Object_internals

May 7, 2021

1 Object internals

Every object in python contains the attribute `__dict__`, it holds all object attributes and their values. The content of the `__dict__` can be **read**, **updated**, **insertd**, and **deleted** like any other python dictionary.

On the other hand, the direct modification of the `__dict__` attribute is frown uppon. The recommended way is to use the build-in functions: `getattr`, `hasattr`, `delattr`, and`setattr`. **This methods are only invoked when the attributes are requested using the dot operator**.

[71]:
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f'{self.__class__.__name__}(x={self.x}, y={self.y})'
```

[3]:
```
v = Vector(5, 3)
dir(v)
```

[3]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'x',
 'y']
```

```
[4]: v.__dict__
```

```
[4]: {'x': 5, 'y': 3}
```

```
[5]: type(v.__dict__)
```

```
[5]: dict
```

```
[6]: v.__dict__['x']
```

```
[6]: 5
```

```
[7]: v.__dict__['x'] = 7
```

```
[8]: v.x
```

```
[8]: 7
```

```
[9]: del v.__dict__['x']
```

```
[10]: v.x
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-185ff259d8bc> in <module>
----> 1 v.x

AttributeError: 'Vector' object has no attribute 'x'
```

```
[11]: v.__dict__['x'] = 'a'
```

```
[12]: v.x
```

```
[12]: 'a'
```

```
[13]: 'y' in v.__dict__

[13]: True

[14]: getattr(v, 'y')

[14]: 3

[15]: hasattr(v, 'x')

[15]: True

[17]: delattr(v, 'x')

[18]: hasattr(v, 'x')

[18]: False

[19]: setattr(v, 'y', 9)

[20]: v.y

[20]: 9

[27]: class GenericVector:
          def __init__(self, **kwargs):
              self.__dict__.update(**kwargs)

          def __repr__(self):
              coordinates = ', '.join(f'{k}={self.__dict__[k]}' for k in self.
       ↪__dict__)
              return f'{self.__class__.__name__}({coordinates})'

[30]: GenericVector(a = 3, b = 4, c = 5)

[30]: GenericVector(a=3, b=4, c=5)

[31]: dir(_)

[31]: ['__class__',
       '__delattr__',
       '__dict__',
       '__dir__',
       '__doc__',
       '__eq__',
       '__format__',
       '__ge__',
       '__getattribute__',
       '__gt__',
       '__hash__',
       '__init__',
       '__init_subclass__',
       '__le__',
       '__lt__',
       '__module__',
       '__ne__',
       '__new__',
       '__reduce__',
       '__reduce_ex__',
       '__repr__',
       '__setattr__',
       '__sizeof__',
       '__str__',
       '__subclasshook__',
       '__weakref__',
       'a',
       'b',
       'c']
```

[32]:
```
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)

    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
 ↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[33]:
```
GenericVector(a = 3, b = 4, c = 5)
```

[33]:
```
GenericVector(a=3, b=4, c=5)
```

[34]:
```
dir(_)
```

[34]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_a',
 '_b',
 '_c']
```

1.1 Differences between `__getattr__` and `__getattribute__`

`__getattribute__` is the method that **all** attribute/properly lookups will call. `__getattr__` is invoked **after** an attribute/property lookup has not been found by a normal lookup.

[43]:
```python
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)

    def __getattr__(self, name):
        print(f'name={name}')

    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
 ↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[44]:
```python
gv = GenericVector(a = 3, b = 4, c = 5)
gv.a
```

```
name=a
```

[42]:
```python
gv._a
```

[42]:
```
3
```

[45]:
```
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)

    def __getattr__(self, name):
        private_name = '_' + name
        return getattr(self, private_name)

    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
 ↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[46]:
```
gv = GenericVector(a = 3, b = 4, c = 5)
gv.a
```

[46]:
```
3
```

[47]:
```
gv.a = 10 # We don't want to allow this!
```

[48]:
```
gv.a
```

[48]:
```
10
```

[49]:
```
dir(gv)
```

[49]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_a',
 '_b',
 '_c',
 'a']
```

[50]:
```
# That's why we didn't want to allow that
```

[51]:
```
gv.x # What happens when we request a non existing value?
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-51-2bff05bc408e> in <module>
----> 1 gv.x # What happens when we request a non existing value?

<ipython-input-45-cbdb71fc9969> in __getattr__(self, name)
      6     def __getattr__(self, name):
      7         private_name = '_' + name
----> 8         return getattr(self, private_name)
      9
     10     def __repr__(self):

… last 1 frames repeated, from the frame below …

<ipython-input-45-cbdb71fc9969> in __getattr__(self, name)
      6     def __getattr__(self, name):
      7         private_name = '_' + name
----> 8         return getattr(self, private_name)
      9
     10     def __repr__(self):

RecursionError: maximum recursion depth exceeded while calling a Python object
```

[52]:
```
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)

    def __getattr__(self, name):
        private_name = '_' + name
        if not hasattr(self, private_name):
            raise AttributeError('{!r} object has no attribute {!r}'.
↪format(self.__class__, name))
        return getattr(self, private_name)

    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[54]:
```
gv = GenericVector(a = 3, b = 4, c = 5)
gv.x # This will still fail because the `hasattr` internally uses the␣
↪`__getattr__` of the object
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-54-31b43d46816e> in <module>
      1 gv = GenericVector(a = 3, b = 4, c = 5)
----> 2 gv.x # This will still fail because the `hasattr` internally uses the␣
 ↪`__getattr__` of the object

<ipython-input-52-7db621324872> in __getattr__(self, name)
      6     def __getattr__(self, name):
      7         private_name = '_' + name
----> 8         if not hasattr(self, private_name):
      9             raise AttributeError('{!r} object has no attribute {!r}'.
 ↪format(self.__class__, name))
     10         return getattr(self, private_name)

… last 1 frames repeated, from the frame below …

<ipython-input-52-7db621324872> in __getattr__(self, name)
      6     def __getattr__(self, name):
      7         private_name = '_' + name
----> 8         if not hasattr(self, private_name):
      9             raise AttributeError('{!r} object has no attribute {!r}'.
 ↪format(self.__class__, name))
     10         return getattr(self, private_name)

RecursionError: maximum recursion depth exceeded while calling a Python object
```

[55]:
```
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)
    def __getattr__(self, name):
        private_name = '_' + name
        if private_name not in self.__dict__:
            raise AttributeError('{!r} object has no attribute {!r}'.
↪format(self.__class__, name))
        return getattr(self, private_name)

    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[56]:
```
gv = GenericVector(a = 3, b = 4, c = 5)
gv.x
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-56-c52e45079e13> in <module>
      1 gv = GenericVector(a = 3, b = 4, c = 5)
----> 2 gv.x

<ipython-input-55-7b931a05c724> in __getattr__(self, name)
      7         private_name = '_' + name
      8         if private_name not in self.__dict__:
----> 9             raise AttributeError('{!r} object has no attribute {!r}'.
 ↪format(self.__class__, name))
     10         return getattr(self, private_name)
     11 

AttributeError: <class '__main__.GenericVector'> object has no attribute 'x'
```

[57]:
```
class GenericVector:
    def __init__(self, **kwargs):
        private_coordinates = {'_' + k:v for k, v in kwargs.items()}
        self.__dict__.update(**private_coordinates)

    def __getattr__(self, name):
        private_name = '_' + name
        # Python is more about, ask for forgiveness than ask for permission, so.
↪..
        try:
            return self.__dict__[private_name]
        except KeyError:
            raise AttributeError('{!r} object has no attribute {!r}'.
↪format(self.__class__, name))

```

```
    def __repr__(self):
        coordinates = ', '.join(f'{k[1:]}={self.__dict__[k]}' for k in self.
↪__dict__)
        return f'{self.__class__.__name__}({coordinates})'
```

[59]:
```
gv = GenericVector(a = 3, b = 4, c = 5)
gv.x
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-57-48aaba18531d> in __getattr__(self, name)
      9         try:
---> 10             return self.__dict__[private_name]
     11         except KeyError:

KeyError: '_x'

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
<ipython-input-59-c52e45079e13> in <module>
      1 gv = GenericVector(a = 3, b = 4, c = 5)
----> 2 gv.x

<ipython-input-57-48aaba18531d> in __getattr__(self, name)
     10             return self.__dict__[private_name]
     11         except KeyError:
---> 12             raise AttributeError('{!r} object has no attribute {!r}'.
 ↪format(self.__class__, name))
     13
     14     def __repr__(self):

AttributeError: <class '__main__.GenericVector'> object has no attribute 'x'
```

2 Vars (build-in function)

There is a more **pythonic** way to access the attributes of an object. It is using the build-in function `vars`.

```
vars(obj)['p'] = "Wololo"
```

Is equivalent to

```
obt.__dict__['p'] = "Wololo"
```

3 Build-in functions special cases

The build-in functions such as `repr` bypass the `getattribute` method. Therefore if you are wrapping an object and you call `wrapper.__repr__()` it will properly forward the call to the wrapped object, but if you call `repr(wrapper)` it will output the `repr` of the wrapping object. In order to avoid that, you would need to implement the wrapping object's `__repr__` method to forward the call to the wrapped object.

4 Where are method stored?

We've already seen that attributes are stored in the `__dict__` attribute of the object, however the methods are stored in another `__dict__` inside the `__class__` attribute.

The `__class__.__dict__` dictionary is not a common dictionary, it is of type `mappingproxy` and it does not support item assignment. In order to add a new entry or modify an existing entry in the map, the `setattr` build-in function must be used.

```
[73]: v = Vector(x=3, y=7)
```

```
[61]: v.__dict__
```

```
[61]: {'x': 3, 'y': 7}
```

```
[62]: v.__class__
```

```
[62]: __main__.Vector
```

```
[63]: v.__class__.__dict__
```

```
[63]: mappingproxy({'__module__': '__main__',
                    '__init__': <function __main__.Vector.__init__(self, x, y)>,
                    '__repr__': <function __main__.Vector.__repr__(self)>,
                    '__dict__': <attribute '__dict__' of 'Vector' objects>,
                    '__weakref__': <attribute '__weakref__' of 'Vector' objects>,
                    '__doc__': None})
```

```
[74]: v.__class__.__dict__['__repr__'](v)
```

```
[74]: 'Vector(x=3, y=7)'
```

```
[75]: v.__class__.__dict__['wololo'] = lambda s, x: print(f'Hello, {x}')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-d81152b7453c> in <module>
----> 1 v.__class__.__dict__['wololo'] = lambda s, x: print(f'Hello, {x}')
```

```
TypeError: 'mappingproxy' object does not support item assignment
```

[78]:
```
setattr(v.__class__, 'wololo', lambda s, x: print(f'Hello, {x}')) # s is the␣
↪self parameter in the method
```

[77]:
```
v.wololo('Oscar')
```

```
Hello, Oscar
```

[ ]:
```
```