Functional Python

Saving Christmas 🎄 with `itertools` & friends

Hi, I’m Edoardo

⚛️ Physicist

👨🏻‍💻 Research Software Engineer at Empa

🐺 Wolfram *aficionado*, 🐍 Python enthusiast, 🦀 Rust learner

🏕️ Long-distance hiker

🧩 Puzzle addicted

Advent of Code 🎄

Have you tried it once? Twice? Every year?

About this talk

Data transformations

Example puzzles from Advent of Code

Picks from `itertools`, `functools`, `operator`

Who did this?

```
def solve(data):
    result = []
    for i in range(len(data)):
        for j in range(len(data[i])):
            if check_condition(data[i][j]):
                for k in range(len(neighbors)):
                    # The doom of nesting goes on...
```

But...

```
def solve(data):
    return sum(
        value
        for row in data
        for value in row
        if check_condition(value)  # or check_neighbors
    )
```

What is Functional Programming?

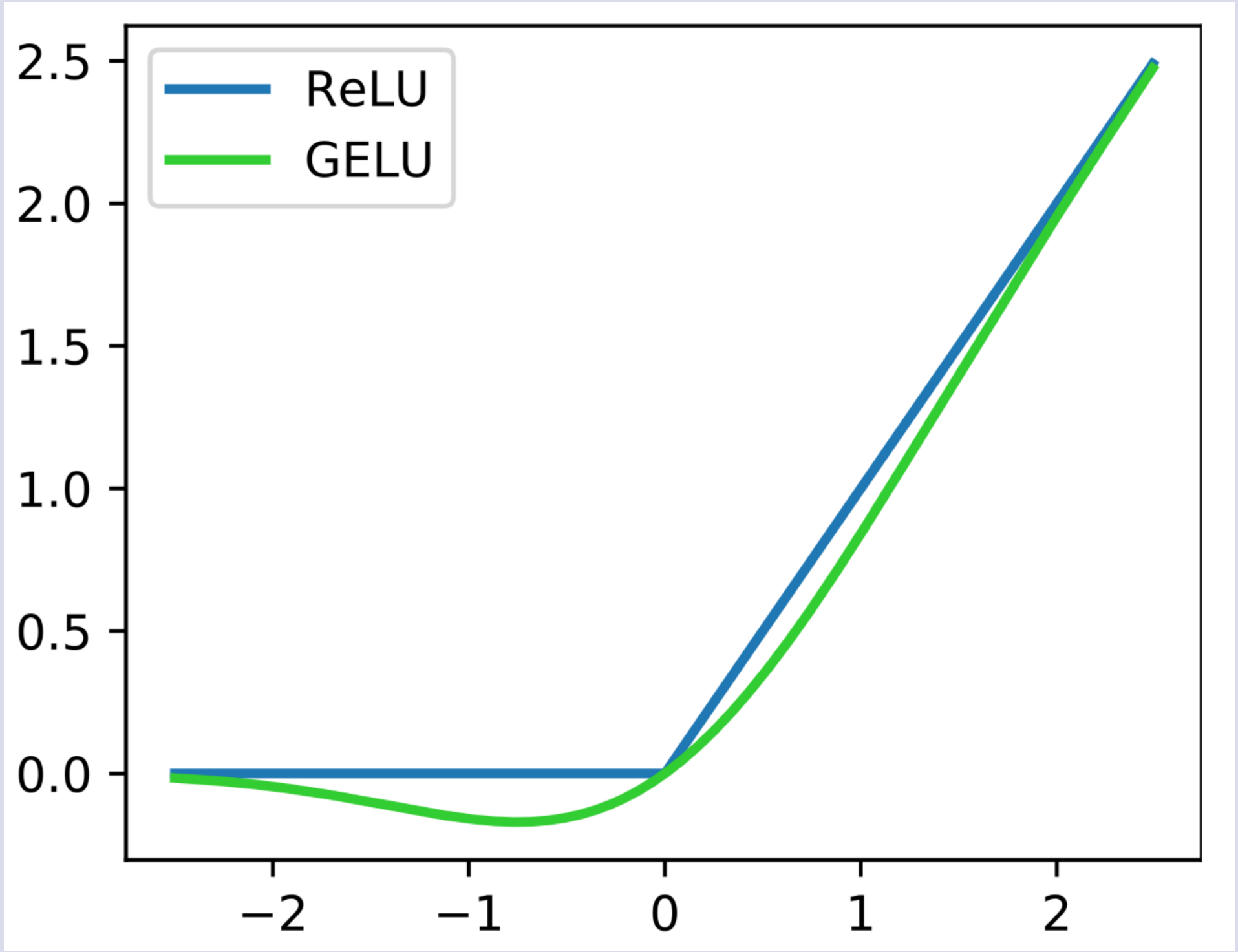
Tenets

Pure functions

Immutability

Higher-order functions

Pure functions & Immutability



Pure functions & Immutability

```python
# ❌ Side effect: data mutation
def ReLU(array: list[int]) -> list[int]:
    for i, value in enumerate(array):
        array[i] = max(value, 0)
    return array
```

Pure functions & Immutability

```
data = [1, -2, 3, -4]
result = ReLU(data)
print(result)  # [1, 0, 3, 0] Expected ✅
print(data)    # [1, 0, 3, 0] Our data are gone 😱
```

Pure functions & Immutability

```python
# ❌ Side effect: data mutation
def ReLU(array: list[int]) -> list[int]:
    for i, value in enumerate(array):
        array[i] = max(value, 0)
    return array

# ✅ Pure: creates new data, leaves input alone
def ReLU(array: tuple[int, ...]) -> tuple[int, ...]:
    return tuple(max(value, 0) for value in array)
```

Higher-order functions

```
def decorator(func):
    def wrapper(*args, **kwargs):
        func(*args, **kwargs)
    return wrapper
```

Higher-order functions

```
from operator import attrgetter

f = attrgetter("name")
f(b)  # returns b.name

g = attrgetter("name.first", "name.last")
g(b)  # returns (b.name.first, b.name.last)
```

Iterators & Generators

Iterator: Stream of data, one element at a time

Generator: Function that creates an iterator with `yield`

Lazy vs Eager evaluation

```
# Lazy
squares = (n**2 for n in range(1_000_000))
>>> squares
<generator object <genexpr> at 0x1076305f0>

# Eager
squares = [n**2 for n in range(1_000_000)]
>>> squares
[0, 1, 4, 9, 16, 25, ...]
```

Functional built-ins

```
map(function, iterable)
map(function, iterable1, iterable2, ...)  # Function must take 2 arguments

filter(function, iterable)

zip(iterable1, iterable2, ...)

enumerate(iterable, start=1)

all(iterable)
any(iterable)
```

`map` and `zip`

```
# map: transform each element
numbers = [1, 2, 3, 4]
list(map(lambda x: x ** 2, numbers))
# → [1, 4, 9, 16]

# zip: combine multiple sequences element-by-element
names = ["Athos", "Porthos", "Aramis"]
ages = [40, 30, 25]
list(zip(names, ages))
# → [("Athos", 40), ("Porthos", 30), ("Aramis", 25)]
```

🧩 Pattern 1

Sequences & neighbors

🤔 Problem

Relationships between adjacent elements

Day 2, 2024

Red-Nosed Reports

Differences between adjacent numbers are between 1–3

Differences have all the same sign

7 6 4 2 1 is safe

1 2 7 8 9 is unsafe

9 7 6 2 1 still unsafe

Day 2, 2024

Imperative

```python
def is_safe_imperative(report):
    # Build a "container"
    differences = []
    for i in range(len(report) - 1):
        diff = report[i+1] - report[i]
        differences.append(diff)

    # Check if all increasing or all decreasing
    all_increasing = all(d > 0 for d in differences)
    all_decreasing = all(d < 0 for d in differences)

    # Check magnitude
    valid_magnitude = all(1 <= abs(d) <= 3 for d in differences)

    return (all_increasing or all_decreasing) and valid_magnitude
```

The magic of `pairwise()`

```
from itertools import pairwise

list(pairwise([1, 2, 3, 4, 5]))
# → [(1, 2), (2, 3), (3, 4), (4, 5)]
```

Day 2, 2024

Functional

```
from itertools import pairwise

def is_safe_functional(report):
    differences = [b - a for a, b in pairwise(report)]

    all_same_sign = all(d > 0 for d in differences) or
                    all(d < 0 for d in differences)

    valid_magnitude = all(1 <= abs(d) <= 3 for d in differences)

    return all_same_sign and valid_magnitude
```

🧩 Pattern 2

Products & Combinations

🤔 Problem

Try all possible combinations

Day 7, 2024

Bridge Repair

```
190: 10 19          →   10 * 19 = 190 ✓
3267: 81 40 27      →   81 + 40 * 27 = 3267 ✓
292: 11 6 16 20     →   11 + 6 * 16 + 20 = 292 ✓
```

Plug the `+` and `*` operators into an integer sequence and obtain a target

Evaluation happens left-to-right

Day 7, 2024

Imperative

```
def can_make(target, nums):
    n = len(nums) - 1
    for combo in range(2**n):
        result = nums[0]
        for i in range(n):
            if combo & (1 << i):
                result *= nums[i+1]
            else:
                result += nums[i+1]
        if result == target:
            return True
    return False
```

Understanding `product`

```python
from itertools import product

# Cartesian product of two sequences
list(product([1, 2], ['a', 'b']))
# → [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]

# Product with itself (repeat)
list(product([0, 1], repeat=3))
# → [(0,0,0), (0,0,1), (0,1,0), (0,1,1),
#    (1,0,0), (1,0,1), (1,1,0), (1,1,1)]

# For Day 7 we need all operator combinations
ops = [add, mul]
list(product(ops, repeat=2))
# → [(add,add), (add,mul), (mul,add), (mul,mul)]
```

Day 7, 2024

Functional

```python
from itertools import product
from operator import add, mul

def can_make(target, nums):
    for operators in product([add, mul], repeat=len(nums) - 1):
        result = nums[0]
        for op, num in zip(operators, nums[1:]):
            result = op(result, num)
        if result == target:
            return True
    return False
```

More combinatorial tools

```python
import itertools as its

# combinations: choose k items (order doesn't matter)
list(its.combinations([1, 2, 3, 4], 2))
# → [(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)]

# permutations: arrange k items (order matters)
list(its.permutations([1, 2, 3], 2))
# → [(1,2), (1,3), (2,1), (2,3), (3,1), (3,2)]

# combinations_with_replacement: can repeat
list(its.combinations_with_replacement([1, 2], 2))
# → [(1,1), (1,2), (2,2)]
```

🧩 Pattern 3

Aggregation & Reduction

🤔 Problem

Combining values across a collection

Day 3, 2021

Binary Diagnostic

Rows of `0` and `1`

Most/least common bits in each **column**

Example: `10110` are the most common bits

```
00100
11110
10110
10111
10101
```

Day 3, 2021

Imperative

```
def solve(data):
    n_cols = len(data[0])
    result = ""

    for col in range(n_cols):
        zeros = 0
        ones = 0
        for row in data:
            if row[col] == '0':
                zeros += 1
            else:
                ones += 1
        result += '1' if ones >= zeros else '0'

    return result
```

`zip(*data)` to the rescue

```
data = ["00100", "11110", "10110"]
columns = list(zip(*data))
# → [('0','1','1'), ('0','1','0'), ('1','1','1'), ...]
#     ↑ col 0         ↑ col 1         ↑ col 2
```

Day 3, 2021

Functional with `zip(*data)`

```
def most_common_bit(data):

    def most_common(bits):
        ones = sum(1 for b in bits if b == '1')
        zeros = len(bits) - ones
        return '1' if ones >= zeros else '0'

    return ''.join(map(most_common, zip(*data)))
```

Reduce, *functionally*

```
from functools import reduce
from operator import add

# Reduction: fold data into single value
reduce(add, [1, 2, 3, 4, 5])
# → ((((1 + 2) + 3) + 4) + 5) = 15
```

Day 3, 2021

Functional with `reduce`

```
from functools import reduce
from operator import add

def solve(data):
    # Build gamma by reducing/joining all most common bits
    gamma = reduce(add, map(most_common_bit, zip(*data)))

    # Epsilon is the bitwise complement
    # i.e., least common bits
    complement = lambda bit: '0' if bit == '1' else '1'
    epsilon = ''.join(map(complement, gamma))

    return gamma, epsilon
```

A functional cheat sheet

When you think...	Reach for...
Adjacent elements	`itertools.pairwise()`
All combinations of X, Y, Z	`itertools.product()`
Choose k items	`itertools.combinations()`
Arrange k items	`itertools.permutations()`
Work on columns not rows	`zip(*data)`
Combine with operation	`functools.reduce()`
Group consecutive elements	`itertools.groupby()`
Chain iterables	`itertools.chain()`
Custom sort comparison	`functools.cmp_to_key()`

Want more?

```
# pip install more-itertools

from more_itertools import windowed, chunked, flatten

# Sliding window of any size
list(windowed([1,2,3,4,5], 3))
# → [(1,2,3), (2,3,4), (3,4,5)]

# Process in batches
list(chunked([1,2,3,4,5,6], 2))
# → [(1,2), (3,4), (5,6)]

# Flatten out nested lists at any level
flatten([[1,2], [3,4], [5]])
# → [1, 2, 3, 4, 5]
```

Day 9, 2024

Disk Fragmenter

When **not** to go functional

Input (compact disk representation):

```
2333133121414131402
```

Becomes:

```
00...111...2...333.44.5555.6666.777.888899
```

Day 9, 2024

A functional breakup 💔

```
class Disk:
    def __init__(self, layout_str):
        self.blocks = self._parse_layout(layout_str)
        self.files = self._extract_files()

    def compact(self):
        """Move blocks left until no gaps"""
        for file_id in reversed(self.files):
            # Find file blocks
            file_positions = [i for i, b in enumerate(self.blocks)
                              if b == file_id]
            # Find leftmost free space
            target = self._find_free_space(len(file_positions))
            # Swap in place
            if target is not None:
                self._move_blocks(file_positions, target)
```

Why functional for coding challenges?

Closer to problem description

Easier to debug

Forces you to think about structure

Takeaways

Think in transformations, not steps (What vs How)

Iterators are your friends

Combinatorics > Nested Loops

Transpose your thinking

Choose the right tool

(Some) Resources

Functional Programming HOWTO

Joel Grus' talk (PyData Seattle 2015)

"Learning Data Science Using Functional Python"

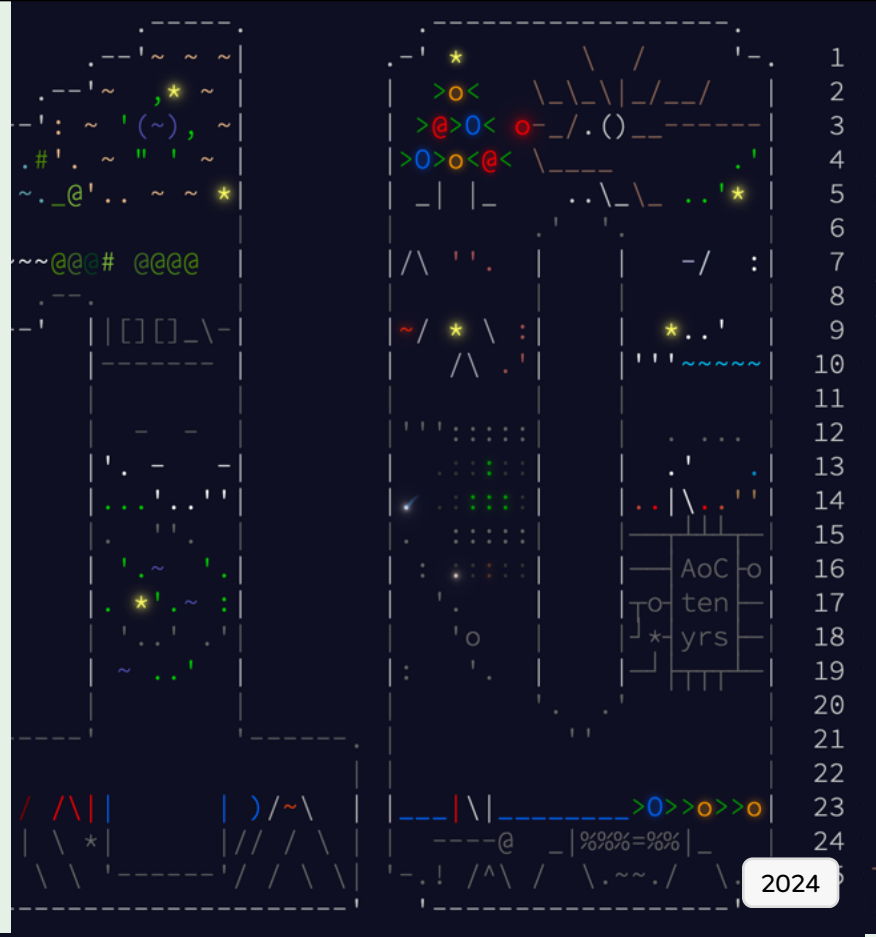
Valerio Maggio's talk (PyConUS 2023)

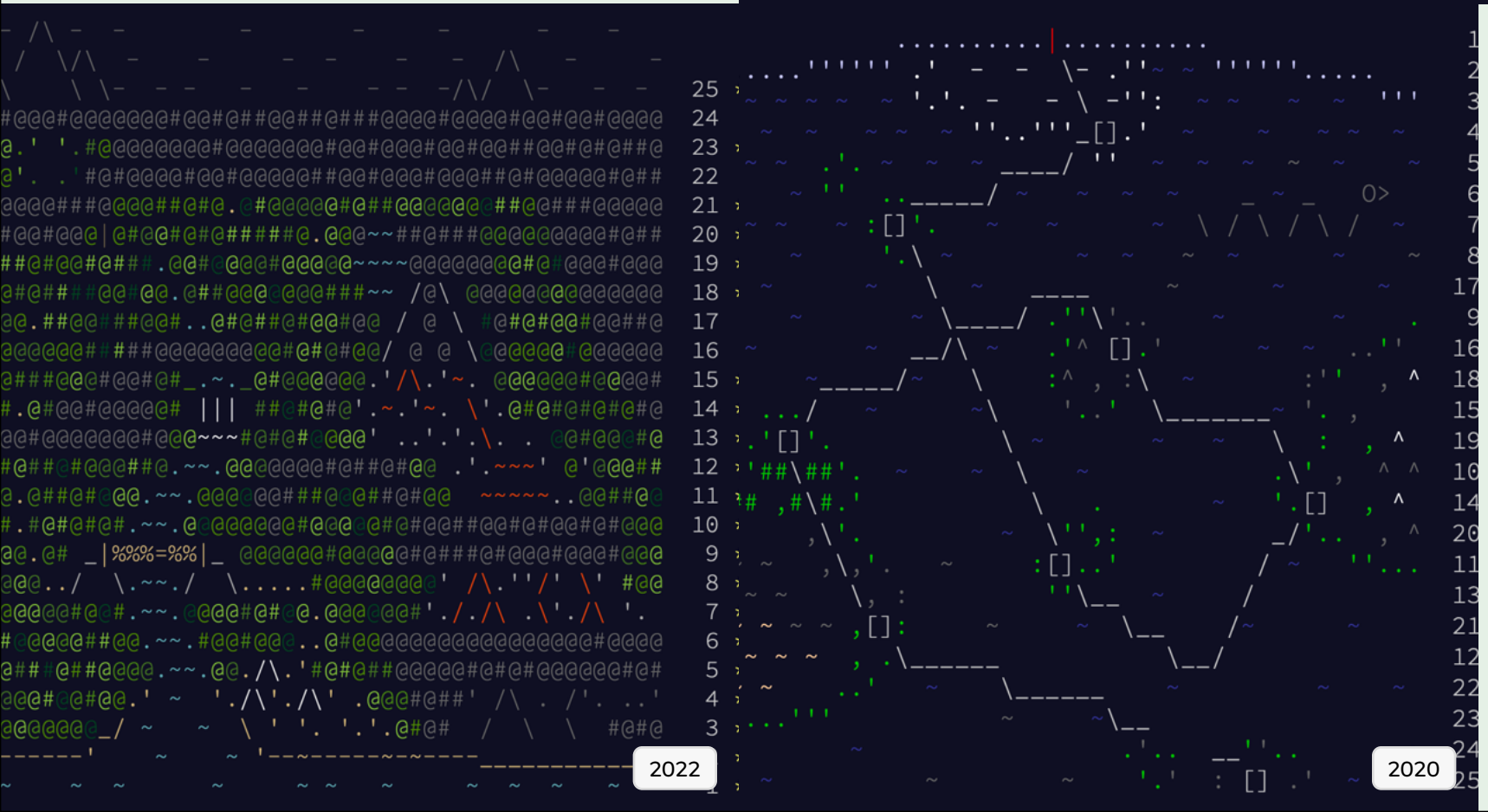
"Pythonic functional (iter)tools for your data challenges

Package **More Itertools**

Rodrigo's "Tour of Python Itertools"

Have yourself a merry, *functional* Christmas 🎄





Where to find me

`edoardob.im`

I'm `edoardob90` on GitHub

(`aoc2024` repo for full solutions)

A few more things

Not mentioned during the talk

Feel free to reach out for any question!

hi@edobld.me

`itertools.pairwise`

Available since Python **3.10**

Otherwise:

```python
from itertools import tee

def pairwise(iterable):
    a, b = tee(iterable)
    next(b, None)
    return zip(a, b)
```

Day 7, 2024

Recursive

```
def can_make(target, nums) -> bool:
    if len(nums) == 1:
        return target == nums[0]
    if target % nums[-1] == 0 and can_make(target // nums[-1], nums[:-1]):
        return True
    if target - nums[-1] > 0 and can_make(target - nums[-1], nums[:-1]):
        return True
    return False
```

Day 7, 2024

A Wolfram approach

```
canMake[target_Integer, {n_Integer}] := target === n

canMake[target_Integer, nums_List] :=
 Which[
  Divisible[target, Last@nums] &&
   canMake[Quotient[target, Last@nums], Most@nums], True,
  With[{diff = target - Last@nums},
   diff > 0 && canMake[diff, Most@nums]], True,
  True, False
  ]
```