Python objects under the hood

by Rodrigo Girão Serrão

EuroPython 2022

(this time, in person 🎉)

About me

Rodrigo Girão Serrão

Formal education: maths

Writing Python for 9 years

Training/teaching

@mathsppblog

Python objects under the hood

Rules

1. Ask questions;
2. Answer *my* questions;
3. OK to interrupt (politely, please 🙃);
4. Write code;
5. Laugh at my jokes.

Plan

1. dunder methods (180 min)

Plan

1. Intro to dunder methods (through `__init__`)
2. Custom arithmetic operations
3. Cookie break
4. `__new__`
5. `__iter__`

Intro to dunder methods

Intro to dunder methods

Plan:

1. `__init__`
2. Dunder methods docs
3. Common dunder methods

Intro to dunder methods

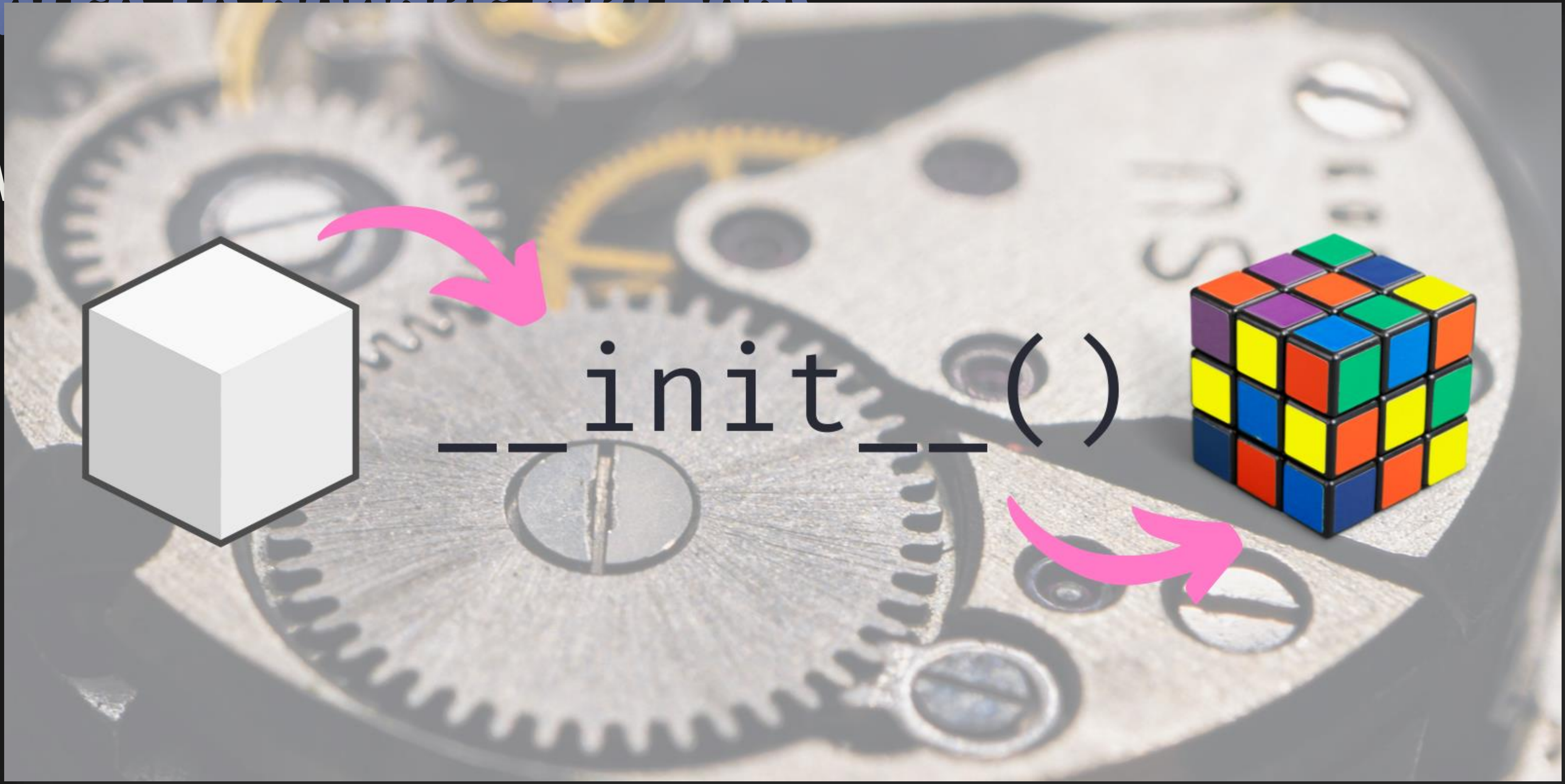
```
Person(name)
```

Intro to dunder methods

What is `__init__`?

Intro to dunder methods

W



Intro to dunder methods

Exercises:

1. `Point2D(x, y)`
2. `Interval(left, right)`
3. `Rectangle(width, height)`

Intro to dunder methods

`__init__` always called implicitly...

Except when it's not: inheritance.

Intro to dunder methods

Exercises:

1. `Point(*coords) > Point2D(x, y)`
2. `Rectangle(width, height) > Square(length)`
3. `Person(name) > Friend(name, nickname)`

Intro to dunder methods

`__init__` is a *dunder* method…

Intro to dunder methods

`__init__` is a *dunder* method…

dunder = *D*ouble *UNDER*score

Intro to dunder methods

Dunder methods AKA magic methods...

Aren't that magic!

Intro to dunder methods

Some common/useful dunder methods:

- `__str__` and `__repr__`
- `__eq__` & (other rich comparison operators)
- `__len__`
- `__contains__`
- `__hash__`
- `__bool__`
- `__getitem__` / `__setitem__` / `__delitem__`
- `__enter__` / `__exit__`
- …

Custom arithmetic operations

Custom arithmetic operations

Plan:

1. `+ - * /`
2. `__add__` vs `__radd__`
3. `NotImplemented`
4. `__add__` vs `__iadd__`

Custom arithmetic operations

Sugar, sugar everywhere!

```
a + b  # so pretty ✨

a.__add__(b)  # 🤮
```

Custom arithmetic operations

Exercises:

1. Implement addition/subtraction for `Point`s of the same length
2. Implement addition/subtraction for `Rectangle`s and numbers

Knowing that:

- `a + b == a.__add__(b)`
- `a - b == a.__sub__(b)`

Custom arithmetic operations

Exercise:

1. Implement multiplication/division between `Point`s and numbers

Knowing that:

- `a * b == a.__mul__(b)`
- `a / b == a.__div__(b)`

Custom arithmetic operations

Exercise:

1. Implement multiplication/division between `Points` and numbers

Knowing that:

- `a * b == a.__mul__(b)`
- ~~`a / b == a.__div__(b)`~~ `# hun?`
- `a / b == a.__truediv__(b)`

Custom arithmetic operations

```
Point2D(3, 56) + 10

==  # ?

10 + Point2D(3, 56)
```

Custom arithmetic operations

Reverse operators: `__radd__`, ...

How does Python know to call `__radd__`?

`NotImplemented`!

Custom arithmetic operations

Exercises:

1. Fix `Rectangle` and number addition/subtraction
2. Fix `Point` and number multiplication/division
3. When possible, defer to the non-reversed dunder

Custom arithmetic operations

`NotImplemented`

vs

`NotImplementedError`

Custom arithmetic operations

Exercises:

1. Return `NotImplemented` when appropriate
2. Implement `Point1D(x)` inheriting from `Point`
3. Implement addition between `Point1D` and `Point` (in `Point1D`)
4. What's running in `Point2D + Point` and `Point + Point2D`?

Custom arithmetic operations

To evaluate `x + y`,

1. `x.__add__(y)`
2. `y.__radd__(x)`

Except if

```
type(y) != type(x) and issubclass(y, x)
```

Custom arithmetic operations

Does this work?

```python
p = Point2D(0, 3)
p += 1
```

But does it *really* work?

Custom arithmetic operations

Augmented assignment: `__iadd__`, …

Exercise:

1. Implement one augmented assignment

Custom arithmetic operations

(Bonus) Exercises:

1. Tabbed printer customisable with `<<` and `>>`
2. Reimplement `pathlib`-like filesystem path joining with `/` and `os.path.join`
3. Regex matcher supporting `+`, `|`, `*`, ...

Cookie break 🍪

(Please check mathspp.com/feedback)

__new__

__new__

Plan:

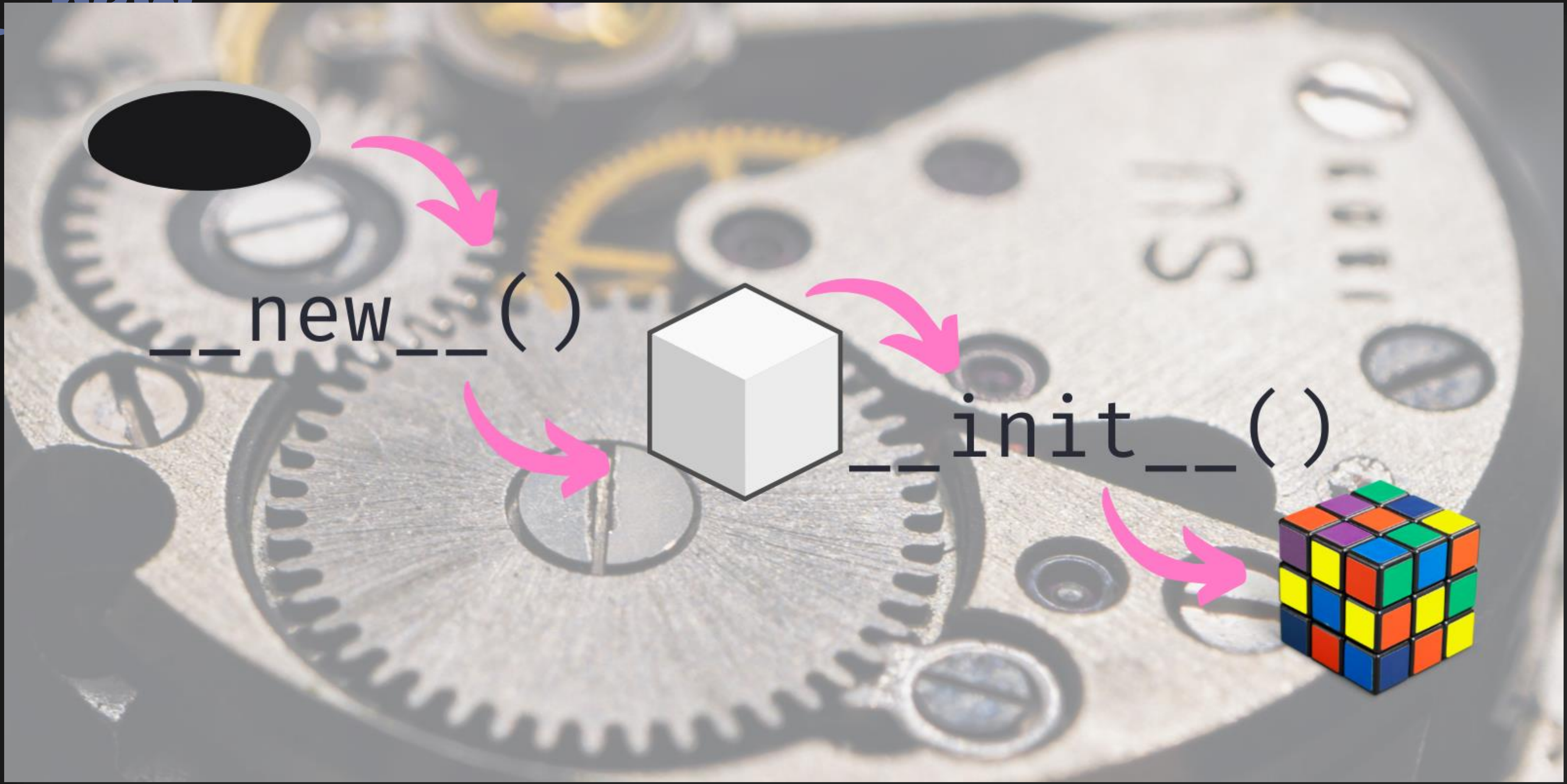
1. `__new__` vs `__init__`
2. Immutable types
3. Inheriting from immutable types
4. Creational patterns

`__new__`

Challenge: prove that `__new__` needs to exist.

Solution: if `__init__` were the sole responsible for object creation, floats/tuples/… would be mutable.

__new



`__new__()`

`__init__()`

__new__

Exercises:

1. Make Point inherit from tuple & fix Point2D
2. Implement FuzzyFloat(x, tol)
    1. Must inherit from float
    2. Overrides __eq__ for equality comparison
    3. Is equal to all numbers within tolerance threshold

__new__

Creational patterns

(Stuff I learn from burying myself in the Standard Library source code.)

__new__

Exercises:

1. Make Point create Point2D or Point1D when needed
2. Rectangle creates Square when needed

Hint: check what `pathlib` does.

__iter__

`__iter__`

1. Plan:
    1. What is `__iter__` for?
    2. Iterators vs iterables
    3. (Lazy) Generators
    4. Strategies to implement `__iter__`

__iter__

What's `__iter__` for?

Turn your objects into iterators.

__iter__

Iterable:

- object implementing __iter__
- __iter__ returns iterator…
- that goes through contents/data of the iterable

Iterator: implements __next__ & is “self-iterable”

__iter__

Get an iterable's iterator with `iter`.

An iterator's iterator is the iterator itself.
(`iter` is idempotent!)

__iter__

Iterables are traversable (with loops, ...).

Iterators are consumable by `next`.

__iter__

Generators are iterators!

Generator for first squares..?

__iter__

Implementing `__iter__`:

1. return an iterator instance (à la `list`, `range`, ...)
2. generator function

`__iter__`

Exercises:

1. Reimplement the iterable behaviour of `range`
2. Reimplement `enumerate`
3. Reimplement `zip`
4. Reimplement `itertools.count`
5. Reimplement `itertools.repeat`
6. Reimplement `itertools.cycle`
7. Reimplement `enumerate` in terms of `zip` and `itertools.count`

(Note: mix & match both strategies)

Feedback appreciated!

mathspp.com/feedback

Recap

- There are way too many dunder methods
- Dunder methods are “hooks” into Python
- `__init__` initialises while `__new__` *creates*
- `__new__` is needed for immutable types
- `__add__` vs `__radd__` vs `__iadd__`
- `NotImplemented` vs `NotImplementedError`
- Iterables (`__iter__`) vs iterators (`__iter__` & `__next__`)

@mathsppblog

References

- Dunder methods, https://mathspp.com/blog/pydonts/dunder-methods
- Object initialisation with __init__, https://mathspp.com/blog/object-initialisation-with-__init__
- Data model, https://docs.python.org/3/reference/datamodel
- Python 3 docs, https://docs.python.org/3

pydonts.com

EuroPython FREE coupon

11-24 July 2022

THE INDIE PYTHON EXTRAVAGANZA

Mike @driscollis

Leonardo @thedigicat

Rodrigo @mathsppblog

Sundeep @learn_byexample

Pydon'ts

Python 101

Python re(gex)?

Practice Python Projects

Clean Architectures in Python

Get the bundle → https://mathspp.com/bundle

$$\overbrace{\underbrace{\text{rodrigo}}_{\text{name}}\text{@}\underbrace{\text{mathspp.com}}_{\text{site}}}^{\text{email}}$$