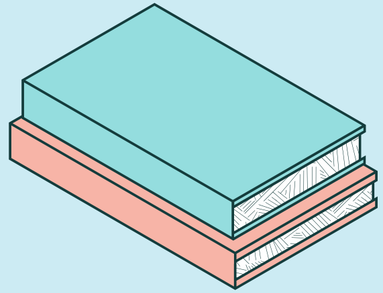
__slots__

=

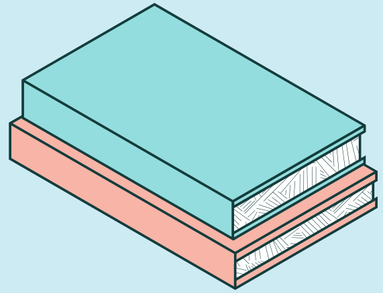
dict -> fixed-size array

recap



- if we know in advance what data attributes a class should support, using slots may offer memory and speed improvements
- slots are defined as a class attribute
- behind the scenes, python switches from a dict to a fixed-size array, mapping each attribute to a specific index

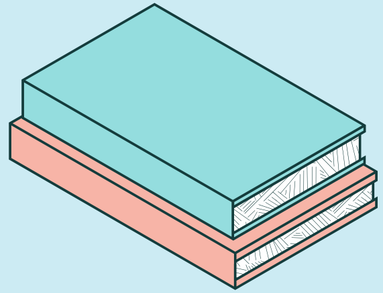
recap



- __slots__ creates a descriptor for each mapped attribute, thereby overriding the default __getattribute__ beahvior

- as a result, just like properties, slotted attribtues reside in the class' mappingproxy, rather than with the instances

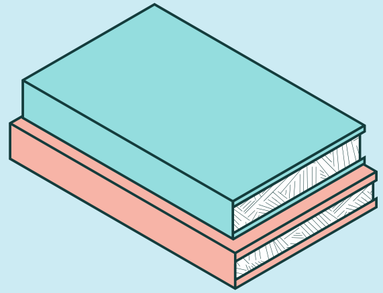
recap



- slots in the parent class will be used for the child's attribute lookup, i.e. they're available
- the child class by default also retains its instance `__dict__`, however
- if both the parent and child classes are slotted, the child loses its `__dict__`
- if the parent class is not slotted, but the child is, the child retains its instance `__dict__`

Something To Avoid

recap



- it is possible to define slotted classes whose instances also maintain their __dict__ without using inheritance
- we do that by adding __dict__ as one of the slotted attributes
- this comes with a performance overhead and mostly defeats the purpose of using slots in the first place

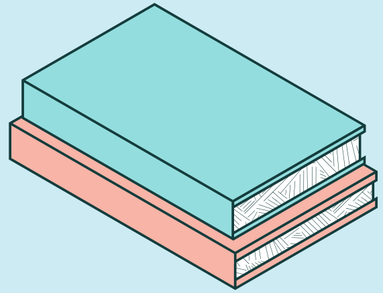
ALWAYS SLOT?

NO

don't use slots for their side effects

but if you do, beware of their side effects

recap



- slots should be used for memory and performance optimization when we have a specific need to optimize, typically indicated by profiling
- when using slots we should be mindful of the side effects, e.g. instance `__dict__`, inheritance rules, etc
- we should not use slots for the side effects

Skill Challenge #9

#slots

Requirements

> Define a new type called Point3D, that encapsulates 3 attributes: x, y, z

> The class should be slotted to those 3 attributes only

> Then define two subclasses of Point3D:

- ColoredPoint - also slotted, but in addition supports a color attribute, defaulting to "black"
- ShapedPoint - also slotted, but in addition supports a shape attribute, defaulting to "sphere"

> All instances of the above 3 classes should produce a representation that makees it easy to recreate the instance

> As a bonus challenge, consider implementing a single __repr__ in Point3D that flexibly returns all the applicable attributes depending on an instance's type, i.e. x, y, z for Point3D, x, y, z, color for ColoredPoint, and so on