

Simplifying Python Code through the Lens of Functional Programming

Artem Kislovskiy
CC-BY-SA 4.0

Teach Yourself Programming in Ten Years

<https://norvig.com/21-days.html>

Learn at least a **half dozen programming languages**. Include one language that emphasizes class abstractions (like Java or C++), one that emphasizes **functional abstraction** (like Lisp or ML or Haskell), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), and one that emphasizes parallelism (like Clojure or Go).

⌚ Tuesday, April 21, 2009

Origins of Python's "Functional" Features

I have never considered Python to be heavily influenced by functional languages, no matter what people say or think. I was much more familiar with imperative languages such as C and Algol 68 and although I had made functions first-class objects, I didn't view Python as a functional programming language. However, earlier on, it was clear that users wanted to do much more with lists and functions.

<https://python-history.blogspot.com/2009/04/origins-of-pythons-functional-features.html>

```
$ bat src/pipeline/data/example.csv
```

```
File: src/pipeline/data/example.csv
```

	Name,Score,Grade
1	Alice,90,A
2	Bob,85,B
3	Charlie,78,C
4	David,92,A
5	Eva,88,B
6	Frank,79,C
7	Grace,95,A

ChatGPT ▾



KI

As an experienced Python software engineer, write a program that reads an `example.csv` file that contains two columns: Name, Score, and Grade. The program should compute the average score for the class.

Here is an example of CSV file:

```
Name,Score,Grade,  
Alice,90,A  
Bob,85,B  
Charlie,78,C  
David,92,A  
Eva,88,B  
Frank,79,C  
Grace,error_two,A  
Monica,error_one,A
```



Certainly! To achieve this, we can use the `'csv'` module in Python for reading the CSV file, and handle any potential errors in the data gracefully. Here's a complete program that reads the CSV file and computes the average score, ignoring any rows with invalid scores.

First, let's create the `'example.csv'` file:

```
import csv  
  
def read_csv_and_compute_average(filename):  
    scores = []  
  
    with open(filename, newline='') as csvfile:  
        reader = csv.DictReader(csvfile)  
        for row in reader:  
            try:  
                # Try converting the Score to an integer  
                score = int(row['Score'])  
                scores.append(score)  
            except ValueError:  
                # Skip the row if the conversion fails  
                print(f"Invalid score '{row['Score']}' for {row['Name']}, skipping.")  
  
    if scores:  
        average_score = sum(scores) / len(scores)  
        print(f"The average score for the class is: {average_score:.2f}")  
    else:  
        print("No valid scores found.")  
  
# Call the function with the CSV file path  
read_csv_and_compute_average('example.csv')
```

```
import csv

def read_csv_and_compute_average(filename):
    scores = []

    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            try:
                # Try converting the Score to an integer
                score = int(row['Score'])
                scores.append(score)
            except ValueError:
                # Skip the row if the conversion fails
                print(f"Invalid score '{row['Score']}' for {row['Name']}, skipping.")

    if scores:
        average_score = sum(scores) / len(scores)
        print(f"The average score for the class is: {average_score:.2f}")
    else:
        print("No valid scores found.")

# Call the function with the CSV file path
read_csv_and_compute_average('example.csv')
```

```
from pathlib import Path
import polars as pl

if __name__ == "__main__":
    current_dir_path = Path(__file__).parent
    csv_file_path = current_dir_path.joinpath("example.csv")

    print(
        pl.read_csv(csv_file_path)
        .select("Score")
        .cast(pl.Float32, strict=False)
        .mean()
    )
```

```

import csv

def read_csv_and_compute_average(filename):
    scores = []

    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            try:
                # Try converting the Score to an integer
                score = int(row['Score'])
                scores.append(score)
            except ValueError:
                # Skip the row if the conversion fails
                print(f"Invalid score '{row['Score']}' for {row['Name']}, skipping.")

    if scores:
        average_score = sum(scores) / len(scores)
        print(f"The average score for the class is: {average_score:.2f}")
    else:
        print("No valid scores found.")

# Call the function with the CSV file path
read_csv_and_compute_average('example.csv')

```

```

from pathlib import Path
import polars as pl

if __name__ == "__main__":
    current_dir_path = Path(__file__).parent
    csv_file_path = current_dir_path.joinpath("example.csv")

    print(pl.read_csv(csv_file_path)
          .select("Score")
          .cast(pl.Float32, strict=False)
          .mean())
)

```

Just Because You Can, It Doesn't Mean You Should...

Time over 100 runs

	n=10		n=1e6
Time ChatGPT:	0.0018s		61.0192s
Time polars:	0.0078s		0.71230s
Time pipe:	0.0031s		29.6276s
Time pymonad:	0.0020s		28.1341s

n - size of the dataset

concept



time

- A function should do one thing and do it well
- Currying
- Higher Order Functions
- Arity
- Function Composition
- Monads

```
from pathlib import Path
import polars as pl

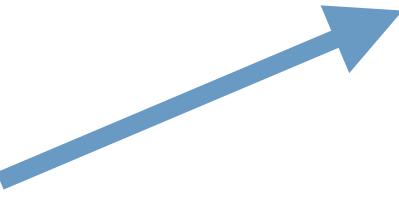
if __name__ == "__main__":
    csv_file_path = Path(__file__).parent / "data" / "example.csv"

    print(pl.read_csv(csv_file_path)
          .select("Score")
          .cast(pl.Float32, strict=False)
          .mean())
)
```

```
2024_PyConIT > src > pipeline > pipeline_b_imperative.py > read_csv_file
1 import csv
2 from pathlib import Path
3
4
5 def read_csv_and_compute_average(filename):
6     scores = []
7
8     with open(filename, newline="") as csvfile:
9         reader = csv.DictReader(csvfile)
10        for row in reader:
11            try:
12                # Try converting the Score to an integer
13                score = int(row["Score"])
14                scores.append(score)
15            except ValueError:
16                # Skip the row if the conversion fails
17                print(f"Invalid score '{row['Score']}' found")
18
19    if scores:
20
21        average_score = sum(scores) / len(scores)
22        print(f"The average score for the class is: {average_score}")
23    else:
24        print("No valid scores found.")
25
26 def main():
27     csv_file_path = Path(__file__).parent / "data" / "example.csv"
28     read_csv_and_compute_average(csv_file_path)
29
30
31 if __name__ == "__main__":
32     # Call the function with the CSV file path
33     read_csv_and_compute_average(Path(__file__).parent / "data" / "example.csv")
```

```
1 import csv
2 from pathlib import Path
3
4
5 def read_csv_file(file_path):
6     try:
7         with open(file_path, "r") as csv_file:
8             reader = csv.reader(csv_file)
9             return [row for row in reader]
10    except FileNotFoundError:
11        return None
12
13
14 def extract_column(index, rows):
15    try:
16        column_values = [float(row[index]) for row in rows[1:]]
17        return column_values
18    except (ValueError, IndexError):
19        return None
20
21
22 def calculate_average(column_values):
23    if column_values is None or not column_values:
24        return None, "Cannot calculate average due to empty or missing data"
25    try:
26        average = sum(column_values) / len(column_values)
27        return average
28    except ZeroDivisionError:
29        return None
30
31
32 if __name__ == "__main__":
33     csv_file_path = Path(__file__).parent / "data" / "example.csv"
34     column_index = 1
35     data = read_csv_file(csv_file_path)
36
37     if data is None:
38         print("Error reading CSV file")
39     else:
40         score_column_values = extract_column(column_index, data)
41         if score_column_values is None:
42             print("Error extracting column")
43         else:
```

How many things
does this
function?



```
def extract_column(index, rows):
    try:
        column_values = [float(row[index]) for row in rows[1:]]
        return column_values
    except (ValueError, IndexError):
        return None

def calculate_average(column_values):
    if column_values is None or not column_values:
        return None
    try:
        average = sum(column_values) / len(column_values)
        return average
    except ZeroDivisionError:
        return None

if __name__ == "__main__":
    ...
    data = read_csv_file(csv_file_path)
    ...
    if data is None:
        ...
    else:
        score_column_values = extract_column(column_index, data)
        if score_column_values is None:
            ...
        else:
            result = calculate_average(score_column_values)
            if result is None:
                ...
            else:
                ...
```

```
def extract_column(index, data):
    try:
        column_values = [float(row[index]) for row in data[1:]]
        return column_values
    except (ValueError, IndexError) as e:
        return None
```

Do one thing and do it well

```
def extract_column(index, rows):
    try:
        column_values = [float(row[index]) for row in rows[1:]]
        return column_values
    except (ValueError, IndexError):
        return None
```



```
def extract_column(column_index, rows):
    try:
        return [row[column_index] for row in rows]
    except (ValueError, IndexError):
        return None
```

```
def remove_row(row_index, rows):
    try:
        return rows[row_index:]
    except IndexError:
        return None
```

```
def convert_to(converter, rows):
    try:
        return [converter(item) for item in rows]
    except ValueError:
        return None
```

```
if __name__ == "__main__":  
  
    csv_file_path = Path(__file__).parent / "data" / "example.csv"  
  
    score_column_index = 1  
    header_row_index = 1  
  
    data = read_csv_file(csv_file_path)  
  
    if data is None:  
        print("Error reading CSV file")  
    else:  
        score_column_values = extract_column(score_column_index, data)  
        removed_header_data = remove_row(header_row_index, score_column_values)  
        score_column_as_float = convert_to(float, removed_header_data)  
  
        if score_column_as_float is None:  
            print("Error extracting column")  
        else:  
            result = calculate_average(score_column_as_float)  
            if result is None:  
                print("Error calculating average")  
            else:  
                print(f"The average is: {result}")
```

```
from pathlib import Path
import polars as pl

if __name__ == "__main__":
    csv_file_path = Path(__file__).parent / "data" / "example.csv"

    print(pl.read_csv(csv_file_path)
          .select("Score")
          .cast(pl.Float32, strict=False)
          .mean()
    )
```

Function composition

```
result = (  
    read_csv_file(csv_file_path)  
    .then(remove_header)  
    .then(extract_score_column)  
    .then(convert_to_float)  
    .then(calculate_average)  
)
```

- Unary Function
- Currying
- Composition Technics

Currying and Higher-Order-Function

Uncurrying

$f(a, b, c)$

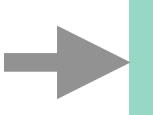
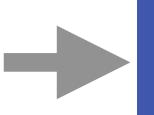
$f(a, b, c) \Rightarrow \text{value};$

(  ) \Rightarrow   

Currying

$f(a)(b)(c)$

$f = a \Rightarrow b \Rightarrow c \Rightarrow \text{value};$

 \rightarrow  \rightarrow  \rightarrow   

Unary functions

```
def plus(a):
    def add(b):
        return a + b
    return add

if __name__ == "__main__":
    result = plus(2)(3)
    # Partial apply, transform binary function to unary function
    plusTwo = plus(2)
    print(plusTwo(3))
```

Arity :> Unary : Why currying is so important

- $f(x)$
- $g(x)$
- $h(x)$
- $\text{compose}(f, g, h)(x)$

$g.f(x) = g(f(x))$

```
basic_call = subtract_five(multiply_by_three(add_two(2)))
```

```
chain_call = pipe(2, add_two, multiply_by_three, subtract_five)
```

```
def extract_column_currying(index):
    def curried(rows):
        try:
            score_column_values = [row[index] for row in rows]
            return score_column_values
        except (ValueError, IndexError):
            return None
    return curried

if __name__ == "__main__":
    data = read_csv_file("data/example.csv")

    name_column_index = 0
    score_column_index = 1

    name_list = extract_column_currying(name_column_index)
    score_list = extract_column_currying(score_column_index)

    print(f"Extracted Column: {score_list(data)}")
    print(f"Extracted Column: {name_list(data)}")
```

```
from toolz import pipe

def add_two(x):
    return x + 2

def multiply_by_three(x):
    return x * 3

def subtract_five(x):
    return x - 5

if __name__ == "__main__":
    basic = subtract_five(multiply_by_three(add_two(2)))
    result = pipe(2, add_two, multiply_by_three, subtract_five)
    print(basic)
    print(result)
```

```

from toolz import curry

@curry
def extract_column(column_index, rows):
    try:
        return [row[column_index] for row in rows]
    except (ValueError, IndexError):
        return None

@curry
def remove_row(row_index, rows):
    try:
        return rows[row_index:]
    except IndexError:
        return None

@curry
def convert_to(converter, rows):
    try:
        return [converter(item) for item in rows]
    except ValueError:
        return None

if __name__ == "__main__":
    csv_file_path = Path(__file__).parent / "data" / "example.csv"
    score_column_index = 1
    header_row_index = 1

    score_column = extract_column(score_column_index)
    removed_header = remove_row(header_row_index)
    score_as_float = convert_to(float)

    average_result = pipe(
        read_csv_file(csv_file_path),
        score_column,
        removed_header,
        score_as_float,
        calculate_average,
    )

    print(f"An average score is {average_result}")

```

What about error handling?

```
@curry
def extract_column(column_index, rows):
    try:
        return [row[column_index] for row in rows]
    except (ValueError, IndexError):
        return None

@curry
def remove_row(row_index, rows):
    try:
        return rows[row_index:]
    except IndexError:
        return None

@curry
def convert_to(converter, rows):
    try:
        return [converter(item) for item in rows]
    except ValueError:
        return None

if __name__ == "__main__":
    current_dir_path = Path(__file__).parent
    csv_file_path = current_dir_path.joinpath(
        "data", "example.csv"
    )

    score_column_index = 1
    header_row_index = 1

    score_column = extract_column(score_column_index)
    removed_header = remove_row(header_row_index)
    score_as_float = convert_to(float)

    average_result = pipe(
        read_csv_file(csv_file_path),
        score_column,
        removed_header,
        score_as_float,
        calculate_average,
    )
    print(f"An average score is {average_result}")
```

What about error handling?

```
from toolz import curry

@curry
def divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError as e:
        raise ValueError(f"Error: {e}")

if __name__ == "__main__":
    try:
        result = divide(10)(0)
    except ValueError as e:
        print(f"Error handling: {e}")
```

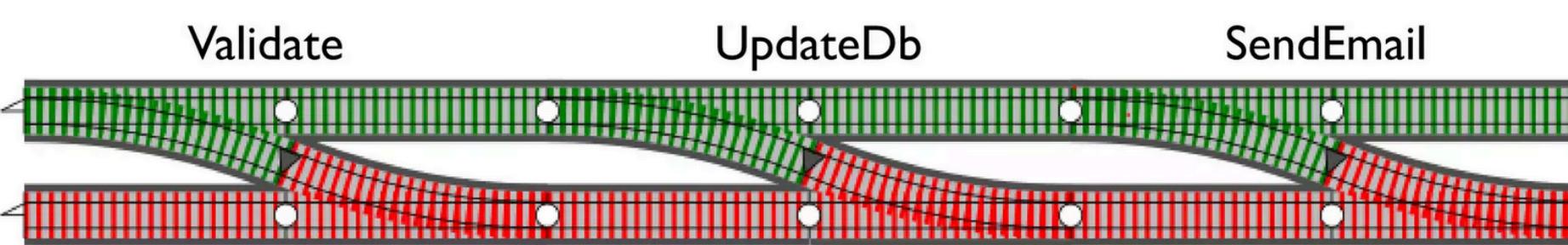
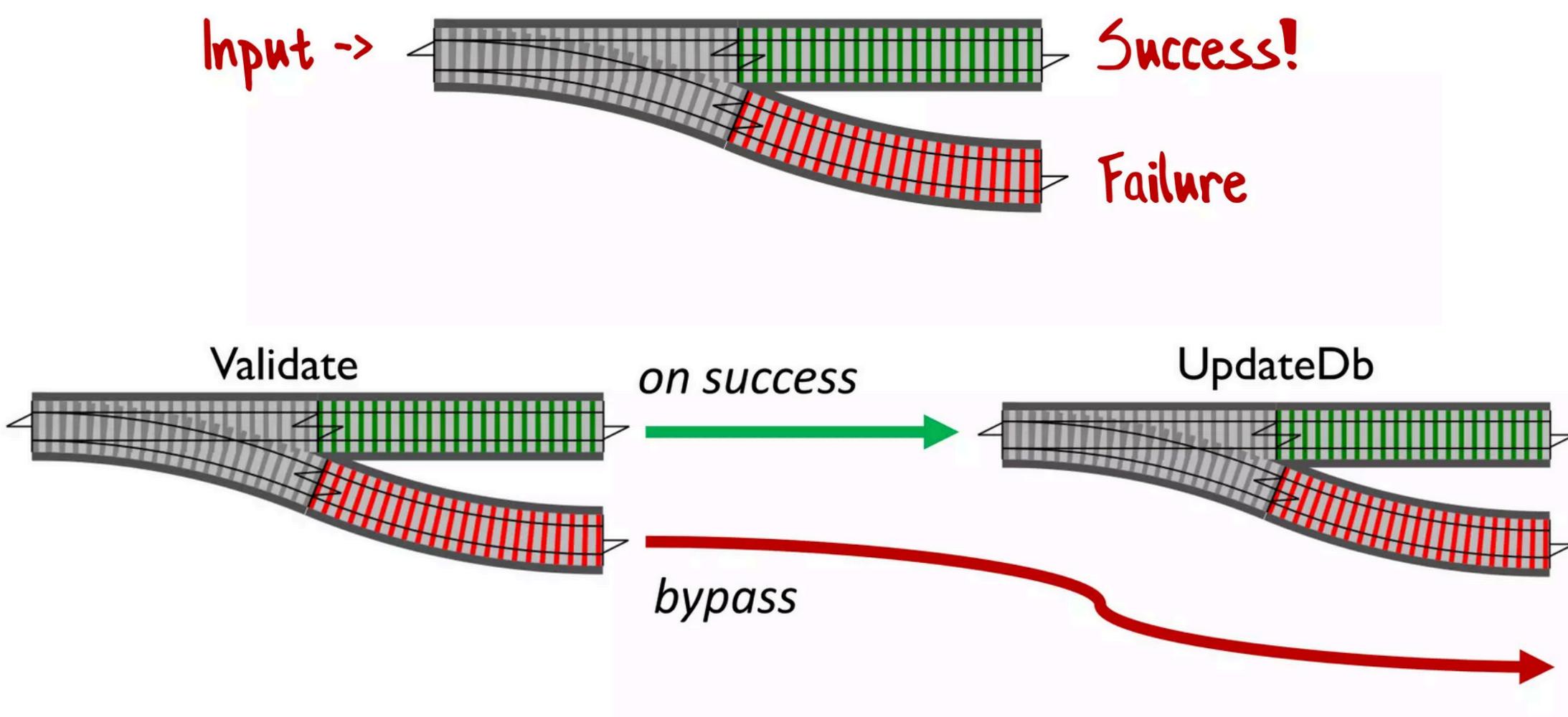
```
if __name__ == "__main__":
    csv_file_path = Path(__file__).parent / "data" / "example.csv"

    score_column_index = 1
    header_row_index = 1

    score_column = extract_column(score_column_index)
    remove_header = remove_row(header_row_index)
    convert_score_to_float = convert_to(float)

    try:
        average_result = pipe(
            read_csv_file(csv_file_path),
            score_column,
            remove_header,
            convert_score_to_float,
            calculate_average,
        )
        print(f"An average score is {average_result}")
    except (FileNotFoundException, ValueError, IndexError, ZeroDivisionError) as e:
        print(f"Exception caught: {e}")
```

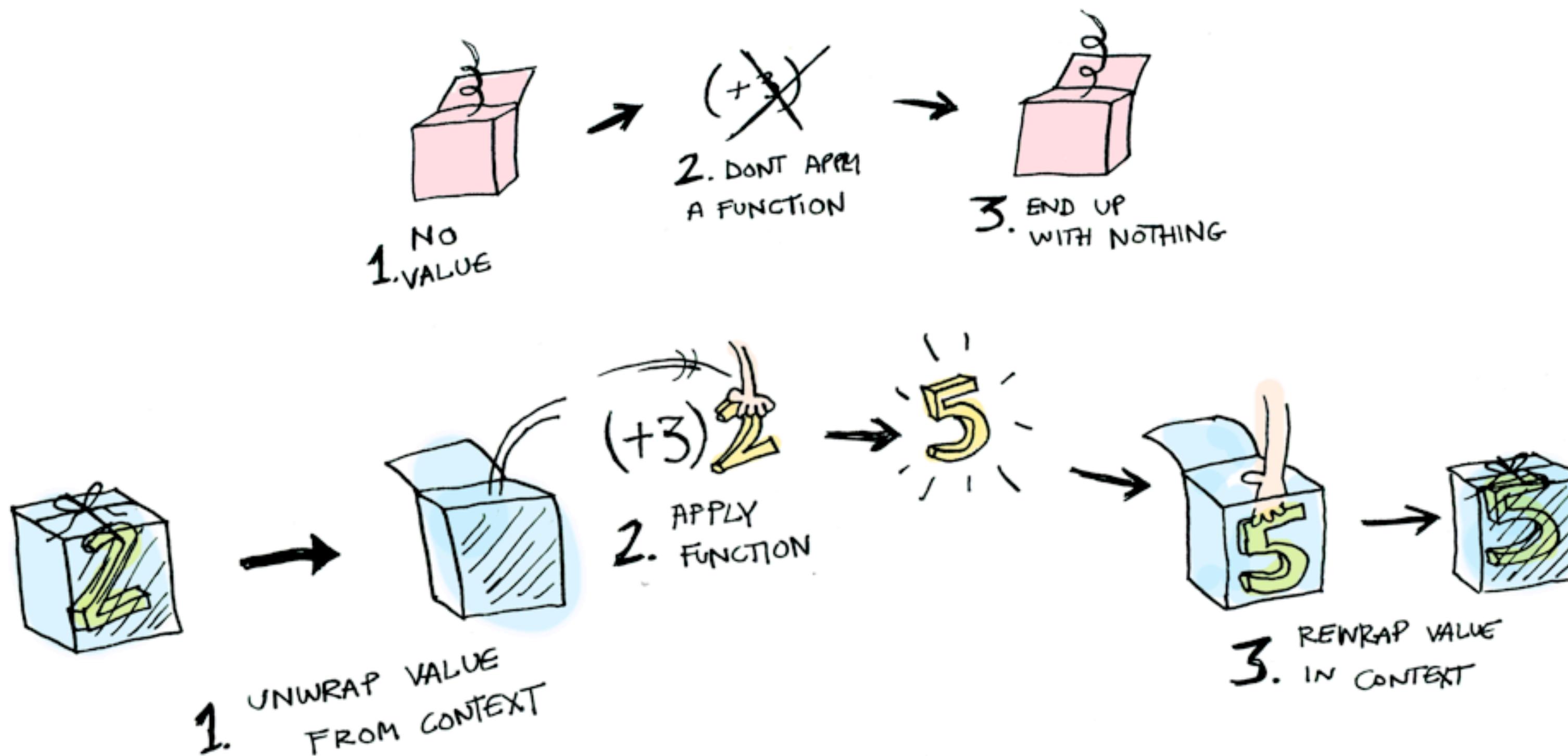
Monad and Railway Oriented Programming



This is the "two track" model –
the basis for the "Railway Oriented Programming"
approach to error handling.

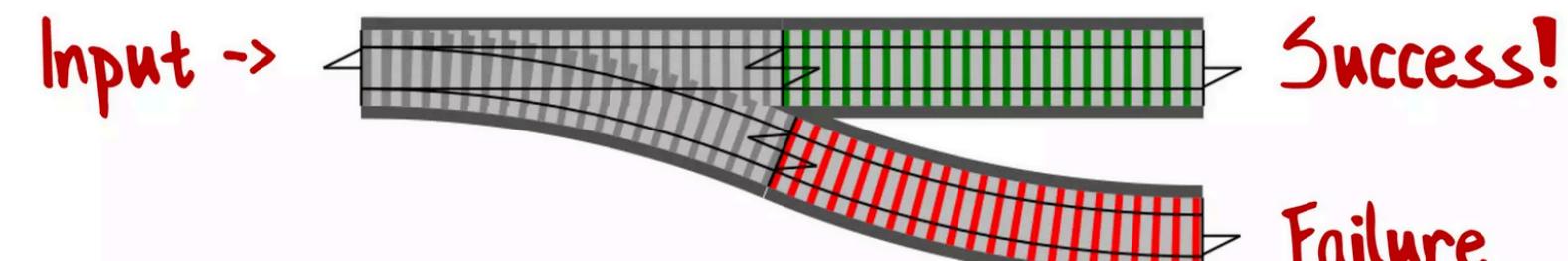
<https://www.slideshare.net/ScottWlaschin/railway-oriented-programming>

Monad

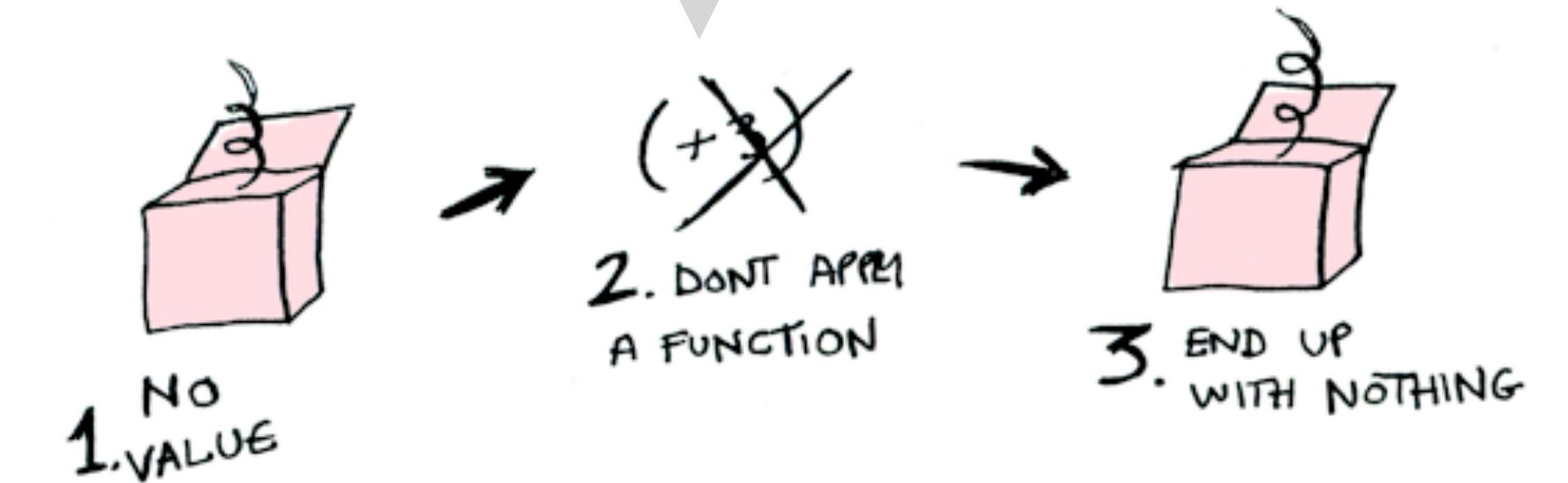
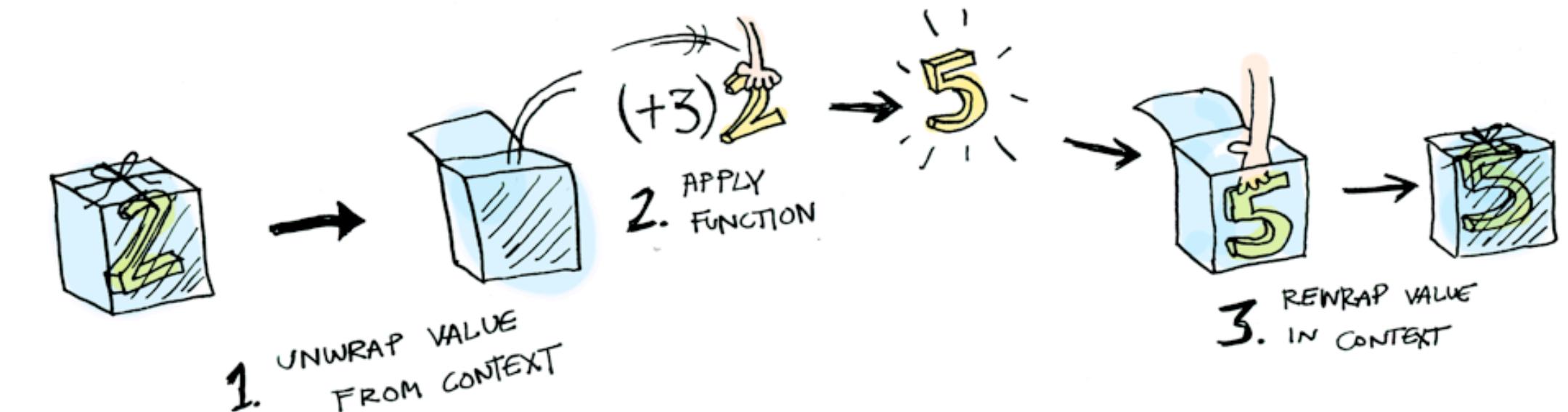


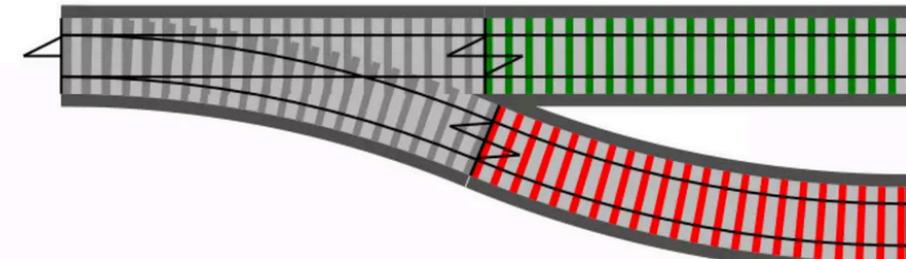
Haskell: https://www.adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html

Python: <https://github.com/dbrattli/0Slash/wiki/Functors,-Applicatives,-And-Monads-In-Pictures#functors>



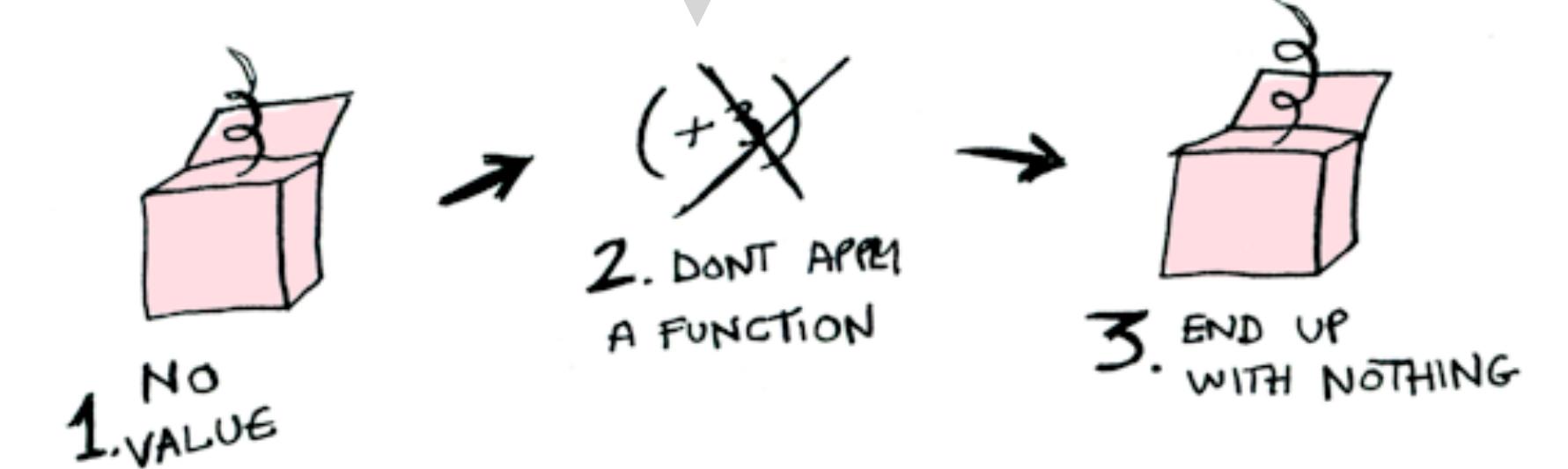
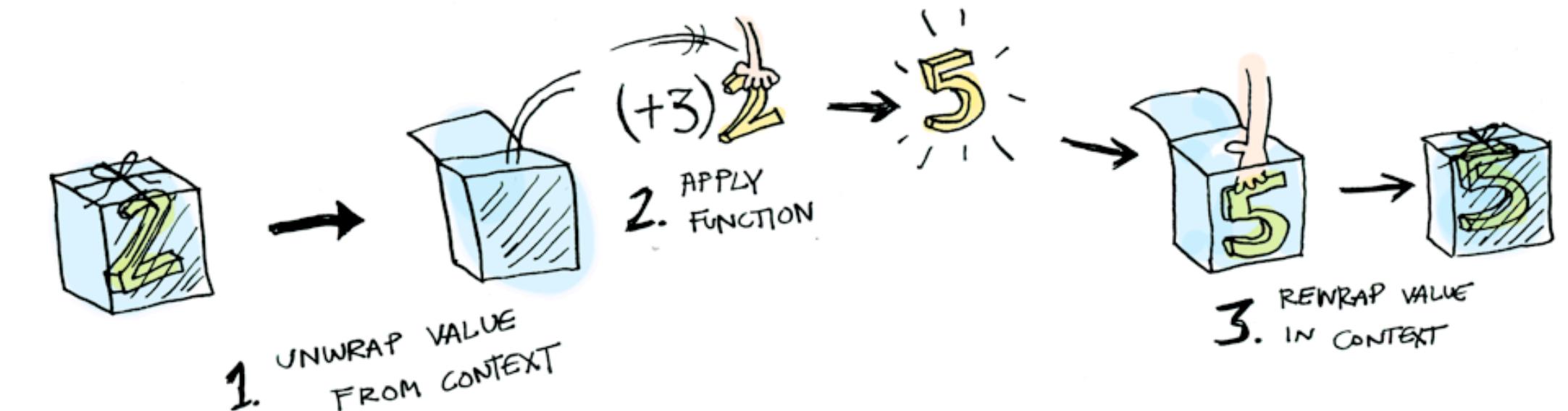
```
def read_csv_file(file_path):
    return (
        Right(list(map(lambda row: row, csv.reader(open(file_path, "r")))))
        if os.path.isfile(file_path)
        else Left("Error: File does not exist")
    )
```



Input ->  Success!

Failure

```
def read_csv_file(file_path):  
    return (  
        Right(list(map(lambda row: row, csv.reader(open(file_path, "r")))))  
        if os.path.isfile(file_path)  
        else Left("Error: File does not exist")  
)
```



```

from pymonad.either import Left, Right
from pmonad.tools import curry

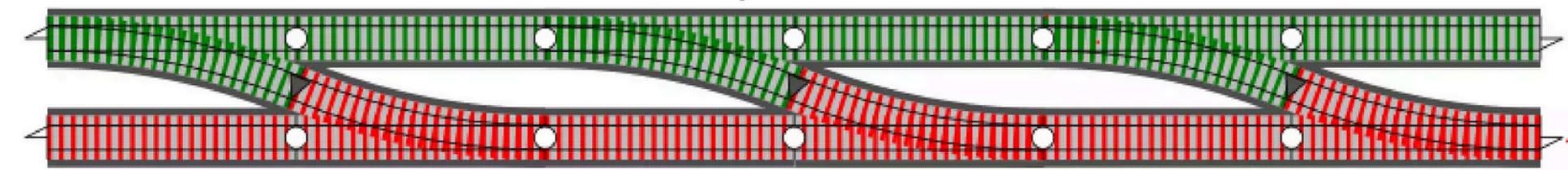
def read_csv_file(file_path):
    return (
        Right(list(map(lambda row: row, csv.reader(open(file_path, "r")))))
        if os.path.isfile(file_path)
        else Left("Error: File does not exist")
    )

def remove_header(data):
    return Right(data[1:]) if len(data) > 1 else Left("Error: U
@curry(2)
def extract_column(index: int, rows):
    return Right(rows).bind(lambda rows: Right(list(map(lambda
def convert_to_float(data):
    return (
        Right(list(map(float, data)))
        if all([num.isnumeric() for num in data])
        else Left("Error: Unable to convert to float")
    )

def calculate_average(column_values):
    return (
        Right(sum(column_values) / len(column_values))
        if column_values
        else Left("Error: Division by zero")
    )

```

Monad and Railway Oriented Programming



```

def compute_average_score(file_path: Path):
    extract_score_column = extract_column(1)

    result = (
        read_csv_file(file_path)
        .then(remove_header)
        .then(extract_score_column)
        .then(convert_to_float)
        .then(calculate_average)
    )

    if result.is_right():
        print(f"An average score of is {result.value}")
    else:
        print(result.either(lambda x: f"Error processing data:"))

```

Final functional implementation

```
def read_csv_file(file_path):
    ...
def remove_header(data):
    ...
@curry(2)
def extract_column(index: int, rows):
    ...
def convert_to_float(data):
    ...
def calculate_average(column_values):
    ...

if __name__ == "__main__":
    extract_score_column = extract_column(1)
    result = (
        read_csv_file(file_path)
        .then(remove_header)
        .then(extract_score_column)
        .then(convert_to_float)
        .then(calculate_average)
    )
    if result.is_right():
        ...
    else:
        ...
```

We don't have to think like a machine

Immutability

- declarative, functional
- relationships between values
- rigorous reasoning
- what is it?

The Machine

- Imperative
- Step-by-step statements
- Operational reasoning
- what does it do?

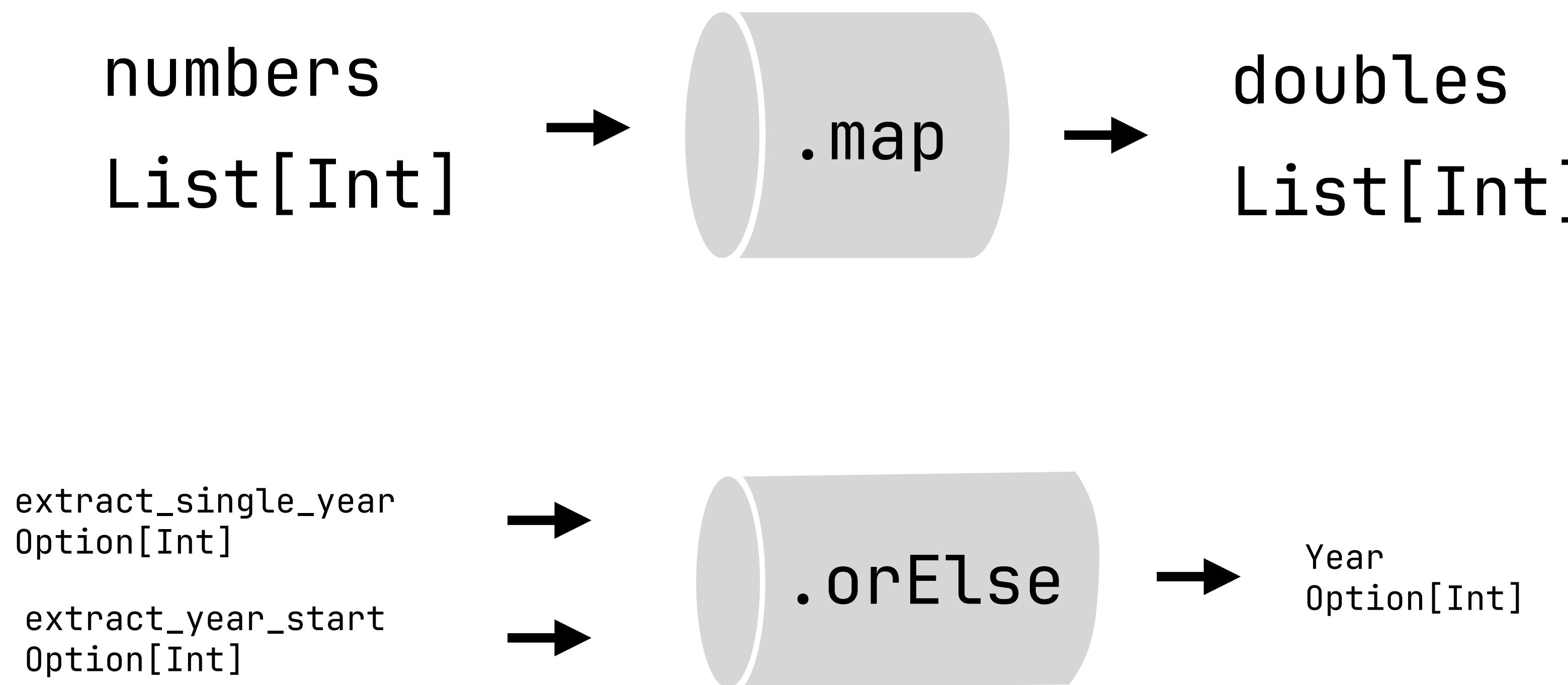


Time over 100 runs

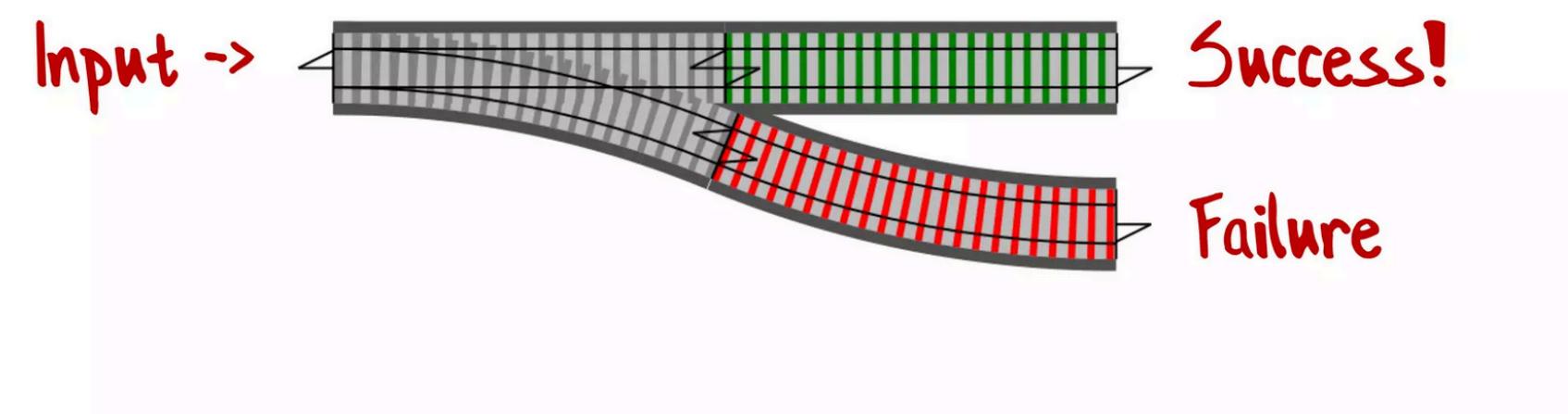
	n=10		n=1e6
Time ChatGPT:	0.0018s		61.0192s
Time polars:	0.0078s		0.71230s
Time pipe:	0.0031s		29.6276s
Time pymonad:	0.0020s		28.1341s

n - size of the dataset

Pure functions. Passing immutable values around

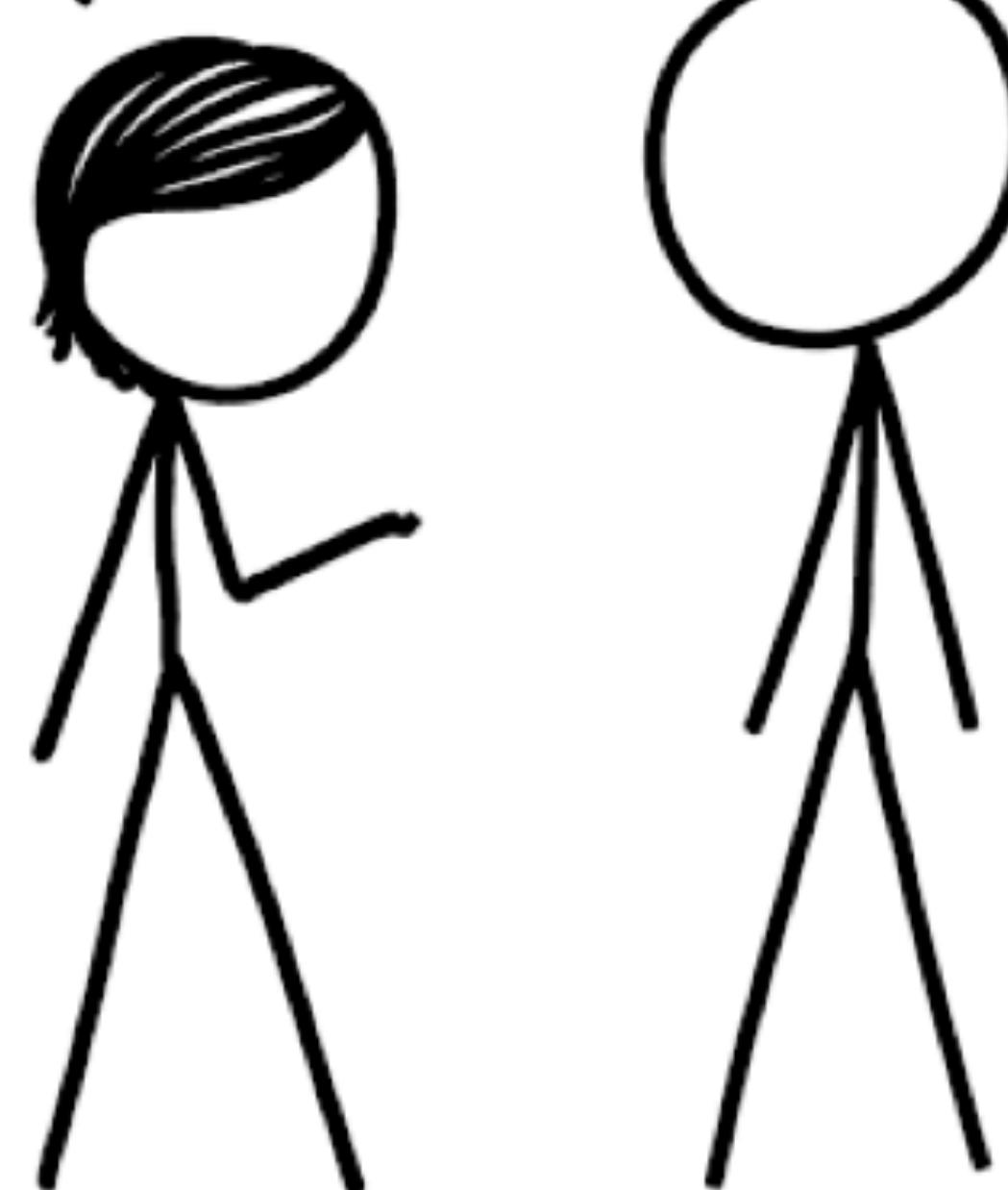


What about talking to the outside world?



CODE WRITTEN IN HASKELL
IS GUARANTEED TO HAVE
NO SIDE EFFECTS.

...BECAUSE NO ONE
WILL EVER RUN IT?



<https://xkcd.com/1312>

Python libraries to enhance functional toolkit

- <https://github.com/tobgu/pyrsistent>
- <https://github.com/pytoolz/toolz>
- <https://github.com/HypothesisWorks/hypothesis>
- <https://github.com/more-itertools/more-itertools>

Teach Yourself Programming in Ten Years

<https://norvig.com/21-days.html>

Learn at least a **half dozen programming languages**. Include one language that emphasizes class abstractions (like Java or C++), one that emphasizes **functional abstraction** (like Lisp or ML or Haskell), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), and one that emphasizes parallelism (like Clojure or Go).

Resources:



github.com/Kislovskiy/talks

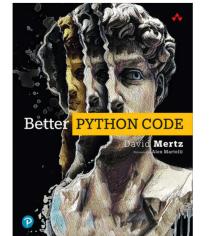


<https://github.com/roof42/python-monad-pymonad>

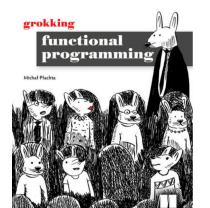
superruzafa.github.io/visual-scala-reference/filter/



learning.oreilly.com/library/view/functional-programming-in/9781492048633/



Better Python Code: A Guide for Aspiring Experts



Grokking Functional Programming by Michal Plachta

“So go ahead and buy that programming book; you'll probably get some use out of it. But you won't change your life, or your real overall expertise as a programmer in 24 hours or 21 days. How about working hard to continually improve over 24 months? Well, now you're starting to get somewhere...”

P.Norvig



Kislovskiy/talks