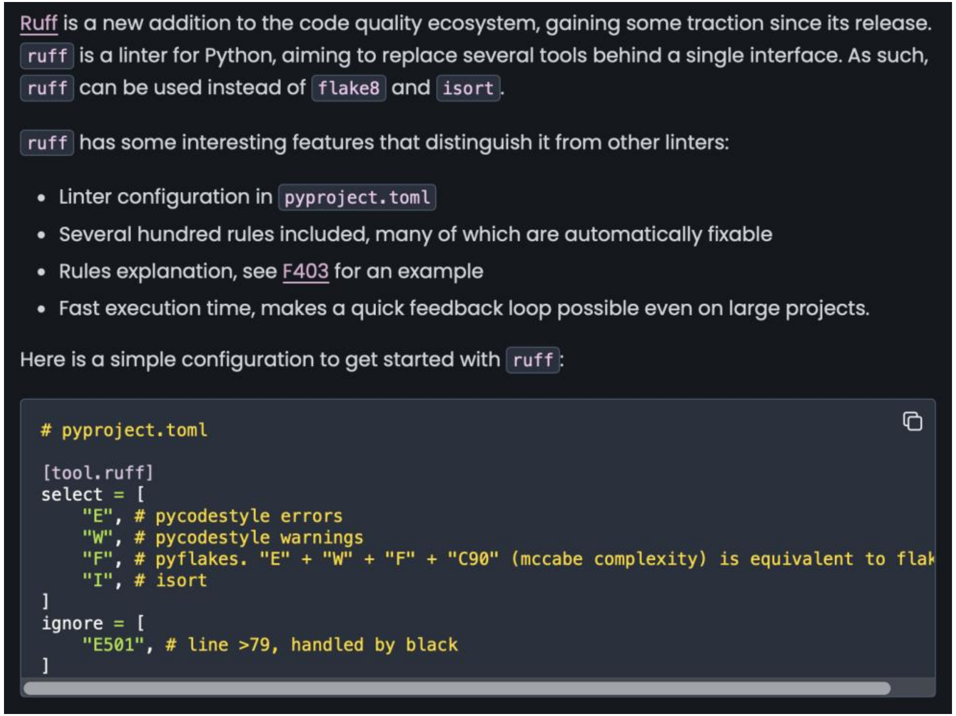
Flake8 also flags unused imports and unused declared variables in your modules.



```
(stravalib-dev) username@computer stravalib % flake8 stravalib/model.py
stravalib/model.py:8:1: F401 'os' imported but unused
stravalib/model.py:29:80: E501 line too long (90 > 79 characters)
stravalib/model.py:34:80: E501 line too long (95 > 79 characters)
stravalib/model.py:442:80: E501 line too long (82 > 79 characters)
stravalib/model.py:443:39: E231 missing whitespace after ','
stravalib/model.py:493:20: E225 missing whitespace around operator
stravalib/model.py:496:80: E501 line too long (82 > 79 characters)
```

https://www.pyopensci.org/python-package-guide/package-structure-code/code-style-linting-format.html

Ruff – new addition to the ecosystem



Ruff is a new addition to the code quality ecosystem, gaining some traction since its release. `ruff` is a linter for Python, aiming to replace several tools behind a single interface. As such, `ruff` can be used instead of `flake8` and `isort`.

`ruff` has some interesting features that distinguish it from other linters:

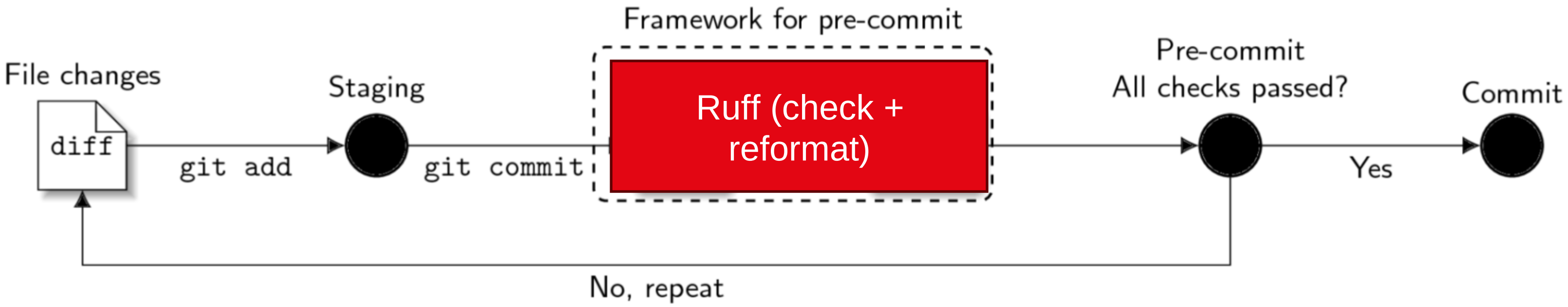
- Linter configuration in `pyproject.toml`
- Several hundred rules included, many of which are automatically fixable
- Rules explanation, see F403 for an example
- Fast execution time, makes a quick feedback loop possible even on large projects.

Here is a simple configuration to get started with `ruff`:

```toml
# pyproject.toml

[tool.ruff]
select = [
    "E", # pycodestyle errors
    "W", # pycodestyle warnings
    "F", # pyflakes. "E" + "W" + "F" + "C90" (mccabe complexity) is equivalent to flak
    "I", # isort
]
ignore = [
    "E501", # line >79, handled by black
]
```

→ Rules set in the pyproject.toml file

Pre-commit hooks (advanced!) – only allow to commit new code, if code format is ok.

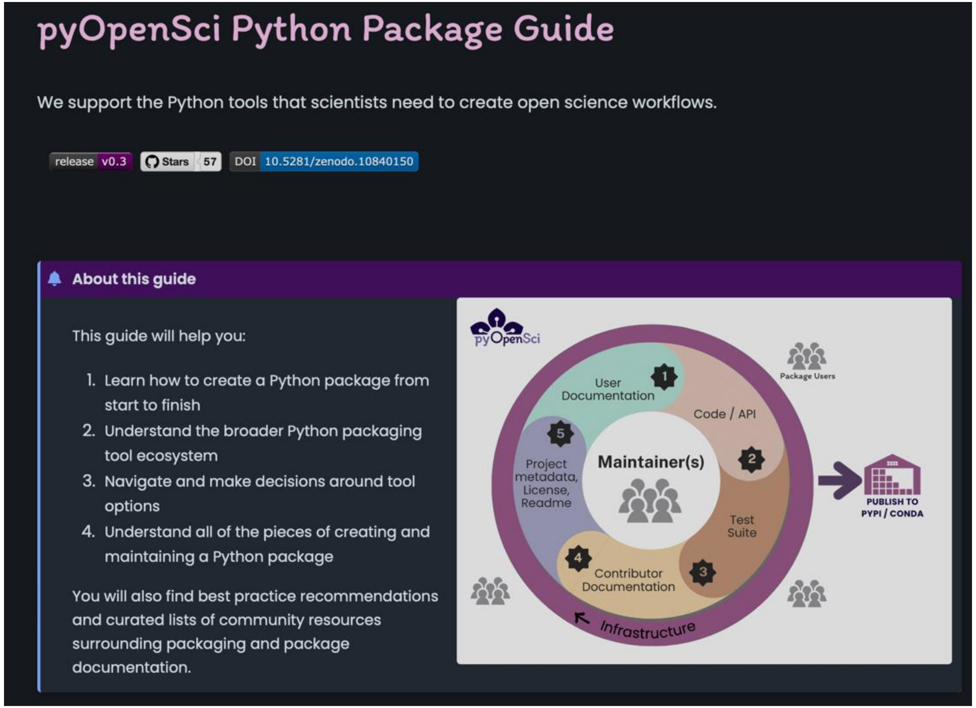


https://www.pyopensci.org/python-package-guide/package-structure-code/code-style-linting-format.html#use-pre-commit-hooks-to-run-code-formatters-and-linters-on-commits

https://github.com/schwallergroup/copier-liac/tree/main.
(template with linters + precommit hooks, advanced!!)

Awesome resource (credits to them)

- https://www.pyopensci.org/python-package-guide/index.html



pyOpenSci Python Package Guide

We support the Python tools that scientists need to create open science workflows.

About this guide

This guide will help you:

1. Learn how to create a Python package from start to finish
2. Understand the broader Python packaging tool ecosystem
3. Navigate and make decisions around tool options
4. Understand all of the pieces of creating and maintaining a Python package

You will also find best practice recommendations and curated lists of community resources surrounding packaging and package documentation.