OOP via Python Session 07

Stephen Leach, Jan 2022

Encapsulation Revisited

- The implementation of an object is limited to an easily identifiable area of the code
- Changing the implementation only affects that area of the code
- Implementation choices are not observable (except via non-functional properties such as performance and storage requirements)
- The implementation is private and cannot be corrupted from the outside

Why Encapsulation?

- To allow us to pretend we're working in the problem domain without worrying about how the implementation works.
- To contain the impact of switching the implementation.
- To reduce the impact of changing the behaviour.

Encapsulation Failure

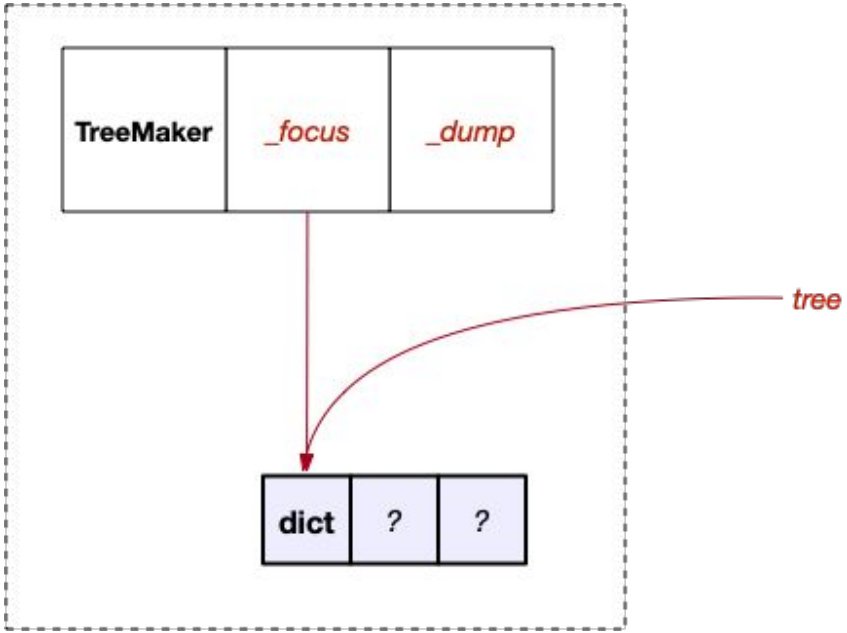
- Now we will work our way through an example (treemaker.py) where encapsulation fails …
- … and this failure will tell us something surprising deep about composing compound objects.

Example: treemaker.py

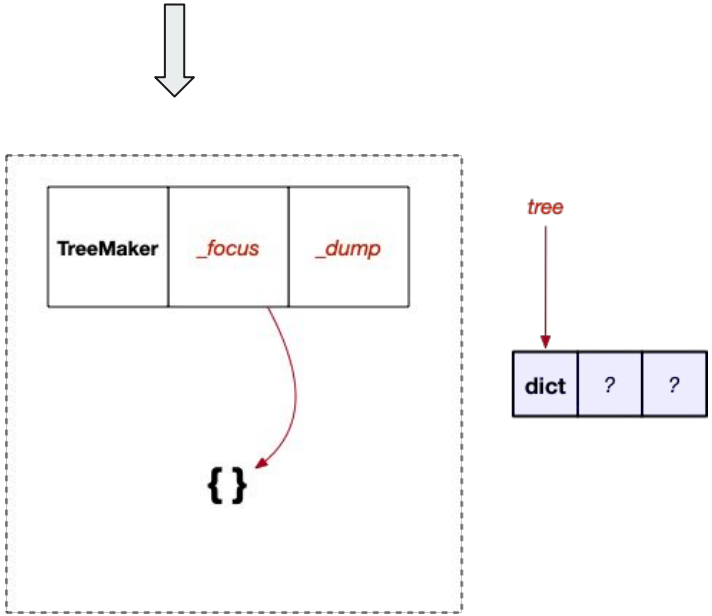
Bad Things Happen

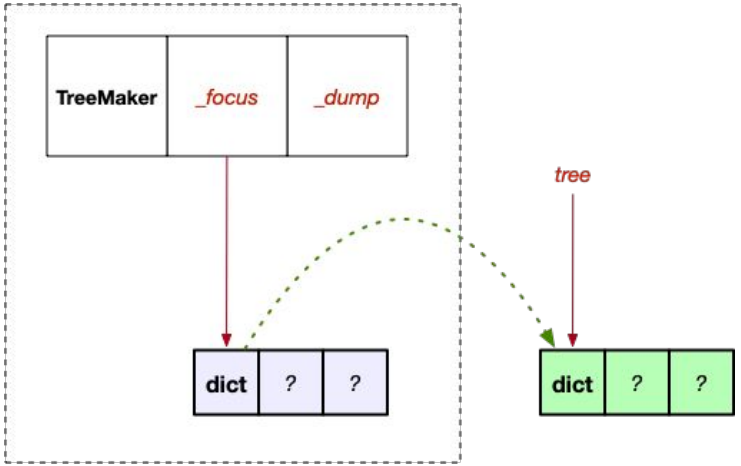
```
steve% python3 -i treemaker.py
>>> tree1 = m.make()
>>> tree2 = m.make()
>>> tree1
{'name': 'Stephen', 'age': 62, 'activities': {'coding': {'duration': 45}, 'birdwatching': {'duration': 40}}}
>>> tree2
{'name': 'Stephen', 'age': 62, 'activities': {'coding': {'duration': 45}, 'birdwatching': {'duration': 40}}}
>>> tree1.clear()
>>> tree2
{}
>>>
```

Escape



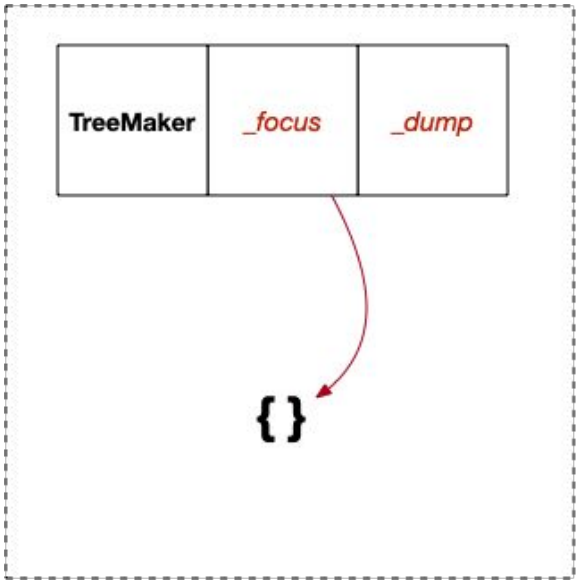
Release or Copy

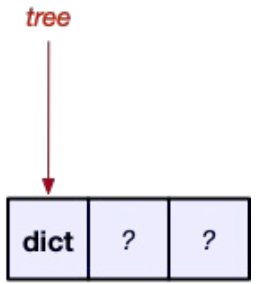




Release

```
class TreeMaker:
    ...
    def make( self ) -> Dict[str, Any]:
        result = self._focus
        self._focus = {}
        return result
```





Copy

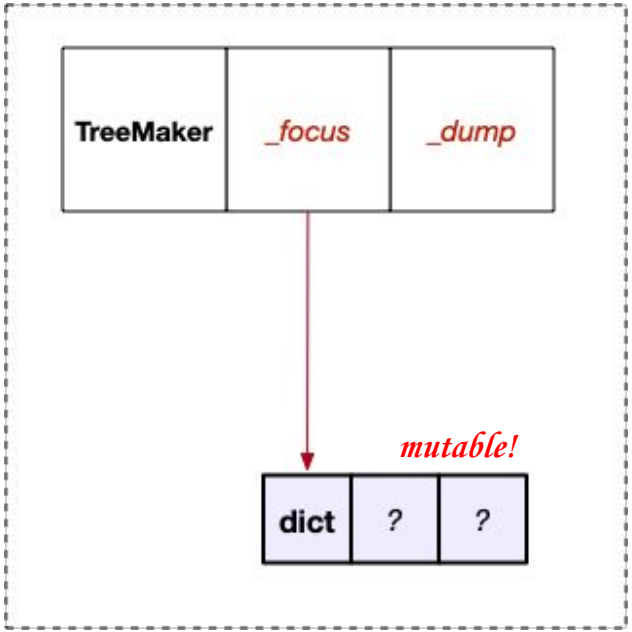
```
from copy import deepcopy

class TreeMaker:
    ...
    def make( self ) -> Dict[str, Any]:
        return deepcopy( self._focus )
```

Definition of "Part-Of"

- An object X is *part-of* a compound object C if:
- C must control all changes to X and
- All the parts-of X, recursively.
- X should never be passed outside of C's control

} Only relevant if C is mutable

"Part-Of"



So what can you return?

- Recursively immutable data (e.g. tuples of numbers and strings)
- Objects that you reference but don't care about how or when they are updated (e.g. members of arrays)
- Copies (Example: Copy)
- Parts for which you sever all connections and dependencies (Example: Release)

Summary

- Methods of compound objects must be carefully written not to accidentally share their inner, mutable parts.
- The notion of sensitivity to independent update is a litmus test for being a part-of a compound object rather than a reference-from a compound object.
- The concept of mutation and objects are profoundly interlinked in today's object oriented languages. Store sharing is an integral part of the design of classes.