RECAP

- type(inst) gives us the type of an instance, which is equivalent to accessing the __class__ from the object
- isinstance(inst, Cls) returns True if inst is an instance of Cls or any of its subclasses, else False
- issubclass(Cls0, Cls1) returns True if Cls0 is a subclass of Cls1, else False

RECAP

- ABCs give us a mechanism through which we could define interfaces, i.e. a set of behaviour and attributes
- any concrete types/classes that inherit the ABC need to implement all that's specified in the interface to be considered valid
- ABCs and the concrete classes deriving from them play well with isinstance() and issubclass() built-ins
- the abc module in python provides us the tools and types to define our own custom ABCs

Excellent Use Cases for ABCs

1

type checking against existing built-in ABCs

2

formalize and document external API

3

enable enhanced static type checking, e.g. through mypy

RECAP

- in most cases, we'll use built-in ABCs that already ship with python
- this is by far the most practical use case for them, at least in my experience
- while defining our own ABCs is fun, it typically ends up being too restrictive, except in very specific contexts as we described earlier

RECAP

- the single most prevalent use case for ABCs is type checking against the Python Standard Library ABCs
- in the Standard Library, ABCs feature most prominently in the collections module, although there's others in numbers and io
- in all these cases the ABC encapsulates a set of behaviours and attributes that concrete implementing classes of the ABC should support
- this offers us a more expressive and syntactically clean alternative to determining that type of the object we're working with

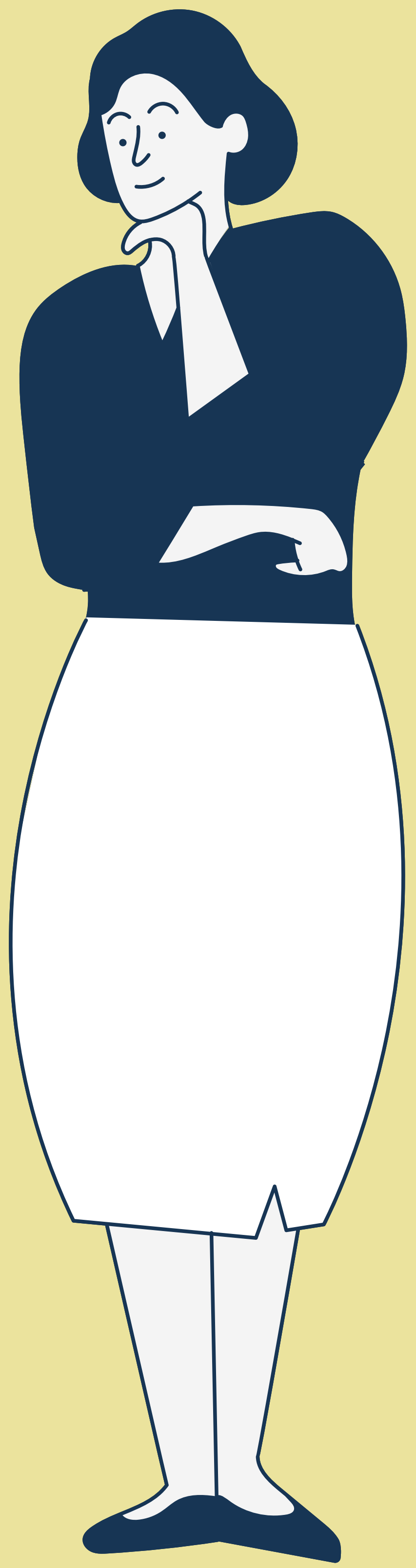
A Quick Look Under The Hood

RECAP

- ABCs work by overriding the behavior of isinstance() and issubclass()
- mechanically that involves hooking into the __instancecheck__ and __subclasscheck__ dunders
- being consistent with the interface laid out in the ABC does not offer any guarantees around the concrete implementation, rather just the presence of the method or attribute itself in the implementing class

Skill Challenge #15

#ABCs



Requirements

> Define two new types: JobApplicant and JobApplicantPool

> JobApplicant will encapsulate the following attributes:

- applicant_id, int
- years_experience, int
- is_recommended, bool
- first_interview_score, float
- second_interview_score, float

> JobApplicantPool will be a sequence container of JobApplicants. As a starting point, inherit from collections.abc.Sequence and try to create an instance. Then, implement the required functionality as indicated in the resulting exceptions until none remain.

> Beyond evaluating as an instance and subclass of the Sequence ABC, the JobApplicantPool type should also reflect through its repr the entire applicant pool in descending order of applicant score

> Applicants should be scored (rounded to 2 decimals) according to the following formula:

years_experience / 2 **+**
1 if the applicant is_recommended, else 0 **+**
applicant.first_interview_score / 2 **+**
applicant.second_interview_score