

Always has been

Wait, it's all monads ?



“A monad is a way of sequencing computations where each step may fail or produce effects, while keeping the plumbing out of your core logic.”

–ChatGPT 4o

$\text{return} :: a \rightarrow$

$>> :: (a \rightarrow b) \rightarrow (b \rightarrow \rightarrow a \rightarrow c)$

$>>= :: (a \rightarrow m c)$



 $[x / y] := \text{img alt="blue diamond icon" data-bbox="500 135 540 195"}[x] / \text{img alt="blue diamond icon" data-bbox="670 135 710 195"}[y]$

What's the value of  $[x / y]$?

How can be  $[x / \emptyset]$ OK?

Let's find out!

(Read $\text{img alt="blue diamond icon" data-bbox="225 785 265 845"}[x]$ as “contains x”)


Kleisli monad in ~Haskell


 is a monad iff:

1. there is a constructor operation

 :: a ->  a - read “ a” as  contains a

2. there is a composition operation

f :: a ->  b

g :: b ->  c

f  g :: a ->  c

(Read `::` as “has type” and `->` as “returns”).)

Writing a Result monad in Python

<https://github.com/bassosimone/kleisli-compose>

```
# Problem statement: measuring internet censorship  
# Initial naive measurement pipeline
```



```
# kliesly_compose/mocks.py
```

```
def dns(domain: str) -> IpAddr: ...
```

```
>>> dns("www.example.com")  
IpAddr("130.192.91.231")
```

```
>>> dns("www.instagram.com")
```

```
Traceback:
```

```
...
```

```
    Raise DNSError("no such domain")
```

```
DNSError: no such domain
```



```
# kliesly_compose/mocks.py
```

```
def fetch(addr: IpAddr) -> WebPage: ...
```

```
>>> fetch(IpAddr("130.192.91.231"))  
WebPage("<html> ...")
```

```
>>> fetch(IpAddr("130.192.91.211"))
```

```
Traceback:
```

```
...
```

```
    raise FetchError("connection reset by peer")
```

```
FetchError: connection reset by peer
```

```
# kliesly_compose/base.py
```

```
def measure(domain: str) -> WebPage:  
    return fetch(dns(domain))
```

```
>>> measure("www.example.com")  
WebPage("<html> ... </html>")
```

```
>>> measure("www.instagram.com")
```

```
Traceback:
```

```
...
```

```
    raise DNSError("no such domain")
```

```
DNSError: no such domain
```

```
# kliesly_compose/base.py & examples/000.py
```

```
def measure_list(domains: list[str]) -> list[WebPage]:  
    result = []  
    for domain in domains:  
        try:  
            result.append(measure(domain))  
        except:  
            pass  
    return result
```

```
>>> measure_list(["www.example.com", "www.instagram.com"])  
[WebPage("<html> ... </html>")]
```

```
# 🤔 "Varo, ubi sunt errores?"
```


Introducing a Monad constructor 

Instead of using the type ``T``, we use ``T | Exception``:

- ``WebPage`` becomes ``WebPage | Exception``
- ``IpAddr`` becomes ``IpAddr | Exception``

We could write this as a `@dataclass`` but it's not as idiomatic as just writing the union type inline.

In other words:

 `T`` corresponds to ``T | Exception``

```
# kliesly_compose/catcher.py & examples/002.py
```

```
def measure_list(
    domains: list[str]) -> list[WebPage | Exception]: # 💎
    result = []
    for domain in domains:
        try:
            result.append(measure(domain))
        except Exception as exc:
            result.append(exc)
    return result
```


```
>>> measure_list(["www.example.com", "www.instagram.com"])
[WebPage("<html> ..."), DNSError("no such host")]
```

Objective: take the exceptions shock away

```
def measure_list(  
    domains: list[str]) -> list[WebPage | Exception]:  
    result = []  
    for domain in domains:  
        result.append(measure(domain))  
    return result
```

How?

```
# kliesly_compose/bettercatcher.py & examples/002.py
```

```
def measure(domain: str) -> WebPage | Exception: # 
```

```
    try:
```

```
        addr = dns(domain)
```

```
    except Exception as exc:
```

```
        return exc
```

```
    try:
```

```
        return fetch(addr)
```


```
    except Exception as exc:
```


```
        return exc
```

Objective: take the exceptions shock away

```
def measure(domain: str) -> WebPage | Exception:  
    return fetchx(dnsx(domain))
```


How?


```
def dnsx(domain: str) -> IpAddr | Exception: #   
    try:  
        return dns(domain)  
    except Exception as exc:  
        return exc
```

```
def fetchx(  
    addr: IpAddr | Exception) -> WebPage | Exception: #   
    if isinstance(addr, Exception):  
        return addr  
    try:  
        return fetch(addr)  
    except Exception as exc:  
        return exc
```

```
# kliesly_compose/landing.py & examples/003.py
# Objective: extract a monadic building block from fetchx
```

```
def fetchx(
    addr: IpAddr | Exception) -> WebPage | Exception:
    if isinstance(addr, Exception):
        return addr
    return fetchxy(addr)
```

```
def fetchxy(addr: IpAddr) -> WebPage | Exception: # 
    try:
        return fetch(addr)
    except Exception as exc:
        return exc
```

```
# Now, this starts to feel very monadic
# f: A -> B | Exception

def dnsx(domain: str) -> IpAddr | Exception: ...

def fetchxy(addr: IpAddr) -> WebPage | Exception: ...

# The time is ripe to write the 🌟 operator!
```

```
class Compose:    # 🌟
    def __init__(
        self,
        fx: Callable[[str], IpAddr | Exception],
        gx: Callable[[IpAddr], WebPage | Exception],
    ):
        self.fx, self.gx = fx, gx

    def __call__(self, value: str) -> WebPage | Exception:
        rv = self.fx(value)
        if isinstance(rv, Exception):
            return rv
        return self.gx(rv)
```

```
# kliesly_compose/full.py & examples/004.py
```

```
def measure(domain: str) -> WebPage | Exception:  
    pipeline = Compose(dnsx, fetchxy) # f 🌟 g  
    return pipeline(domain)
```

```
class Compose[A, B, C]: # Make 🌟 generic!
  def __init__(
    self,
    fx: Callable[[A], B | Exception],
    gx: Callable[[B], C | Exception],
  ):
    self.fx, self.gx = fx, gx

  def __call__(self, value: A) -> C | Exception:
    rv = self.fx(value)
    if isinstance(rv, Exception):
      return rv
    return self.gx(rv)
```

```
# kliesly_compose/fancy.py & examples/005.py
```

```
class Func[A, B]:  
    """Generic monad-aware function."""  
  
    def __init__(self, fx: Callable[[A], B | Exception]):  
        self.fx = fx  
  
    def __call__(self, a: A) -> B | Exception:  
        return self.fx(a)  
  
    def __or__[C](self, other: Func[B, C]) -> Func[A, C]:  
        return Func(Compose(self.fx, other.fx))
```

```
measure = Func(dnsx) | Func(fetchxy) # f 🌻 g
```

So, the measurement algorithm now reduces to

```
def dnsx(domain: str) -> IpAddr | Exception:  
    try:  
        return dns(domain)  
    except Exception as exc:  
        return exc
```

```
def fetchxy(addr: IpAddr) -> WebPage | Exception:  
    try:  
        return fetch(addr)  
    except Exception as exc:  
        return exc
```

```
def measure_list(domains: list[str]) -> list[WebPage | Exception]:  
    return list(map(Func(dnsx) | Func(fetchxy), domains))
```




Sono intorno a noi
sono in mezzo a noi
you cannot unsee this



Digital Hub

Installation

Concepts and Tasks ▼

Projects

Functions and Runtimes

Workspaces

Resource Management with
KRM

Data and transformations

ML Models

Using Kubernetes Resources
for Runs

Secret Management

Workflows

Code source

Using the platform externally

Components ▼

Landing Page

Core UI

CLI

Kubernetes Resource Manager

KubeFlow Pipelines

Interactive Workspaces ▼

Workflows

Workflows allow for organizing the single operations in a advanced management pipelines, to perform a series operation of data processing, ML model training and serving, etc. Workflows represent long-running procedures defined as Directed Acyclic Graphs (DAGs) where each node is a single unit of work performed by the platform (e.g., as a Kubernetes Job).

As in case of functions, it is possible for the platform to have different workflow runtimes. Currently, the only workflow runtime implemented is the one based on Kubeflow Pipelines infrastructure. See [KFP Runtime](#) for further details about how the workflow is defined and executed with the Kubeflow Pipelines component of the platform.

Similarly, to functions the workflows may be managed via console UI or via Python SDK.

Management via UI

Workflows can be created and managed as *entities* from the console. You can access them from the dashboard or the left menu. You can:

- `create` a new workflow
- `expand` a workflow to see its 5 latest versions
- `show` the details of a workflow

Table of contents

Management via UI

Create

Read

Update

Delete

Management via SDK

← → ↺ 🌐 github.com/fbk-most/civic-digital-twins/blob/main/.github/workflows/test.yml

📁 main ▾ civic-digital-twins / .github / workflows / test.yml

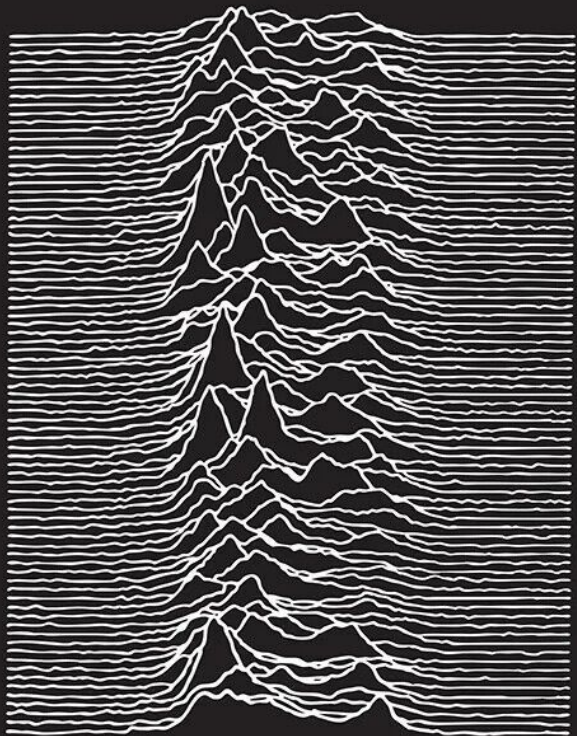
Code Blame 51 lines (40 loc) · 1.29 KB

```
13     matrix:
14         python_version:
15             - "3.11"
16             - "3.12"
17             - "3.13"
18
19     name: Test with Python ${ matrix.python_version }
20
21     steps:
22     - uses: actions/checkout@v4
23
24     - name: Install uv
25       uses: astral-sh/setup-uv@v5
26
27     - name: Install dependencies
28       run: uv sync --python ${ matrix.python_version } --dev
29
30     - name: Check formatting with ruff
31       if: matrix['python_version'] == '3.11'
32       run: uv run --python ${ matrix.python_version } ruff format --check .
33
34     - name: Lint with ruff
35       if: matrix['python_version'] == '3.11'
36       run: uv run --python ${ matrix.python_version } ruff check .
37
38     - name: Check types with pyright
39       if: matrix['python_version'] == '3.11'
40       run: uv run --python ${ matrix.python_version } pyright
41
42     - name: Run tests with pytest
43       run: uv run --python ${ matrix.python_version } pytest
44
45     - name: Upload coverage to Codecov
46       uses: codecov/codecov-action@v5
47       if: matrix['python_version'] == '3.11'
```

```
% set -euo pipefail
```

```
% find . -type f -name *.py |  
    awk -F/ '{print $NF}' |  
    cat |  
    uniq -c |  
    cat |  
    sort -rnk1 |  
    head -n1
```

JOY DIVISION



UNKNOWN PLEASURES

We (aim to) build robust systems by accepting failure as part of the computation, not as an afterthought.

Monads—like ``Compose``—help us structure this with grace.

But, is this real fun?

—

I've got the spirit
And I've lost the feeling
Take the shock away 💎