

# Doc-tracing

---

# **Traçage d'exécution du code à des fins documentaires**

---

Outiller les discussions d'architecture avec des diagrammes d'exécution des tests fonctionnels.


Luc Sorel-Giffo — jeudi 6 avril 2023 — Soirée des communautés techniques rennaises



[@lucsorelgiffo@floss.social](mailto:@lucsorelgiffo@floss.social)

## Genèse de l'idée : du doc-as-code ++

---

- documenter une base de code, c'est bien
- écrire de la documentation, c'est pénible / *improductif*
- maintenir de la documentation, c'est... rarement fait

 doc - as - code :

-  écrire la doc comme on écrit du code (fichiers *texte*, DSL)
-  générer la doc à partir du code !

# Il y a 2 types de code dans le doc-as-code

Le code source (vérité)

```
# school.py
from dataclasses import \
    dataclass

@dataclass
class Person:
    firstname: str
    lastname: str

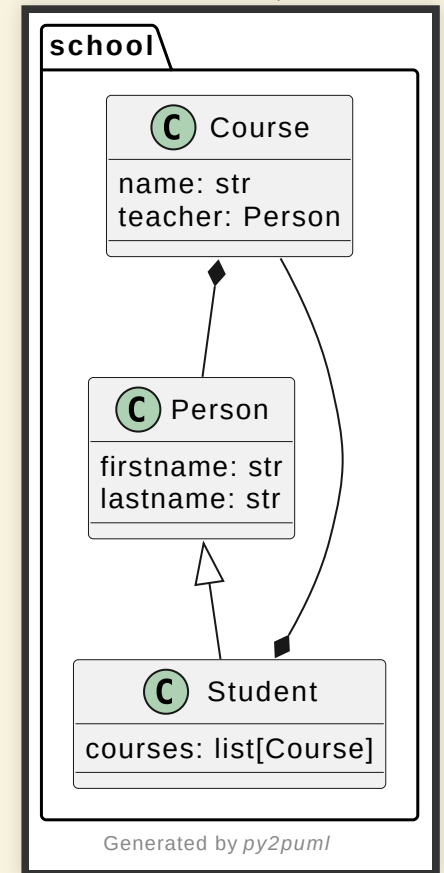
@dataclass
class Course:
    name: str
    teacher: Person

@dataclass
class Student(Person):
    courses: list[Course]
```

Le code de la doc (PlantUML, AsciiDoc, etc.)

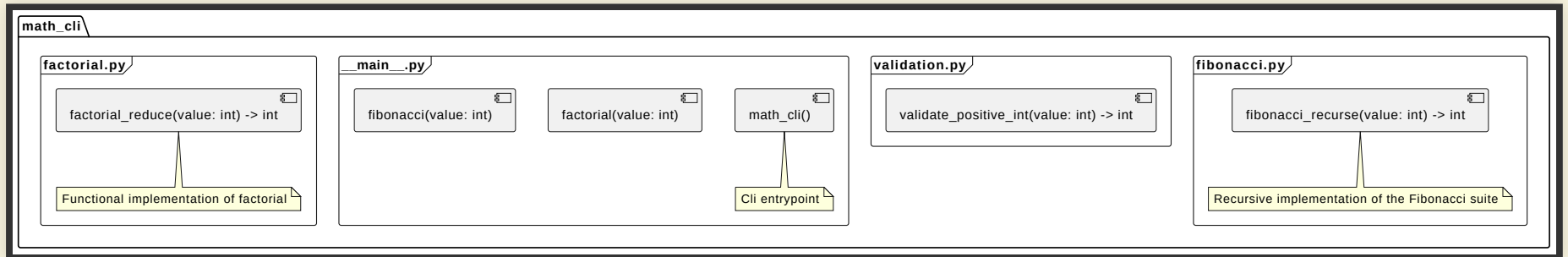
```
# school.puml
@startuml school
class school.Course {
    name: str
    teacher: Person
}
class school.Person {
    firstname: str
    lastname: str
}
class school.Student {
    courses: list[Course]
}
school.Course *-- school.Person
school.Student *-- school.Course
school.Person <|-- school.Student
footer Generated by //py2puml//
@enduml
```

(le code des outils de  
génération et de rendu de  
la doc)



# Doc-as-code par analyse statique de code - composants

Exemple d'une CLI proposant de calculer `factoriel(n)` ou `fibonacci(n)` :



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).

# Doc-as-code par analyse statique de code - pydoc

```
# 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.

# Apports et limites de l'analyse statique de code

---



- code as doc : génération à partir d'une source de vérité
- valorisation : docstring & annotations de typage



- on sait où se trouvent les fonctions mais pas **la façon dont elles s'articulent**
- on ne voit pas **comment sont gérées les erreurs**

# Traçage d'exécution

```
from sys import argv

def factorial(n: int) -> int:
    assert n > 0
    if n == 1:
        return 1
    return n * factorial(n - 1)

factorial(int(argv[1]))
```

```
python -m trace --trace trace_factorial.py 3

--- modulename: trace_factorial, funcname: <module>
trace_factorial.py(1): from sys import argv
trace_factorial.py(3): def factorial(n: int) -> int:
trace_factorial.py(9): factorial(int(argv[1]))
--- modulename: trace_factorial, funcname: factorial
trace_factorial.py(4):     assert n > 0
trace_factorial.py(5):     if n == 1:
trace_factorial.py(7):     return n * factorial(n - 1)
--- modulename: trace_factorial, funcname: factorial
trace_factorial.py(4):     assert n > 0
trace_factorial.py(5):     if n == 1:
trace_factorial.py(7):     return n * factorial(n - 1)
--- modulename: trace_factorial, funcname: factorial
trace_factorial.py(4):     assert n > 0
trace_factorial.py(5):     if n == 1:
trace_factorial.py(6):         return 1
```

*Et si on utilisait des regexp dessus pour faire  
un diagramme de séquence ? 😁*



## Exploration de sys.settrace

---

Le module `sys` propose d'ajouter un hook avec `sys.settrace` qui va être appelé à chaque étape d'exécution du code :

```
from sys import gettrace, settrace

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

    settrace(global_doctracer)
    try:
        return func(*args, **kwargs)
    finally:
        settrace(tracer)
```

# Global tracer : appel d'un bloc de code

L'exécution du code (script, 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):
        # appel tracé
        return local_tracer
    # return None -> appel non tracé
```

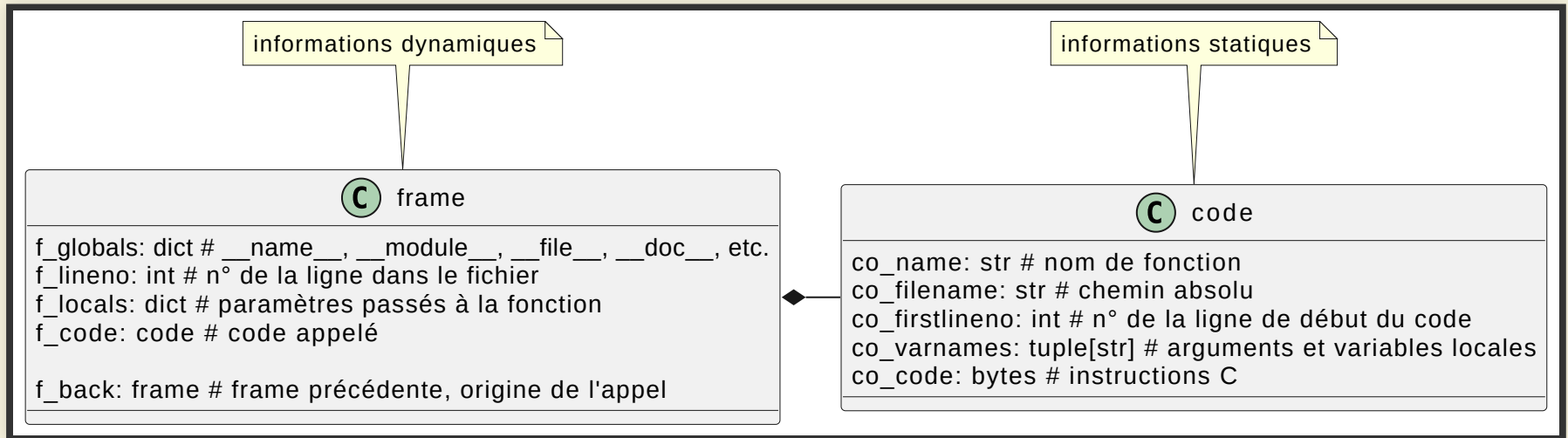


Figure 1. Doc officielle : [frame](#), [code](#)

## Local tracer : exécution d'un bloc de code

---

```
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
    if event == 'exception':
        error_class, error, traceback = arg
        # 💡 stocker l'erreur et sa ligne d'émission

        # poursuite du traçage avec un traceur spécifique
        return error_tracer

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

# Error tracer : gestion de l'erreur dans le bloc de code

---

Similaire à `local_tracer`:

```
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, une autre qui la remplace ou l'enrobe)
        # 💡 stocker l'erreur et sa ligne d'émission
        ...

    elif event == 'return':
        # si il y a une erreur stockée :
        # - si ligne de sortie "return" == ligne de 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
```

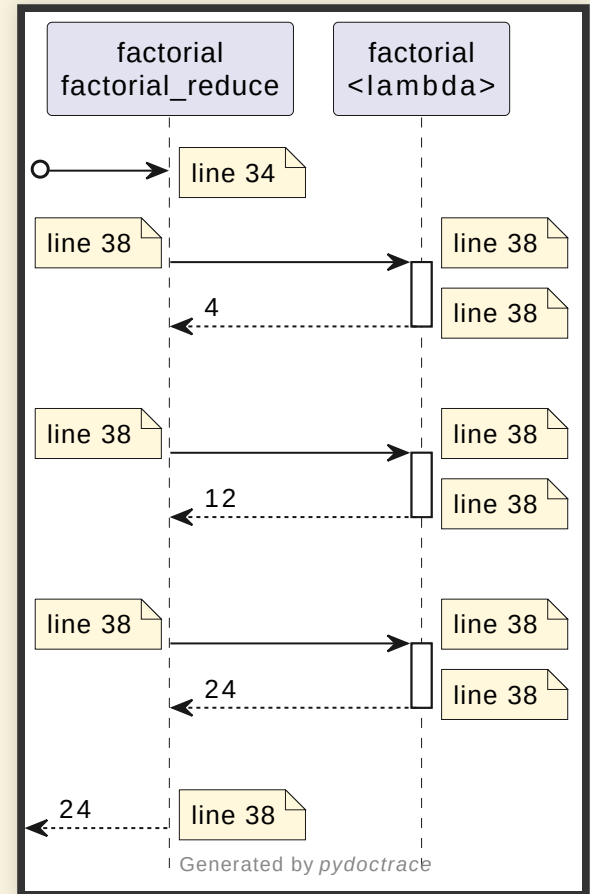
# pydoctrace 0.1.0 (7 février 2023) 🎉

```
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

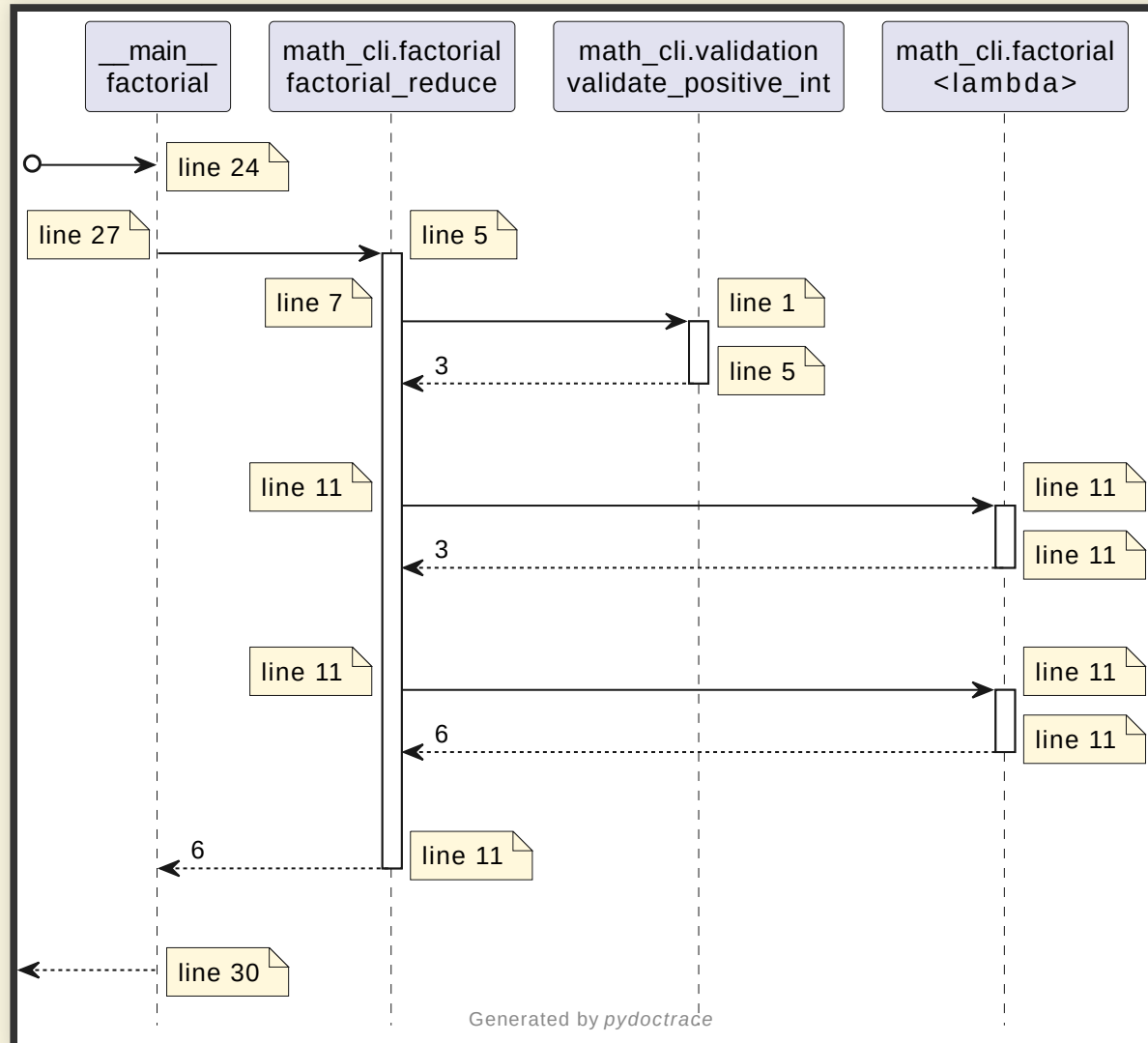
    return reduce(
        lambda agg, index: agg * index,
        range(value, 1, -1),
        1
    )
```

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



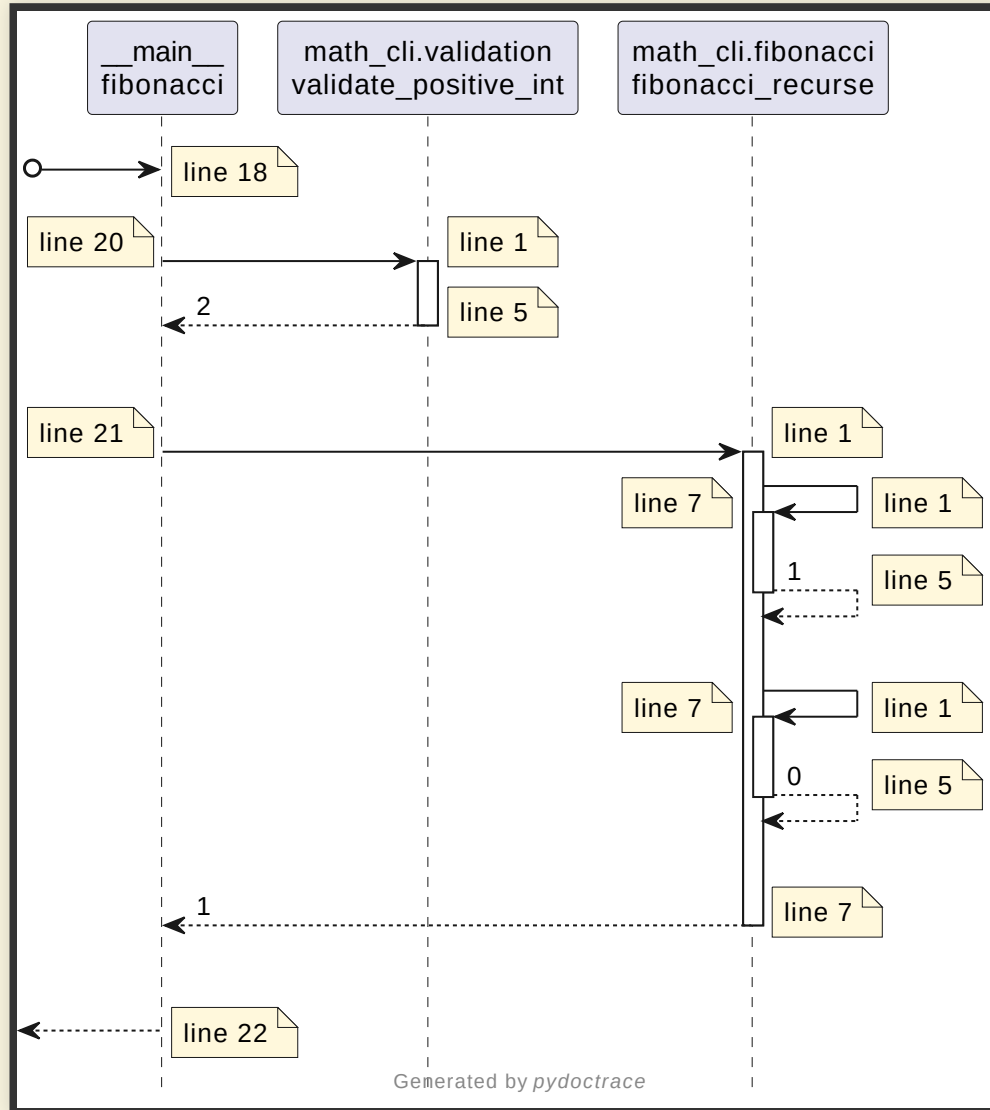
# pydoctrace : factorial reduce

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



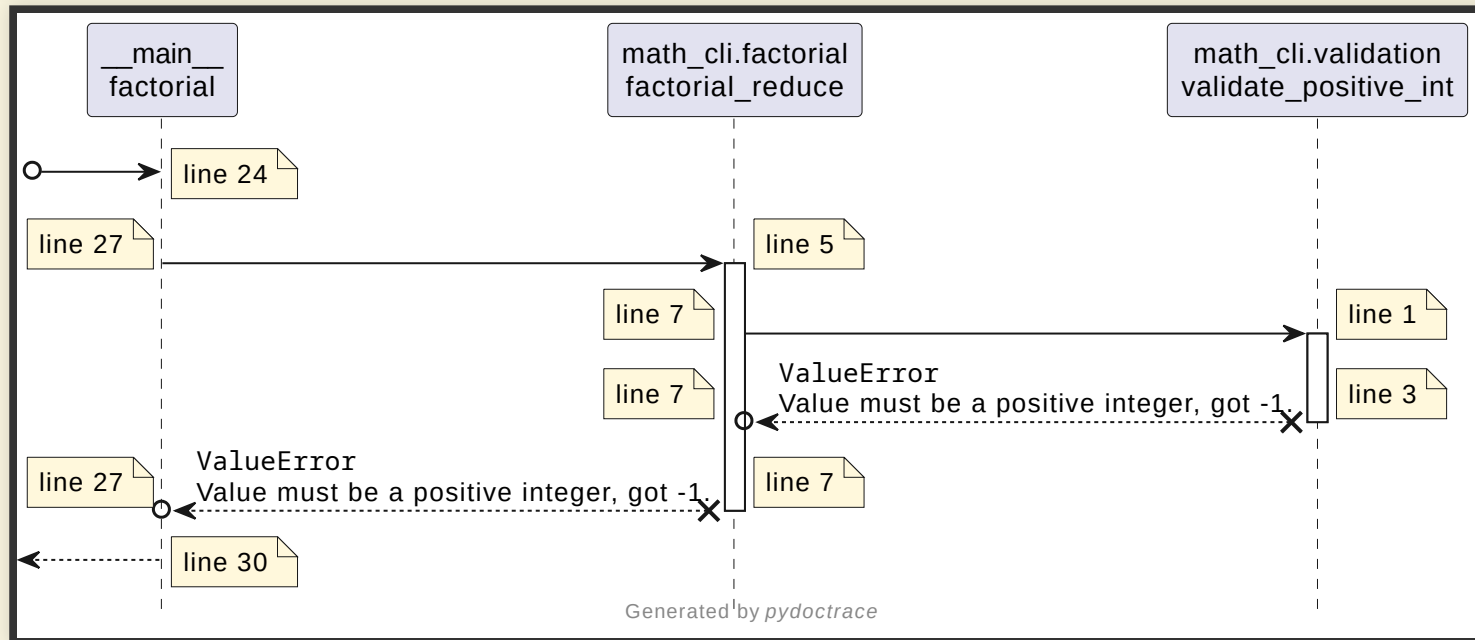
# pydoctrace : fibonacci

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



# pydoctrace : factorial reduce, handled error

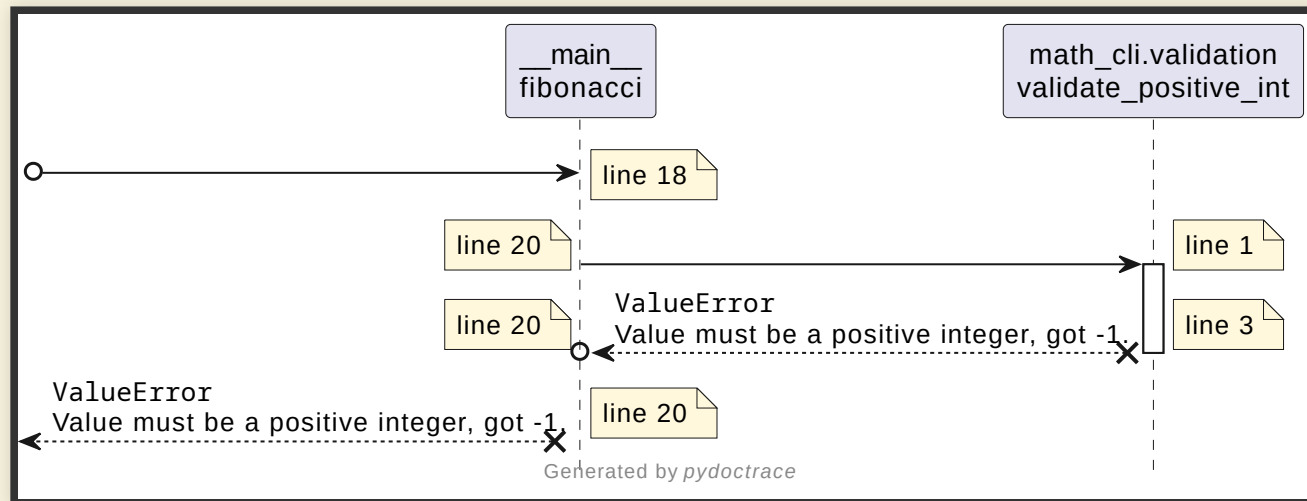
```
python -m math_cli --factorial -1
```





# pydoctrace : fibonacci, unhandled error

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



## Conclusion - doc-tracing


---

- mise en œuvre :
  - API hook des langages interprétés
  - agents pour langages compilés
- architecture & diagramme de séquence : le bon niveau ?

## pydoctrace - pistes d'évolutions

---

- grouper les fonctions par module, les modules par packages
- diagramme de composants avec les flèches d'appels (+ concis)
- export mermaidjs ou structurizr ?
- personnaliser les diagrammes produits
  - exclure des modules du traçage (builtins, outils de tests)
  - fichier de diagramme : destination, nommage (utilisation des valeurs des paramètres ?)
  - skinparams pour PlantUML

À prioriser en fonction des besoins : à vos issues et vos  sur [github.com/lucsorel/pydoctrace](https://github.com/lucsorel/pydoctrace)

Envie d'une présentation plus détaillée ?



Figure 2. Pour rejoindre le slack : <https://tinyurl.com/slack-pythonrennes>

**Merci ! Des questions ?**

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