MONADIC ERROR HANDLING

https://przemyslawjanpietrzak.github.io/przemyslawjanpietrzak.github.io/monadic-error-handling/dist

CLASSIC ERROR HANDLING

```
def do():
  if err
      raise MyCustomException
  return data


try:
    data = do()
    handle(data)
except MyCustomException as e:
    handle(e)
```

LATER

```
def do():
    if err
        raise MyCustomException
    elif other_err:
        raise MyAnotherCustomException
    return data

try:
    data = do()
    handle(data)
except MyCustomException as e:
    handle(e)
except MyAnotherCustomException as e:
    handle(e)
```

SOMETHING SIMPLER?

```
err, f := os.Open("filename.ext")
if err != nil {
    log.Fatal(err)
}
// do something with the open *File f:

err, _ := os.Create("/tmp/file")
if err != nil {
    panic(err)
}
```

WHAT ABOUT SCALA?

```
val in = Console.readLine("Type Either a string or an Int: ")
val result: Either[String, Int] = try {
    Right(in.toInt)
  } catch {
    case e: Exception =>
      Left(in)
}

println(result match {
  case Right(x) => "Success :)"
  case Left(x)  => "Error :("
})
```

MORE EITHER

```
from pymonet.either import Left, Right

def divide(divided, divider):
    if divider == 0:
        return Left('can not divide by 0')
    return Right(divided / divider)

def handle_error(value):
    print ('error {}'.format(value))

def handle_success(value):
    print ('success {}'.format(value))

divide(42, 0)
    .map(lambda value: value + 1)
    .case(error=handle_error, success=handle_success)
# error can not divide by 0

divide(42, 1)
    .map(identity, lambda value: value + 1)
    .case(error=handle_error, success=handle_success)
# success 43
```

LET'S GO DEEPER

```
from pymonet.either import Left, Right

divide(42, 1)
    .map(lambda value: value + 1)
    .bind(lambda value: divide(43, value))
    .case(error=handle_error, success=handle_success)
# success 1

map(self, mapper: (A) -> B): Either[B]
bind(self, mapper: (A) -> Either[B]): Either[B]
```

LET'S GO DEEPER

```
from pymonet.either import Left, Right

divide(2, 1)
    .map(lambda value: value - 1)
    .fold(lambda value: divide(42, value))
    .case(error=handle_error, success=handle_success)
# error DivisionError...
```

MONAD FOR ABSENCE ... MAYBE?

```
from pymonet.Maybe import Maybe

def find_something(item):
    if item in special_list:
        return Maybe.just(item)
    return Maybe.nothing()

find_something(42).get_or_else(0)  # 42
find_something(41).get_or_else(0)  # 0
```

LAZINESS

```
from pymonet.lazy import Lazy

def fn():
    print('fn call')
    return 42

def mapper(value):
    print('mapper side effect of ' + value)
    return value + 1

def side_effect(value):
    print('side effect of ' + value)

lazy = Lazy(fn)
mapped_lazy = lazy.map(mapper)
mapped_lazy.fold(side_effect)
# fn call
# mapper side effect of 42
# side effect of 43

mapped_lazy.fold(side_effect)
# side effect of 43
```

MORE LAZY

```
from pymonet.task import Task

def resolvable_fn(reject, resolve):
    print('resolve side effect')
    resolve(42)

def rejectable_fn(reject, resolve):
    print('reject side effect')
    reject(0)

resolvable_task = Task.of(resolvable_fn)
rejectable_task = Task.of(rejectable_fn)

def mapper(value):
  print('mapper side effect ' + value)
  return value + 1

resolvable_task.fold(mapper)
# resolve side effect
# mapper side effect 42

rejectable_task.fold(mapper)
# reject side effect
```

TRY (TECHNICALLY IT'S NOT A MONAD)

```
from pymonet.monad_try import Try

def divide(dividend, divisor):
    return dividend / divisor

def success_callback(value):
    print('success: {}'.format(value))

def fail_callback(error):
    print('error: {}'.format(value))

(Try.of(divide, 42, 2)
    .on_success(success_callback)
    .on_fail(fail_callback))
# success: 21

(Try.of(divide, 42, 0)
    .on_success(success_callback)
    .on_fail(fail_callback))
#error: division by zero
```

APPLICATIVES

```
from pymonet.maybe import Maybe

applicative = Maybe.of(lambda value: value + 1)
applicative.ap(Maybe.of(42)) # Maybe[43]

nothing = Maybe.nothing()
applicative.ap(Maybe.of(42)) # Maybe[]
```

WHY?

- Less exceptions
- More expressive
- More modular
- More testable
- Immutable

HOW?

- Composable Functional JavaScript
- JavaScript: fantasy-land, immutable-ext
- Java: vavr.io
- Python: OSlash, pyMonet

LINKS

- https://github.com/przemyslawjanpietrzak/pyMonet
- https://github.com/fantasyland/fantasy-land
- https://egghead.io/courses/professor-frisby-introduces-composable-functional-javascript
- https://github.com/DrBoolean/immutable-ext
- http://www.vavr.io/
- https://github.com/FormidableLabs/spectacle (for presentatnion)

THANK YOU ;*