gitconnected

Python Dunder Methods

A guide to use Python magic methods

Fernando Souza

Jun 29 · 6 min read ★

Photo by Cristian Escobar on Unsplash

In Python, a class can have special methods that are invoked by special syntax (such as arithmetic operations or slicing) by defining methods with certain names.

These methods are called **dunder methods** or **magic methods**, although they don't have any magic in it. They are Python's approach to *operator overloading*.

They are used also to emulate some built-in types and can be used to enrich your classes in a more pythonic way.

Note: this guide was based on Python 3.8.

. . .

Initialization

Methods to help and change the class initialization behavior.

Image by author.

Remarks

- The return of `__new__` should be an instance of the class. If nothing is returned, the new instance's `__init__` is not called.
- No value should be returned by `__init__`. A `TypeError` is raised if anything than `None` is returned.
- The operation `del x` does not directly call `x.__del__`. It decrements the reference count of `x` and only when it is zero that it is really called.
- Any exception that occurs during `__del__` is ignored and a warning is printed to `sys.stderr` instead.

Example

. . .

Representation

Useful to get a string that represents the class instance.

Image by author.

Remarks

- If a class declares `__repr__` but not `__str__`, then `__repr__` is also used as "informal" representation, that is, is called by `str()`.

- The `spec` argument is a string that contains a description of format options desired and is up to the class to interpret it. Most of the time, the class will either delegate to one of the built-in types or follow a similar syntax.
- The major difference between `__repr__` and `__str__` is the intended audience. The `__repr__` function is intended to produce an output that is machine-readable (it can be even a Python expression), while `__str__` is more a human-readable output (it must be a string).

Example

. . .

Rich comparison

Some methods to enable a class to behave like built-in types and implement intuitive comparison between objects using operators.

Image by author.

Remarks

- It may return `NotImplemented` error if does not implement the operation for the given pair of arguments.
- By convention, it should returns `False` and `True`. But these methods can return any value, and if it is used in a Boolean context (such as `if` statement) Python will call `bool()` on the value to determine if it is true or false.
- If `__bool__` is not defined, Python calls `__len__`, and the object is defined true if its result is different than zero.
- If a class does not define `__eq__` then it should not define `__hash__` either.
- A class that defines `__eq__` but does not define `__hash__` will have its `__hash__` implicitly set to `None`. In this case, it is not possible to use them in hashable collections such as `set`.
- `__hash__` should return an integer. And `a == b` means that `hash(a) == hash(b)`.

Example

. . .

Attribute access

Can customize the meaning of access to an attribute (use, assignment or deletion).

Image by author.

Remarks

- If the method is found through normal mechanism, `__getattr__` is never called.
- If the class defines both `__getattr__` and `__getattribute__`, `__getattr__` is only called if `__getattribute__` raises an `AttributeError` or explicitly calls it.
- If `__setattr__` wants to assign to an instance attribute, it should call the base class method with the same name, for example, `object.__setattr__(self, name, value)` or `self.__dict__[name]=value`.

Example

. . .

Descriptors

Descriptors are classes which, when accessed through either getting, setting, or deleting, can also alter other objects. Here the descriptor is a class that declares one or many of theses methods and appears in one attribute of the owner class.

Image by author.

Remarks

- Descriptors aren't meant to stand alone; rather, they're meant to be held by an owner class. Descriptors can be useful when building object-oriented databases or classes that have attributes whose values are dependent on each other
- To be a descriptor, a class must have at least one of `__get__`, `__set__`, and `__delete__` implemented
- The same behavior can be achieved by using `property` decorator.

Example

. . .

Container methods

The following methods can be defined to implement container objects. Containers usually are *sequences* (such as lists or tuples) or *mappings* (like dictionaries), but can represent other containers as well.

Image by author.

Remarks

For methods `__getitem__`, `__setitem__` and `__delitem__`:

- If *key* is of an inappropriate type, it should raise `TypeError`.
- If *key* is a value outside the set of indexes (for sequences type), it should raise `IndexError`.
- If *key* is missing (for mapping types), it should raise `KeyError`.

Example

. . .

Numeric operations

The following methods can be defined to emulate numeric objects.

Image by author.

Remarks

- The augmented arithmetic methods (such as `__iadd__`) should attempt to do the operation in-place, modifying *self*, and return the result (that should be, but not have to be, *self*).
- The reversed binary arithmetic methods (such as `__radd__`) are only called if left operand does not support the corresponding operation and the operands are of different type. That is, in a expression `x + y`, if `x` does not implement `__add__` and `y` does implement `__radd__`, then `y.__radd__(x)` is called.
- If `__int__`, `__float__` and `__complex__` are not defined then the corresponding functions `int()`, `float()` and `complex()` fall back to `__index__`.
- If `__int__` is not defined, then the function `int()` fall back to `__trunc__`.

Example

. . .

Context Managers

Context managers allow setup and cleanup actions to be executed when wrapped by `with` statements or called directly. This can be used to save and restore states, lock and unlock resources, or close files and connections.

Remarks

- If the context was exited without any exception, all three parameters `exc`, `exc_value` and `traceback` will be `None`.
- If an exception is supplied, the method should return `True` to suppress it. Otherwise, it will continue normally upon exit from this method.

Example

. . .

Callable Objects

Allows an instance of an object to be have like a function, that is, you can "call" them.

Image by author.

Remarks

- The `__call__` method can be declare as any other function, as you can declare as many arguments as you want.
- It is useful when the instance must change its current state a lot, like in class that represents a point in plane.

Example

. . .

Conclusion

You can use the *dunder* methods to enrich your classes and emulate built-in types in Python. You can also make some operations a little more "pythonic".

But be careful not to overuse them. As any other feature, this must be used with responsibility.

I hope you enjoyed the reading. If you have any doubt or suggestion, please leave a comment below. You can have more information about the methods in

the official documentation.

Thanks to Yenson Lau.

Python3 Dunder Class Programming

164

WRITTEN BY

Fernando Souza

Follow

Enthusiast of programming, electronics, technology and beer, not necessarily in that order.
Linkedin: https://www.linkedin.com/in/fernandocleber/

Level Up Coding

Follow

Coding tutorials and news. The developer homepage gitconnected.com

More From Medium

Turn Your Google Sheet Into A Web Application

Maulin Tolia in Level Up Coding

Hello (New) World! — Living With and Developing On Apple

Silicon M1

Attila Vágó in Level Up Coding

15 Hilarious Jokes by the Programmers for the Programmers

Lokajit Tikayatray in Level Up Coding

15 Useful VS Code Extensions for a Better Workflow

Nehal Khan in Level Up Coding

Persistent Data Structures for Gophers: Persistent Stack

Oleg Stotsky in Level Up Coding

Top lessons learned from working with a 10x developer

Jeffrey Bakker in Level Up Coding

Handling user authentication and authorization after load balancing your web app.

Dan Mitreanu in Level Up Coding

React: How I learned to create optimized contexts

Thomas Juster in Level Up Coding