Multi-paradigm programming in Python

Elias Mistler | Machine Learning Engineer

[Previse (https://previ.se/)](https://previ.se/)

Quick Intro

- Elias Mistler
- Previse
    - Invoice financing
    - based on ML
    - corporate data
    - improve SME cashflow
- Machine Learning Engineer
    - ML integration into invoice processing platform
    - Buyer data intake and mapping
    - Operational tooling

Contents

- Introduction
- Code Structure
- Data Structures
- State Handling
- Multiple implementations
- Summary

Introduction

- Python = multi-paradigm (unlike OO Java / FP Clojure)
- OOP and FP are **concepts**, not tied to syntax (`class` or `def`)

Object-oriented principles

- mutable data structures
- (relies on rich type system)
- class hierarchies
    - inheritance
    - abstraction
    - encapsulation
    - polymorphism

Functional programming Principles

- immutable data structures
- (relies on simple data types)
- pure functions
  - no side-effects
  - idempotent
- (often lazy evaluation)

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- 9 x 9 grid
- numbers from 1 - 9
- each row/column/block should contain each digit

Code structure: high- vs. low-context

Example: parse raw Sudoku string (from OpenSudoku (https://opensudoku.moire.org/)) to array

In [2]:

```
raw_example = '7001500000030020978004701265003902000300100500080270019750310041207
009000000065002'
```

Factory function (OO)

In [3]:

```
@dataclass
class Sudoku:
    grid: np.array

    @classmethod
    def from_string(cls, raw):
        values = []
        for digit in raw:
            values.append(int(digit))
        grid = np.array(values, dtype='int64').reshape((9, 9))
        return cls(grid)
```

In [4]:

```
Sudoku.from_string(raw_example)
```

Out[4]:

```
Sudoku(grid=array([[7, 0, 0, 1, 5, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 2, 0, 9, 7],
       [8, 0, 0, 4, 7, 0, 1, 2, 6],
       [5, 0, 0, 3, 9, 0, 2, 0, 0],
       [0, 3, 0, 0, 1, 0, 0, 5, 0],
       [0, 0, 8, 0, 2, 7, 0, 0, 1],
       [9, 7, 5, 0, 3, 1, 0, 0, 4],
       [1, 2, 0, 7, 0, 0, 9, 0, 0],
       [0, 0, 0, 0, 6, 5, 0, 0, 2]]))
```

- explicit, high-context
- easy to find and use

Isolated function (FP)

```
In [5]: def parse_raw(raw):
            return np.array(list(map(int, raw)), dtype='int64').reshape((9, 9))
```

```
In [6]: parse_raw(raw_example)
```

```
Out[6]: array([[7, 0, 0, 1, 5, 0, 0, 0, 0],
               [0, 0, 3, 0, 0, 2, 0, 9, 7],
               [8, 0, 0, 4, 7, 0, 1, 2, 6],
               [5, 0, 0, 3, 9, 0, 2, 0, 0],
               [0, 3, 0, 0, 1, 0, 0, 5, 0],
               [0, 0, 8, 0, 2, 7, 0, 0, 1],
               [9, 7, 5, 0, 3, 1, 0, 0, 4],
               [1, 2, 0, 7, 0, 0, 9, 0, 0],
               [0, 0, 0, 0, 6, 5, 0, 0, 2]])
```

- free of assumptions about the use case
- easy to reuse or generalise

Multi-paradigm solution

Generalised, low-context pure function, use in high-context class

In [7]:

```
def parse_raw(raw):
    size = int(math.sqrt(len(raw)))
    return np.array(list(map(int, raw)), dtype='int64').reshape((size, size))


class Sudoku:
    @classmethod
    def from_string(cls, raw):
        values = parse_raw(raw)
        return cls(values)
```

- low-context pure functions *and* high-context class
- tidy, reusable code
- generalises well
- works in any context
- easy to use and explore

That tedious `for`-loop

In [8]:
```python
values = []
for digit in raw_example:
    values.append(int(digit))

values[:5]
```

Out[8]:
```
[7, 0, 0, 1, 5]
```

- easy to write
- tedious to read and reconstruct
- comparatively far from high-level intention
- error prone

"I WOULD HAVE WRITTEN A SHORTER LETTER, BUT I DID NOT HAVE THE TIME."

Blaise Pascal

The alternative:

```
In [9]: values = tuple(map(int, raw_example))

        values[:5]

Out[9]: (7, 0, 0, 1, 5)
```

```
In [10]: values = thread_last(raw_example, (map, int), tuple)
```

- concise
- reflects the intention
- easy to read
- can take longer to write

Also useful: *list comprehension*

```
In [11]: values = [int(digit) for digit in raw_example]
```

- Pythonic middle ground
- easy to both read *and* write
- do not use *lambda functions* in list comprehensions!
- combine with pure functions for best results

Further example - display/format

Object-oriented

Implement `__repr__`

In [12]:

```
from sudoku.oo.base import Sudoku

Sudoku.from_string(raw_example)
```

Out[12]:

```
+---+---+---+---+---+---+---+---+---+
| 7 |   |   | 1 | 5 |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   | 3 |   |   | 2 |   | 9 | 7 |
+---+---+---+---+---+---+---+---+---+
| 8 |   |   | 4 | 7 |   | 1 | 2 | 6 |
+---+---+---+---+---+---+---+---+---+
| 5 |   |   | 3 | 9 |   | 2 |   |   |
+---+---+---+---+---+---+---+---+---+
|   | 3 |   |   | 1 |   |   | 5 |   |
+---+---+---+---+---+---+---+---+---+
|   |   | 8 |   | 2 | 7 |   |   | 1 |
+---+---+---+---+---+---+---+---+---+
| 9 | 7 | 5 |   | 3 | 1 |   |   | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 2 |   | 7 |   |   | 9 |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 6 | 5 |   |   | 2 |
+---+---+---+---+---+---+---+---+---+
```

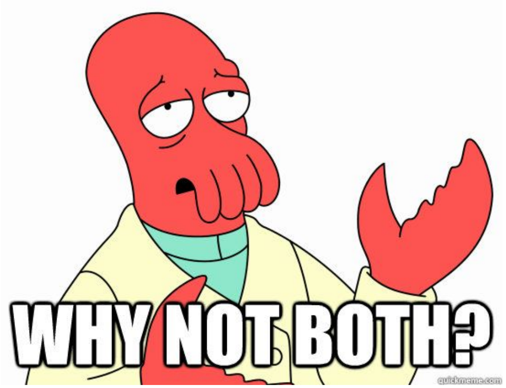
Functional

explicit functions

```
In [13]: from sudoku.fp.load import format_sudoku

         thread_last(raw_example, parse_raw, format_sudoku, print)
```

```
+---+---+---+---+---+---+---+---+---+
| 7 |   |   | 1 | 5 |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   | 3 |   |   | 2 |   | 9 | 7 |
+---+---+---+---+---+---+---+---+---+
| 8 |   |   | 4 | 7 |   | 1 | 2 | 6 |
+---+---+---+---+---+---+---+---+---+
| 5 |   |   | 3 | 9 |   | 2 |   |   |
+---+---+---+---+---+---+---+---+---+
|   | 3 |   |   | 1 |   |   | 5 |   |
+---+---+---+---+---+---+---+---+---+
|   |   | 8 |   | 2 | 7 |   |   | 1 |
+---+---+---+---+---+---+---+---+---+
| 9 | 7 | 5 |   | 3 | 1 |   |   | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 2 |   | 7 |   |   | 9 |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 6 | 5 |   |   | 2 |
+---+---+---+---+---+---+---+---+---+
```

Multi-paradigm



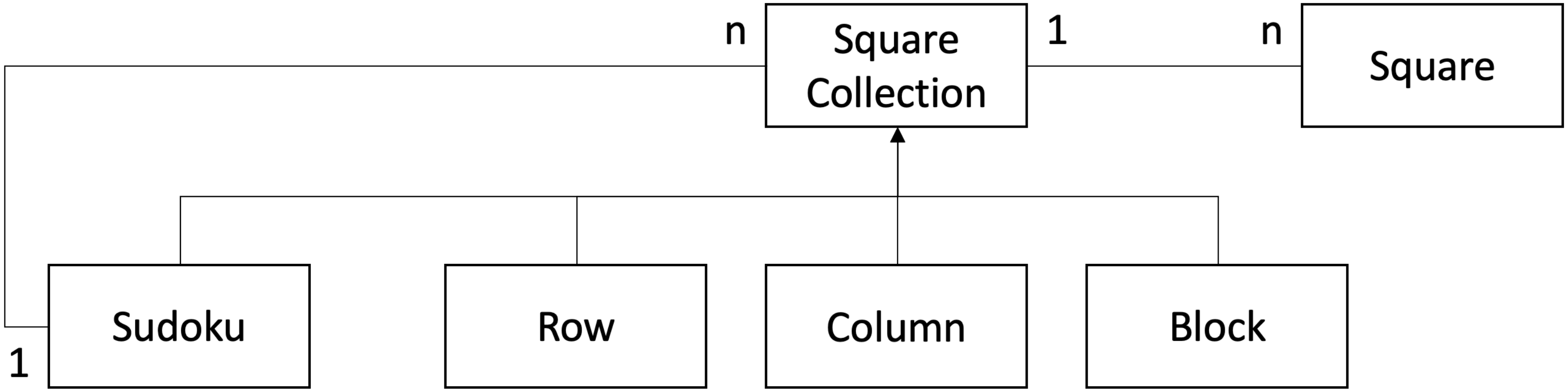
In [14]:

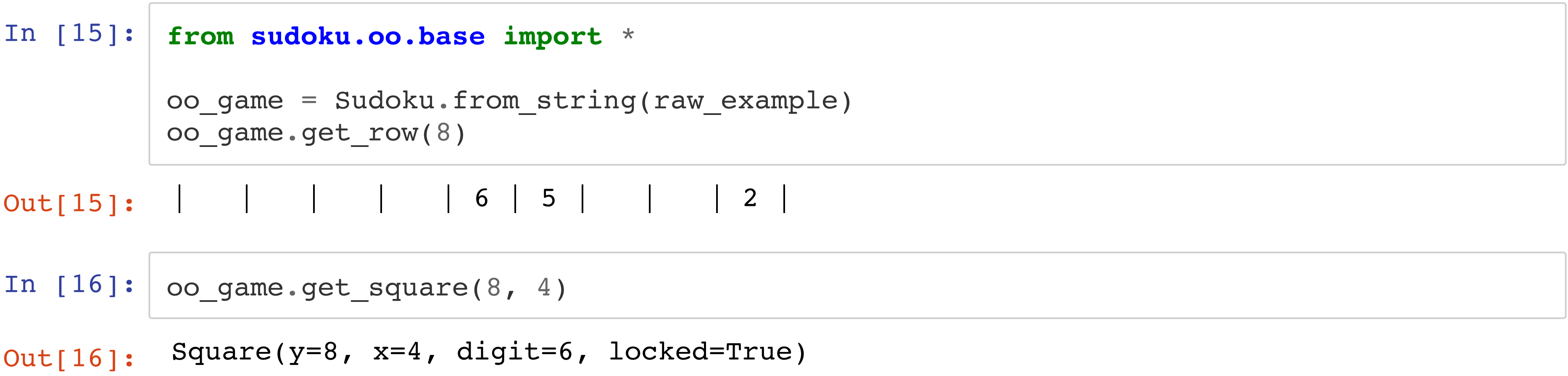
```python
def format_sudoku(grid):
    ...


class Sudoku:
    ...

    def __repr__(self):
        return format_sudoku(self.grid)
```

Data Structures: explicit vs. minimalist

Example: The Sudoku grid

Class Hierarchy (OO)





```
In [15]: from sudoku.oo.base import *

         oo_game = Sudoku.from_string(raw_example)
         oo_game.get_row(8)

Out[15]: |   |   |   |   | 6 | 5 |   |   | 2 |

In [16]: oo_game.get_square(8, 4)

Out[16]: Square(y=8, x=4, digit=6, locked=True)
```

- assumes certain usage patterns
- intuitive to explore
- fairly rigid
- requires lots of boilerplate

```
In [17]: # even with `dataclass` and without many getters, setters etc:
         !wc ./sudoku/oo/base.py

      121     335    3148 ./sudoku/oo/base.py
```

Simplicity (FP)

```
In [18]: import schema

         sudoku_schema = schema.And(np.ndarray,
                                    lambda a: a.shape == (9, 9),
                                    lambda a: a.dtype == 'int64')
```

```
In [19]: thread_last(raw_example, parse_raw, sudoku_schema.validate)
```

```
Out[19]: array([[7, 0, 0, 1, 5, 0, 0, 0, 0],
                [0, 0, 3, 0, 0, 2, 0, 9, 7],
                [8, 0, 0, 4, 7, 0, 1, 2, 6],
                [5, 0, 0, 3, 9, 0, 2, 0, 0],
                [0, 3, 0, 0, 1, 0, 0, 5, 0],
                [0, 0, 8, 0, 2, 7, 0, 0, 1],
                [9, 7, 5, 0, 3, 1, 0, 0, 4],
                [1, 2, 0, 7, 0, 0, 9, 0, 0],
                [0, 0, 0, 0, 6, 5, 0, 0, 2]])
```

- minimalist approach with basic data types
- zero boilerplate
- no context on the data structure itself
- harder to explore
- easier to reuse

Multi-paradigm solution

In [20]:

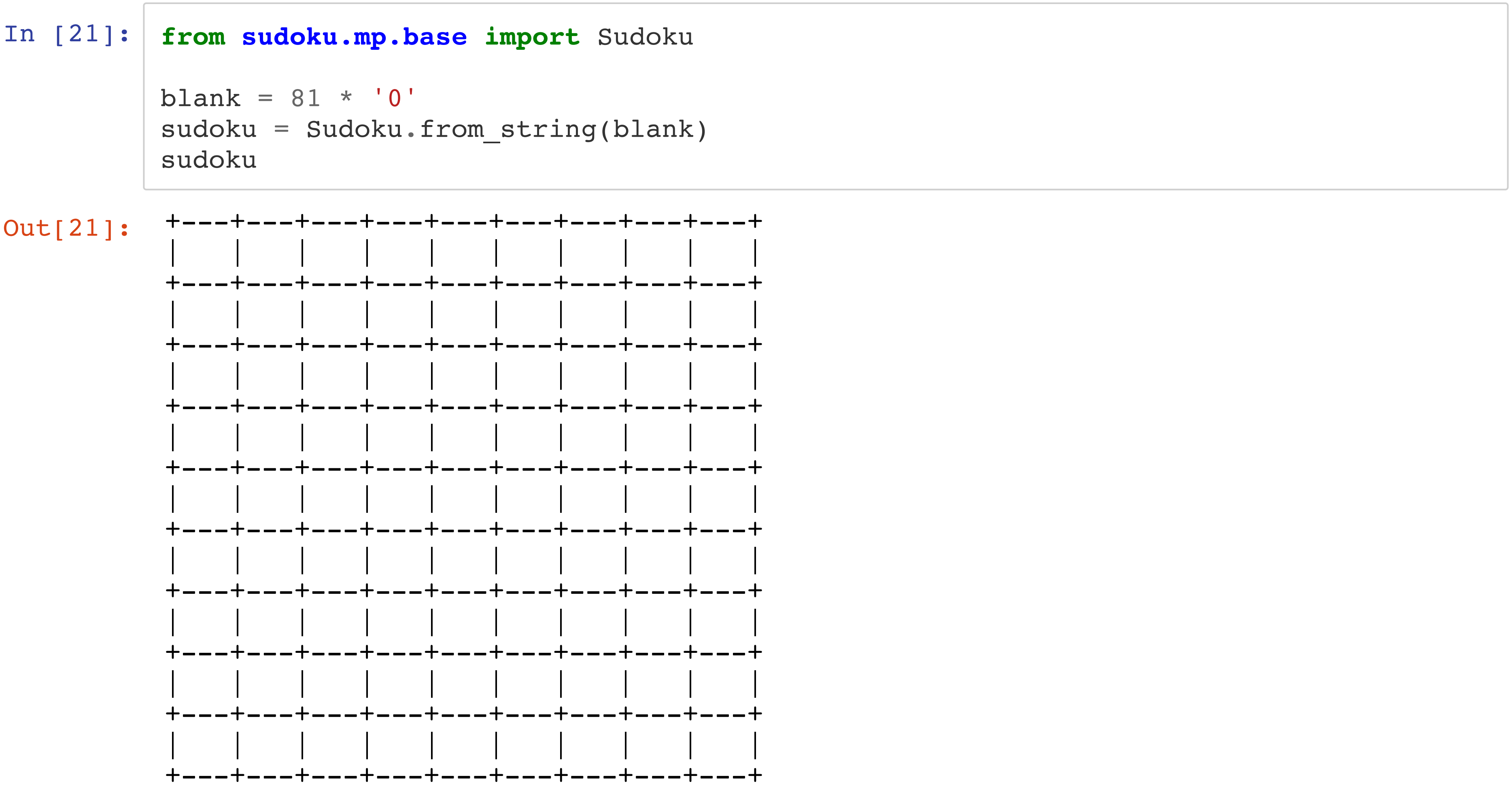
```
@dataclass
class Sudoku:
    grid: np.ndarray

    @property
    def remaining_blanks(self):
        return (self.grid == 0).sum()

    def __repr__(self):
        ...
```

- "shallow" class
- saves a lot of boilerplate code
- adds context for user

State handling - mutable vs. immutable

Example: Fill digits into Sudoku

Using a multi-paradigm implementation, inspired by `pandas` :



```
In [21]: from sudoku.mp.base import Sudoku

         blank = 81 * '0'
         sudoku = Sudoku.from_string(blank)
         sudoku
```

```
Out[21]: +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
```

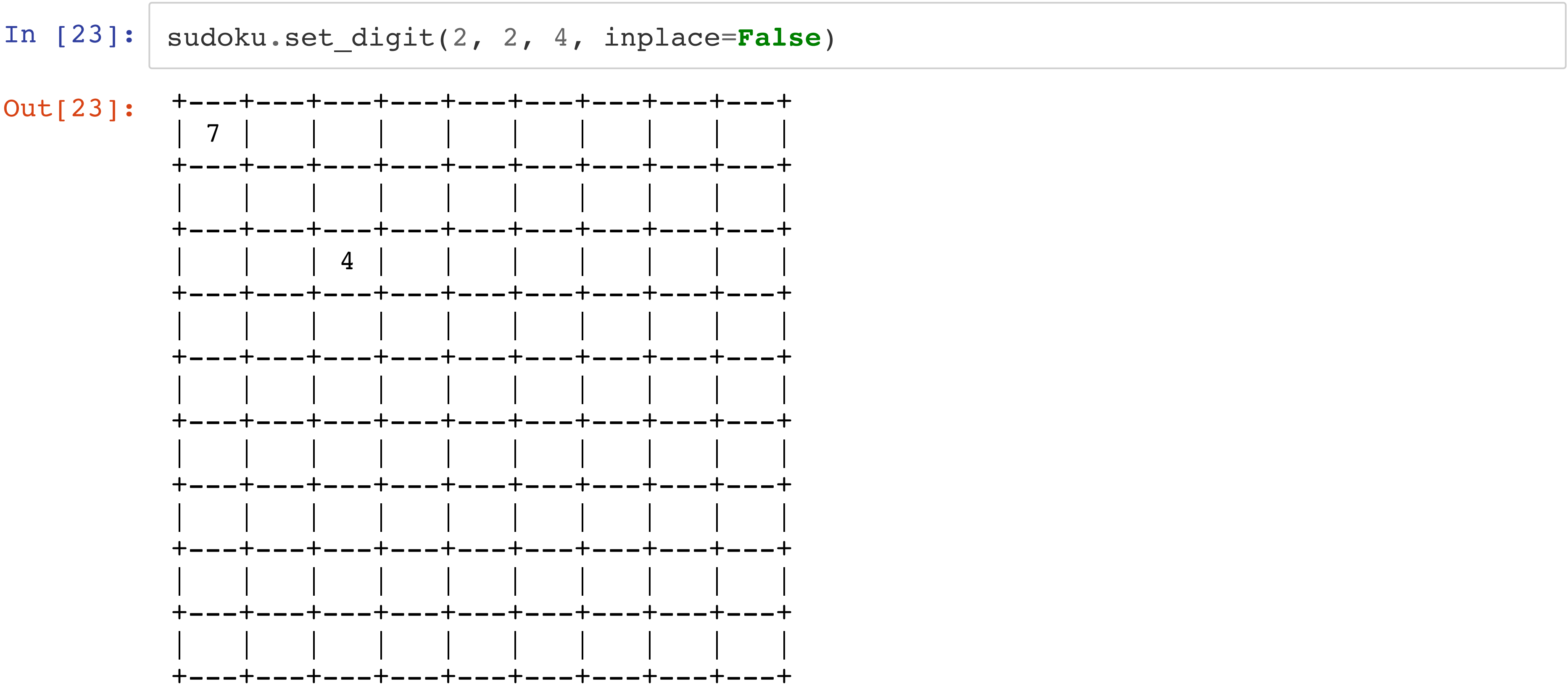
Mutable (OO)

```
In [22]: sudoku.set_digit(0, 0, 7, inplace=True)
         sudoku
```

```
Out[22]: +---+---+---+---+---+---+---+---+---+
         | 7 |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
```

- changed in-place
- seems "natural"
- no way back / history

Immutable (FP)

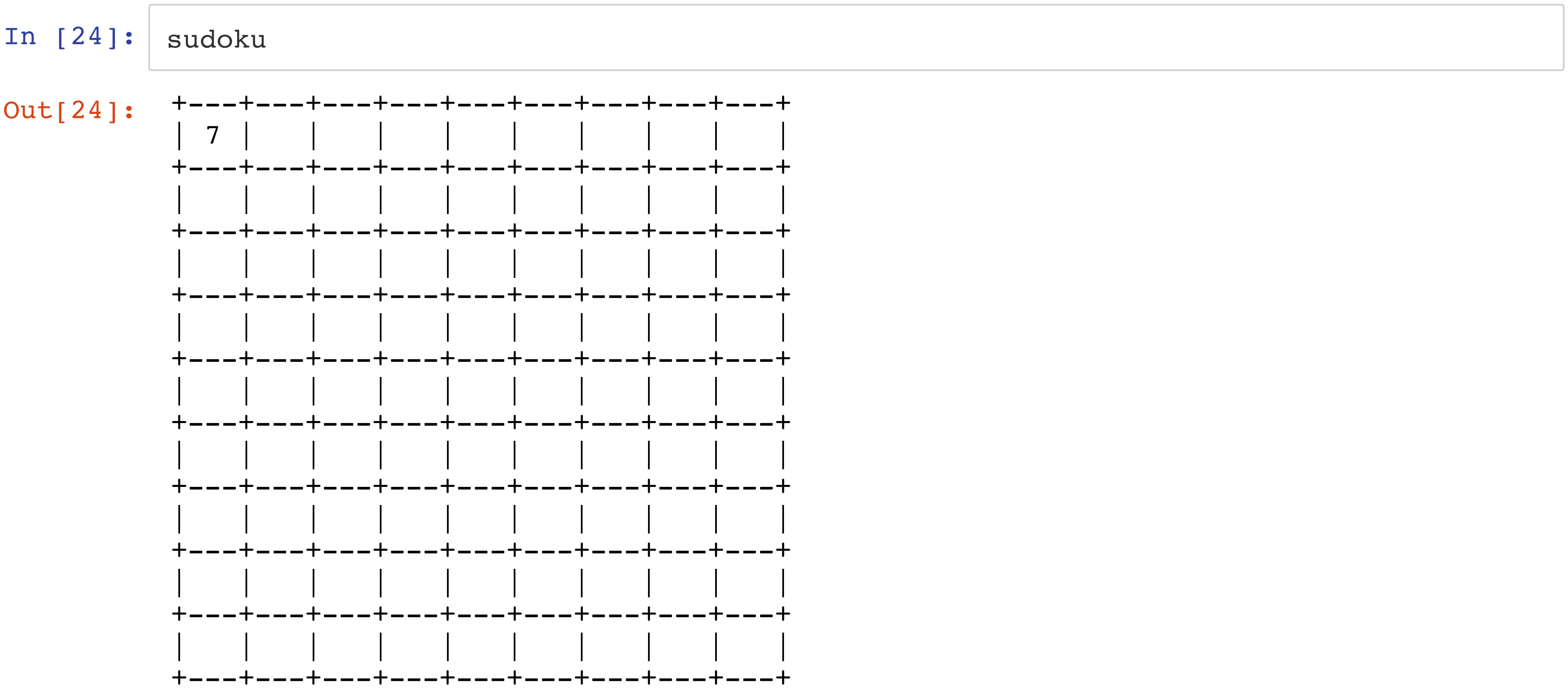


In [23]:

```
sudoku.set_digit(2, 2, 4, inplace=False)
```

Out[23]:

```
+---+---+---+---+---+---+---+---+---+
| 7 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   | 4 |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
```



```
In [24]: sudoku

Out[24]: +---+---+---+---+---+---+---+---+---+
         | 7 |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
         |   |   |   |   |   |   |   |   |   |
         +---+---+---+---+---+---+---+---+---+
```

- easy to reuse or parallelise (efficienct, avoids concurrecny errors)
- natural versioning
- lends itself well to pipelines or method chaining

Method Chaining

In [25]:

```
(sudoku
 .set_digit(2, 8, 9)
 .set_digit(1, 0, 9)
 .set_digit(0, 3, 9))
```

Out[25]:

```
+---+---+---+---+---+---+---+---+---+
| 7 |   |   | 9 |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
| 9 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   | 9 |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+
```

Recommendation

- make use of immutable data structures like `@dataclass(frozen=True)`, `NamedTuple`, `frozendict` and `pyrsistent.pmap`)
- use mutable data structures in immutable ways (try the `toolz` library!)
- keep functions pure and idempotent - use classes where configuration and state is required

Example in Pandas

In [26]:

```
import pandas as pd

df = pd.DataFrame(np.random.random((5,3)), columns=list('abc'))
df
```

Out[26]:

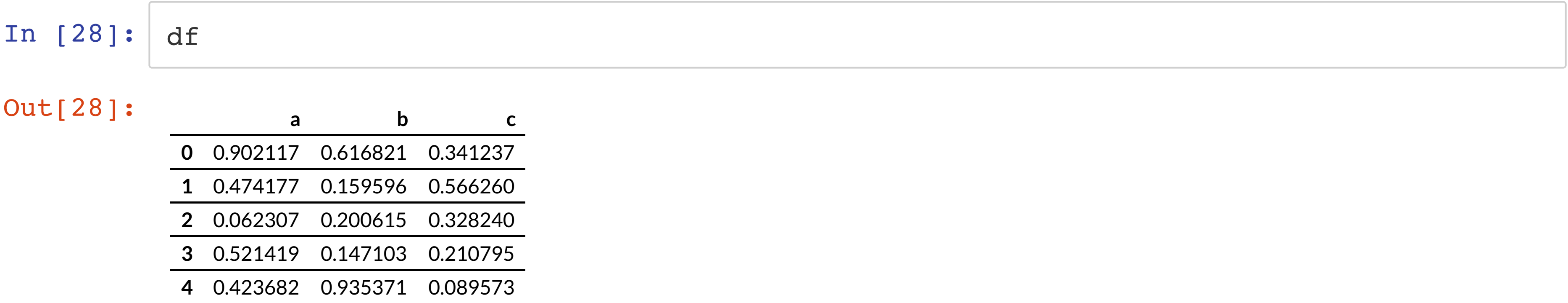
	a	b	c
0	0.902117	0.616821	0.341237
1	0.474177	0.159596	0.566260
2	0.062307	0.200615	0.328240
3	0.521419	0.147103	0.210795
4	0.423682	0.935371	0.089573

In [27]:

```
(df
 .assign(sum=lambda df: df.sum(axis=1))
 .assign(a_percent=lambda df: df['a'] / df['sum'])
 .drop(index=[1,3]))
```

Out[27]:

	a	b	c	sum	a_percent
0	0.902117	0.616821	0.341237	1.860174	0.484964
2	0.062307	0.200615	0.328240	0.591162	0.105397
4	0.423682	0.935371	0.089573	1.448627	0.292472



```
In [28]: df
```

Out[28]:

	a	b	c
0	0.902117	0.616821	0.341237
1	0.474177	0.159596	0.566260
2	0.062307	0.200615	0.328240
3	0.521419	0.147103	0.210795
4	0.423682	0.935371	0.089573

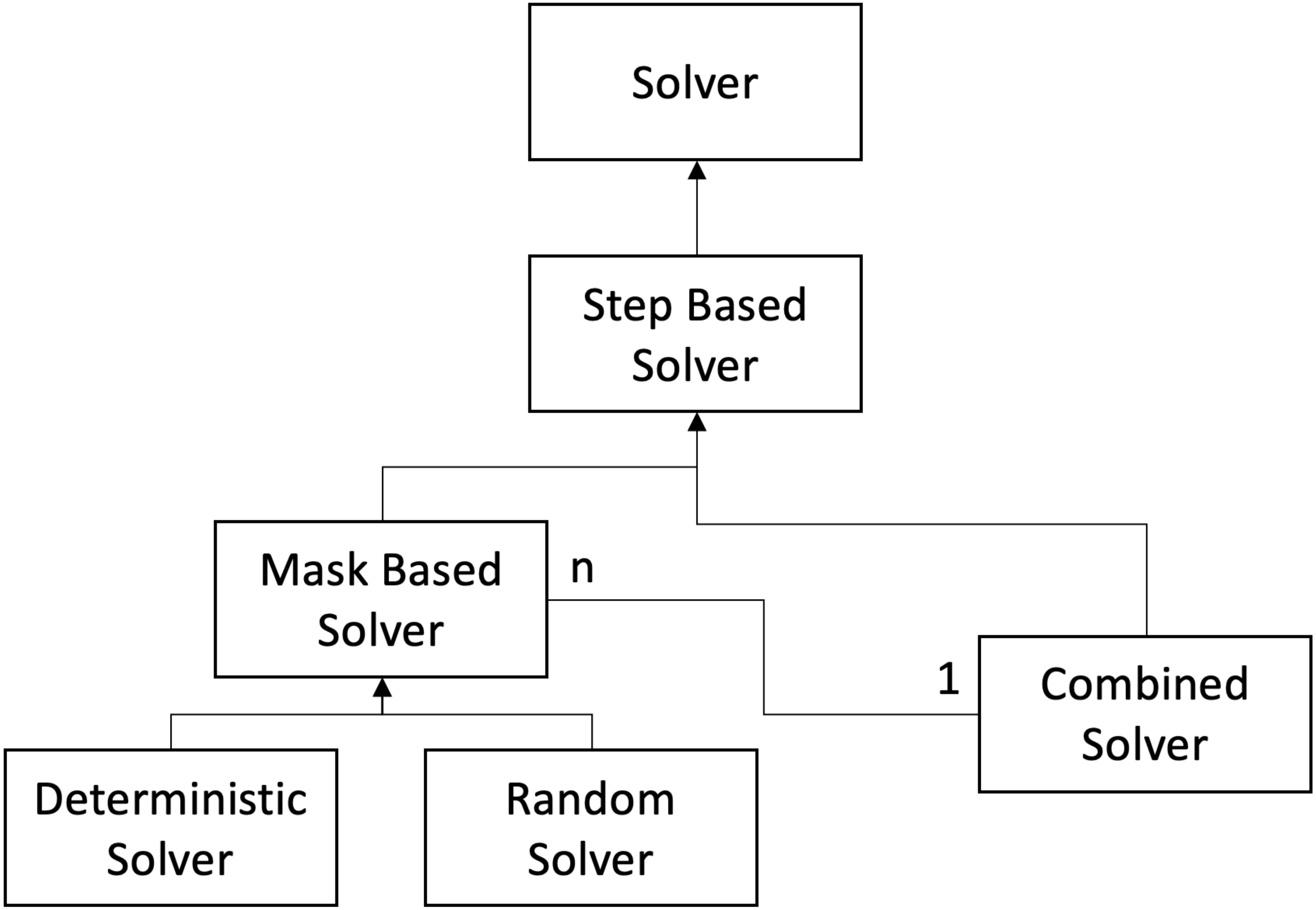
- cleaner Jupyter notebooks (execution order...)
- better reusability
- close to production-ready

Multiple implementations: polymorphism vs. function composition

Example: Different Sudoku solvers

- Deterministic (mask, fill unambiguous, repeat) - insufficient
- Random (mask, fill random, repeat) - prohibitively slow
- Combined (deterministic as much as possible, random step, repeat)

OO - Solver class hierarchy



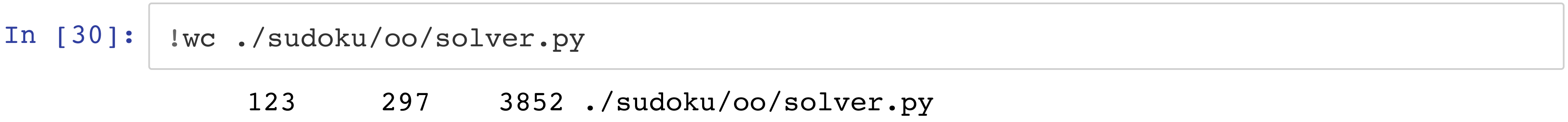
In [29]:

```
from sudoku.oo.solver import *

sudoku = Sudoku.from_string(raw_example)
solver = DeterministicSolver(sudoku)
solver.solve()

sudoku
```

Out[29]:

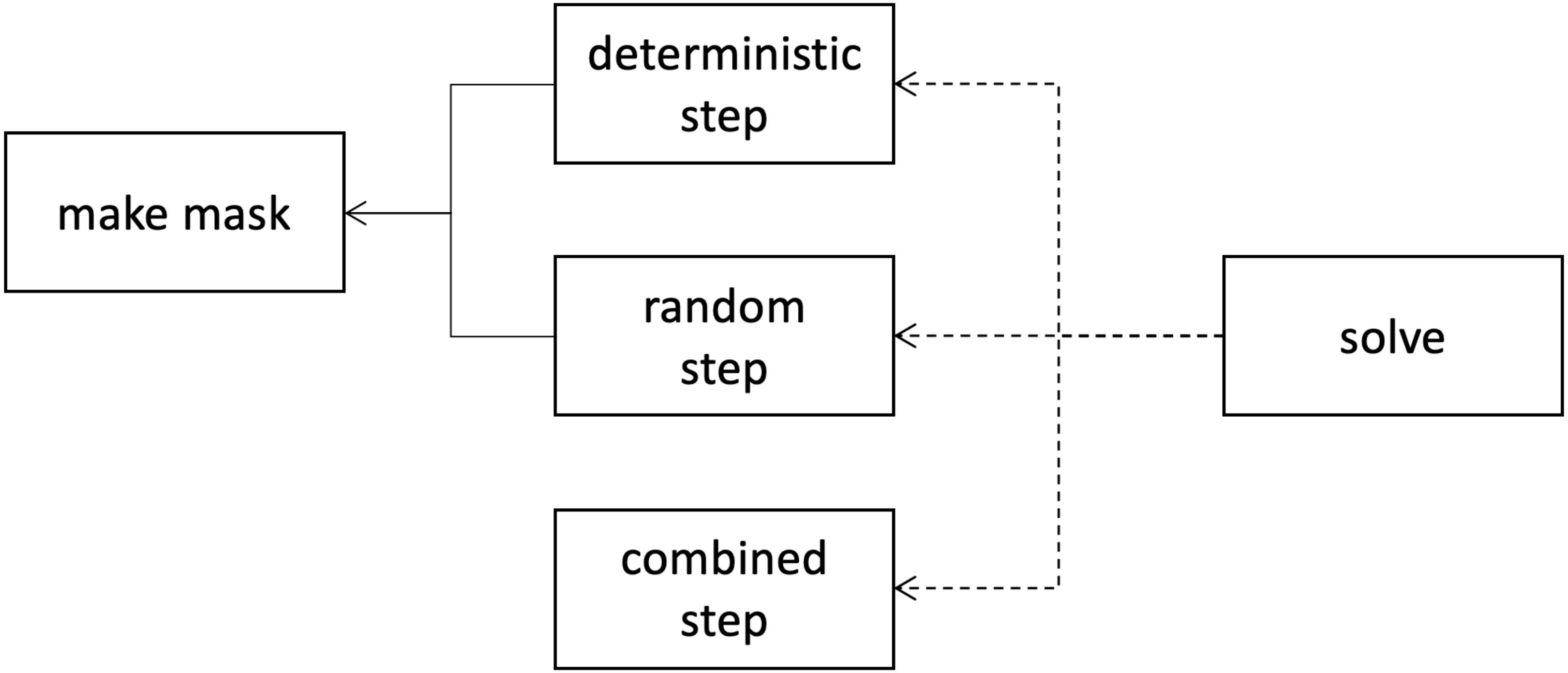
```
+---+---+---+---+---+---+---+---+---+
| 7 | 6 | 2 | 1 | 5 | 9 | 4 | 8 | 3 |
+---+---+---+---+---+---+---+---+---+
| 4 | 1 | 3 | 6 | 8 | 2 | 5 | 9 | 7 |
+---+---+---+---+---+---+---+---+---+
| 8 | 5 | 9 | 4 | 7 | 3 | 1 | 2 | 6 |
+---+---+---+---+---+---+---+---+---+
| 5 | 4 | 1 | 3 | 9 | 6 | 2 | 7 | 8 |
+---+---+---+---+---+---+---+---+---+
| 2 | 3 | 7 | 8 | 1 | 4 | 6 | 5 | 9 |
+---+---+---+---+---+---+---+---+---+
| 6 | 9 | 8 | 5 | 2 | 7 | 3 | 4 | 1 |
+---+---+---+---+---+---+---+---+---+
| 9 | 7 | 5 | 2 | 3 | 1 | 8 | 6 | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 2 | 6 | 7 | 4 | 8 | 9 | 3 | 5 |
+---+---+---+---+---+---+---+---+---+
| 3 | 8 | 4 | 9 | 6 | 5 | 7 | 1 | 2 |
+---+---+---+---+---+---+---+---+---+
```

- Mutable data access (as before)
- Single-method classes excessive (boilerplate!)
- Complicated design for simple functionality



```
In [30]: !wc ./sudoku/oo/solver.py

      123     297    3852 ./sudoku/oo/solver.py
```

FP - solving function composition



In [31]:

```
from sudoku.fp.solve import *
from sudoku.fp.load import *

solve_combined = partial(solve, step_function=combined_step)

thread_last(raw_example, parse_raw, solve_combined, format_sudoku, print)
```

```
+---+---+---+---+---+---+---+---+---+
| 7 | 6 | 2 | 1 | 5 | 9 | 4 | 8 | 3 |
+---+---+---+---+---+---+---+---+---+
| 4 | 1 | 3 | 6 | 8 | 2 | 5 | 9 | 7 |
+---+---+---+---+---+---+---+---+---+
| 8 | 5 | 9 | 4 | 7 | 3 | 1 | 2 | 6 |
+---+---+---+---+---+---+---+---+---+
| 5 | 4 | 1 | 3 | 9 | 6 | 2 | 7 | 8 |
+---+---+---+---+---+---+---+---+---+
| 2 | 3 | 7 | 8 | 1 | 4 | 6 | 5 | 9 |
+---+---+---+---+---+---+---+---+---+
| 6 | 9 | 8 | 5 | 2 | 7 | 3 | 4 | 1 |
+---+---+---+---+---+---+---+---+---+
| 9 | 7 | 5 | 2 | 3 | 1 | 8 | 6 | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 2 | 6 | 7 | 4 | 8 | 9 | 3 | 5 |
+---+---+---+---+---+---+---+---+---+
| 3 | 8 | 4 | 9 | 6 | 5 | 7 | 1 | 2 |
+---+---+---+---+---+---+---+---+---+
```

- very clear responsibilities per function
- simple, pragmatic design
- easy to introspect
- much more concise (*and* no base module!)

Multi-paradigm solution

In [32]:

```
from sudoku.fp import solve as _fp_solve
from sudoku.mp.base import Sudoku


def solve_sudoku(sudoku: Sudoku, step_function: Callable, max_tries: int = 1):
    if max_tries == 1:
        solved_grid = _fp_solve.solve(sudoku.grid, step_function)
    else:
        solved_grid = _fp_solve.repeat_solve(sudoku.grid,
                                             partial(_fp_solve.solve, step_functio
n=step_function),
                                             max_tries=max_tries)
    return Sudoku(solved_grid)
```

- simplicity and clarity of FP
- takes and returns high-context Sudoku objects

Or, with more context:

In [33]:

```
from sudoku.mp.solve import solve


@dataclass(frozen=True)
class Solver:
    step_function: Callable
    max_tries: int = 1

    def __call__(self, sudoku: Sudoku):
        return solve(sudoku, self.step_function, self.max_tries)
```

In [34]:

```
thread_last(raw_example,
            Sudoku.from_string,
            Solver(combined_step, max_tries=100))
```

Out[34]:

```
+---+---+---+---+---+---+---+---+---+
| 7 | 6 | 2 | 1 | 5 | 9 | 4 | 8 | 3 |
+---+---+---+---+---+---+---+---+---+
| 4 | 1 | 3 | 6 | 8 | 2 | 5 | 9 | 7 |
+---+---+---+---+---+---+---+---+---+
| 8 | 5 | 9 | 4 | 7 | 3 | 1 | 2 | 6 |
+---+---+---+---+---+---+---+---+---+
| 5 | 4 | 1 | 3 | 9 | 6 | 2 | 7 | 8 |
+---+---+---+---+---+---+---+---+---+
| 2 | 3 | 7 | 8 | 1 | 4 | 6 | 5 | 9 |
+---+---+---+---+---+---+---+---+---+
| 6 | 9 | 8 | 5 | 2 | 7 | 3 | 4 | 1 |
+---+---+---+---+---+---+---+---+---+
| 9 | 7 | 5 | 2 | 3 | 1 | 8 | 6 | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 2 | 6 | 7 | 4 | 8 | 9 | 3 | 5 |
+---+---+---+---+---+---+---+---+---+
| 3 | 8 | 4 | 9 | 6 | 5 | 7 | 1 | 2 |
+---+---+---+---+---+---+---+---+---+
```

Key Takeaways

Object-orientation

- "top-down" design
- larger, topical structures
- explicit, high-context
- functionality and data intertwined

leads to:

- intuitive use cases
- high explorability

Functional programming

- "bottom-up" design
- simplistic thinking
- small chunks of reusable logic, separate from data
- high isolation, low context

leads to

- high reusability
- tidy, concise code
- flexible use cases

Multi-paradigm programming

pick & mix of both worlds:

- pure functions in mutable context
    - brings the simplicity and elegance of FP into OO
    - make your code explorable and easy to understand
    - *remember*: no side effects, no problem!
- mutable data in immutable context
    - use your favourite OO libaries in concise FP code
    - *remember*: copy-and-modify mutable data structures!

leads to (ideally) - best of both worlds:

- intuitive **and** flexible use cases
- high explorability **and** reusability

My preferred Approach

- iterate with a REPL
- use immutable data types and pure functions where possible
- create classes where either:
    - required due to syntax or library
    - high-context use cases are required

Thank you for your attention!

References & Further Reading

Full notebook and code available at https://github.com/eliasmistler/europython2020-multi-paradigm-sudoku (https://github.com/eliasmistler/europython2020-multi-paradigm-sudoku)