Magic of Python

Darshan Markandaiah

@darshanMark dmark11235@gmail.com

bit.ly/magic-of-python

Agenda

What are Magic Methods?

Groups of Magic Methods

- Define Custom Behaviour
- Enable Operator Overloading
- Emulate Built in types

Code for a basic Shopping Cart

```
class BaseCart:
    def __init__(self, user_id):
        self.user_id = user_id
        # Mapping of item_id -> item_count
        self.cart_items = dict()

    def add_item(self, item_id, item_count=1):
        self.cart_items[item_id] = self.cart_items.get(item_id, 0) + item_count
```

```
>>> cart_1 = BaseCart("user_id_1")
>>> cart_1.add_item("item_1")
>>> print(cart_1)

<__main__.BaseCart object at 0x10a196f28>
```

Define Custom Behavior

`__repr__` / `__str__`

```python
class Cart(BaseCart):
    def __str__(self):
        return f"Cart for user_id:{self.user_id} has {len(self.cart_items)} items."

    def __repr__(self):
        return f"Cart('{self.user_id}')"
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1.add_item("item_id_2")
>>> cart_1
Cart('user_id_1')
>>> print(cart_1)
Cart for user_id:user_id_1 has 2 items.
>>> print(f"String formatting: {cart_1}")
String formatting: Cart for user_id:user_id_1 has 2 items.
```

`__bool__`

```
class Cart(BaseCart):
    def __bool__(self):
        return len(self.cart_items) > 0
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> if cart_1:
        # operations for a non empty cart object
        print("Cart contains at least 1 item.")
    else:
        # operations for an empty cart object
        print("Cart is empty.")

Cart contains at least 1 item.
```

Method	Invocation
`__hash__`	hash(object)
`__del__`	del object*

Define Custom Behavior; *: called when reference count reaches 0

Enable Operator Overloading

__gt__

```
class Cart(BaseCart):
    def __gt__(self, other):
        return len(self.cart_items) > len(other.cart_items)
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_2 = Cart("user_id_2")
>>> cart_1.add_item("item_id_1")
>>> cart_2.add_item("item_id_1")
>>> print(cart_1 > cart_2)
False
cart_1.add_item("item_id_2")
>>> print(cart_1 > cart_2)
True
```

Other Comparison Methods

Method	Invocation
`__ge__`	self >= other
`__lt__`	self < other
`__le__`	self <= other
`__eq__`	self == other
`__ne__`	self != other

@functools.total_ordering

```
from functools import total_ordering
@total_ordering
class Cart(BaseCart):
    def __gt__(self, other):
        return len(self.cart_items) > len(other.cart_items)

    def __eq__(self, other):
        return len(self.cart_items) == len(other.cart_items)
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_2 = Cart("user_id_2")
>>> cart_1.add_item("item_id_1")
>>> cart_2.add_item("item_id_1")
>>> print(cart_1 < cart_2)
False
>>> print(cart_1 >= cart_2)
True
```

__add__

```
class Cart(BaseCart):
    def __add__(self, other):
        new_cart = Cart(self.user_id)
        for item_id, item_count in self.cart_items.items():
            new_cart.add_item(item_id, item_count)

        for item_id, item_count in other.cart_items.items():
            new_cart.add_item(item_id, item_count)

        return new_cart
```

```
>>> cart_1, cart_2 = Cart("user_id_1"), Cart("user_id_2")
>>> cart_1.add_item("item_id_1")
>>> cart_2.add_item("item_id_1")
>>> cart_2.add_item("item_id_2")
>>> cart_3 = cart_1 + cart_2
>>> print(cart_3.cart_items)

{'item_id_1': 2, 'item_id_2': 1}
```

Other arithmetic magic methods

Method	Invocation
`__sub__`	self - other
`__mul__`	self * other
`__div__`	self / other
`__pow__`	self ** other

Reflected numerical magic methods

Method	Invocation
`__radd__`	other + self
`__rsub__`	other - self
`__rmul__`	other * self
`__rdiv__`	other / self
`__rpow__`	other ** self

Unary Operations

Method	Invocation
__abs__	abs(self)
__neg__	-self
__pos__	+self

and others found in the documentation...

Emulate Container Types

`__len__`

```
class Cart(BaseCart):
    def __len__(self):
        return sum(self.cart_items.values())
```

```
>>> cart_1 = Cart("user_id_1")

>>> cart_1.add_item("item_id_1", 2)
>>> cart_1.add_item("item_id_2")

>>> len(cart_1)
3
```

`__contains__`

```
class Cart(BaseCart):
    def __contains__(self, key):
        return key in self.cart_items
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> print("item_id_1" in cart_1)
True
>>> print("item_id_2" in cart_1)
False
```

`__iter__`

Iterator Protocol

- Iterables
- Iterators

```
>>> iterable = [1, 2]
>>> iterator = iter(iterable)
>>> print(iterator)
<list_iterator at 0x108c2e048>
>>> print(next(iterator))
1
>>> print(next(iterator))
2
>>> print(next(iterator))
StopIteration
```

`__iter__`

```
class Cart(BaseCart):
    def __iter__(self):
        for item_id in self.cart_items:
            yield item_id
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1.add_item("item_id_2")
>>> for item in cart_1:
        print(item)
'item_id_1'
'item_id_2'
```

`__getitem__`

```
class Cart(BaseCart):
    def __getitem__(self, key):
        return self.cart_items[key]
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1["item_id_1"]
2
```

Related Magic Methods

Method Name	Similar
`__setitem__`	self[key] = value
`__delitem__`	del(self[key])

and others found in the documentation...

collections.abc

Use Case: Checking for interfaces

```
>>> from collections import abc

>>> isinstance(list(), abc.Sequence)
True

>>> isinstance(dict(), abc.Mapping)
True

>>> isinstance(list(), abc.Hashable)
False

>>> isinstance(str(), abc.Hashable)
True
```

Use Case: Offer Free Mixin Methods

```
from collections import abc

class Cart(BaseCart, abc.Mapping):
    def __iter__(self):
        for key in self.cart_items:
            yield key

    def __len__(self):
        return len(self.cart_items)

    def __getitem__(self, key):
        return self.cart_items[key]
```

```
>>> cart_1 = Cart("user_id_1")
>>> cart_1.add_item("item_id_1")
>>> cart_1.add_item("item_id_2")

>>> 'item_id_1' in cart_1
True

>>> list(cart_1.keys())
['item_id_1', 'item_id_2']
```

other ABCs that offer free mixin methods

Context Managers

```
file_handle = open("/path/to/file")
try:
    content = file_handle.read()
finally:
    file_handle.close()

with open("/path/to/file") as file_handle:
    content = file_handle.read()
```

Defining your own Context Managers

- `__enter__`
- `__exit__`

```
class FileReader:
    def __init__(self, filename):
        self.filename = filename

    def __enter__(self):
        self.file_handle = open(self.filename)
        return self.file_handle

    def __exit__(self, exc_type, exc_value, traceback):
        if self.file_handle:
            self.file_handle.close()

with FileReader("/path/to/file") as file_handle:
    content = file_handle.read()
```

@contextlib.contextmanager

Recap

- Enumerated over Magic Methods
- Using `abc` in the `collections` module to get free mixin functionality
- Using contextmanagers to improve pre and post processing steps

Resources

- Rafe Kettler's guide to Python's Magic Methods
- Enriching Your Python Classes With Dunder Methods
- How to make an iterator in Python
- How for loops work in Python
- Python Data Model Documentation
- Python collections module documentation

Thank You!