FRI:DAY

FRIDAY code challnge

Street Address Automatic Parser

Jon Fernández Bedia

_

Table of contents

1. FRIDAY_code_challenge 3
1.1 Requisites 3
1.2 Start 3
1.3 Install dependencies 3
1.4 Add/remove dependency 3
2. Install dependency 3
3. Uninstall dependency 3
3.1 Run 3
3.2 Run tests 3
3.3 Format code 4
3.4 Check format 4
3.5 Check types 4
3.6 Generate docs 4
4. CODE CHALLENGE 5
4.1 Addressline 5
5. Techinal documentation 6
5.1 Code Structure 6
5.2 Documentation 6
5.3 Makefile 6
6. Modus operandi 8
7. Faced problems 9
7.1 Multiple SSH keys on same machine 9
7.2 Regex 9
7.3 Deepparse 9
7.4 Testing 9
7.5 Pipenv 10
8. Future implementations 11
8.1 Regular expressions 11
8.2 Testing 11
8.3 Database connection 11
8.4 Github actions 11
8.5 Code structure 11
8.6 Deepparse 11
8.7 Documentation 11

1. FRIDAY_code_challenge

Code challenge for FRIDAY, German branch of FRIDAY Insurance S.A.

To access project documentation go to https://jonfernandez12.github.io/FRIDAY_code_challenge/public/index.html

Jon Fernández Bedia

1.1 Requisites

- pipenv
- If you are using Visual Studio Code, you need to install Python extension

1.2 Start

`make start`

1.2.1 Git Hooks

- **Pre-commit**: before each commit the `make run-tests` command will be executed.

Note: If you want to ignore the pre-commit hook, add the flag `--no-verify` to the commit command.

1.3 Install dependencies

`make install-deps`

1.4 Add/remove dependency

```

# 2. Install dependency

make install dep={{dependency}} ver={{dependency_version}} # Example: make install dep=requests ver=2.26.0 make install-dev dep={{dependency}} ver={{dependency_version}} # Example: make install-dev dep=requests ver=2.26.0

# 3. Uninstall dependency

make uninstall dep={{dependency}} # Example: make uninstall dep=requests ```

## 3.1 Run

Runs main application, located in app/main.py.

`make run`

## 3.2 Run tests

Run all tests defined in app/services/tests/test.py.
```

```
make run-tests
```

3.3 Format code

Orders dependencies with Isort and formats the code using Black.

```
make format
```

3.4 Check format

Checks code format based on Isort, Black and Flake8 requirements.

```
make check-format
```

3.5 Check types

Checks if all types are well typed.

```
make check-types
```

3.6 Generate docs

Generate all documentation in HTML format and leaves it in public/index.html, open with your favorite browser.

```
make docs
```

4. CODE CHALLENGE

4.1 Addressline

An address provider returns addresses only with concatenated street names and numbers. Our own system on the other hand has separate fields for street name and street number.

Input: string of address

Output: string of street and string of street-number as JSON object

1. Write a simple program that does the task for the most simple cases, e.g.
2. `"Winterallee 3"` -> `{"street": "Winterallee", "housenumber": "3"}`
3. `"Musterstrasse 45"` -> `{"street": "Musterstrasse", "housenumber": "45"}`
4. `"Blaufeldweg 123B"` -> `{"street": "Blaufeldweg", "housenumber": "123B"}`
5. Consider more complicated cases
6. `"Am Bächle 23"` -> `{"street": "Am Bächle", "housenumber": "23"}`
7. `"Auf der Vogelwiese 23 b"` -> `{"street": "Auf der Vogelwiese", "housenumber": "23 b"}`
8. Consider other countries (complex cases)
9. `"4, rue de la revolution"` -> `{"street": "rue de la revolution", "housenumber": "4"}`
10. `"200 Broadway Av"` -> `{"street": "Broadway Av", "housenumber": "200"}`
11. `"Calle Aduana, 29"` -> `{"street": "Calle Aduana", "housenumber": "29"}`
12. `"Calle 39 No 1540"` -> `{"street": "Calle 39", "housenumber": "No 1540"}`

Your Task: Solve the task using your best programming skills and share the code in a public repository via a public hoster like Github. Contact us via jobs@friday.de.

When choosing your approach of implementation, please keep following things in mind:

- While we will try to evaluate all code that is sent to us, keep in mind that we will feel most comfortable with the tools we know:
  - Python
  - Pandas
  - PySpark
  - SQL
- We prefer quality over speed. It does not only matter if your solution yields correct results, but we will also take a close look on your overall project structure, the tools used, test coverage, documentation, etc.

5. Techinal documentation

The following section will cover technical aspects of the development.

5.1 Code Structure

The selected code structure has been designed based on the purpose I was asked, this is, data processing. First of all I start thinking about the processor, without taking care of how I was initializing data. Second, came testing, so as to validate data and develop processors at the same time. Then, the need for a repository came up, because data need to be initialized in a proper and secure way and looking into future to possible connections to database. Last, when I was developing the main thread, the need for a validation and proper results arose. Also together with validation a proper way of logging was needed and I created a logger.

5.1.1 Repository

The repository class is meant to be the one that accesses the data, most commonly a database. In this case, only plays the role of getter as data is hardcoded. From this class I can obtain raw data as well as expected data.

5.1.2 Processor

Processor, as the world itself describes it, has all the methods to process the data: - simple processor: First approach, it was meant to process simplest data, this is, data from task 1. - middle processor: Second approach, it was meant to process middle complex data, this is, data from task 2. - complex processor: Third approach, it was meant to process more complex data, this is, data from task 3. - complex deeparse processor: Fourth approach, it was meant to process complex data using machine learning classificator.

5.1.3 Validator

Finally, the validator, only has one method and is used to validate data between the streets I have processed and the expected ones, It gives us information about how many streets have been successfully processed and which are them.

5.1.4 Testing

For testing pytest and unittest have been used, at first, I try to compare json objects but then I figures out it was easier to compare python lists using assertCountEqual() function. Also pytest fixtures have been used to provide fixed data and instances for the tests.

5.2 Documentation

Plenty documentation is offered, coverage documentation with coverage library, format related documentation using Flake8 and project documentation using mkdocs.

5.3 Makefile

Finally a makefile have been provided so that program execution, testing and documenting is easier and more flexible when switching between IDEs. Also some tasks have been automatized, for example, when "make docs" command is executed, all tests, coverage and markdown text is processed and grouped in html format in public/ folder. Is worth mentioning that Makefile

interacts also with Pipfile, as they have been described also less abstract scripts in this file. For example the command flow for command "make format" would be the following:

Makefile	Pipfile	Python
make format	pipenv run black &&	python -m black . &&
	pipenv run isort	python -m isort --sp pyproject.toml --skip .venv .

6. Modus operandi

The following section will cover which has been the procedure while developing the challenge, just for curiosity.

[2022-06-16]

1. Read statement
2. Free thinking:
    - First challenge looks easy, thinking about split() / strip().
    - Second challenge also easy, read until a numeric is found.
    - Third challenge looks like it needs another field such as postal code or an IA app to identify the country so as to know how to split streets or maybe an IA that learns how to split streets.
    - Do I need a country, street and number database to cover all cases?
3. Looks like regex is needed for the second challenge, did not think about german dictionary.
    - Nice link!
    - Nice link 2! Regex Example
    - Need to try to understand how does regex exactly works and adapt to my needs.

[2022-06-17]

1. I found a good regex that works 7/9 of the times (all streets but "4, rue de la revolution" and "Calle 39 No 1540") , I am going to try another kind of process to parse streets. `"^(\b\D+\b)?\s*(\b.*?\d.*?\b)\s*(\b\D+\b)?$"`
2. I found an state-of-the-art library for parsing multinational street addresses using deep learning, let's see how it works.

[2022-06-19]

1. I read again the challenge statement and as I was blocked with some streets, I decide to start using TDD and use as many python best practices as I know, incorporating the following concepts:
2. Start developing following TDD, this is, start using automated unit tests to drive the development of our code.
3. Add code formatting frameworks such as Black, Flake8 and Isort and automatizing code format checks whenever a commit was done.
4. Add Makefile so it simplifies code running, testing and formatting no matter what IDE is being used.
5. Add docs folder and organizing better documentation. Also adding testing and coverage reports into the documentation.

[2022-06-21]

1. Understand how Actions in github work, trying to automate the execution of all docs an leave them in public/ folder, finally not achieved, I create public in local and uploaded to Github so is accessible via https://jonfernandez12.github.io/FRIDAY_code_challenge/public/.
2. Thinking about how to organize project. First 3 files where developed with one processing function each of them, then I think it was a better idea to create a Processor class, Repository class and Validator class to organize concepts in the code structure.

7. Faced problems

The following section will cover the problems I have faced during challenge implementation.

7.1 Multiple SSH keys on same machine

1. To clone and interact with another github account, just create a new ssh-key:

```
ssh-keygen -t rsa
```

1. Save it on **/home/jon/.ssh/** with a different name if one already exists.

```
ssh-add ~/.ssh/<new_private_key_file>
```

1. Copy public key and paste on Github>Settings>SSH and GPG keys
2. Add configuration on **/home/jon/.ssh/config** `Host <alias_name>`

```
    Hostname github.com
    User git
    IdentityFile ~/.ssh/<new_private_key_file>
    IdentitiesOnly yes
```

3. Finally clone repository with `git clone <alias_name>:<github_name>/<proyect_name>.git`

7.2 Regex

I need to learn what is regex from the very beginning, I find some useful websites as that given a regex explain what was defining each character. Finally I decided to use "^(\b\D+\b)?\s(\b.?\d.?\b)\s(\b\D+\b)?$" as is the one with better splitting results for the data given.

7.3 Deepparse

I try a pre-trained machine learning classification method to see if I get better results than with the regex method, I need to understand how to extract data from the output returned by the model and which models where available using the library and which one was better for this purpose. As I was not worried about cpu limitations I used 'best' model.

```
{bibliography}
    author = {Marouane Yassine and David Beauchemin},
    title  = {{Deepparse: A State-Of-The-Art Deep Learning Multinational Addresses Parser}},
    year   = {2020},
    note   = [url](https://deepparse.org)
```

7.4 Testing

I discovered helpful documentation about python testing and fixtures as I am not used to make tests. Although I knew about TDD methodology.

Also, I modify the output of tests from a boolean type to a comparison between actual processed streets and expected streets, as a boolean gives no information about which streets are well parsed and which not. I did not find out a good library for asserting json lists so I convert them into python lists first, so I could make use of unittest TestCase library.

7.5 Pipenv

Somehow, **pipenv run** commands was not working, giving the following error: `Traceback (most recent call last):`
`File "/usr/bin/pipenv", line 11, in <module>`
`load_entry_point('pipenv==11.9.0', 'console_scripts', 'pipenv')()`
`File "/usr/lib/python3/dist-packages/pipenv/vendor/click/core.py", line 722, in __call__`
`return self.main(*args, **kwargs)`
`File "/usr/lib/python3/dist-packages/pipenv/vendor/click/core.py", line 697, in main`
`rv = self.invoke(ctx)`
`File "/usr/lib/python3/dist-packages/pipenv/vendor/click/core.py", line 1066, in invoke`
`return _process_result(sub_ctx.command.invoke(sub_ctx))`
`File "/usr/lib/python3/dist-packages/pipenv/vendor/click/core.py", line 895, in invoke`
`return ctx.invoke(self.callback, **ctx.params)`
`File "/usr/lib/python3/dist-packages/pipenv/vendor/click/core.py", line 535, in invoke`
`return callback(*args, **kwargs)`
`File "/usr/lib/python3/dist-packages/pipenv/cli.py", line 602, in run`
`core.do_run(command=command, args=args, three=three, python=python)`
`File "/usr/lib/python3/dist-packages/pipenv/core.py", line 2200, in do_run`
`command = ' '.join(project.scripts[command])`
`File "/usr/lib/python3/dist-packages/pipenv/project.py", line 374, in scripts`
`scripts[k] = shlex.split(v, posix=True)`
`File "/usr/lib/python3.8/shlex.py", line 311, in split`
`return list(lex)`
`File "/usr/lib/python3.8/shlex.py", line 300, in __next__`
`token = self.get_token()`
`File "/usr/lib/python3.8/shlex.py", line 109, in get_token`
`raw = self.read_token()`
`File "/usr/lib/python3.8/shlex.py", line 140, in read_token`
`nextchar = self.instream.read(1)`
`AttributeError: 'list' object has no attribute 'read'`

I fix it reinstalling pipenv:

`sudo apt-get remove pipenv`
`pip3 install pipenv` ## Flake8

Makes the following warning: `./app/services/middle_processor.py:14:39: W605 invalid escape sequence '\d'`

It seems that Python 3 interprets string literals as Unicode strings, and therefore our \d is treated as an escaped Unicode character. As I was following Flake8 rules regarding W605 and Flake8 rules regarding W503 to avoid the warning I add --ignore flag when checking flake8 format:

`check-flake8 = "python -m flake8 --exclude=.venv/ --ignore=W605"` Even so, a warning is raised when I run tests with pytest.

8. Future implementations

The following section will cover future features I would like to implement.

8.1 Regular expressions

The actual regular expression splits 7 from the proposed 9 streets and only 2 of the 4 complex ones. I am sure there must be a better regular expression that covers all of them. If not, maybe it would be a good idea once I have processed 7 of 9 streets, to try to reprocess the missing streets with another type of regular expression or method.

8.2 Testing

There is 14% of coverage missing because exception paths have not been covered. Also I miss adding JSON validation with pydantic by creating classes with the same structure as the expected json objects, so I can return more strong values in each function.

8.3 Database connection

We have only tried 9 different streets and I would like to download a whole database with streets from all over the world to see how the program runs and make stronger the actual logic.

8.4 Github actions

As I am more used to using Gitlab, I was not able to generate all HTML documentation directly in the cloud using Github actions and I decided to leave it in the project in "public/" folder.

8.5 Code structure

I am not sure if simple and middle processors are somehow useful, I leave them because they were my first approach, but in a real world application they should be deleted. Also, I think about implementing an abstract processor and then all processors to inherit from that abstract class, but again, I do not think that really makes sense, because in real world there is no need for more than one processor.

8.6 Deepparse

Using a pre-trained machine learning model its been hopeless and has lead to bad results, however I would not exclude to train some model with a huge database to get better results.

8.7 Documentation

Regarding documentation, I did not have time to use some profiler like pylint to generate an UML diagram so as to show in a visual graph the relation between classes and methods .