
Aflowey

Marc Dubois

Jan 30, 2023

CONTENTS

1	Features	3
2	Requirements	5
3	Installation	7
4	Usage	9
5	Contributing	11
6	License	13
7	Issues	15
8	Credits	17
	Python Module Index	27
	Index	29

FEATURES

- Utilities to describe and execute flow with coroutine functions
- Easily launch several flows simultaneously
- Strong focus on readability

<https://aflowey.readthedocs.io>

REQUIREMENTS

- python 3.7 +

This library is easier to use with third party libraries for manipulating function such as **fn_** (flip function, function composition...), and **tenacity_** (retry library).

INSTALLATION

You can install *Aflowey* via `pip` from PyPI:

```
$ pip install aflowey
```


USAGE

Chain function to execute an async flow !

```
from aflowey import aflow, CANCEL_FLOW, aexec, flog, partial

db = ... # get client db

# add some other library
from tenacity import retry

async def fetch_url(url):
    return await ...

def process_data(html):
    processed_data = ... # process data
    if processed_data is None:
        return CANCEL_FLOW

    return processed_data

async def insert_into_db(content):
    return await db.insert_one(content)

def get_url_flow(url):
    # defining a flow for working with url
    return (
        aflow.from_args("http://goole.fr")
        >> retry(wait=2)(fetch_url)
        >> flog("url fetched", print_arg=True)
        >> process_data # this one is synchronous but may included in the flow
        >> insert_into_db
    )
```

Execute the flow for one url:

```
result = await get_url_flow("http://google.com/...").run()
```

Execute several flows asynchronously:

```
from fn import flip

name = "Marco"
```

(continues on next page)

(continued from previous page)

```

user_flow = (
    aflow.empty()
    >> partial(db.find_one, search={"username": name})
    >> User.from_dict
    # the impure indicate that this step does not return a new result
    # i.e the result of User.from_dict will be sended
    >> impure(partial(flip(setattr), datetime.now(), 'created_at'))
)

organization_id = "Not employed"

organization_flow = (
    aflow.empty()
    >> partial(db.find_one, search={"organization_id": organization_id})
    >> Organization.from_dict
)

urls = [
    "http://google.com/...",
    "http://google.com/...",
    "http://google.com/...",
    "http://google.com/...",
]

url_flows = [get_url_flow(url) for url in urls]

# execute all flow with asyncio gather method
executor = aexec().from_flows(url_flows) | user_flow | organization_flow
(url1, url2, url3, url4), user, organization = await executor.run()

```

It can be boring to create function that exactly matches arity of the flow. Aflowey provide some higher order functions to help, see:

- lift: create a new method accepting transformed arguments
- F0: from a 0 argument function, create one argument function to fit the arity of the flow
- F1: create a new function with an extra parameter to process input of the flow step
- spread: create a new function which spread an iterable of arguments into the given function
- spread_kw: create a new function which spread kw arguments into the given function

The fn library provide other interesting functions like:

- flip
- first

If you have any other ideas...

CONTRIBUTING

Contributions are very welcome. To learn more, see the [Contributor Guide](#).

LICENSE

Distributed under the terms of the [MIT license](#), *Aflowey* is free and open source software.

ISSUES

If you encounter any problems, please [file an issue](#) along with a detailed description.

This project was generated from [@cjolowicz's Hypermodern Python Cookiecutter](#) template.

8.1 Reference

- *[aflowey.async_flow](#)*
- *[aflowey.single_executor](#)*
- *[aflowey.executor](#)*
- *[aflowey.f](#)*
- *[aflowey.functions](#)*

8.1.1 aflowey.async_flow

class `aflowey.async_flow.AsyncFlow(*args, **kwargs)`

Describe an async flow chaining function

```
>>>flow = (AsyncFlow() >> gen1 >> gen2 >> gen3)
```

```
>>>await flow.run()
```

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

static `empty()`

create an empty flow

Return type

AsyncFlow

static `from_args(*args, **kwargs)`

create a flow with given arguments as first input

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type*AsyncFlow***static from_flow(flow)**

create a new flow from given flow, copying it (args, kwargs, and aws functions)

Parameters**flow** (*AsyncFlow*) –**Return type***AsyncFlow***async run(**kwargs)**

run the flow

Returns

coroutine

Return type*Any***aflowey.async_flow.afflow**alias of *AsyncFlow***aflowey.async_flow.async_flow**alias of *AsyncFlow*

8.1.2 aflowey.single_executor

class aflowey.single_executor.SingleFlowExecutor(flow)

Single flow executor

Parameters**flow** (*AsyncFlow*) –**async static check_and_execute_flow_if_needed(maybe_flow, **kwargs)**

check if we have an async flow and execute it

Parameters• **maybe_flow** (*Union*[*Any*, *AsyncFlow*]) –• **kwargs** (*Any*) –**Return type***Any***async execute_flow(is_root, **kwargs)**

Main function to execute a flow

Parameters• **is_root** (*bool*) –• **kwargs** (*Any*) –**Return type***Any*

static `get_step_name(func, index)`

get the step name

Parameters

- **func** (F) –
- **index** (int) –

Return type

str

static `need_to_cancel_flow(result)`

check if we need to cancel flow checking sentinel

Parameters

result (Any) –

Return type

bool

save_step(task, index, current_args)

save step state in flow attribute

Parameters

- **task** (F) –
- **index** (int) –
- **current_args** (Any) –

Return type

None

8.1.3 aflowey.executor

class `aflowey.executor.AsyncFlowExecutor`

Execute several flows concurrently

```
>>> await (aexec().from_flows(flows) | flow).run()
```

from_flows(flows)

create a new executor from one flow or array of flows

Parameters

flows (Any) –

Return type

`AsyncFlowExecutor`

async run(**kwargs)

main function to run stuff in parallel

Parameters

kwargs (Any) –

Return type*Any*`aflowey.executor.aexec`alias of *AsyncFlowExecutor*`aflowey.executor.async_exec`alias of *AsyncFlowExecutor*

8.1.4 aflowey.f

`class aflowey.f.F(func)`

tiny wrapper around a function

Parameters**func** (*Function*) – callable

8.1.5 aflowey.functions

`aflowey.functions.F0(func)`

create a new function from a 0 arity function (takes 0 args). The new function takes exactly one argument and does not pass it to the wrapped function. It allows using a 0 arity function in a flow relatively easily.

Parameters**func** (*Function*) –**Return type***F*`aflowey.functions.F1(func, extractor=None)`

wraps a one argument function (with arity 1) and allows to add an extractor to work on the input argument.

Parameters

- **func** (*Function*) –
- **extractor** (*Optional[Function]*) –

Return type*F*`aflowey.functions.apartial(func, *args, **kwargs)`

make a partial function of the given func and ensure it will work in an async context

Parameters

- **func** (*Function*) –
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type*F*`aflowey.functions.breaker(func)`

simply for readability

Parameters**func** (*Function*) –**Return type***Function*`aflowey.functions.ensure_callable(x)`

ensure a given args is a callable by returning a new callable if not

Parameters**x** (*Union[Any, Function]*) –**Return type***Function*`aflowey.functions.ensure_f(func)`

wrap the given function into an F instance

Parameters**func** (*Function*) –**Return type***F*`aflowey.functions.erratic(func)`

simply for readability

Parameters**func** (*Function*) –**Return type***Function*`aflowey.functions.f0(func)`

create a new function from a 0 arity function (takes 0 args). The new function takes exactly one argument and does not pass it to the wrapped function. It allows using a 0 arity function in a flow relatively easily.

Parameters**func** (*Function*) –**Return type***F*`aflowey.functions.f1(func, extractor=None)`

wraps a one argument function (with arity 1) and allows to add an extractor to work on the input argument.

Parameters

- **func** (*Function*) –
- **extractor** (*Optional[Function]*) –

Return type*F*`aflowey.functions.flog(log_str="", print_arg=False)`

utility function to log between steps, printing argument if needed

Parameters

- **log_str** (*str*) –
- **print_arg** (*bool*) –

Return type*Any*`aflowey.functions.imp(*func)`

take an array of function and tag it as side effects function

Parameters`func (Union[Function, F]) –`**Return type***Union[List[F], F]*`aflowey.functions.impure(*func)`

take an array of function and tag it as side effects function

Parameters`func (Union[Function, F]) –`**Return type***Union[List[F], F]*`aflowey.functions.log(log_str="", print_arg=False)`

utility function to log between steps, printing argument if needed

Parameters

- `log_str (str) –`
- `print_arg (bool) –`

Return type*Any*`aflowey.functions.make_impure(func)`

tags the given function as impure, i.e. consume an argument but does not create new one

Parameters`func (Union[Function, F]) –`**Return type***F*`aflowey.functions.may_fail(func)`

simply for readability

Parameters`func (Function) –`**Return type***Function*`aflowey.functions.named(func, name)`

tags a function as a named function

Parameters

- `func (Union[Function, F, Any]) –`
- `name (str) –`

Return type*F*

`aflowey.functions.p(func, *args, **kwargs)`

make a partial function of the given func and ensure it will work in an async context

Parameters

- **func** (*Function*) –
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type

F

`aflowey.functions.partial(func, *args, **kwargs)`

make a partial function of the given func and ensure it will work in an async context

Parameters

- **func** (*Function*) –
- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type

F

`aflowey.functions.side_effect(*func)`

take an array of function and tag it as side effects function

Parameters

func (*Union[Function, F]*) –

Return type

Union[List[F], F]

`aflowey.functions.spread(func)`

create a function which takes an iterable of args and spread it into the given function

Parameters

func (*Function*) –

Return type

F

`aflowey.functions.spread_args(func)`

create a function which takes an iterable of args and spread it into the given function

Parameters

func (*Function*) –

Return type

F

`aflowey.functions.spread_kw(func)`

create a function which takes a mapping of kwargs and spread it into the given function

Parameters

func (*Function*) –

Return type

F

`aflowey.functions.spread_kwargs(func)`

create a function which takes a mapping of kwargs and spread it into the given function

Parameters

func (*Function*) –

Return type

F

`aflowey.functions.wrapper_async(func)`

wrap a function into a coroutine function :param func: callable

Returns

F instance

Parameters

func (*Function*) –

Return type

F

8.2 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the [MIT license](#) and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- [Source Code](#)
- [Documentation](#)
- [Issue Tracker](#)
- [Code of Conduct](#)

8.2.1 How to report a bug

Report bugs on the [Issue Tracker](#).

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.2.2 How to request a feature

Request features on the [Issue Tracker](#).

8.2.3 How to set up your development environment

You need Python 3.6+ and the following tools:

- [Poetry](#)
- [Nox](#)
- [nox-poetry](#)

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run aflow
```

8.2.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the [pytest](#) testing framework.

8.2.5 How to submit changes

Open a [pull request](#) to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.3 MIT License

Copyright © 2021 Marc Dubois

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

PYTHON MODULE INDEX

a

`aflowey.async_flow`, [17](#)
`aflowey.executor`, [19](#)
`aflowey.f`, [20](#)
`aflowey.functions`, [20](#)
`aflowey.single_executor`, [18](#)

A

aflowey.executor), 20
 aflow (in module *aflowey.async_flow*), 18
aflowey.async_flow
 module, 17
aflowey.executor
 module, 19
aflowey.f
 module, 20
aflowey.functions
 module, 20
aflowey.single_executor
 module, 18
 apartial() (in module *aflowey.functions*), 20
 async_exec (in module *aflowey.executor*), 20
 async_flow (in module *aflowey.async_flow*), 18
 AsyncFlow (class in *aflowey.async_flow*), 17
 AsyncFlowExecutor (class in *aflowey.executor*), 19

B

breaker() (in module *aflowey.functions*), 20

C

check_and_execute_flow_if_needed()
 (*aflowey.single_executor.SingleFlowExecutor*
 static method), 18

E

empty() (*aflowey.async_flow.AsyncFlow* static method),
 17
 ensure_callable() (in module *aflowey.functions*), 21
 ensure_f() (in module *aflowey.functions*), 21
 erratic() (in module *aflowey.functions*), 21
 execute_flow() (*aflowey.single_executor.SingleFlowExecutor*
 method), 18

F

F (class in *aflowey.f*), 20
 F0() (in module *aflowey.functions*), 20
 f0() (in module *aflowey.functions*), 21
 F1() (in module *aflowey.functions*), 20

f1() (in module *aflowey.functions*), 21
 flog() (in module *aflowey.functions*), 21
 from_args() (*aflowey.async_flow.AsyncFlow* static
 method), 17
 from_flow() (*aflowey.async_flow.AsyncFlow* static
 method), 18
 from_flows() (*aflowey.executor.AsyncFlowExecutor*
 method), 19

G

get_step_name() (*aflowey.single_executor.SingleFlowExecutor*
 static method), 18

I

imp() (in module *aflowey.functions*), 22
 impure() (in module *aflowey.functions*), 22

L

log() (in module *aflowey.functions*), 22

M

make_impure() (in module *aflowey.functions*), 22
 may_fail() (in module *aflowey.functions*), 22
 module
 aflowey.async_flow, 17
 aflowey.executor, 19
 aflowey.f, 20
 aflowey.functions, 20
 aflowey.single_executor, 18

N

named() (in module *aflowey.functions*), 22
 need_to_cancel_flow()
 (*aflowey.single_executor.SingleFlowExecutor*
 static method), 19

P

p() (in module *aflowey.functions*), 22
 partial() (in module *aflowey.functions*), 23

R

run() (*aflowey.async_flow.AsyncFlow* method), 18

`run()` (*aflowey.executor.AsyncFlowExecutor method*), [19](#)

S

`save_step()` (*aflowey.single_executor.SingleFlowExecutor method*), [19](#)

`side_effect()` (*in module aflowey.functions*), [23](#)

`SingleFlowExecutor` (class *in aflowey.single_executor*), [18](#)

`spread()` (*in module aflowey.functions*), [23](#)

`spread_args()` (*in module aflowey.functions*), [23](#)

`spread_kw()` (*in module aflowey.functions*), [23](#)

`spread_kwargs()` (*in module aflowey.functions*), [23](#)

W

`wrapper_async()` (*in module aflowey.functions*), [24](#)