Inheritance

Speaker notes

No notes on this slide.

Class Inheritance (Refresher)

Given a base class…

```
class Regressor:

    def __init__(self, penalty=None):
        self.penalty = penalty

    def predict(self, x):
        return np.dot(x, self.beta)
```

One can inherit and override

```
class Classifier(Regressor):
    def __init__(self, threshold=0, **kwargs):
        super().__init__(**kwargs)
        self.threshold = threshold

    def predict(self, x):
        # Go up the chain
        return super().predict(x) >= self.threshold
```


Speaker notes

Here, a `Classifier` reuses the same prediction logic as a `Regressor`, so it starts by using the same exact classification. It then binarizes the output (it is a classifier!) by evaluating the regression against a threshold.

It can use `super` in the `__init__` method as well to capture similar sets of parameters.

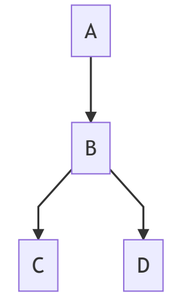
Class Inheritance (Refresher)

```
class A():
    _X = 5
    def f(self):
        return self._X
```

```
class B(A):
    _X = 7

class C(B):
    def f(self):
        return super().f() + 1

class D(B):
    def f(self):
        return 2 * super().f()
```

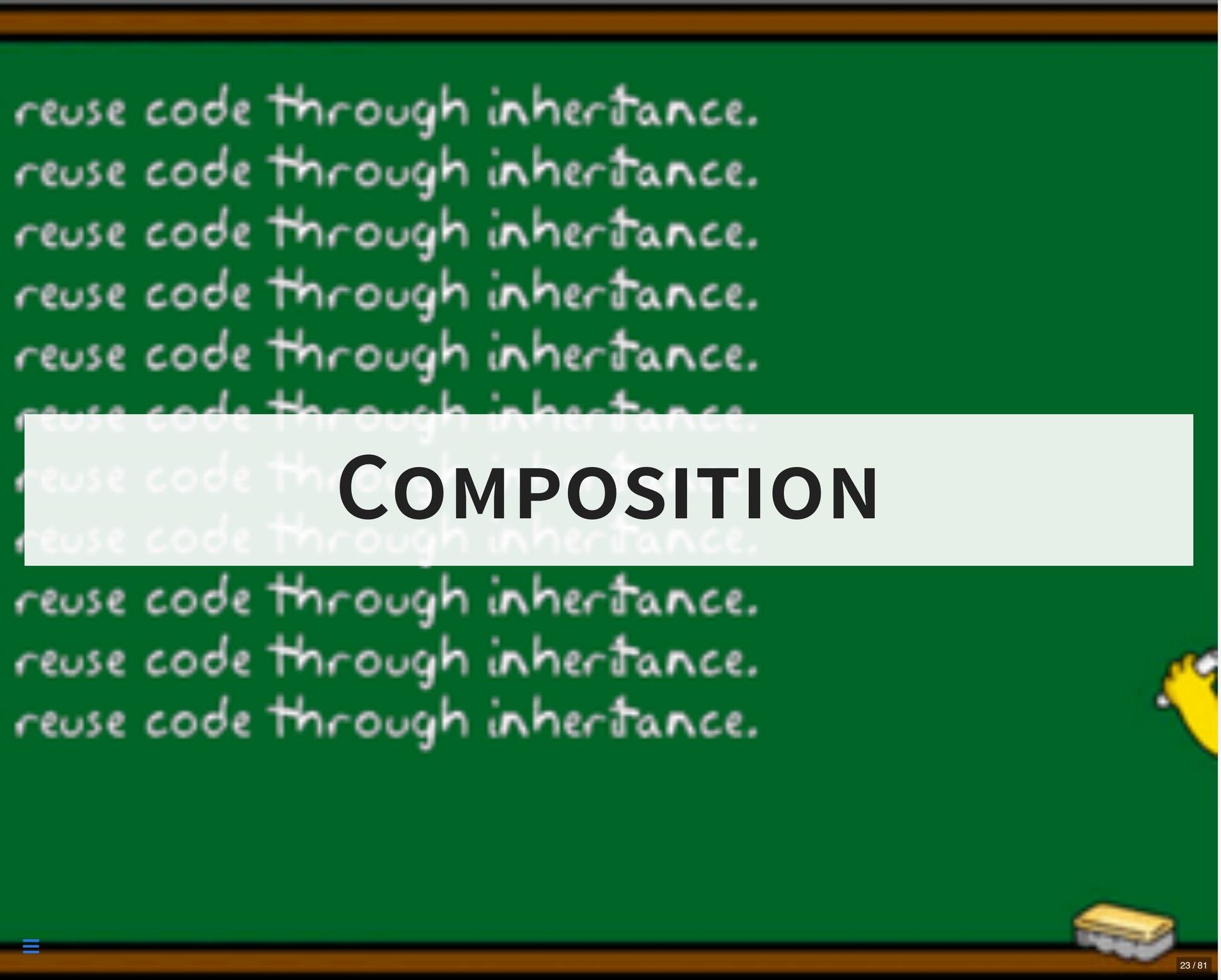


Note the use of `super()` or `super(cls, instance)` to call up the inheritance tree

Speaker notes

`super()` is shorthand for `super(self.__class__, self)` when called within an instance method (or `super(cls, cls)` in a class method).

Here we have a little family of classes - any child class can effectively walk up the tree as it wishes.



Speaker notes

No notes on this slide.

Composition

… an orthogonal organization

Inheritance

Mary ***is a*** Data Scientist

```
mary.compute()
```

Composition

Mary ***has a*** MacBookPro

```
mary.computer.compute()
```

Give classes their sophistication by giving them rich properties, and delegating responsibilities to them

Speaker notes

No notes on this slide.

Composition

As instance properties:

```
class Student:
    def __init__(self, computer=None):
        self.computer = computer or Laptop()
    def compute(self, *args):
        return self.computer.compute(*args)
```

As *declarative classes*:

```
class Student:
    COMPUTER_CLASS = Laptop
    def __init__(self):
        self.computer = self.COMPUTER_CLASS()

class MacStudent(Student):
    COMPUTER_CLASS = MacBookPro
```

Speaker notes

No notes on this slide.