recap

1. a class is a blueprint that represents a type of object
2. even the simplest class has state and behaviour
3. calling the class creates instances (or objects) of that class

```
obj = Class()
```

recap

1. we traditionally define class state in the class body

```
...but we could also redefine (or
add to it) outside
```

2. class state is stored in a mappingproxy object and retrieved using __dict__

3. class state is shared and accessible by all instances of that class

recap

1. we add behaviour to our classes by defining functions
2. these functions are special in that they always have at least one parameter
3. that parameter is by convention called self
4. when functions are defined within the body of a class, they become bound (or attached) to instances of that class

```
...and they're then called methods
```

recap

1. attributes are simply variables associated with objects
2. instance attributes could be set before or after the instance object is returned
3. that's said, it's a best practice to set them in __init__, a special method which exists specifically for this purpose

recap

1. in addition to instance methods, python has static and class methods
2. in class methods, the class is implicitly passed as the first argument, whereas in static methods, neither the instance object nor the class is passed
3. static methods are like regular functions that are grouped with the class namespace because they're somehow conceptually related to the class

recap

1. self is always the first argument passed to instance methods
2. it represents the instance object bound to the method
3. it is called self by (good) convention only, and self is neither a reserved keyword nor does it have any special meaning in python

Skill Challenge #1

#classes

Requirements

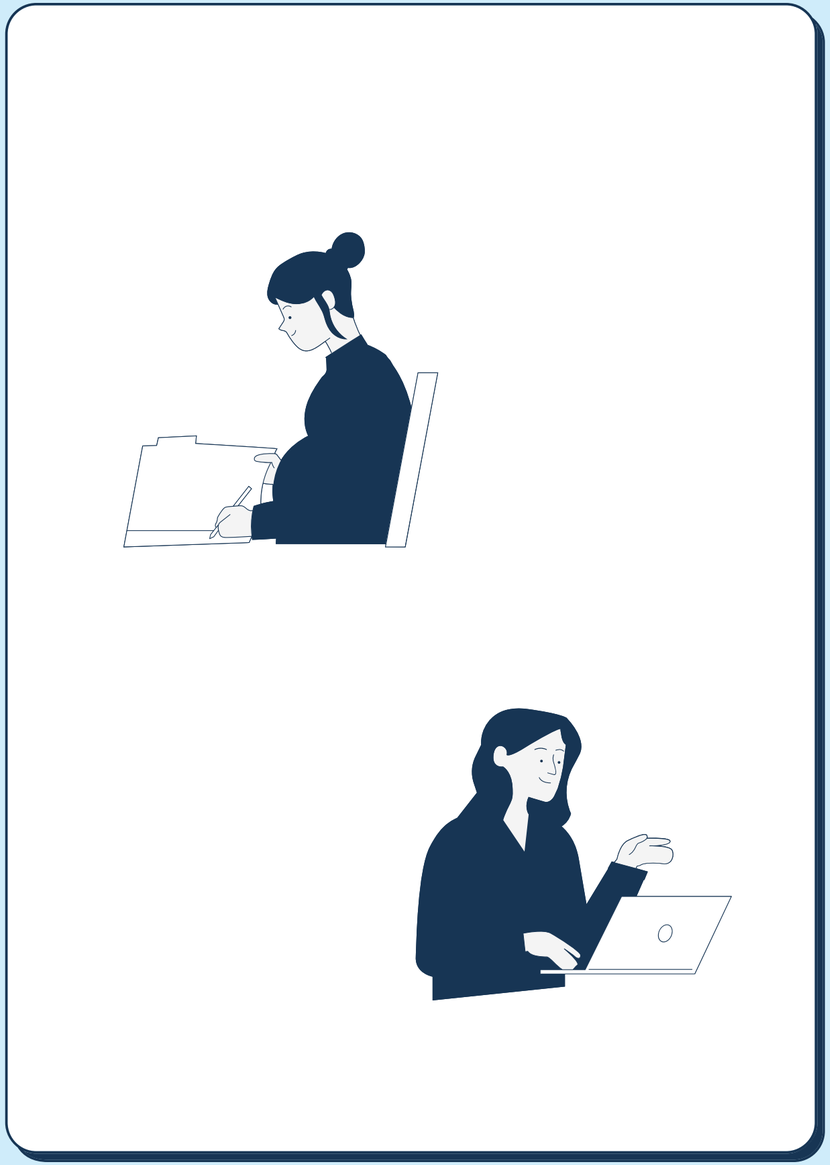
> Define a Student class with 2 instance attributes (name, age) and 1 class attribute (educational_platform), defaulting to udemy.

> Make it so that instances of Student could be created by simply specifying the name. Hint: set the default age to a number

> Define a greet() method which alternates between various name greetings. When invoked, the method should randomly select a greeting and interpolate in the name of the student

- Hi, I'm...
- Hey there, my name is...
- Hi. Oh, my name is...

> Starting with a list of several student names, create student instances from each, and have each student introduce themselves



recap

1. in addition to instance methods, python has static and class methods
2. in class methods, the class is implicitly passed as the first argument, whereas in static methods, neither the instance object nor the class is passed
3. static methods are like regular functions that are grouped with the class namespace because they're somehow conceptually related to the class

recap

1. all instance attributes are stored in an instance-specific mapping object
2. for instances, that mapping object is a plain python dictionary
3. it is accessed using instance.__dict__ syntax

recap

1. just like instances, classes have their own attribute namespace
2. just as for instances, it too is accessed using __dict__
3. unlike instances, however, the class __dict__ is a mappingproxy, which is a more restricted type of read-only dictionary where all the keys are strings
4. the class __dict__ contains all the instance, class, and static methods we define, in addition to class variables
5. it also contains some descriptors and other class dunders

recap

1. in python, attributes are by default public, so there's no access control in the classical sense
2. you shouldn't define getters and setters, unless you have a good reason to do so
3. this is not a bug or a missing feature. It's a language design decision that aligns with the Uniform Access Principle and it's something that gives python a much more expressive syntax
4. there's really no loss of functionality either, because specific logic around getting, setting, or deleting attributes could be implemented through properties while maintaining exactly the same syntax

recap

1. python docstrings are strings written as the first statement of a class, function or module
2. the python compiler binds them to the __doc__ attribute of the object, and they are also reflected in help()
3. there are several alternative styles for writing docstrings in python
4. docstrings are quite different from comments

Skill Challenge #2

#classes

Requirements

> Define a Password class that supports two instance arugments: strength and length

> The class should generate a random password having the following characteristics, depending on strength

- low: include a mix of 8 lowercase and uppercase letters only
- mid: a mix of 12 lowercase and uppercase letters and numbers
- high: a mix of 16 lowercase and uppercase letters and numbers and punctuation signs

> If the user specifies a length at instance creation, the user-specified length overrides the defaults above

> If the user does not specify a strength or a length, assume "mid" strength

> Finally, the class should also implement a method called show_input_universe() which is not specific to the instance. The method should return a dict of lists exposing the pools of characters from where the sampling is done, e.g. {"letters": ["a", "b"...], "numbers": [0, 1, 2...], "punctuation": ["!", "?"...]}.