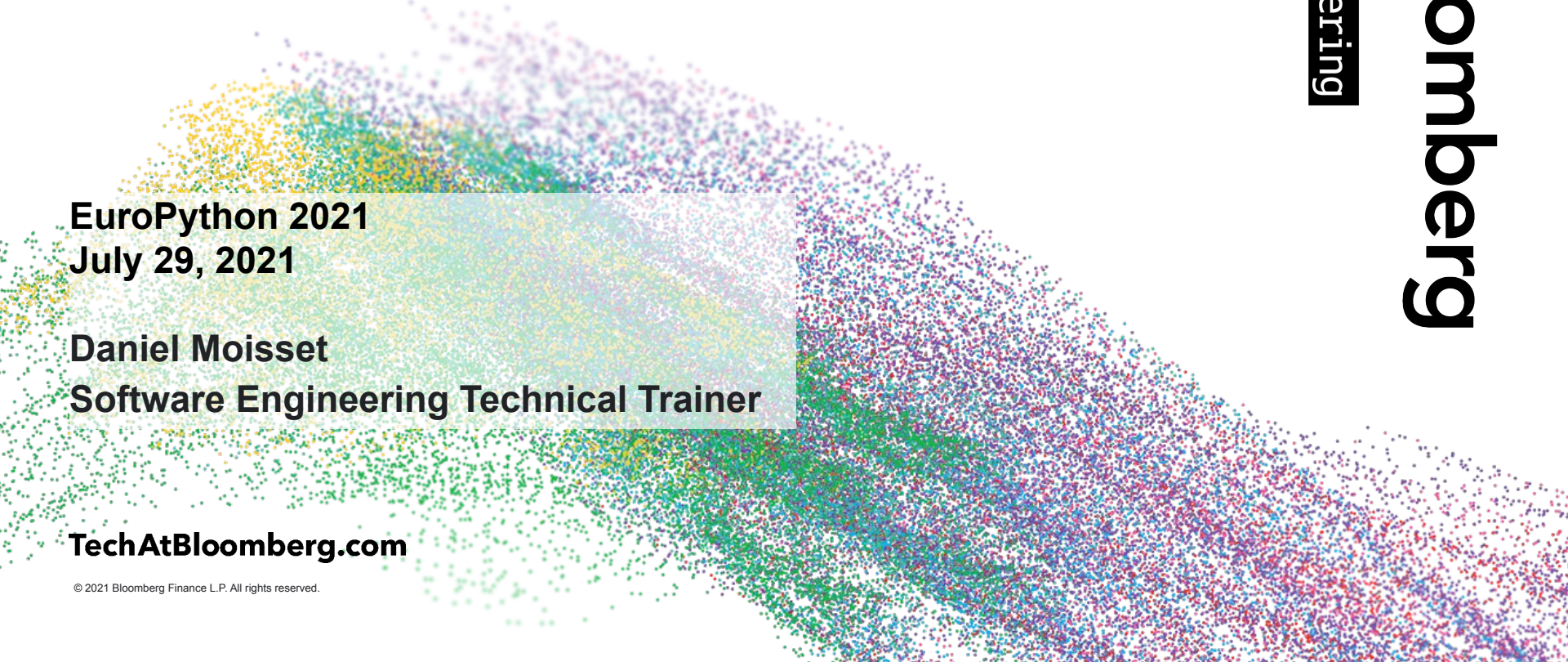
Pattern Matching in Python

Learning why Python and patterns are a perfect match

Bloomberg

Engineering



EuroPython 2021
July 29, 2021

Daniel Moisset
Software Engineering Technical Trainer

TechAtBloomberg.com

Hello!

I am Daniel Moisset

I co-authored the Pattern Matching PEP “trilogy” (PEP 634-636)

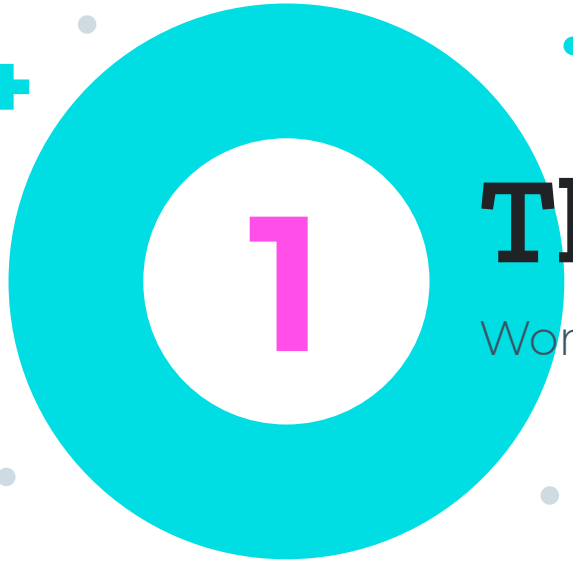
You can find me at @dmoisset

It's out!

Pattern Matching comes with the latest Python 3.10b4 (since alpha 6)

Final 3.10 is planned for October 2021

You're in a maze of twisty little objects, all alike.



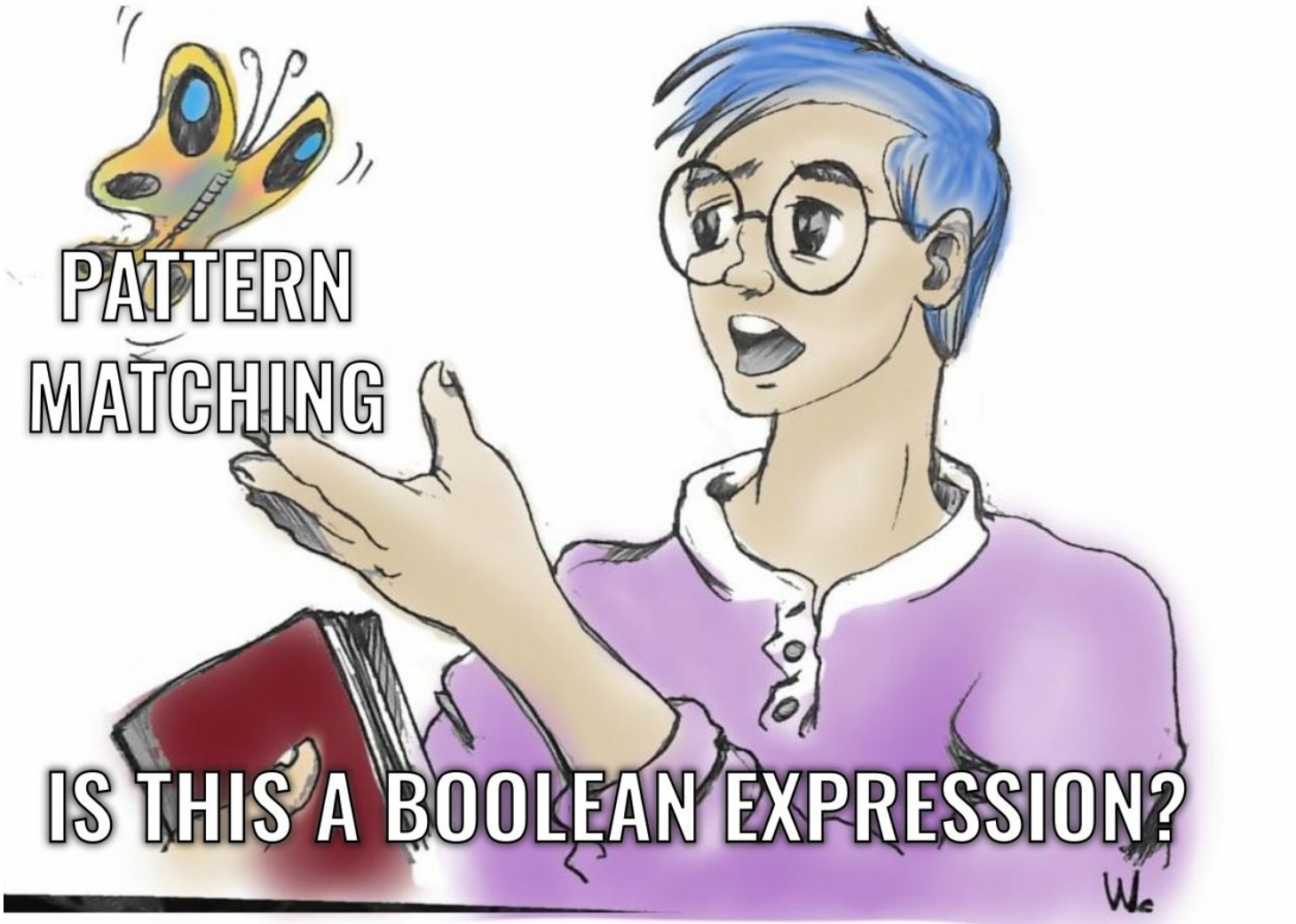
The problem

Working with the “shape” of an object

What is the "shape" of an object?

- Type
- Length (of a sequence)
- Present keys (of a mapping)
- Present attributes (of an instance)
- Exact value

And we can combine and nest these...


PATTERN
MATCHING
IS THIS A BOOLEAN EXPRESSION?

Insight: you're checking the shape of your object to extract data from it!

Condition		Data extraction
`isinstance(x, Duck)`	⟷	`foo(x.quack())`
`len(x) > 2`	⟷	`foo(x[0], x[1])`
`"key" in x`	⟷	`foo(x["key"])`
`hasattr(x, "beak")`	⟷	`foo(x.beak)`

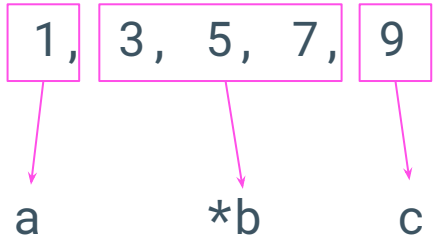
Patterns

Describing shape+extraction

We need some syntax to decompose an object and assign into variables...

...wait, we already have something like that:

```
a, *b, c = [1, 3, 5, 7, 9]
```



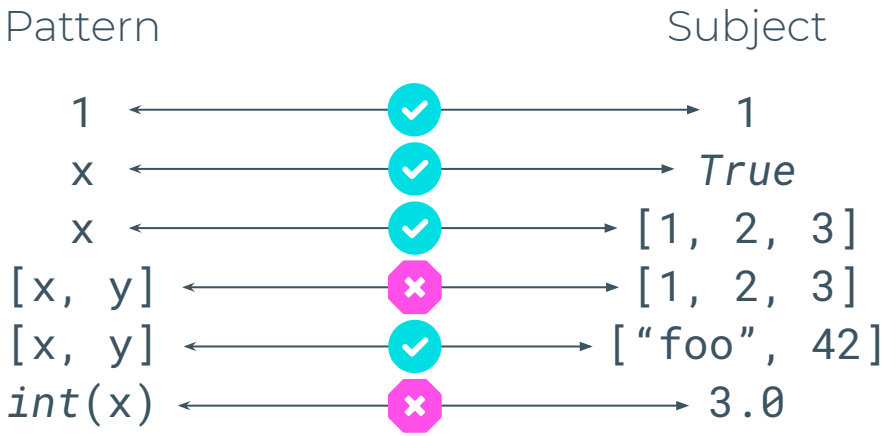
Let's crank this idea up to 11!

Object decomposition mimics object creation:

```
print([a, *b, c])
```

```
[1, 3, 5, 7, 9]
```

Can the *pattern* generate the *subject*?



Some simple patterns:

Pattern	Matches if	Extracts
`123`	`obj == 123`	-
`True`	`obj is True`	-
`os.SEEK_END`	`obj == os.SEEK_END`	-
`x`	always!	`x = obj`
`_`	always!	-
`Duck()`	*`isinstance`*`(obj, Duck)`	-

Some composite patterns:

Pattern	Matches if	Extracts

`(123, a)`	`obj` is a sequence
	`len(obj) == 2`
	`obj[0] == 123`
		`a = obj[1]`

`[int(), *mid, int()]`	`obj` is a sequence
	`len(obj) >= 2`
	`isinstance(obj[0], int)`
	`isinstance(obj[-1], int)`
		`mid = obj[1:-1]`

`{"name": n, "age": a}`	`obj` is a mapping
	`"name" in obj`
	`"age" in obj`
		`n = obj["name"]`
		`a = obj["age"]`

Extra syntax: alternatives and as

Pattern	Matches if	Extracts
`200 | 404 | 500`	`obj in (200, 404, 500)`	-
`[bool()|"yes"|"no" as flag, *_]`	`obj` is a sequence `len(obj) >= 1 and isinstance(obj[0], bool) or obj[0] in ("yes", "no")`	`flag = obj[0]`

`[x]|{"value": x}|x`	always!	One of
		`x = obj[0]`
		`x = obj["value"]`
		`x = obj`

Class patterns

Pattern	Matches if	Extracts
`Duck(color="green")`	*isinstance*(obj, Duck) *hasattr*(obj, "color") obj.color == "green"	-
`Duck(color="green", age=a)`	*isinstance*(obj, Duck) *hasattr*(obj, "color") obj.color == "green" hasattr(obj, "age")	a = obj.age
`Duck(mom=Duck(age=a))`	*isinstance*(obj, Duck) *hasattr*(obj, "mom") *isinstance*(obj.mom, Duck) hasattr(obj.mom, "age")	a = obj.mom.age

Putting it all together

The **`match/case`** statement

Overall syntax

```
match <subject>:
    case <pattern 1>:
        <code block 1>
    case <pattern 2> if <guard>:
        <code block 2>
    case <pattern 3>:
        <code block 3>
```

Let's write a "slicer"

Usage: `slicer(seq, [start], stop, [step])`

```
slicer(seq, stop) → seq[:stop]
slicer(seq, start, stop) → seq[start:stop]
slicer(seq, start, stop, step) → seq[start:stop:step]
```

Ok, let's implement:

```
def slicer(seq, start=0, stop, step=1):
```

Hmm... **SyntaxError**: non-default argument follows default argument

Pattern match the arguments!

```
def slicer(seq, *args):
    match args:
        case [stop]: return seq[:stop]
        case [start, stop]: return seq[start:stop]
        case [start, stop, step]: return seq[start:stop:step]
        case _: raise TypeError("invalid arguments")
```

Let's look at the spec again:

```
slicer(seq, stop) → seq[:stop]
slicer(seq, start, stop) → seq[start:stop]
slicer(seq, start, stop, step) → seq[start:stop:step]
```

As Bruce Eckel said: "Python is executable pseudocode"

FizzBuzz

```
for i in range(1, 101):
    match (i % 3 == 0), (i % 5 == 0):
        case True, True: print("Fizzbuzz")
        case True, _: print("Fizz")
        case _, True: print("Buzz")
        case _: print(i)
```

Matching on API calls

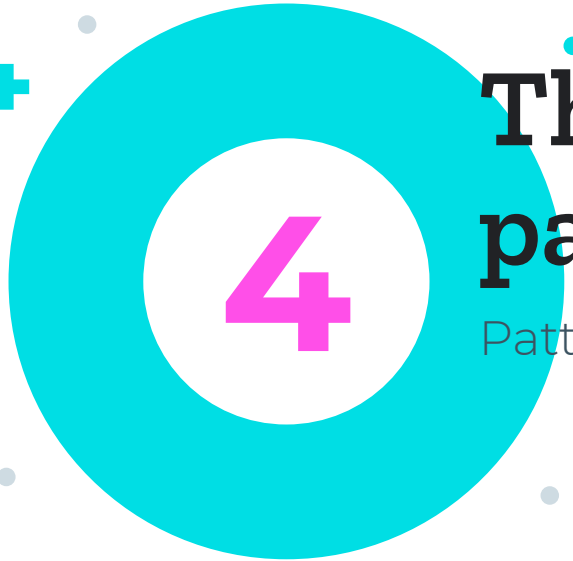
```
resp = get_user_info(uuid).json
match resp:
    case {"error": msg}:
        raise APIError(msg)
    case {"user": {"name": n, "dob": d}} if is_birthday(d):
        render(f"Happy Birthday {n}! 🥳")
    case {"user": {"name": n}}:
        render(f"Welcome {n}!")
    case _:
        raise APIError("Unexpected get_user_info response")
```

A thermonuclear "switch" statement

```
resp = get_user_info(uuid).status
match resp.status:
    case 200:
        process_response(resp.json)
    case 403:
        raise InvalidUser
    case 301:
        process_redirect(resp.location)
    case code if 500 <= code <= 599:
        raise ServerError
    case _:
        raise InvalidStatus
```

Other use cases

- Simple parsers
  - See PEP-636 for an example
- Traverse recursive structures (trees!)
  - Example: github.com/dmoisset/patma/blob/rbtree/examples/rbtree.py



The pattern matching paradigm

Pattern matching as a decomposition approach

An alternative to method dispatch

```
def area(s: Shape):
    match s:
        case Rectangle(w, h): return w * h
        case Square(side): return side ** 2
        case Circle(r): return math.pi * (r ** 2)
        case _: raise InvalidShape

# versus
class Circle(Shape):
    def area(self):
        return math.pi * self.radius ** 2
```

Pattern matching vs. OOP

- PM comes from functional programming
- “Dumb values and smart functions”
- Comparable to method dispatch
  - Emphasis on adding verbs vs. nouns
- Comparable to visitor pattern
  - But straightforward syntax
  - Additional power

Pattern matching vs. switch statement

- Checking is sequential! (a compiler *could* optimise)
- Switch is usually based on literals
- Match can use constant values, but be careful with semantics (“`MY_CONST`” pattern vs. “`mod.MY_CONST`”)

Further Reading

- PEP-636: The tutorial
- PEP-635: Motivational text
- PEP-634: The spec
- github.com/gvanrossum/patma/: Some code examples (possibly outdated)

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Other original images and photographs used with permission from their creators.

Thanks!

You can find me at @dmoisset
and dfmoisset@gmail.com

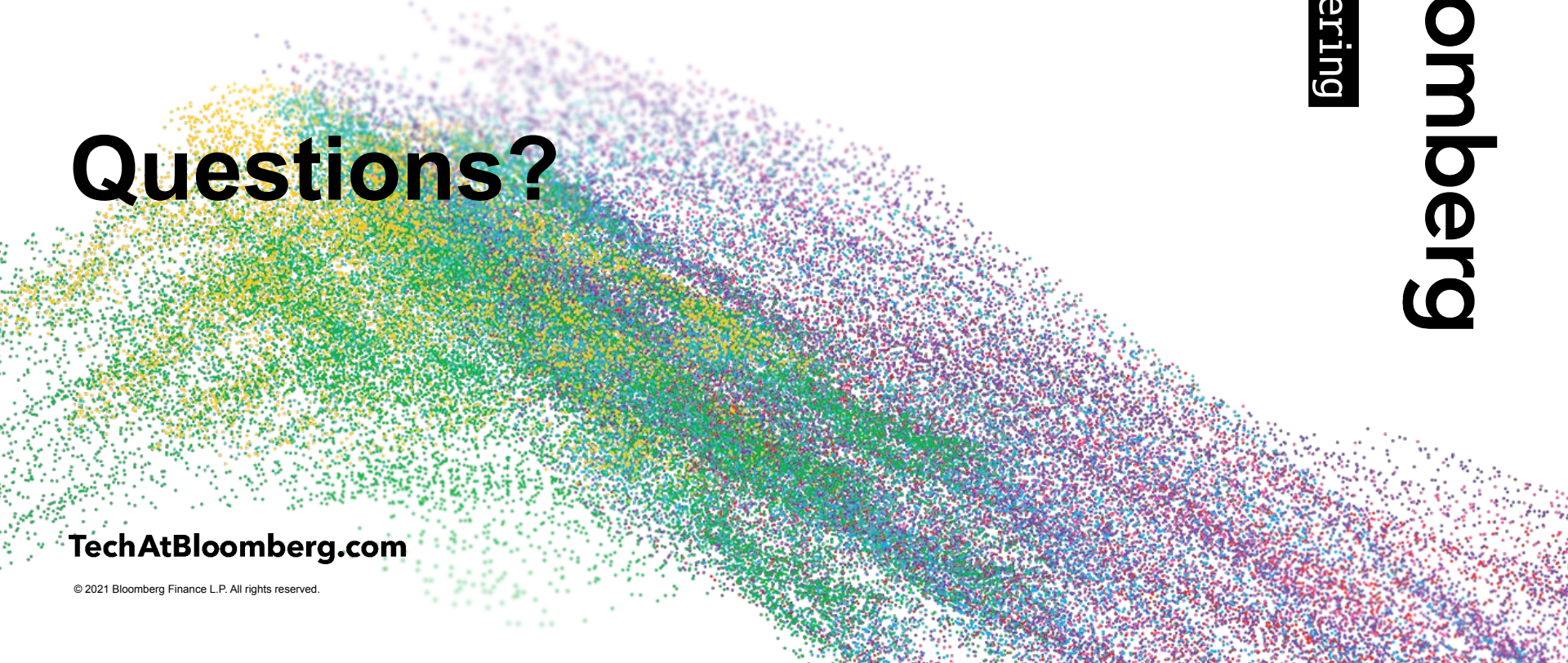
Thank you!

https://www.bloomberg.com/careers

Questions?

Bloomberg

Engineering



TechAtBloomberg.com