https://youtu.be/u15HcpiBeRc

Polymorphism: Overriding

"Overriding" a function occurs when:

1. A base class declares a `virtual` function.
2. A derived class *overrides* that virtual function by defining its own implementation with an identical function signature (i.e. the same function name and argument types).

```
class Animal {
public:
  virtual std::string Talk() const = 0;
};

class Cat {
public:
  std::string Talk() const { return std::string("Meow"); }
};
```

In this example, `Animal` exposes a `virtual` function: `Talk()`, but does not define it. Because `Animal::Talk()` is undefined, it is called a ***pure virtual function***, as opposed to an ordinary (impure? 😉) ***virtual function***.

Furthermore, because `Animal` contains a pure virtual function, the user cannot instantiate an object of type `Animal`. This makes `Animal` an ***abstract class***.

`Cat`, however, inherits from `Animal` and overrides `Animal::Talk()` with `Cat::Talk()`, which is defined. Therefore, it is possible to instantiate an object of type `Cat`.

Instructions

1. Create a class `Dog` to inherit from `Animal`.
2. Define `Dog::Talk()` to override the virtual function `Animal::Talk()`.
3. Confirm that the tests pass.

Saving Graffiti Recording. Please wait...

↑ Menu Expand

Function Hiding

Function hiding is **closely related, but distinct from**, overriding.

A derived class hides a base class function, as opposed to overriding it, if the base class function is not specified to be `virtual`.

```
class Cat { // Here, Cat does not derive from a base class
public:
  std::string Talk() const { return std::string("Meow"); }
};

class Lion : public Cat {
public:
  std::string Talk() const { return std::string("Roar"); }
};
```

In this example, `Cat` is the base class and `Lion` is the derived class. Both `Cat` and `Lion` have `Talk()` member functions.

When an object of type `Lion` calls `Talk()`, the object will run `Lion::Talk()`, not `Cat::Talk()`.

In this situation, `Lion::Talk()` is *hiding* `Cat::Talk()`. If `Cat::Talk()` were `virtual`, then `Lion::Talk()` would *override* `Cat::Talk()`, instead of *hiding* it. *Overriding* requires a `virtual` function in the base class.

The distinction between *overriding* and *hiding* is subtle and not terribly significant, but in certain situations *hiding* **can lead to bizarre errors**, particularly when the two functions have slightly different function signatures.