How dunder methods rule Python under the hood

EuroPython 2023

by Rodrigo Girão Serrão

Rodrigo 🐍🚀

@mathsppblog

Rodrigo 🐍🚀

🐦 @mathsppblog

Textualize

Rodrigo 🐍🚀

@mathsppblog

Textualize

mathspp.com

Slides & code

github.com/mathspp/talks

Goal

What *are* dunder methods?

dunder = ***double under***score

dunder = ***double under***score

- `__init__`

dunder = ***double under***score

- `__init__`
- `__str__`

dunder = ***double under***score

- `__init__`
- `__str__`
- `__eq__`

dunder = ***double under***score

- `__init__`
- `__str__`
- `__eq__`
- …

__str__ & __init__

Let's see it in action!

Called when instance is created.

```python
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    def __str__(self):
        return f"{self.first} {self.last}"
```

Called when instance is converted to string.

Common & useful dunders

- `__init__`
- `__str__` & `__repr__`
- `__eq__`
- `__hash__`

Sequences
& dunder methods

Sequences rely on:

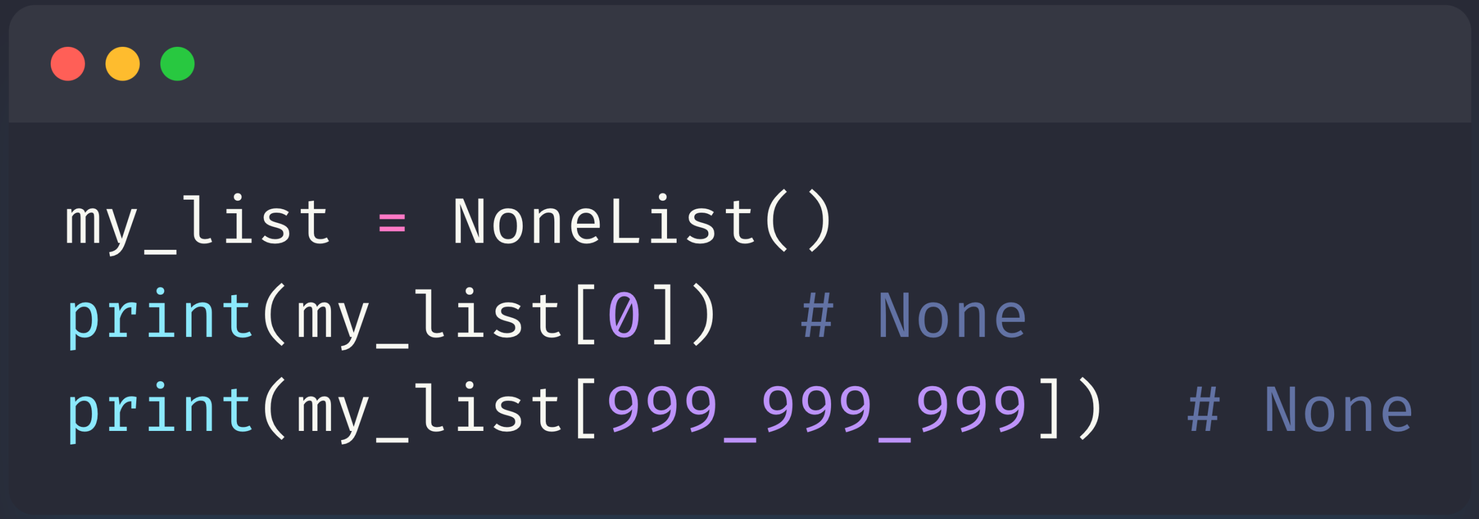
- `__getitem__`

Sequences rely on:

- `__getitem__`
- `__setitem__`

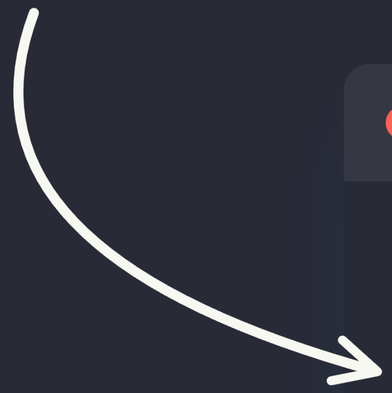
Sequences rely on:

- `__getitem__`
- `__setitem__`
- (and others)



```
my_list = NoneList()
print(my_list[0])  # None
print(my_list[999_999_999])  # None
```

Item access calls `__getitem__`



```
my_list = NoneList()
print(my_list[0])  # None
print(my_list[999_999_999])  # None
```

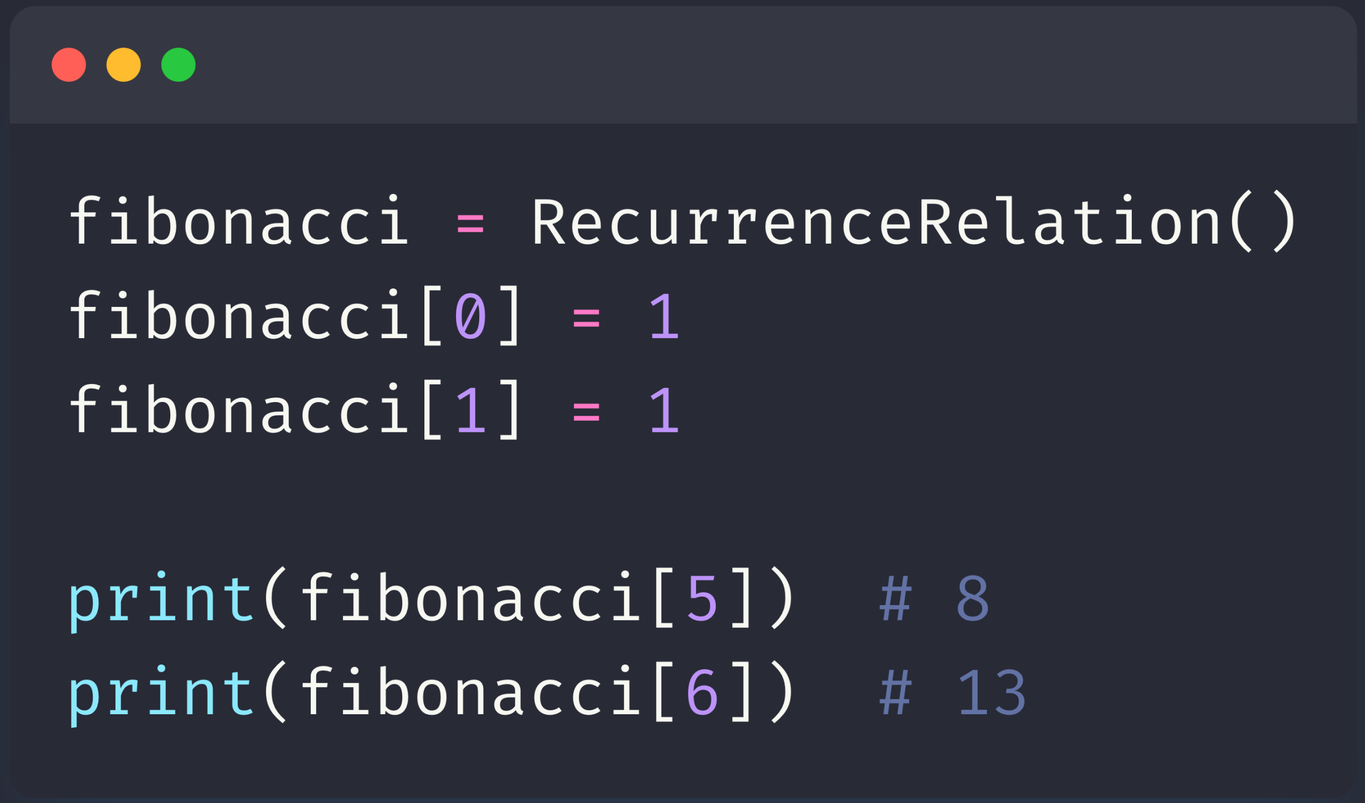
Item access calls `__getitem__`

```
my_list = NoneList()
print(my_list[0])  # None
print(my_list[999_999_999])  # None

my_list[42] = 73
```

Item assignment calls `__setitem__`

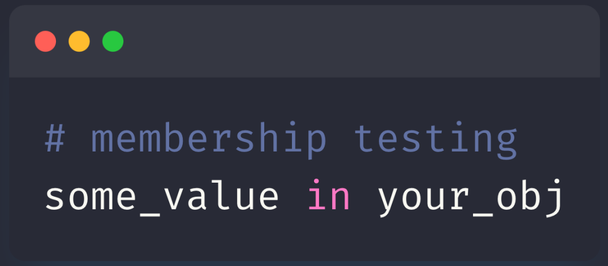
Let's see it in action!



```
fibonacci = RecurrenceRelation()
fibonacci[0] = 1
fibonacci[1] = 1

print(fibonacci[5])  # 8
print(fibonacci[6])  # 13
```

Even more dunders!


membership testing
some_value in your_obj

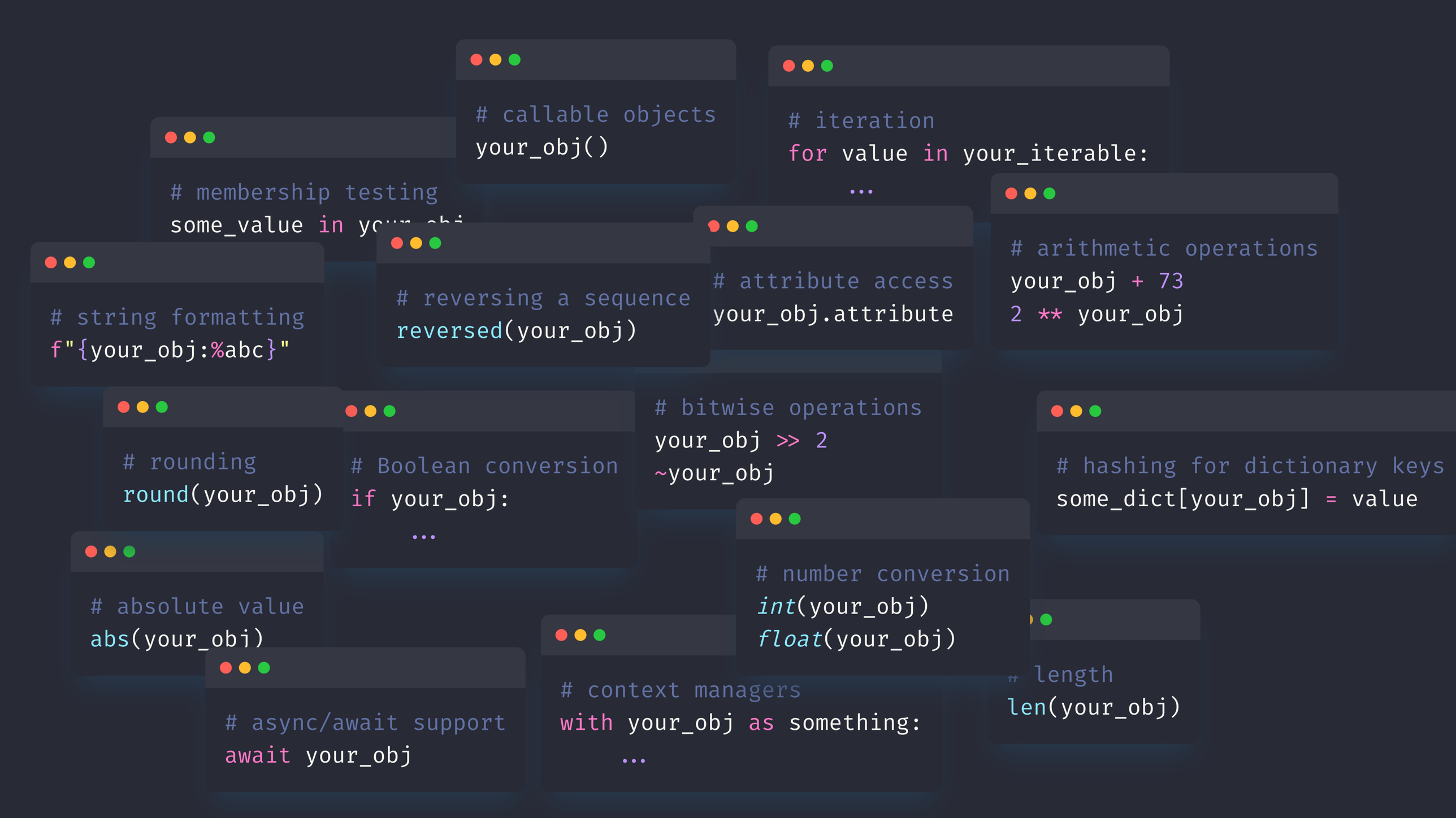
```
# membership testing
some_value in your_obj
```

```
# context managers
with your_obj as something:
    ...
```

```python
# membership testing
some_value in your_obj
```

```python
# attribute access
your_obj.attribute
```

```python
# context managers
with your_obj as something:
    ...
```


# callable objects
your_obj()
# iteration
for value in your_iterable:
...
membership testing
some_value in your_obj
string formatting
f"{your_obj:%abc}"
# reversing a sequence
reversed(your_obj)
# attribute access
your_obj.attribute
arithmetic operations
your_obj + 73
2 ** your_obj
# rounding
round(your_obj)
# Boolean conversion
if your_obj:
...
bitwise operations
your_obj >> 2
~your_obj
# hashing for dictionary keys
some_dict[your_obj] = value
# absolute value
abs(your_obj)
# number conversion
int(your_obj)
float(your_obj)
# async/await support
await your_obj
context managers
with your_obj as something:
...
# length
len(your_obj)

References

References

- mathspp.com/blog/pydonts/dunder-methods
- docs.python.org/3/reference/datamodel

mathspp.com/pydonts

presented with snappify*

*no affiliation, they're just awesome

rodrigo@mathspp.com