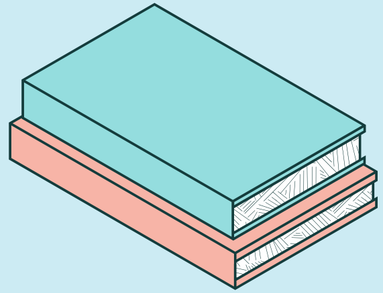
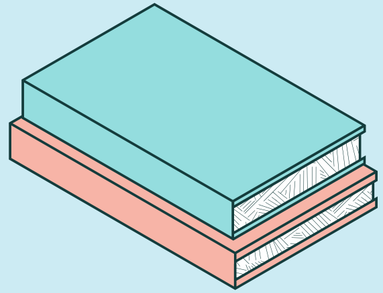
recap



- often in programming we need instances whose sole purpose is to encapsulate data
- the traditional data structures (list, tuple, dict) are less than ideal to use, maintain, and document in larger projects
- defining a new class (or type) is the OOP way to address this, but that too often comes with some overhead
- dataclasses will help us easily elminiate that overhead

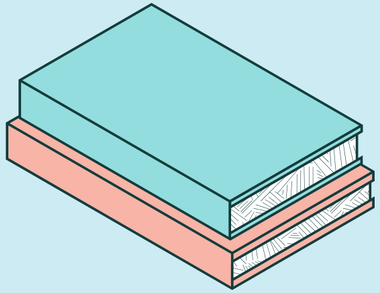
recap



- a great choice for encapsulating data with named attribtues is the namedtuple container class

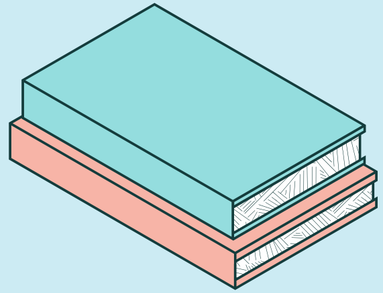
- it lets us create tuple-like objects, with convenient defaults, nice representations, and easy access by index or attribute name

recap



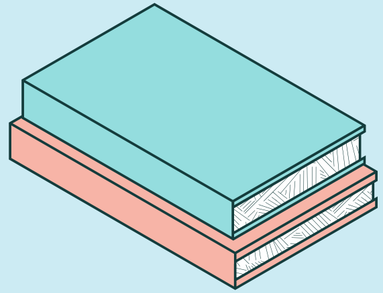
- dataclasses make it easy to create classes fine tuned to storing state (i.e. data) with little code
- though syntactically similar to typed namedtuples, dataclasses are most similar to regular classes
- dataclasses get rid of the boilerplate and provide additional functionality for free

recap



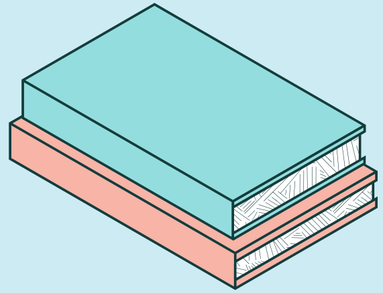
- when defining dataclass fields, adding type hints is recommended
- this helps us clarify our intent to code editors and readers of our code, and in turn provides better code hints and intellisense since most modern editors ship with some sort of type checker built in
- if you strongly feel about preserving the core dynamically typed style of python, you could always use Any, object, or simply '...' in your definitions

recap



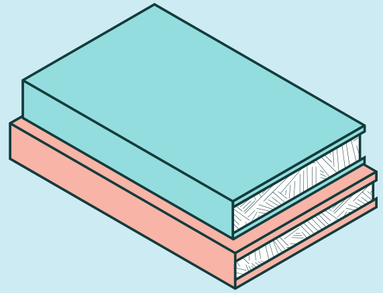
- we could customize our fields by using the various parameters supported by the fields method of the dataclass module
- these include defaults, treatments in repr, init, hashing, and comparison
- if this built-in convenience is not enough for our needs, we could always turn it off and customize our dataclass as we would a regular python class

recap



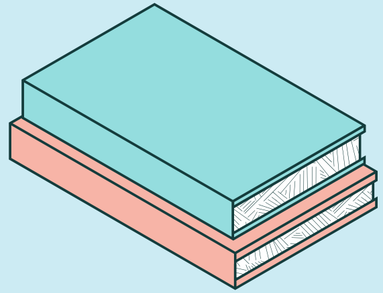
- dataclasses could be made immutable by toggling the frozen parameter of the dataclass decorating function
- attempting to change a value on an instance of a frozen dataset will raise a FrozenInstanceError
- freezing a dataclass makes it hashable, expanding the range of use cases for all instances, e.g. as keys in dicts
- what fields are included in the hash could be customized using the hash parameter of the respective field function

recap



- dataclasses could be extended through inheritance as we would normally expect
- the final list of fields is the combination of all fields defined in the inhertiance tree
- this means that if a field is defined in both parent and child, the latter will shadow the former
- defaulted fields may only be supported in inheritance trees that have only defaulted fields preceding them

recap



- dataclasses are often compared, with good reason, to namedtuples
- despite the similarity in use cases (and even syntax) the two are fundamentally different under the hood
- namedtuples are essentially tuples, always immutable, hashable, with type-agnostic comparison and equality going solely by attribute value
- dataclasses are much more extensible, somewhat more performant, and loaded with functionality often needed by classes geared toward data storage

Skill Challenge #10

#dataclasses

Requirements

> Define 3 new types mimicking a financial markets equity portfolio hierarchy:

a. Stock

- has 4 attributes: ticker, price, dividend, dividend_frequency, e.g. MSFT, 360, 0.62, 4
- dividend defaults to 0, whereas dividend_frequency to 4, i.e. quarterly
- the type should expose annual_dividend as a calculated/managed attribute
- the type should be immutable

b. Position

- has 2 attributes: stock and shares, e.g. MSFT, 100
- compares with other instances of Position based on dollar value alone, e.g. a position of 100 shares of MSFT
- should compare equal to a position of 100 shares of LMT, if both MSFT and LMT trade at the same price

c. *Portfolio*

- has 1 attribute: holdings, which is a list of positions, e.g. [Position(MSFT...), Position(LMT...), ...]
- exposes two managed attributes (i.e. dot access notation): value and portfolio_yield
- value is the total dollar value (shares * stock.price) of all positions in the portfolio
- portfolio_yield is simply total annual portfolio dividends divided by portfolio value

> Try to define all of the above types as dataclasses, with applicable customization and overrides.