Descriptor HowTo Guide

Author: Raymond Hettinger
Contact: <python at rcn dot com>

Contents

:term:`Descriptors <descriptor>` let objects customize attribute lookup, storage, and deletion.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main)(Doc)(howto)descriptor.rst`, **line 13**); ***backlink***
>
> Unknown interpreted text role "term".

This guide has four major sections:

1. The "primer" gives a basic overview, moving gently from simple examples, adding one feature at a time. Start here if you're new to descriptors.
2. The second section shows a complete, practical descriptor example. If you already know the basics, start there.
3. The third section provides a more technical tutorial that goes into the detailed mechanics of how descriptors work. Most people don't need this level of detail.
4. The last section has pure Python equivalents for built-in descriptors that are written in C. Read this if you're curious about how functions turn into bound methods or about the implementation of common tools like :func:`classmethod`, :func:`staticmethod`, :func:`property`, and :term:`__slots__`.

    > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main)(Doc)(howto)descriptor.rst`, **line 28**); ***backlink***
    >
    > Unknown interpreted text role "func".

    > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main)(Doc)(howto)descriptor.rst`, **line 28**); ***backlink***
    >
    > Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 28); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 28); *backlink*

Unknown interpreted text role "term".

Primer

In this primer, we start with the most basic possible example and then we'll add new capabilities one by one.

Simple example: A descriptor that returns a constant

The :class:\`Ten\` class is a descriptor whose :meth:\`__get__\` method always returns the constant `10`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 45); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 45); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 48)

Unknown directive type "testcode".

```
.. testcode::

    class Ten:
        def __get__(self, obj, objtype=None):
            return 10
```

To use the descriptor, it must be stored as a class variable in another class:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 56)

Unknown directive type "testcode".

```
.. testcode::

    class A:
        x = 5                       # Regular class attribute
        y = Ten()                   # Descriptor instance
```

An interactive session shows the difference between normal attribute lookup and descriptor lookup:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 65)

Unknown directive type "doctest".

```
.. doctest::

    >>> a = A()                     # Make an instance of class A
    >>> a.x                         # Normal attribute lookup
    5
    >>> a.y                         # Descriptor lookup
    10
```

In the `a.x` attribute lookup, the dot operator finds `'x': 5` in the class dictionary. In the `a.y` lookup, the dot operator finds a descriptor instance, recognized by its `__get__` method. Calling that method returns `10`.

Note that the value `10` is not stored in either the class dictionary or the instance dictionary. Instead, the value `10` is computed on demand.

This example shows how a simple descriptor works, but it isn't very useful. For retrieving constants, normal attribute lookup would be better.

In the next section, we'll create something more useful, a dynamic lookup.

Dynamic lookups

Interesting descriptors typically run computations instead of returning constants:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 93)

Unknown directive type "testcode".

```
.. testcode::

    import os

    class DirectorySize:

        def __get__(self, obj, objtype=None):
            return len(os.listdir(obj.dirname))

    class Directory:

        size = DirectorySize()              # Descriptor instance

        def __init__(self, dirname):
            self.dirname = dirname          # Regular instance attribute
```

An interactive session shows that the lookup is dynamic â€” it computes different, updated answers each time:

```
>>> s = Directory('songs')
>>> g = Directory('games')
>>> s.size                              # The songs directory has twenty files
20
>>> g.size                              # The games directory has three files
3
>>> os.remove('games/chess')            # Delete a game
>>> g.size                              # File count is automatically updated
2
```

Besides showing how descriptors can run computations, this example also reveals the purpose of the parameters to :meth:`__get__`. The *self* parameter is *size*, an instance of *DirectorySize*. The *obj* parameter is either *g* or *s*, an instance of *Directory*. It is the *obj* parameter that lets the :meth:`__get__` method learn the target directory. The *objtype* parameter is the class *Directory*.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 122); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 122); *backlink*

Unknown interpreted text role "meth".

Managed attributes

A popular use for descriptors is managing access to instance data. The descriptor is assigned to a public attribute in the class dictionary while the actual data is stored as a private attribute in the instance dictionary. The descriptor's :meth:`__get__` and :meth:`__set__` methods are triggered when the public attribute is accessed.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 133); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 133); *backlink*

Unknown interpreted text role "meth".

In the following example, *age* is the public attribute and *_age* is the private attribute. When the public attribute is accessed, the descriptor logs the lookup or update:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 143)

Unknown directive type "testcode".

```
.. testcode::

    import logging

    logging.basicConfig(level=logging.INFO)

    class LoggedAgeAccess:

        def __get__(self, obj, objtype=None):
            value = obj._age
            logging.info('Accessing %r giving %r', 'age', value)
            return value

        def __set__(self, obj, value):
            logging.info('Updating %r to %r', 'age', value)
            obj._age = value

    class Person:

        age = LoggedAgeAccess()             # Descriptor instance

        def __init__(self, name, age):
            self.name = name                # Regular instance attribute
            self.age = age                  # Calls __set__()

        def birthday(self):
            self.age += 1                   # Calls both __get__() and __set__()
```

An interactive session shows that all access to the managed attribute *age* is logged, but that the regular attribute *name* is not logged:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 175)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    import logging, sys
    logging.basicConfig(level=logging.INFO, stream=sys.stdout, force=True)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 181)

Unknown directive type "doctest".

```
.. doctest::

    >>> mary = Person('Mary M', 30)         # The initial age update is logged
    INFO:root:Updating 'age' to 30
    >>> dave = Person('David D', 40)
    INFO:root:Updating 'age' to 40

    >>> vars(mary)                          # The actual data is in a private attribute
    {'name': 'Mary M', '_age': 30}
    >>> vars(dave)
    {'name': 'David D', '_age': 40}
```

```
>>> mary.age                          # Access the data and log the lookup
INFO:root:Accessing 'age' giving 30
30
>>> mary.birthday()                   # Updates are logged as well
INFO:root:Accessing 'age' giving 30
INFO:root:Updating 'age' to 31

>>> dave.name                         # Regular attribute lookup isn't logged
'David D'
>>> dave.age                          # Only the managed attribute is logged
INFO:root:Accessing 'age' giving 40
40
```

One major issue with this example is that the private name *_age* is hardwired in the *LoggedAgeAccess* class. That means that each instance can only have one logged attribute and that its name is unchangeable. In the next example, we'll fix that problem.

Customized names

When a class uses descriptors, it can inform each descriptor about which variable name was used.

In this example, the :class:`Person` class has two descriptor instances, *name* and *age*. When the :class:`Person` class is defined, it makes a callback to :meth:`__set_name__` in *LoggedAccess* so that the field names can be recorded, giving each descriptor its own *public_name* and *private_name*:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 218); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 218); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 218); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 223)

Unknown directive type "testcode".

```
.. testcode::

    import logging

    logging.basicConfig(level=logging.INFO)

    class LoggedAccess:

        def __set_name__(self, owner, name):
            self.public_name = name
            self.private_name = '_' + name

        def __get__(self, obj, objtype=None):
            value = getattr(obj, self.private_name)
            logging.info('Accessing %r giving %r', self.public_name, value)
            return value

        def __set__(self, obj, value):
            logging.info('Updating %r to %r', self.public_name, value)
            setattr(obj, self.private_name, value)

    class Person:

        name = LoggedAccess()                # First descriptor instance
        age = LoggedAccess()                 # Second descriptor instance

        def __init__(self, name, age):
            self.name = name                 # Calls the first descriptor
            self.age = age                   # Calls the second descriptor
    def birthday(self):
        self.age += 1
```

An interactive session shows that the :class:`Person` class has called :meth:`__set_name__` so that the field names would be recorded. Here we call :func:`vars` to look up the descriptor without triggering it:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 256); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 256); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 256); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 260)

Unknown directive type "doctest".

```
.. doctest::

    >>> vars(vars(Person)['name'])
    {'public_name': 'name', 'private_name': '_name'}
    >>> vars(vars(Person)['age'])
    {'public_name': 'age', 'private_name': '_age'}
```

The new class now logs access to both *name* and *age*:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 269)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    import logging, sys
    logging.basicConfig(level=logging.INFO, stream=sys.stdout, force=True)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 275)

Unknown directive type "doctest".

```
.. doctest::

    >>> pete = Person('Peter P', 10)
    INFO:root:Updating 'name' to 'Peter P'
    INFO:root:Updating 'age' to 10
    >>> kate = Person('Catherine C', 20)
    INFO:root:Updating 'name' to 'Catherine C'
    INFO:root:Updating 'age' to 20
```

The two *Person* instances contain only the private names:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 286)

Unknown directive type "doctest".

```
.. doctest::
```

```
>>> vars(pete)
{'_name': 'Peter P', '_age': 10}
>>> vars(kate)
{'_name': 'Catherine C', '_age': 20}
```

Closing thoughts

A :term:`descriptor` is what we call any object that defines :meth:`__get__`, :meth:`__set__`, or :meth:`__delete__`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 297); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 297); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 297); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 297); *backlink*

Unknown interpreted text role "meth".

Optionally, descriptors can have a :meth:`__set_name__` method. This is only used in cases where a descriptor needs to know either the class where it was created or the name of class variable it was assigned to. (This method, if present, is called even if the class is not a descriptor.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 300); *backlink*

Unknown interpreted text role "meth".

Descriptors get invoked by the dot operator during attribute lookup. If a descriptor is accessed indirectly with `vars(some_class)[descriptor_name]`, the descriptor instance is returned without invoking it.

Descriptors only work when used as class variables. When put in instances, they have no effect.

The main motivation for descriptors is to provide a hook allowing objects stored in class variables to control what happens during attribute lookup.

Traditionally, the calling class controls what happens during lookup. Descriptors invert that relationship and allow the data being looked-up to have a say in the matter.

Descriptors are used throughout the language. It is how functions turn into bound methods. Common tools like :func:`classmethod`, :func:`staticmethod`, :func:`property`, and :func:`functools.cached_property` are all implemented as descriptors.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 319); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 319); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 319); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 319); *backlink*

Unknown interpreted text role "func".

Complete Practical Example

In this example, we create a practical and powerful tool for locating notoriously hard to find data corruption bugs.

Validator class

A validator is a descriptor for managed attribute access. Prior to storing any data, it verifies that the new value meets various type and range restrictions. If those restrictions aren't met, it raises an exception to prevent data corruption at its source.

This :class:`Validator` class is both an :term:`abstract base class` and a managed attribute descriptor:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 340); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 340); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 343)

Unknown directive type "testcode".

```
.. testcode::

    from abc import ABC, abstractmethod

    class Validator(ABC):

        def __set_name__(self, owner, name):
            self.private_name = '_' + name

        def __get__(self, obj, objtype=None):
            return getattr(obj, self.private_name)

        def __set__(self, obj, value):
            self.validate(value)
            setattr(obj, self.private_name, value)

        @abstractmethod
        def validate(self, value):
            pass
```

Custom validators need to inherit from :class:`Validator` and must supply a :meth:`validate` method to test various restrictions as needed.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 363); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 363); *backlink*

Unknown interpreted text role "meth".

Custom validators

Here are three practical data validation utilities:

1. :class:`OneOf` verifies that a value is one of a restricted set of options.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 372**); ***backlink***

Unknown interpreted text role "class".

2. :class:`Number` verifies that a value is either an :class:`int` or :class:`float`. Optionally, it verifies that a value is between a given minimum or maximum.

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 374**); ***backlink***

   Unknown interpreted text role "class".

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 374**); ***backlink***

   Unknown interpreted text role "class".

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 374**); ***backlink***

   Unknown interpreted text role "class".

3. :class:`String` verifies that a value is a :class:`str`. Optionally, it validates a given minimum or maximum length. It can validate a user-defined predicate as well.

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 378**); ***backlink***

   Unknown interpreted text role "class".

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 378**); ***backlink***

   Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 383**)

Unknown directive type "testcode".

```
.. testcode::

    class OneOf(Validator):

        def __init__(self, *options):
            self.options = set(options)

        def validate(self, value):
            if value not in self.options:
                raise ValueError(f'Expected {value!r} to be one of {self.options!r}')

    class Number(Validator):

        def __init__(self, minvalue=None, maxvalue=None):
            self.minvalue = minvalue
            self.maxvalue = maxvalue

        def validate(self, value):
            if not isinstance(value, (int, float)):
                raise TypeError(f'Expected {value!r} to be an int or float')
            if self.minvalue is not None and value < self.minvalue:
                raise ValueError(
                    f'Expected {value!r} to be at least {self.minvalue!r}'
                )
            if self.maxvalue is not None and value > self.maxvalue:
                raise ValueError(
                    f'Expected {value!r} to be no more than {self.maxvalue!r}'
                )

    class String(Validator):

        def __init__(self, minsize=None, maxsize=None, predicate=None):
            self.minsize = minsize
            self.maxsize = maxsize
            self.predicate = predicate

        def validate(self, value):
            if not isinstance(value, str):
                raise TypeError(f'Expected {value!r} to be an str')
            if self.minsize is not None and len(value) < self.minsize:
                raise ValueError(
                    f'Expected {value!r} to be no smaller than {self.minsize!r}'
                )
            if self.maxsize is not None and len(value) > self.maxsize:
                raise ValueError(
                    f'Expected {value!r} to be no bigger than {self.maxsize!r}'
                )
            if self.predicate is not None and not self.predicate(value):
                raise ValueError(
                    f'Expected {self.predicate} to be true for {value!r}'
                )
```

Practical application

Here's how the data validators can be used in a real class:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main)(Doc)(howto)descriptor.rst`, line 441)

Unknown directive type "testcode".

```
.. testcode::

    class Component:

        name = String(minsize=3, maxsize=10, predicate=str.isupper)
        kind = OneOf('wood', 'metal', 'plastic')
        quantity = Number(minvalue=0)

        def __init__(self, name, kind, quantity):
            self.name = name
            self.kind = kind
            self.quantity = quantity
```

The descriptors prevent invalid instances from being created:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main)(Doc)(howto)descriptor.rst`, line 456)

Unknown directive type "doctest".

```
.. doctest::

    >>> Component('Widget', 'metal', 5)      # Blocked: 'Widget' is not all uppercase
    Traceback (most recent call last):
        ...
    ValueError: Expected <method 'isupper' of 'str' objects> to be true for 'Widget'

    >>> Component('WIDGET', 'metle', 5)      # Blocked: 'metle' is misspelled
    Traceback (most recent call last):
        ...
    ValueError: Expected 'metle' to be one of {'metal', 'plastic', 'wood'}

    >>> Component('WIDGET', 'metal', -5)     # Blocked: -5 is negative
    Traceback (most recent call last):
        ...
    ValueError: Expected -5 to be at least 0
    >>> Component('WIDGET', 'metal', 'V')    # Blocked: 'V' isn't a number
    Traceback (most recent call last):
```

```
    ...
TypeError: Expected 'V' to be an int or float

>>> c = Component('WIDGET', 'metal', 5)  # Allowed:  The inputs are valid
```

Technical Tutorial

What follows is a more technical tutorial for the mechanics and details of how descriptors work.

Abstract

Defines descriptors, summarizes the protocol, and shows how descriptors are called. Provides an example showing how object relational mappings work.

Learning about descriptors not only provides access to a larger toolset, it creates a deeper understanding of how Python works.

Definition and introduction

In general, a descriptor is an attribute value that has one of the methods in the descriptor protocol. Those methods are :meth:`__get__`, :meth:`__set__`, and :meth:`__delete__`. If any of those methods are defined for an attribute, it is said to be a :term:`descriptor`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 500); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 500); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 500); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 500); *backlink*

Unknown interpreted text role "term".

The default behavior for attribute access is to get, set, or delete the attribute from an object's dictionary. For instance, `a.x` has a lookup chain starting with `a.__dict__['x']`, then `type(a).__dict__['x']`, and continuing through the method resolution order of `type(a)`. If the looked-up value is an object defining one of the descriptor methods, then Python may override the default behavior and invoke the descriptor method instead. Where this occurs in the precedence chain depends on which descriptor methods were defined.

Descriptors are a powerful, general purpose protocol. They are the mechanism behind properties, methods, static methods, class methods, and :func:`super()`. They are used throughout Python itself. Descriptors simplify the underlying C code and offer a flexible set of new tools for everyday Python programs.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 514); *backlink*

Unknown interpreted text role "func".

Descriptor protocol

`descr.__get__(self, obj, type=None) -> value`

`descr.__set__(self, obj, value) -> None`

`descr.__delete__(self, obj) -> None`

That is all there is to it. Define any of these methods and an object is considered a descriptor and can override default behavior upon being looked up as an attribute.

If an object defines :meth:`__set__` or :meth:`__delete__`, it is considered a data descriptor. Descriptors that only define :meth:`__get__` are called non-data descriptors (they are often used for methods but other uses are possible).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 534); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 534); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 534); *backlink*

Unknown interpreted text role "meth".

Data and non-data descriptors differ in how overrides are calculated with respect to entries in an instance's dictionary. If an instance's dictionary has an entry with the same name as a data descriptor, the data descriptor takes precedence. If an instance's dictionary has an entry with the same name as a non-data descriptor, the dictionary entry takes precedence.

To make a read-only data descriptor, define both :meth:`__get__` and :meth:`__set__` with the :meth:`__set__` raising an :exc:`AttributeError` when called. Defining the :meth:`__set__` method with an exception raising placeholder is enough to make it a data descriptor.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 545); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 545); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 545); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 545); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 545); *backlink*

Unknown interpreted text role "meth".

Overview of descriptor invocation

A descriptor can be called directly with `desc.__get__(obj)` or `desc.__get__(None, cls)`.

But it is more common for a descriptor to be invoked automatically from attribute access.

The expression `obj.x` looks up the attribute `x` in the chain of namespaces for `obj`. If the search finds a descriptor outside of the instance `__dict__`, its :meth:`__get__` method is invoked according to the precedence rules listed below.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 560); *backlink*

Unknown interpreted text role "meth".

The details of invocation depend on whether `obj` is an object, class, or instance of super.

Invocation from an instance

Instance lookup scans through a chain of namespaces giving data descriptors the highest priority, followed by instance variables, then non-data descriptors, then class variables, and lastly :meth:`__getattr__` if it is provided.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 572); *backlink*

Unknown interpreted text role "meth".

If a descriptor is found for `a.x`, then it is invoked with: `desc.__get__(a, type(a))`.

The logic for a dotted lookup is in :meth:`object.__getattribute__`. Here is a pure Python equivalent:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 580); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 583)

Unknown directive type "testcode".

```
.. testcode::

    def object_getattribute(obj, name):
        "Emulate PyObject_GenericGetAttr() in Objects/object.c"
        null = object()
        objtype = type(obj)
        cls_var = getattr(objtype, name, null)
        descr_get = getattr(type(cls_var), '__get__', null)
        if descr_get is not null:
            if (hasattr(type(cls_var), '__set__')
                or hasattr(type(cls_var), '__delete__')):
                return descr_get(cls_var, obj, objtype)     # data descriptor
        if hasattr(obj, '__dict__') and name in vars(obj):
            return vars(obj)[name]                          # instance variable
        if descr_get is not null:
            return descr_get(cls_var, obj, objtype)         # non-data descriptor
        if cls_var is not null:
            return cls_var                                  # class variable
        raise AttributeError(name)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 604)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    # Test the fidelity of object_getattribute() by comparing it with the
    # normal object.__getattribute__().  The former will be accessed by
    # square brackets and the latter by the dot operator.

    class Object:

        def __getitem__(obj, name):
            try:
                return object_getattribute(obj, name)
            except AttributeError:
                if not hasattr(type(obj), '__getattr__'):
                    raise
            return type(obj).__getattr__(obj, name)             # __getattr__

    class DualOperator(Object):

        x = 10

        def __init__(self, z):
            self.z = z

        @property
        def p2(self):
            return 2 * self.x

        @property
        def p3(self):
            return 3 * self.x

        def m5(self, y):
            return 5 * y

        def m7(self, y):
            return 7 * y

        def __getattr__(self, name):
            return ('getattr_hook', self, name)

    class DualOperatorWithSlots:

        __getitem__ = Object.__getitem__

        __slots__ = ['z']

        x = 15

        def __init__(self, z):
            self.z = z

        @property
        def p2(self):
            return 2 * self.x

        def m5(self, y):
            return 5 * y

        def __getattr__(self, name):
            return ('getattr_hook', self, name)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 667)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> a = DualOperator(11)
    >>> vars(a).update(p3 = '_p3', m7 = '_m7')
    >>> a.x == a['x'] == 10
    True
    >>> a.z == a['z'] == 11
    True
    >>> a.p2 == a['p2'] == 20
    True
    >>> a.p3 == a['p3'] == 30
    True
    >>> a.m5(100) == a.m5(100) == 500
    True
    >>> a.m7 == a['m7'] == '_m7'
    True
    >>> a.g == a['g'] == ('getattr_hook', a, 'g')
    True

    >>> b = DualOperatorWithSlots(22)
    >>> b.x == b['x'] == 15
    True
    >>> b.z == b['z'] == 22
    True
    >>> b.p2 == b['p2'] == 30
    True
    >>> b.m5(200) == b['m5'](200) == 1000
    True
    >>> b.g == b['g'] == ('getattr_hook', b, 'g')
    True
```

Note, there is no :meth:`__getattr__` hook in the :meth:`__getattribute__` code. That is why calling :meth:`__getattribute__` directly or with `super().__getattribute__` will bypass :meth:`__getattr__` entirely.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 699); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 699); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 699); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 699); *backlink*

Unknown interpreted text role "meth".

Instead, it is the dot operator and the :func:`getattr` function that are responsible for invoking :meth:`__getattr__` whenever :meth:`__getattribute__` raises an :exc:`AttributeError`. Their logic is encapsulated in a helper function:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 703); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 703); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 703); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 703); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 708)

Unknown directive type "testcode".

```
.. testcode::

    def getattr_hook(obj, name):
        "Emulate slot_tp_getattr_hook() in Objects/typeobject.c"
        try:
            return obj.__getattribute__(name)
        except AttributeError:
            if not hasattr(type(obj), '__getattr__'):
                raise
        return type(obj).__getattr__(obj, name)             # __getattr__
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 719)

Unknown directive type "doctest".

```
.. doctest::
    :hide:
```

```
>>> class ClassWithGetAttr:
...     x = 123
...     def __getattr__(self, attr):
...         return attr.upper()
...
>>> cw = ClassWithGetAttr()
>>> cw.y = 456
>>> getattr_hook(cw, 'x')
123
>>> getattr_hook(cw, 'y')
456
>>> getattr_hook(cw, 'z')
'Z'

>>> class ClassWithoutGetAttr:
...     x = 123
...
>>> cwo = ClassWithoutGetAttr()
>>> cwo.y = 456
>>> getattr_hook(cwo, 'x')
123
>>> getattr_hook(cwo, 'y')
456
>>> getattr_hook(cwo, 'z')
Traceback (most recent call last):
    ...
AttributeError: 'ClassWithoutGetAttr' object has no attribute 'z'
```

Invocation from a class

The logic for a dotted lookup such as `A.x` is in :meth:`type.__getattribute__`. The steps are similar to those for :meth:`object.__getattribute__` but the instance dictionary lookup is replaced by a search through the class's :term:`method resolution order`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 755); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 755); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 755); *backlink*

Unknown interpreted text role "term".

If a descriptor is found, it is invoked with `desc.__get__(None, A)`.

The full C implementation can be found in :c:func:`type_getattro()` and :c:func:`_PyType_Lookup()` in :source:`Objects/typeobject.c`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 762); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 762); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 762); *backlink*

Unknown interpreted text role "source".

Invocation from super

The logic for super's dotted lookup is in the :meth:`__getattribute__` method for object returned by :class:`super()`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 769**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 769**); ***backlink***

Unknown interpreted text role "class".

A dotted lookup such as `super(A, obj).m` searches `obj.__class__.__mro__` for the base class `B` immediately following `A` and then returns `B.__dict__['m'].__get__(obj, A)`. If not a descriptor, `m` is returned unchanged.

The full C implementation can be found in :c:func:`super_getattro()` in :source:`Objects/typeobject.c`. A pure Python equivalent can be found in Guido's Tutorial.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 777**); ***backlink***

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 777**); ***backlink***

Unknown interpreted text role "source".

Summary of invocation logic

The mechanism for descriptors is embedded in the :meth:`__getattribute__()` methods for :class:`object`, :class:`type`, and :func:`super`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 786**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 786**); ***backlink***

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 786**); ***backlink***

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 786**); ***backlink***

Unknown interpreted text role "func".

The important points to remember are:

- Descriptors are invoked by the :meth:`__getattribute__` method.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, **line 791**); ***backlink***

  Unknown interpreted text role "meth".

- Classes inherit this machinery from :class:`object`, :class:`type`, or :func:`super`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 793); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 793); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 793); *backlink*

Unknown interpreted text role "func".

- Overriding :meth:`__getattribute__` prevents automatic descriptor calls because all the descriptor logic is in that method.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 796); *backlink*

Unknown interpreted text role "meth".

- :meth:`object.__getattribute__` and :meth:`type.__getattribute__` make different calls to :meth:`__get__`. The first includes the instance and may include the class. The second puts in `None` for the instance and always includes the class.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 799); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 799); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 799); *backlink*

Unknown interpreted text role "meth".

- Data descriptors always override instance dictionaries.
- Non-data descriptors may be overridden by instance dictionaries.

Automatic name notification

Sometimes it is desirable for a descriptor to know what class variable name it was assigned to. When a new class is created, the :class:`type` metaclass scans the dictionary of the new class. If any of the entries are descriptors and if they define :meth:`__set_name__`, that method is called with two arguments. The *owner* is the class where the descriptor is used, and the *name* is the class variable the descriptor was assigned to.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 812); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 812); *backlink*

Unknown interpreted text role "meth".

The implementation details are in :c:func:`type_new()` and :c:func:`set_names()` in :source:`Objects/typeobject.c`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 819); ***backlink***

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 819); ***backlink***

Unknown interpreted text role "c:func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 819); ***backlink***

Unknown interpreted text role "source".

Since the update logic is in :meth:`type.__new__`, notifications only take place at the time of class creation. If descriptors are added to the class afterwards, :meth:`__set_name__` will need to be called manually.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 822); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 822); ***backlink***

Unknown interpreted text role "meth".

ORM example

The following code is simplified skeleton showing how data descriptors could be used to implement an object relational mapping.

The essential idea is that the data is stored in an external database. The Python instances only hold keys to the database's tables. Descriptors take care of lookups or updates:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 838)

Unknown directive type "testcode".

```
.. testcode::

    class Field:

        def __set_name__(self, owner, name):
            self.fetch = f'SELECT {name} FROM {owner.table} WHERE {owner.key}=?;'
            self.store = f'UPDATE {owner.table} SET {name}=? WHERE {owner.key}=?;'

        def __get__(self, obj, objtype=None):
            return conn.execute(self.fetch, [obj.key]).fetchone()[0]

        def __set__(self, obj, value):
            conn.execute(self.store, [value, obj.key])
            conn.commit()
```

We can use the :class:`Field` class to define models that describe the schema for each table in a database:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 853); ***backlink***

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 857)

Unknown directive type "testcode".

```
.. testcode::

    class Movie:
        table = 'Movies'                    # Table name
        key = 'title'                       # Primary key
        director = Field()
        year = Field()

        def __init__(self, key):
            self.key = key

    class Song:
        table = 'Music'
        key = 'title'
        artist = Field()
        year = Field()
        genre = Field()

        def __init__(self, key):
            self.key = key
```

To use the models, first connect to the database:

```
>>> import sqlite3
>>> conn = sqlite3.connect('entertainment.db')
```

An interactive session shows how data is retrieved from the database and how it can be updated:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 886)

Unknown directive type "testsetup".

```
.. testsetup::

    song_data = [
        ('Country Roads', 'John Denver', 1972),
        ('Me and Bobby McGee', 'Janice Joplin', 1971),
        ('Coal Miners Daughter', 'Loretta Lynn', 1970),
    ]

    movie_data = [
        ('Star Wars', 'George Lucas', 1977),
        ('Jaws', 'Steven Spielberg', 1975),
        ('Aliens', 'James Cameron', 1986),
    ]

    import sqlite3

    conn = sqlite3.connect(':memory:')
    conn.execute('CREATE TABLE Music (title text, artist text, year integer);')
    conn.execute('CREATE INDEX MusicNdx ON Music (title);')
    conn.executemany('INSERT INTO Music VALUES (?, ?, ?);', song_data)
    conn.execute('CREATE TABLE Movies (title text, director text, year integer);')
    conn.execute('CREATE INDEX MovieNdx ON Music (title);')
    conn.executemany('INSERT INTO Movies VALUES (?, ?, ?);', movie_data)
    conn.commit()
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 911)

Unknown directive type "doctest".

```
.. doctest::

    >>> Movie('Star Wars').director
    'George Lucas'
    >>> jaws = Movie('Jaws')
    >>> f'Released in {jaws.year} by {jaws.director}'
    'Released in 1975 by Steven Spielberg'

    >>> Song('Country Roads').artist
    'John Denver'

    >>> Movie('Star Wars').director = 'J.J. Abrams'
    >>> Movie('Star Wars').director
    'J.J. Abrams'
```

Pure Python Equivalents

The descriptor protocol is simple and offers exciting possibilities. Several use cases are so common that they have been prepackaged into built-in tools. Properties, bound methods, static methods, class methods, and __slots__ are all based on the descriptor protocol.

Properties

Calling :func:`property` is a succinct way of building a data descriptor that triggers a function call upon access to an attribute. Its signature is:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 939); *backlink*

Unknown interpreted text role "func".

```
property(fget=None, fset=None, fdel=None, doc=None) -> property
```

The documentation shows a typical use to define a managed attribute `x`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 946)

Unknown directive type "testcode".

```
.. testcode::

    class C:
        def getx(self): return self.__x
        def setx(self, value): self.__x = value
        def delx(self): del self.__x
        x = property(getx, setx, delx, "I'm the 'x' property.")
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 954)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> C.x.__doc__
    "I'm the 'x' property."
    >>> c.x = 2.71828
    >>> c.x
    2.71828
    >>> del c.x
    >>> c.x
    Traceback (most recent call last):
      ...
    AttributeError: 'C' object has no attribute '_C__x'
```

To see how :func:`property` is implemented in terms of the descriptor protocol, here is a pure Python equivalent:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 968); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 971)

Unknown directive type "testcode".

```
.. testcode::

    class Property:
```

```
    "Emulate PyProperty_Type() in Objects/descrobject.c"

    def __init__(self, fget=None, fset=None, fdel=None, doc=None):
        self.fget = fget
        self.fset = fset
        self.fdel = fdel
        if doc is None and fget is not None:
            doc = fget.__doc__
        self.__doc__ = doc
        self._name = ''

    def __set_name__(self, owner, name):
        self._name = name

    def __get__(self, obj, objtype=None):
        if obj is None:
            return self
        if self.fget is None:
            raise AttributeError(f"property '{self._name}' has no getter")
        return self.fget(obj)

    def __set__(self, obj, value):
        if self.fset is None:
            raise AttributeError(f"property '{self._name}' has no setter")
        self.fset(obj, value)

    def __delete__(self, obj):
        if self.fdel is None:
            raise AttributeError(f"property '{self._name}' has no deleter")
        self.fdel(obj)

    def getter(self, fget):
        prop = type(self)(fget, self.fset, self.fdel, self.__doc__)
        prop._name = self._name
        return prop

    def setter(self, fset):
        prop = type(self)(self.fget, fset, self.fdel, self.__doc__)
        prop._name = self._name
        return prop

    def deleter(self, fdel):
        prop = type(self)(self.fget, self.fset, fdel, self.__doc__)
        prop._name = self._name
        return prop
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1020)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    # Verify the Property() emulation

    class CC:
        def getx(self):
            return self.__x
        def setx(self, value):
            self.__x = value
        def delx(self):
            del self.__x
        x = Property(getx, setx, delx, "I'm the 'x' property.")

    # Now do it again but use the decorator style

    class CCC:
        @Property
        def x(self):
            return self.__x
        @x.setter
        def x(self, value):
            self.__x = value
        @x.deleter
        def x(self):
            del self.__x
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1048)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> cc = CC()
    >>> hasattr(cc, 'x')
    False
    >>> cc.x = 33
    >>> cc.x
    33
    >>> del cc.x
    >>> hasattr(cc, 'x')
    False

    >>> ccc = CCC()
    >>> hasattr(ccc, 'x')
    False
    >>> ccc.x = 333
    >>> ccc.x == 333
    True
    >>> del ccc.x
    >>> hasattr(ccc, 'x')
    False
```

The :func:`property` builtin helps whenever a user interface has granted attribute access and then subsequent changes require the intervention of a method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1071); *backlink*

Unknown interpreted text role "func".

For instance, a spreadsheet class may grant access to a cell value through `Cell('b10').value`. Subsequent improvements to the program require the cell to be recalculated on every access; however, the programmer does not want to affect existing client code accessing the attribute directly. The solution is to wrap access to the value attribute in a property data descriptor:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1081)

Unknown directive type "testcode".

```
.. testcode::

    class Cell:
        ...

        @property
        def value(self):
            "Recalculate the cell before returning value"
            self.recalc()
            return self._value
```

Either the built-in :func:`property` or our :func:`Property` equivalent would work in this example.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1092); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1092); *backlink*

Unknown interpreted text role "func".

Functions and methods

Python's object oriented features are built upon a function based environment. Using non-data descriptors, the two are merged seamlessly.

Functions stored in class dictionaries get turned into methods when invoked. Methods only differ from regular functions in that the object instance is prepended to the other arguments. By convention, the instance is called *self* but could be called *this* or any other variable name.

Methods can be created manually with :class:`types.MethodType` which is roughly equivalent to:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1107); ***backlink***

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1110)

Unknown directive type "testcode".

```
.. testcode::

    class MethodType:
        "Emulate PyMethod_Type in Objects/classobject.c"

        def __init__(self, func, obj):
            self.__func__ = func
            self.__self__ = obj

        def __call__(self, *args, **kwargs):
            func = self.__func__
            obj = self.__self__
            return func(obj, *args, **kwargs)
```

To support automatic creation of methods, functions include the :meth:`__get__` method for binding methods during attribute access. This means that functions are non-data descriptors that return bound methods during dotted lookup from an instance. Here's how it works:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1124); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1129)

Unknown directive type "testcode".

```
.. testcode::

    class Function:
        ...

        def __get__(self, obj, objtype=None):
            "Simulate func_descr_get() in Objects/funcobject.c"
            if obj is None:
                return self
            return MethodType(self, obj)
```

Running the following class in the interpreter shows how the function descriptor works in practice:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1143)

Unknown directive type "testcode".

```
.. testcode::

    class D:
        def f(self, x):
             return x
```

The function has a :term:`qualified name` attribute to support introspection:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1149); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1151)

Unknown directive type "doctest".

```
.. doctest::

    >>> D.f.__qualname__
    'D.f'
```

Accessing the function through the class dictionary does not invoke :meth:`__get__`. Instead, it just returns the underlying function object:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1156); *backlink*

Unknown interpreted text role "meth".

```
>>> D.__dict__['f']
<function D.f at 0x00C45070>
```

Dotted access from a class calls :meth:`__get__` which just returns the underlying function unchanged:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1162); *backlink*

Unknown interpreted text role "meth".

```
>>> D.f
<function D.f at 0x00C45070>
```

The interesting behavior occurs during dotted access from an instance. The dotted lookup calls :meth:`__get__` which returns a bound method object:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1168); *backlink*

Unknown interpreted text role "meth".

```
>>> d = D()
>>> d.f
<bound method D.f of <__main__.D object at 0x00B18C90>>
```

Internally, the bound method stores the underlying function and the bound instance:

```
>>> d.f.__func__
<function D.f at 0x00C45070>

>>> d.f.__self__
<__main__.D object at 0x1012e1f98>
```

If you have ever wondered where *self* comes from in regular methods or where *cls* comes from in class methods, this is it!

Kinds of methods

Non-data descriptors provide a simple mechanism for variations on the usual patterns of binding functions into methods.

To recap, functions have a :meth:`__get__` method so that they can be converted to a method when accessed as attributes. The non-data descriptor transforms an `obj.f(*args)` call into `f(obj, *args)`. Calling `cls.f(*args)` becomes `f(*args)`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1194); *backlink*

Unknown interpreted text role "meth".

This chart summarizes the binding and its two most useful variants:

Transformation	Called from an object	Called from a class
function	f(obj, *args)	f(*args)
staticmethod	f(*args)	f(*args)
classmethod	f(type(obj), *args)	f(cls, *args)

Static methods

Static methods return the underlying function without changes. Calling either `c.f` or `C.f` is the equivalent of a direct lookup into `object.__getattribute__(c, "f")` or `object.__getattribute__(C, "f")`. As a result, the function becomes identically accessible from either an object or a class.

Good candidates for static methods are methods that do not reference the `self` variable.

For instance, a statistics package may include a container class for experimental data. The class provides normal methods for computing the average, mean, median, and other descriptive statistics that depend on the data. However, there may be useful functions which are conceptually related but do not depend on the data. For instance, `erf(x)` is handy conversion routine that comes up in statistical work but does not directly depend on a particular dataset. It can be called either from an object or the class: `s.erf(1.5) --> .9332` or `Sample.erf(1.5) --> .9332`.

Since static methods return the underlying function with no changes, the example calls are unexciting:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1237)

Unknown directive type "testcode".

```
.. testcode::

    class E:
        @staticmethod
        def f(x):
            return x * 10
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1244)

Unknown directive type "doctest".

```
.. doctest::

    >>> E.f(3)
    30
    >>> E().f(3)
    30
```

Using the non-data descriptor protocol, a pure Python version of :func:`staticmethod` would look like this:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1251); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1254)

Unknown directive type "testcode".

```
.. testcode::

    class StaticMethod:
        "Emulate PyStaticMethod_Type() in Objects/funcobject.c"

        def __init__(self, f):
            self.f = f

        def __get__(self, obj, objtype=None):
            return self.f

        def __call__(self, *args, **kwds):
            return self.f(*args, **kwds)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1268)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    class E_sim:
        @StaticMethod
        def f(x):
            return x * 10

    wrapped_ord = StaticMethod(ord)
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1278)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> E_sim.f(3)
    30
    >>> E_sim().f(3)
    30
    >>> wrapped_ord('A')
    65
```

Class methods

Unlike static methods, class methods prepend the class reference to the argument list before calling the function. This format is the same for whether the caller is an object or a class:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1296)

Unknown directive type "testcode".

```
.. testcode::

    class F:
        @classmethod
        def f(cls, x):
            return cls.__name__, x
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1303)

Unknown directive type "doctest".

```
.. doctest::

    >>> F.f(3)
    ('F', 3)
    >>> F().f(3)
    ('F', 3)
```

This behavior is useful whenever the method only needs to have a class reference and does not rely on data stored in a specific instance. One use for class methods is to create alternate class constructors. For example, the classmethod :func:`dict.fromkeys` creates a new dictionary from a list of keys. The pure Python equivalent is:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1310); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1316)

Unknown directive type "testcode".

```
.. testcode::

    class Dict(dict):
        @classmethod
        def fromkeys(cls, iterable, value=None):
            "Emulate dict_fromkeys() in Objects/dictobject.c"
            d = cls()
            for key in iterable:
                d[key] = value
            return d
```

Now a new dictionary of unique keys can be constructed like this:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1329)

Unknown directive type "doctest".

```
.. doctest::

    >>> d = Dict.fromkeys('abracadabra')
    >>> type(d) is Dict
    True
    >>> d
    {'a': None, 'b': None, 'r': None, 'c': None, 'd': None}
```

Using the non-data descriptor protocol, a pure Python version of :func:`classmethod` would look like this:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1337); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1340)

Unknown directive type "testcode".

```
.. testcode::

    class ClassMethod:
        "Emulate PyClassMethod_Type() in Objects/funcobject.c"

        def __init__(self, f):
            self.f = f

        def __get__(self, obj, cls=None):
            if cls is None:
                cls = type(obj)
            if hasattr(type(self.f), '__get__'):
                return self.f.__get__(cls, cls)
            return MethodType(self.f, cls)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst, line 1355)

Unknown directive type "testcode".

```
.. testcode::
    :hide:

    # Verify the emulation works
    class T:
        @ClassMethod
        def cm(cls, x, y):
            return (cls, x, y)

        @ClassMethod
        @property
        def __doc__(cls):
            return f'A doc for {cls.__name__!r}'
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1370)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> T.cm(11, 22)
    (<class 'T'>, 11, 22)

    # Also call it from an instance
    >>> t = T()
    >>> t.cm(11, 22)
    (<class 'T'>, 11, 22)

    # Check the alternate path for chained descriptors
    >>> T.__doc__
    "A doc for 'T'"
```

The code path for `hasattr(type(self.f), '__get__')` was added in Python 3.9 and makes it possible for :func:`classmethod` to support chained decorators. For example, a classmethod and property could be chained together:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1386); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1391)

Unknown directive type "testcode".

```
.. testcode::

    class G:
        @classmethod
        @property
        def __doc__(cls):
            return f'A doc for {cls.__name__!r}'
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1399)

Unknown directive type "doctest".

```
.. doctest::

    >>> G.__doc__
    "A doc for 'G'"
```

Member objects and __slots__

When a class defines `__slots__`, it replaces instance dictionaries with a fixed-length array of slot values. From a user point of view that has several effects:

1. Provides immediate detection of bugs due to misspelled attribute assignments. Only attribute names specified in `__slots__` are allowed:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1415)

Unknown directive type "testcode".

```
.. testcode::
        class Vehicle:
            __slots__ = ('id_number', 'make', 'model')
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1420)

Unknown directive type "doctest".

```
.. doctest::

        >>> auto = Vehicle()
        >>> auto.id_nubmer = 'VYE483814LQEX'
        Traceback (most recent call last):
            ...
        AttributeError: 'Vehicle' object has no attribute 'id_nubmer'
```

2. Helps create immutable objects where descriptors manage access to private attributes stored in `__slots__`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1431)

Unknown directive type "testcode".

```
.. testcode::

    class Immutable:

        __slots__ = ('_dept', '_name')          # Replace the instance dictionary

        def __init__(self, dept, name):
            self._dept = dept                   # Store to private attribute
            self._name = name                   # Store to private attribute

        @property                               # Read-only descriptor
        def dept(self):
            return self._dept

        @property
        def name(self):                         # Read-only descriptor
            return self._name
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1449)

Unknown directive type "doctest".

```
.. doctest::

    >>> mark = Immutable('Botany', 'Mark Watney')
    >>> mark.dept
    'Botany'
    >>> mark.dept = 'Space Pirate'
    Traceback (most recent call last):
        ...
    AttributeError: property 'dept' of 'Immutable' object has no setter
    >>> mark.location = 'Mars'
    Traceback (most recent call last):
        ...
    AttributeError: 'Immutable' object has no attribute 'location'
```

3. Saves memory. On a 64-bit Linux build, an instance with two attributes takes 48 bytes with `__slots__` and 152 bytes without. This flyweight design pattern likely only matters when a large number of instances are going to be created.

4. Improves speed. Reading instance variables is 35% faster with `__slots__` (as measured with Python 3.10 on an Apple M1 processor).

5. Blocks tools like :func:`functools.cached_property` which require an instance dictionary to function correctly:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1471); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1474)

Unknown directive type "testcode".

```
.. testcode::

    from functools import cached_property

    class CP:
        __slots__ = ()                          # Eliminates the instance dict

        @cached_property                        # Requires an instance dict
        def pi(self):
            return 4 * sum((-1.0)**n / (2.0*n + 1.0)
                           for n in reversed(range(100_000)))
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1486)

Unknown directive type "doctest".

```
.. doctest::

    >>> CP().pi
    Traceback (most recent call last):
      ...
    TypeError: No '__dict__' attribute on 'CP' instance to cache 'pi' property.
```

It is not possible to create an exact drop-in pure Python version of `__slots__` because it requires direct access to C structures and control over object memory allocation. However, we can build a mostly faithful simulation where the actual C structure for slots is emulated by a private `_slotvalues` list. Reads and writes to that private structure are managed by member descriptors:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1500)

Unknown directive type "testcode".

```
.. testcode::

    null = object()

    class Member:

        def __init__(self, name, clsname, offset):
            'Emulate PyMemberDef in Include/structmember.h'
            # Also see descr_new() in Objects/descrobject.c
            self.name = name
            self.clsname = clsname
            self.offset = offset

        def __get__(self, obj, objtype=None):
            'Emulate member_get() in Objects/descrobject.c'
            # Also see PyMember_GetOne() in Python/structmember.c
            value = obj._slotvalues[self.offset]
            if value is null:
                raise AttributeError(self.name)
            return value

        def __set__(self, obj, value):
            'Emulate member_set() in Objects/descrobject.c'
            obj._slotvalues[self.offset] = value

        def __delete__(self, obj):
            'Emulate member_delete() in Objects/descrobject.c'
            value = obj._slotvalues[self.offset]
            if value is null:
                raise AttributeError(self.name)
            obj._slotvalues[self.offset] = null

        def __repr__(self):
            'Emulate member_repr() in Objects/descrobject.c'
            return f'<Member {self.name!r} of {self.clsname!r}>'
```

The :meth:`type.__new__` method takes care of adding member objects to class variables:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1536); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1539)

Unknown directive type "testcode".

```
.. testcode::

    class Type(type):
        'Simulate how the type metaclass adds member objects for slots'

        def __new__(mcls, clsname, bases, mapping):
            'Emulate type_new() in Objects/typeobject.c'
            # type_new() calls PyTypeReady() which calls add_methods()
            slot_names = mapping.get('slot_names', [])
            for offset, name in enumerate(slot_names):
                mapping[name] = Member(name, clsname, offset)
            return type.__new__(mcls, clsname, bases, mapping)
```

The :meth:`object.__new__` method takes care of creating instances that have slots instead of an instance dictionary. Here is a rough simulation in pure Python:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1552); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1556)

Unknown directive type "testcode".

```
.. testcode::

    class Object:
        'Simulate how object.__new__() allocates memory for __slots__'

        def __new__(cls, *args):
            'Emulate object_new() in Objects/typeobject.c'
            inst = super().__new__(cls)
            if hasattr(cls, 'slot_names'):
                empty_slots = [null] * len(cls.slot_names)
                object.__setattr__(inst, '_slotvalues', empty_slots)
            return inst

        def __setattr__(self, name, value):
            'Emulate _PyObject_GenericSetAttrWithDict() Objects/object.c'
            cls = type(self)
            if hasattr(cls, 'slot_names') and name not in cls.slot_names:
                raise AttributeError(
                    f'{type(self).__name__!r} object has no attribute {name!r}'
                )
            super().__setattr__(name, value)

        def __delattr__(self, name):
            'Emulate _PyObject_GenericSetAttrWithDict() Objects/object.c'
            cls = type(self)
            if hasattr(cls, 'slot_names') and name not in cls.slot_names:
                raise AttributeError(
                    f'{type(self).__name__!r} object has no attribute {name!r}'
                )
            super().__delattr__(name)
```

To use the simulation in a real class, just inherit from :class:`Object` and set the :term:`metaclass` to :class:`Type`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1587); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1587); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1587); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1590)

Unknown directive type "testcode".

```
.. testcode::

    class H(Object, metaclass=Type):
        'Instance variables stored in slots'

        slot_names = ['x', 'y']

        def __init__(self, x, y):
            self.x = x
            self.y = y
```

At this point, the metaclass has loaded member objects for *x* and *y*:

```
>>> from pprint import pp
>>> pp(dict(vars(H)))
{'__module__': '__main__',
 '__doc__': 'Instance variables stored in slots',
 'slot_names': ['x', 'y'],
 '__init__': <function H.__init__ at 0x7fb5d302f9d0>,
 'x': <Member 'x' of 'H'>,
 'y': <Member 'y' of 'H'>}
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1612)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    # We test this separately because the preceding section is not
    # doctestable due to the hex memory address for the __init__ function
    >>> isinstance(vars(H)['x'], Member)
    True
    >>> isinstance(vars(H)['y'], Member)
    True
```

When instances are created, they have a `slot_values` list where the attributes are stored:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1625)

Unknown directive type "doctest".

```
.. doctest::

    >>> h = H(10, 20)
    >>> vars(h)
    {'_slotvalues': [10, 20]}
    >>> h.x = 55
    >>> vars(h)
    {'_slotvalues': [55, 20]}
```

Misspelled or unassigned attributes will raise an exception:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1636)

Unknown directive type "doctest".

```
.. doctest::

    >>> h.xz
    Traceback (most recent call last):
        ...
    AttributeError: 'H' object has no attribute 'xz'
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\(cpython-main) (Doc) (howto)descriptor.rst`, line 1643)

Unknown directive type "doctest".

```
.. doctest::
   :hide:

    # Examples for deleted attributes are not shown because this section
    # is already a bit lengthy.  We still test that code here.
    >>> del h.x
    >>> hasattr(h, 'x')
    False

    # Also test the code for uninitialized slots
    >>> class HU(Object, metaclass=Type):
    ...     slot_names = ['x', 'y']
    ...
    >>> hu = HU()
    >>> hasattr(hu, 'x')
    False
    >>> hasattr(hu, 'y')
    False
```