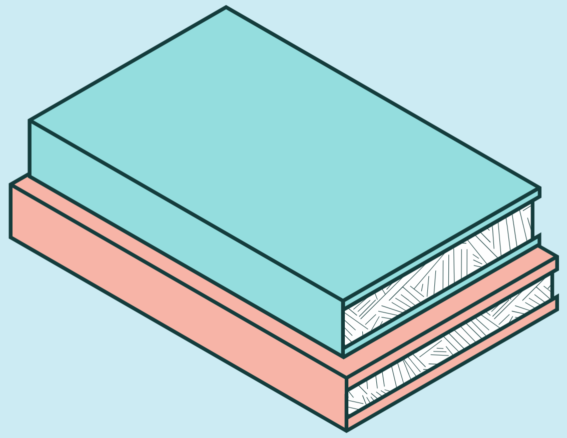
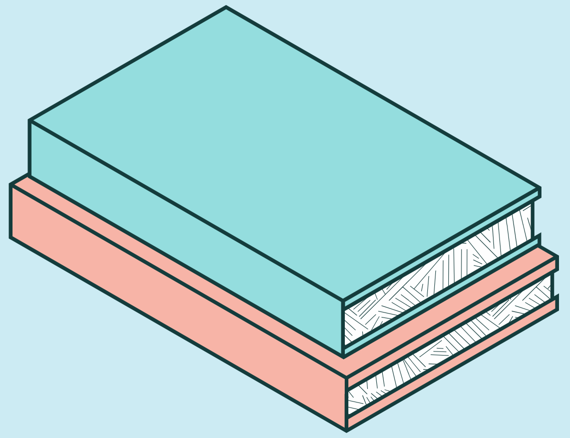
Attribute Lookup Chain Review



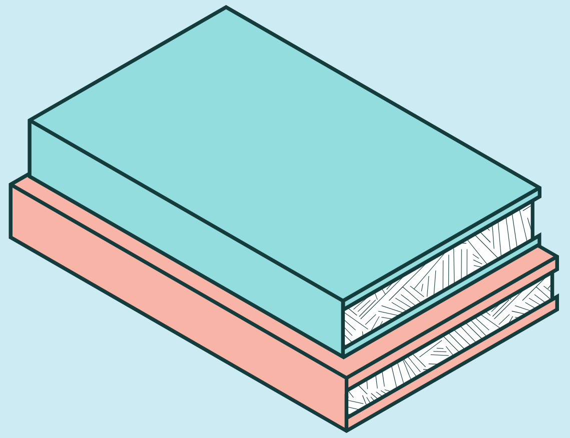
1 look in the instance (i.e. object) __dict__ for a key with the attribute's name

Attribute Lookup Chain Review



1. look in the instance (i.e. object) __dict__ for a key with the attribute's name
2. look in the instance's type (i.e. class) __dict__ for a key with the attribute's name

Attribute Lookup Chain Review



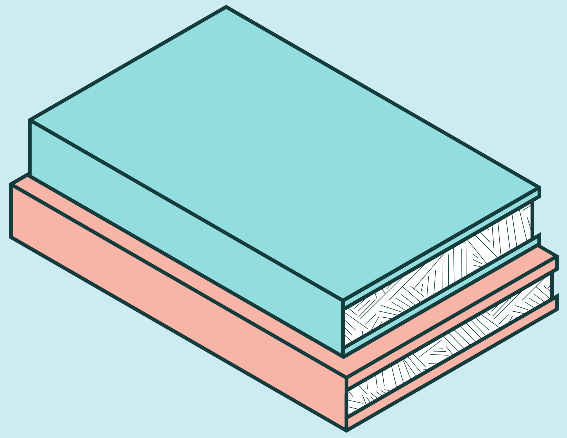
1 look in the instance (i.e. object) __dict__ for a key with the attribute's name

2 look in the instance's type (i.e. class) __dict__ for a key with the attribute's name

3 look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name

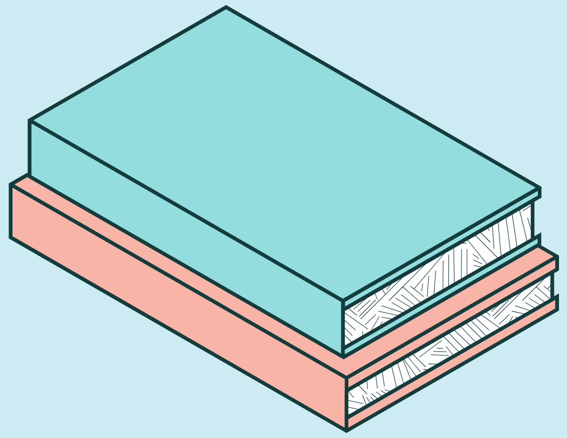
4 if not found, repeat for each parent type in mro order

Attribute Lookup Chain Review



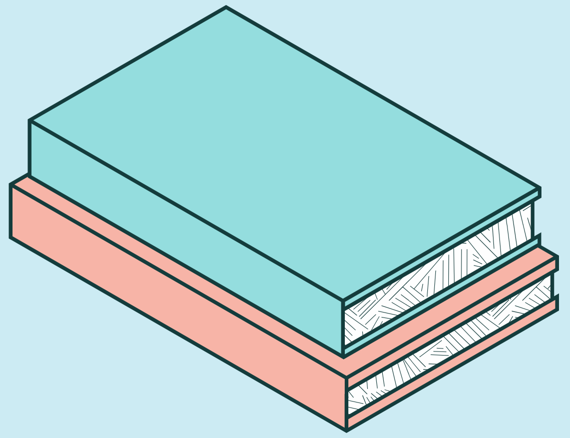
1. look in the instance (i.e. object) __dict__ for a key with the attribute's name
2. look in the instance's type (i.e. class) __dict__ for a key with the attribute's name
3. look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name
4. if not found, repeat for each parent type in mro order
5. if not found, raise AttributeError

Attribute Lookup Chain Review



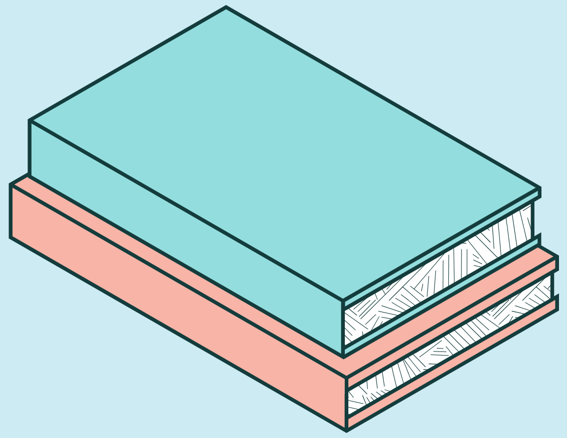
1. look in the instance (i.e. object) __dict__ for a key with the attribute's name
2. look in the instance's type (i.e. class) __dict__ for a key with the attribute's name
3. look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name
4. if not found, repeat for each parent type in mro order
5. if not found, raise AttributeError

The Descriptor Protocol



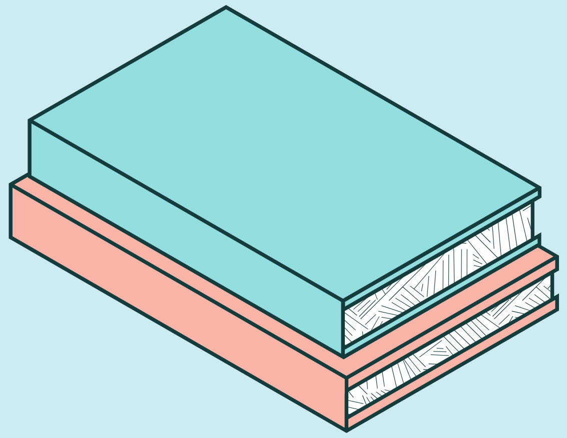
- the descriptor protocol consists of dunder get/set/delete
- any object that implements a combination of these methods is a descriptor

Using A Descriptor



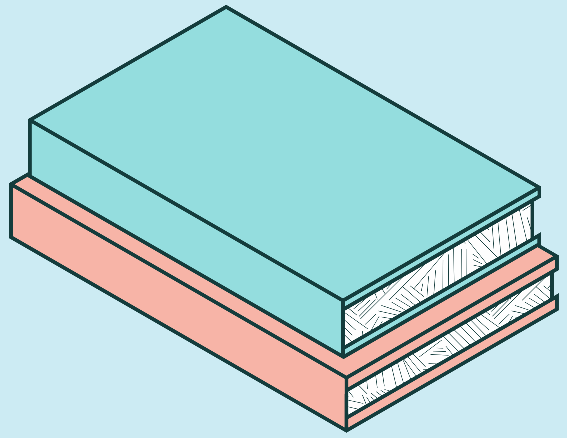
- descriptors are objects that define some or all of the descriptor protocol
- when pointed to attributes in other objects, they take on special behaviour and allow us to customize attribute access for that attribute alone
- the resulting behaviour takes precedence over all attribute lookup rules for that attribute alone (binding behaviour)
- descriptors are only instantiated at the class level; never put them in __init__ or other methods

Attribute Lookup Chain Review



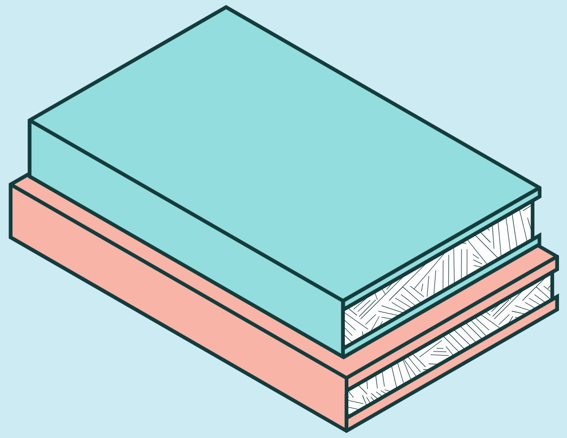
1. call the __get__ of the descriptor having the same name as the attribute
2. look in the instance (i.e. object) __dict__ for a key with the attribute's name
3. look in the instance's type (i.e. class) __dict__ for a key with the attribute's name
4. look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name
5. if not found, repeat for each parent type in mro order
6. if not found, raise AttributeError

Descriptor Storage



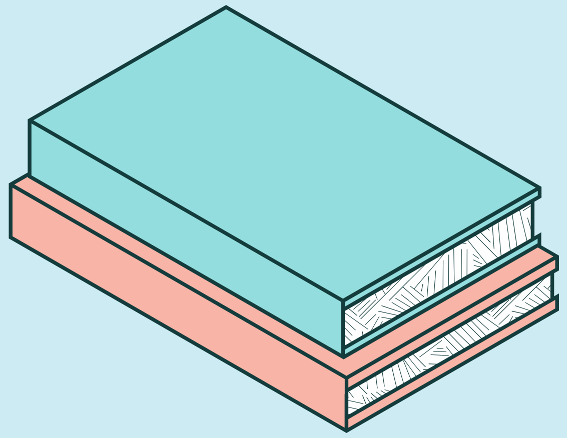
- a more meaningful descriptor needs to allocate separate storage across instances to allow them to store and retrieve different values
- when using the descriptor itself for storage, we need to be careful about avoiding memory leaks
- using instance memory addresses as keys or weakkey data structures may help but they do comes with their own caveats

Even Better: Instance Storage



- using the instance __dict__ for storing descriptor field values is neat both conceptually (aligns with the instances lifecycle) and practically (avoids the need to exhaustively consider what could go wrong with memory management)

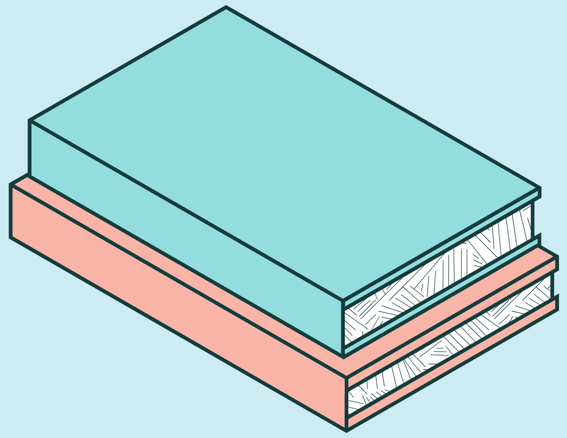
Using __set_name__



- reusing a descriptor within the same class requires the ability to separate its storage within the instance dictionary
- __set_name__ (python 3.6+) offers the pythonic solution to this problem in modern codebases
- __set_name__ is defined in the descriptor class and called each time the descriptor is instantiated
- the second parameter (name) captures the name of the class attribtue the instance of the descriptor is assigned to

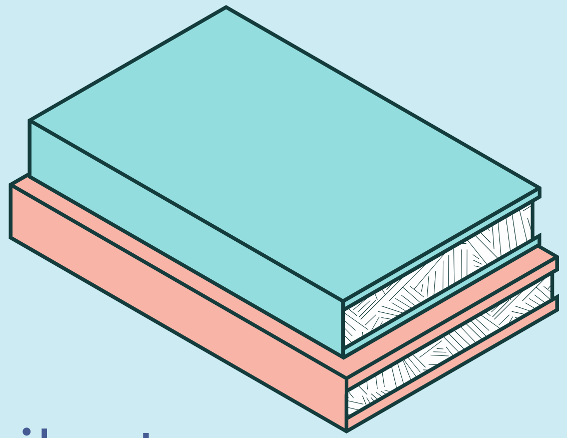
Tying Up Loose Ends

recap



- self in the descriptor class refers to the instance of the descriptor
- owner refers to the class from where the descriptor is invoked (and set to a class variable)
- instance refers to the instance of the owning class
- when the descriptor attribute is accessed from the class directly, the instance argument is set to None; it may be a good idea to return the instance of the descriptor in such cases

Attribute Lookup Chain Review



1 call the __get__ of the descriptor having the same name as the attribute

2 look in the instance (i.e. object) __dict__ for a key with the attribute's name

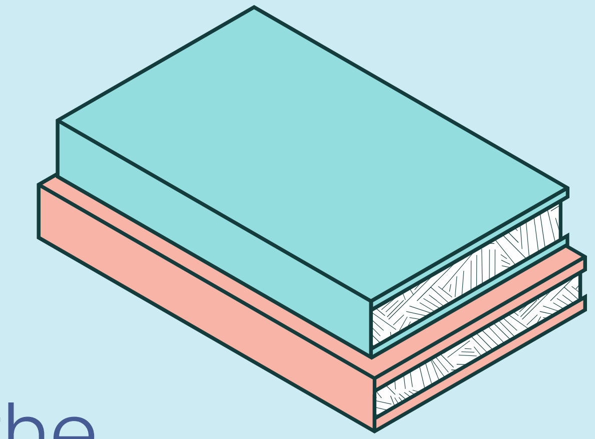
3 look in the instance's type (i.e. class) __dict__ for a key with the attribute's name

4 look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name

5 if not found, repeat for each parent type in mro order

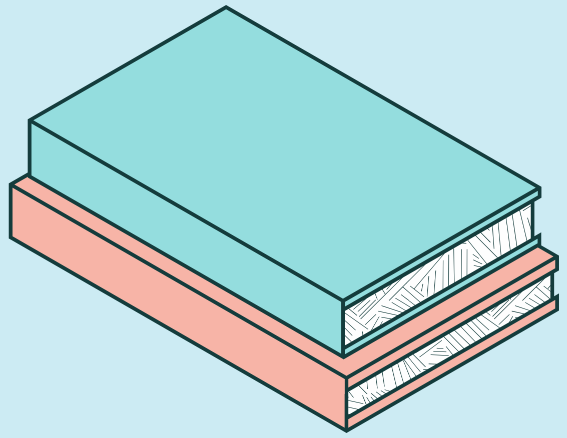
6 if not found, raise AttributeError

Attribute Lookup Chain Review



1. call the __get__ of the data descriptor having the same name as the attribute
2. look in the instance (i.e. object) __dict__ for a key with the attribute's name
3. call the __get__ of the non-data descriptor having the same name as the attribute
4. look in the instance's type (i.e. class) __dict__ for a key with the attribute's name
5. look in the instance's parent type (i.e. parent class) __dict__ for a key with the attribute's name
6. if not found, repeat for each parent type in mro order
7. if not found, raise AttributeError

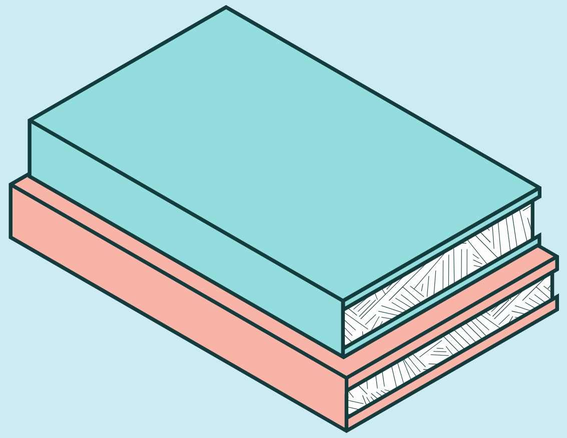
recap



- if a descriptor implements only __get__ it is known as a non-data descriptor
- if __set__ and/or __delete__ are also implemented it becomes a data-descriptor
- data (overriding) descriptors reign supreme in the attribute lookup chain for a given attribute name
- non-data descriptors (non-overriding) are secondary to instance dictionaries

Aren't Properties Just Better?

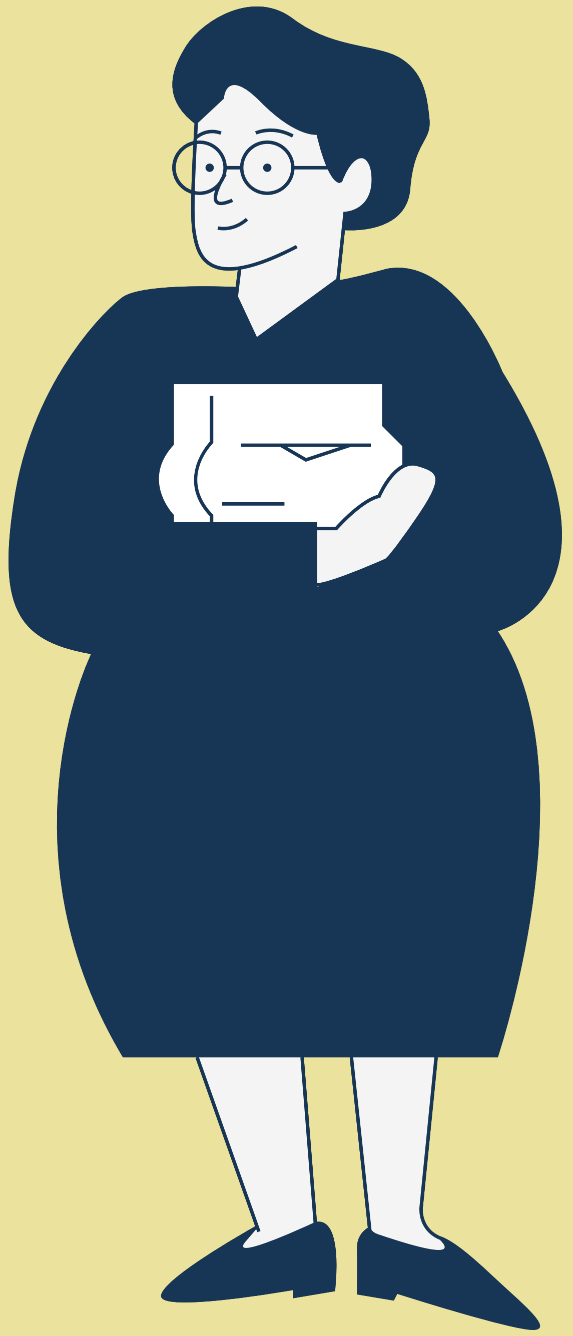
recap



- properties provide syntactic sugar over the descriptor protocol
- descriptors are significantly more reusable
- "properties are better than descriptors" inherently makes no sense and indicates the speaker does not understand properties or descriptors in enough depth
- "properties are more appropriate than descriptors" should be taken with a heavy dose of context awareness, e.g. in a simple class used just once, maybe, but in a large project with multiple dependencies, most definitely not

Skill Challenge #11

#descriptors



andybek.com

Requirements

> Define a new type called StudentProfile, whose instances should encapsulate the following attributes:

- the student's name
- the student's GRE score (integers between 130 and 340), and
- the student's SAT score (integers between 400 and 1600)

> StudentProfile instances should have a customized representation

> The score fields should be validated for the correct type and value, i.e. they should be ints that fall in the expected range

> If a score field is not specified at instantiation, it must default to the minimum of its respective valid range

> Use descriptors with instance-specific storage to implement these validations

> As an extra challenge, try to maximize code reuse by writing a single general descriptor