

# DOC-TRACING

---

# FOUILLER UNE BASE DE CODE FOSSILE PAR TRACAGE D'EXECUTION

---

**NUMERIC PARK**

**LA DOCUMENTATION DISPARUE**

Luc Sorel-Giffo — jeudi 29 juin 2023 - 12h45 amphi D — BreizhCamp

@lucsorelgiffo@floss.social

# SYNOPSIS

---

## LE SITE DE FOUILLES : LE CODE

---

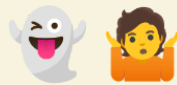
```
math_cli
├── __main__.py
├── factorial.py
├── fibonacci.py
└── validation.py
```



Tous les exemples de code se trouvent dans [github.com/lucsorel/doc-tracing/tree/main/examples](https://github.com/lucsorel/doc-tracing/tree/main/examples).

# LA DOC

---



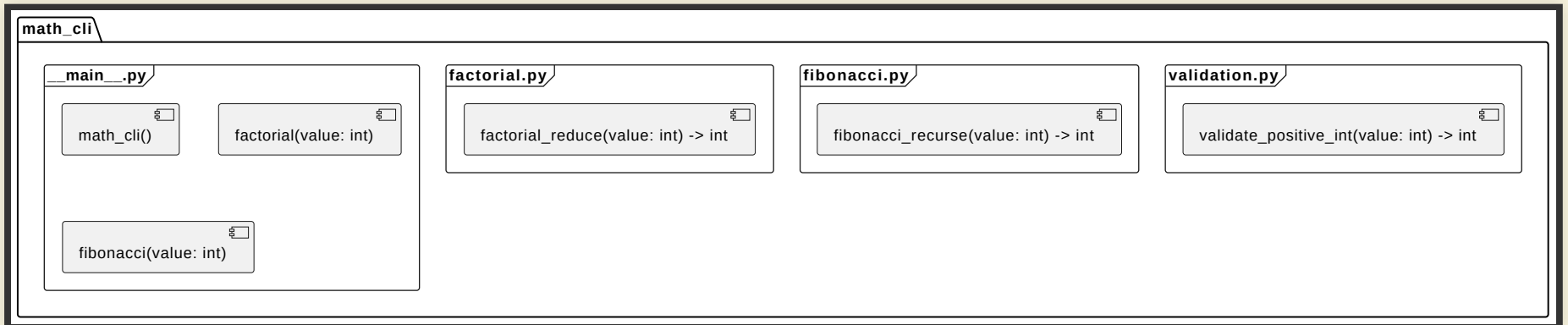
# DOC-AS-CODE

```
@startuml math_cli
package math_cli {
  frame __main__.py {
    component "math_cli()"
    component "factorial(value: int)"
    component "fibonacci(value: int)"
  }

  frame factorial.py { component "factorial_reduce(value: int) -> int" }

  frame fibonacci.py { component "fibonacci_recurse(value: int) -> int" }

  frame validation.py { component "validate_positive_int(value: int) -> int" }
}
@enduml
```



# ANALYSE STATIQUE DE CODE – "CODE-AS-DOC" 🤪

```
# math_cli/factorial.py
from functools import reduce

def factorial_reduce(value: int) -> int:
    '''Functional implementation of factorial'''
    if value == 1:
        return 1

    return reduce(lambda agg, index: agg * index, range(value, 1, -1), 1)
```

```
python -m pydoc -p 8080 -b # -> http://localhost:8080
```

Python 3.10.9 [main, GCC 11.3.0]  
Linux-5.15.0-53-lowlatency-x86\_64-with-glibc2.35

[Module Index](#) : [Topics](#) : [Keywords](#)

---

**math\_cli.factorial** [index](#)  
[{...}/math\\_cli/factorial.py](#)

---

**Functions**

**factorial\_reduce(value: int) -> int**  
Functional implementation of factorial

**reduce(...)**  
[reduce](#)(function, iterable[, initial]) -> value

Apply a function of two arguments cumulatively to the items of a sequence or iterable, from left to right, so as to reduce the iterable to a single value. For example, [reduce](#)(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates (((1+2)+3)+4)+5. If initial is present, it is placed before the items of the iterable in the calculation, and serves as a default when the iterable is empty.

# ANALYSE STATIQUE DE CODE – APPORTS ET LIMITES

---



- génération / mise à jour de la doc à partir d'une source de vérité
- valorisation : docstring & annotations de typage



- comment **s'articulent** les fonctions ?
- comment sont gérées **les erreurs** ?



# TRACAGE D'EXECUTION AVEC SYS.SETTRACE

---

`sys.settrace` permet de déclarer un hook qui va être appelé à chaque étape d'exécution du code :

```
from sys import settrace

def trace_func(tracer: Callable, func: Callable, *args, **kwargs) -> Any:
    '''Applies the tracer hook during the execution of the given func'''

    settrace(tracer)
    try:
        return func(*args, **kwargs)
    finally:
        settrace(None)
```

# GLOBAL TRACER : APPEL D'UN BLOC DE CODE

Rôle : l'exécution du bloc de code (script, corps de fonction) doit-elle être tracée ?

```
def global_tracer(frame, event: str, arg: Any) -> Callable:
    # ici : event vaut toujours 'call', arg est toujours None
    if should_trace_call(frame):
        return local_tracer # appel tracé

    return None # appel non tracé
```

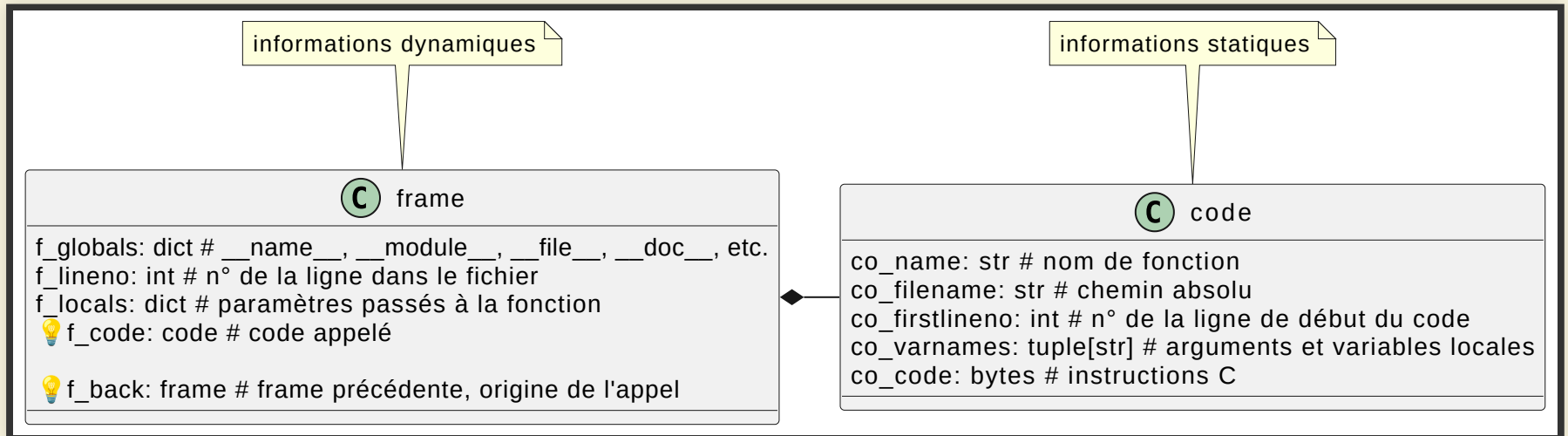


Figure 1. Doc officielle : `frame`, `code`

# LOCAL TRACER : EXECUTION INTERNE D'UN BLOC DE CODE

---

Rôle : poursuivre le traçage après chaque expression ou lors d'une levée d'erreur ?

```
def local_tracer(frame, event: str, arg: Any) -> Callable:
    # ici : event peut valoir 'line', 'return' ou 'exception'
    if event == 'line':
        ... # arg : toujours None
    if event == 'return':
        ... # arg : la valeur renvoyée ; local_tracer ne sera plus appelée
    if event == 'exception':
        error_class, error, traceback = arg
        # 💡 stocker l'erreur et sa ligne d'émission (frame.f_lineno)

        # poursuite du traçage avec un traceur spécifique
        return error_tracer

    # pour continuer à tracer l'exécution du bloc
    return local_tracer
```



créer `error_tracer` avec un pattern *factory* pour lui associer le contexte de l'erreur.

# ERROR TRACER : GESTION DE L'ERREUR DANS LE BLOC DE CODE

---

Rôle : tracer la gestion de l'erreur par le code.

```
def error_tracer(self, frame, event: str, arg: Any):
    # ici : event peut valoir 'line', 'return' ou 'exception'

    if event == 'exception':
        # levée d'une erreur (la même, ou une autre qui la remplace ou l'enrobe)
        # 💡 stocker l'erreur et sa ligne d'émission (frame.f_lineno)
        ...

    elif event == 'return':
        # si il y a une erreur stockée :
        # - si n° ligne "return" == n° ligne "levée d'erreur" : propagation de l'erreur
        # - sinon : l'erreur a été traitée dans un except, sortie du bloc avec `arg`
        # 💡 déréférencer l'erreur stockée

        # s'il n'y a plus d'erreur stockée : sortie du bloc de code

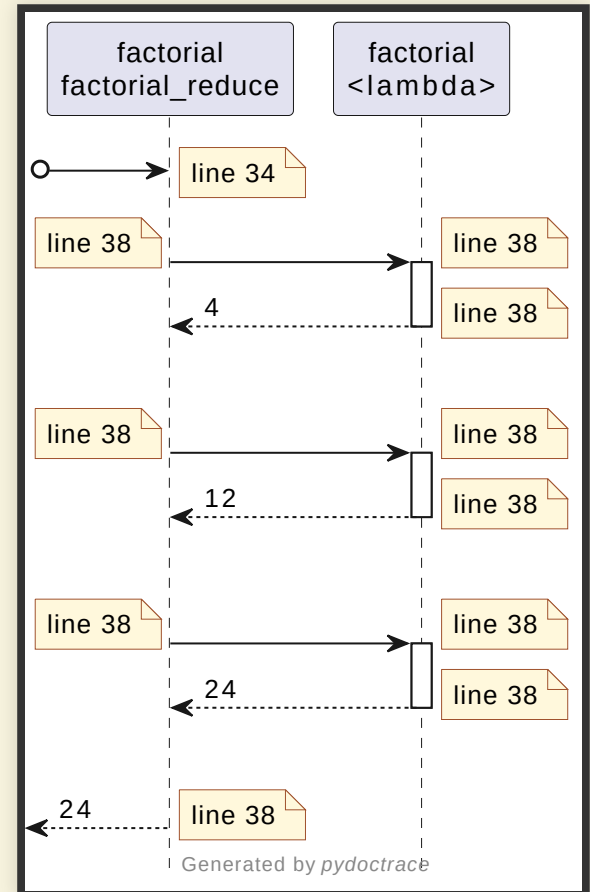
    return error_tracer
```

- décorateur (peu intrusif) → diagramme de séquence **PlantUML**
- [github.com/lucsorel/pydoctrace](https://github.com/lucsorel/pydoctrace)

```
from pydoctrace.doctrace import trace_to_puml

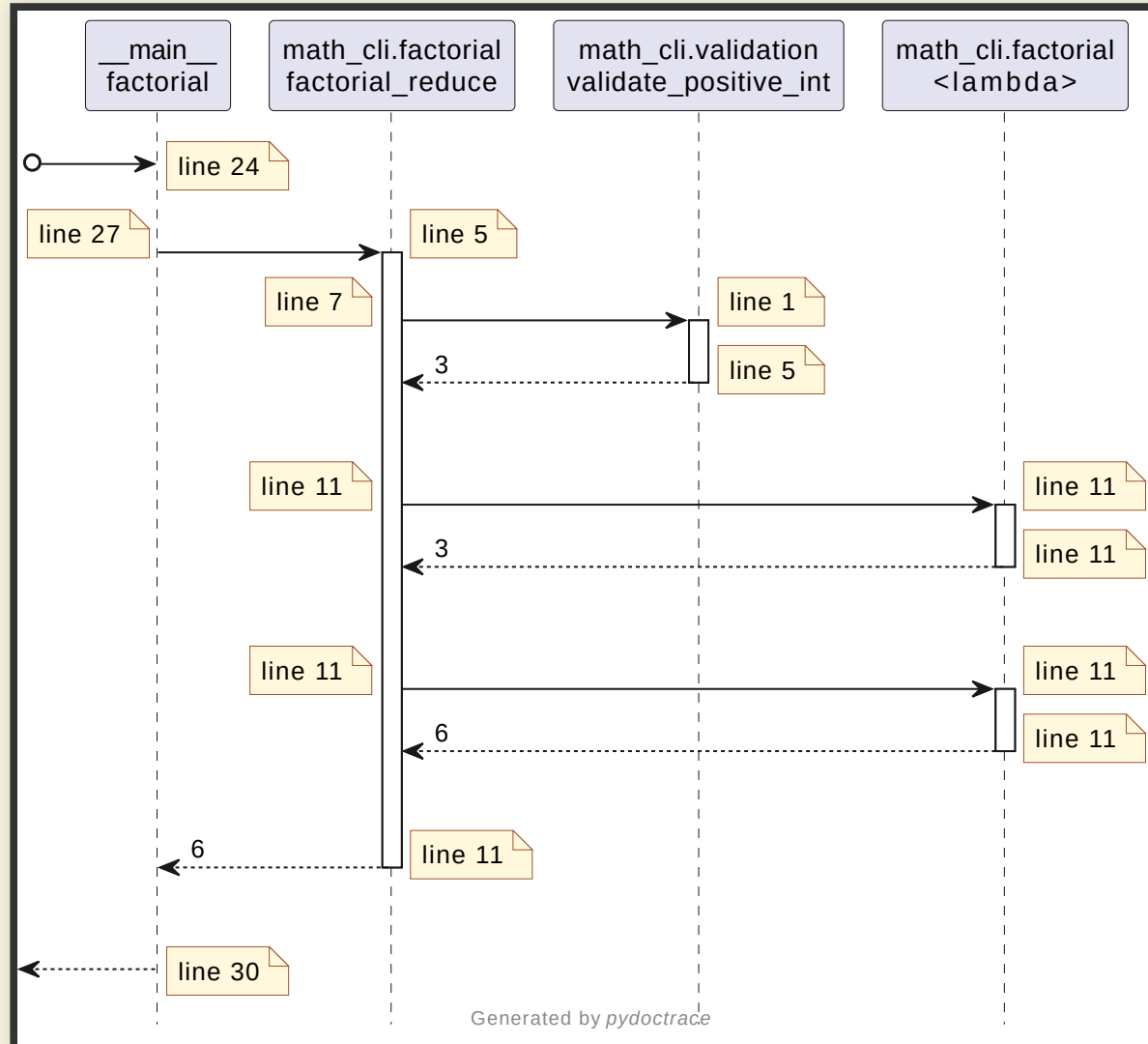
@trace_to_puml
def factorial_reduce(value: int) -> int:
    '''Functional implementation of factorial'''
    value = validate_positive_int(value)
    if value == 1:
        return 1

    return reduce(
        lambda agg, index: agg * index,
        range(value, 1, -1),
        1
    )
```



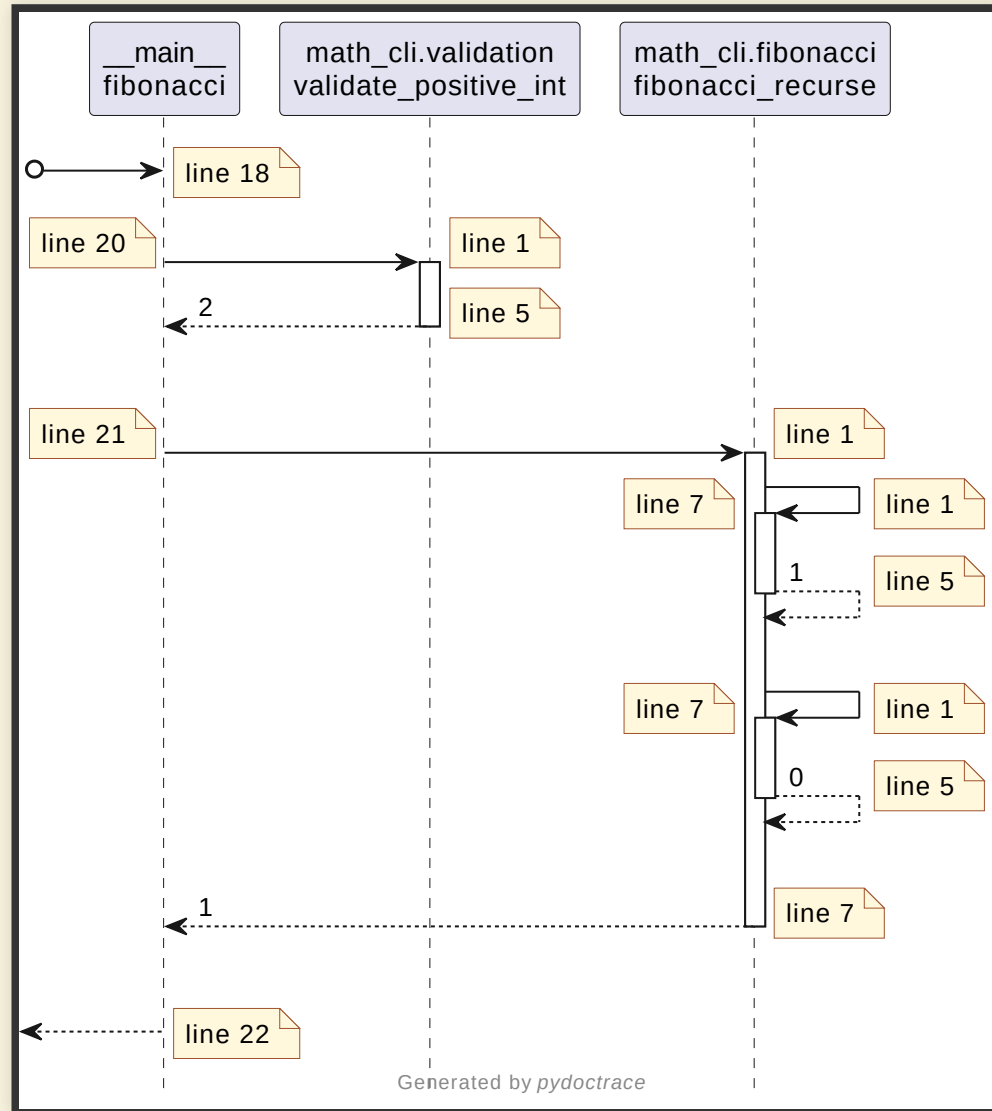
# PYDOCTRACE : FACTORIAL REDUCE

```
python -m math_cli --factorial 3
```



# PYDOCTRACE : FIBONACCI

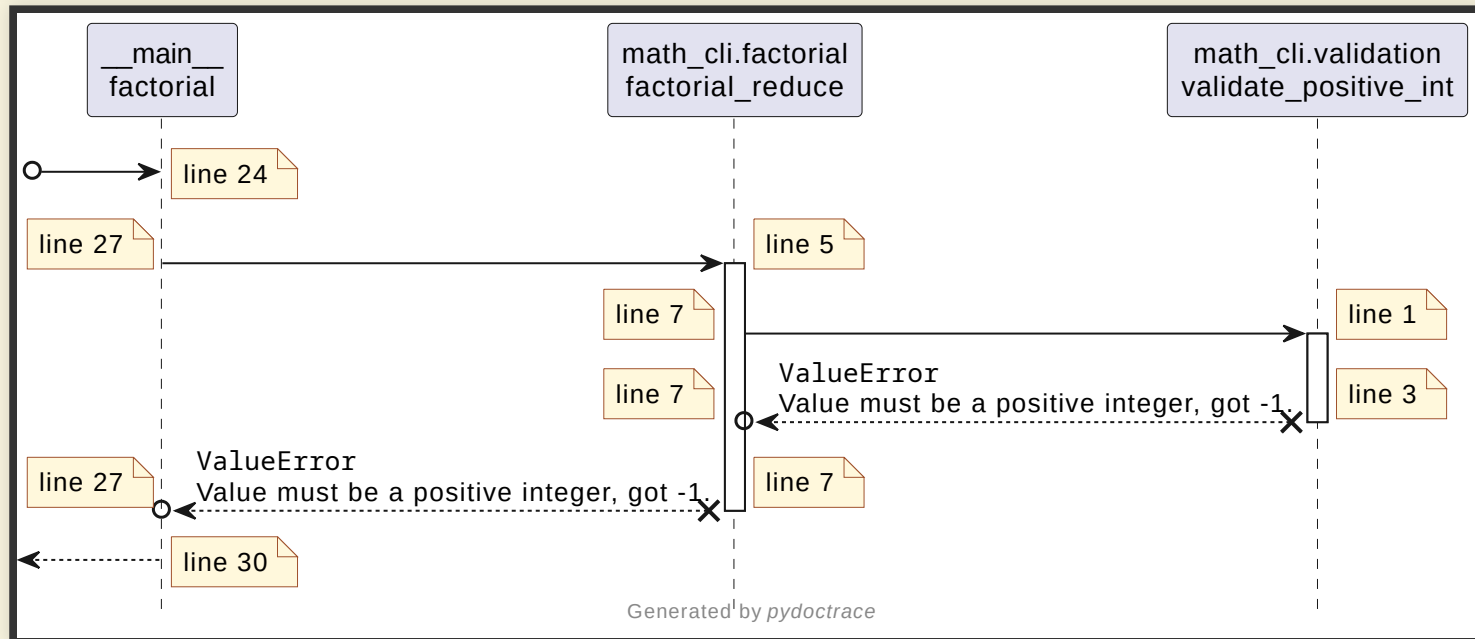
```
python -m math_cli --fibonacci 2
```



# PYDOCTRACE : FACTORIAL REDUCE, ERREUR INTERCEPTEE



```
python -m math_cli --factorial -1
```

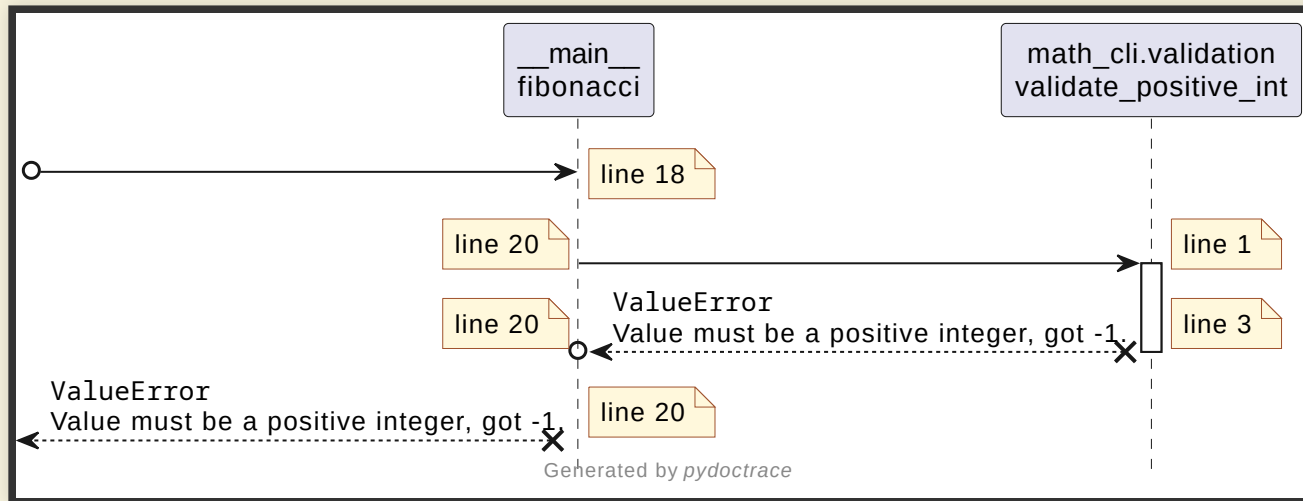




# PYDOCTRACE : FIBONACCI, ERREUR NON GEREE



```
python -m math_cli --fibonacci -1
```



# ARCHITECTURE & DIAGRAMME DE SEQUENCE : LE BON NIVEAU ?

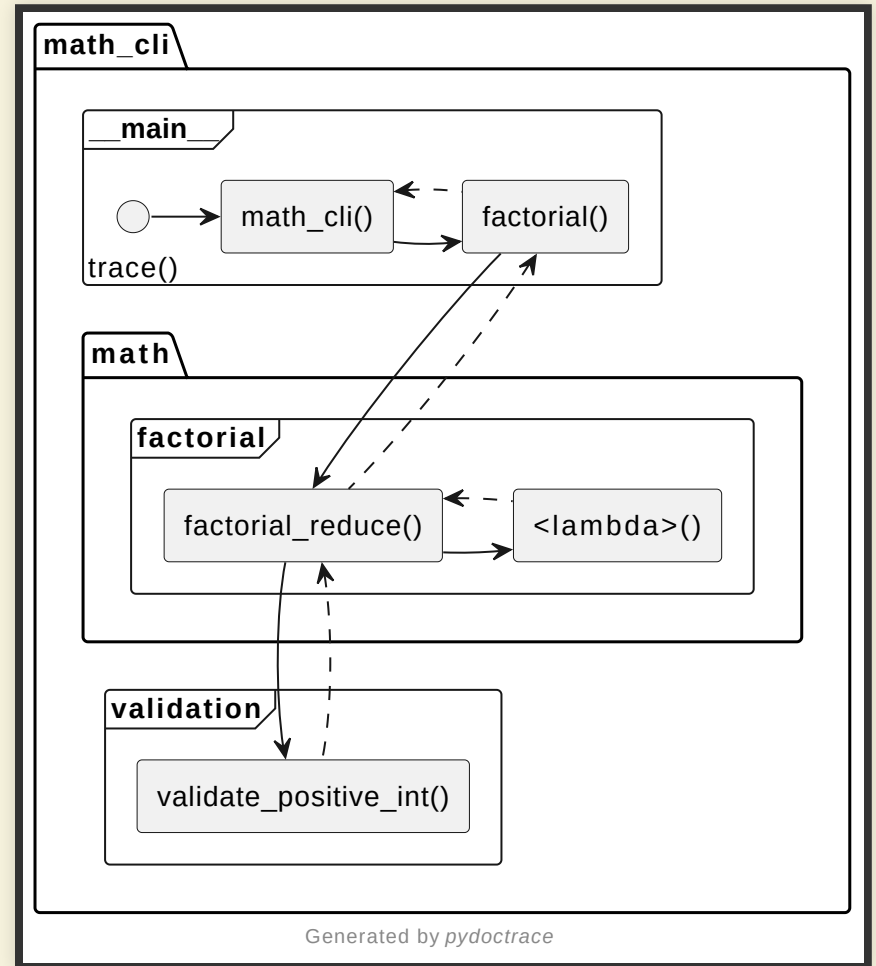
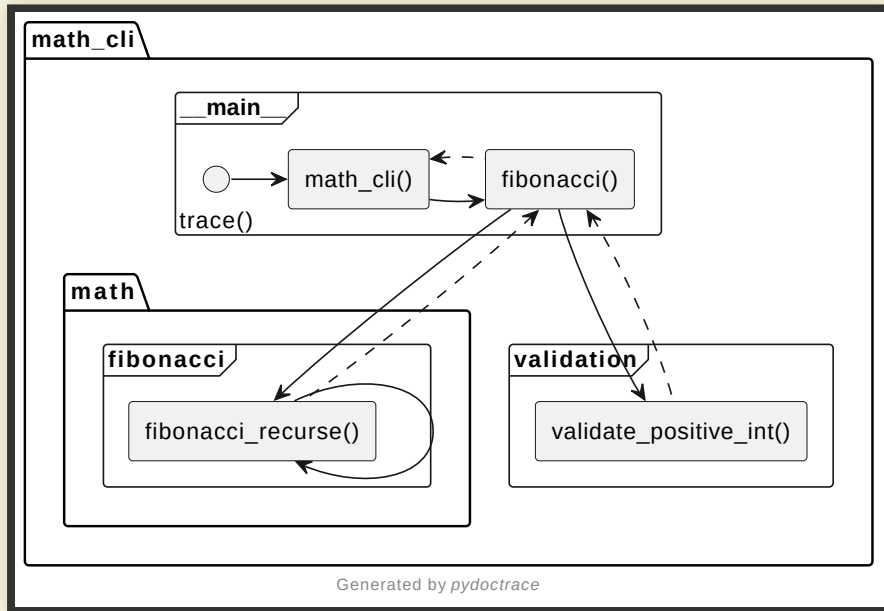
---



# ARCHITECTURE & DIAGRAMME DE COMPOSANTS



- @trace\_to\_sequence\_puml
- 🧑 @trace\_to\_components\_puml



# CONCLUSIONS CONCERNANT LE DOC-TRACING

---

Mise en œuvre :

- API du débogueur Python
- programmation orientée aspect, agents

Documenter la vraie vie du code :

- les tests fonctionnels associés à une *user story*
- le code d'une base qu'on découvre
- observabilité applicative ? (OpenTelemetry)

Issues et ★ : [github.com/lucsorel/pydoctrace](https://github.com/lucsorel/pydoctrace) 👍

MERCI !

---

DES QUESTIONS ?

Présentation à retrouver sur [github.com/lucsorel/doc-tracing](https://github.com/lucsorel/doc-tracing) 

# REJOIGNEZ PYTHON RENNES !

---

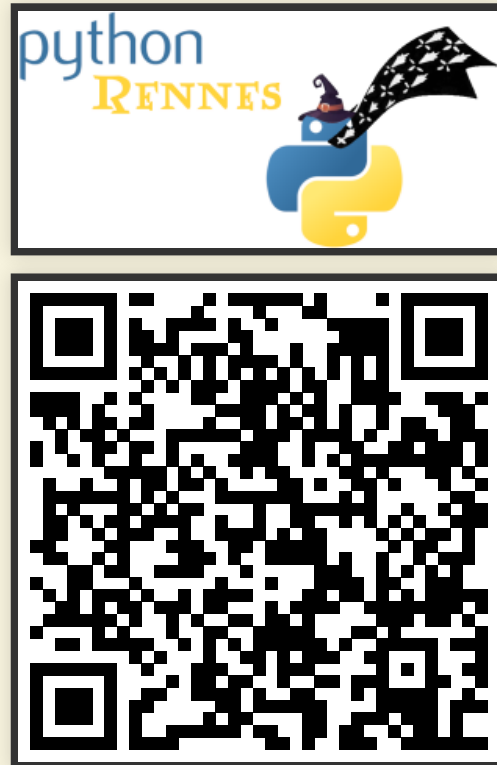


Figure 2. Pour rejoindre le slack : [https://join.slack.com/t/pythonrennes/shared\\_invite/zt-1yd4yioap-IBAngm3Q0jxAKLP6fYJR8w](https://join.slack.com/t/pythonrennes/shared_invite/zt-1yd4yioap-IBAngm3Q0jxAKLP6fYJR8w)