marp
true

Inheritance

- 🧱 Re-use common patterns and standards easily
- 🏗️ Child contracts inherit functions (except `private`) and state variables
- 🧰 Child contracts also have access to enum, struct, error and event definitions
- 🔩 Build your functionality on top or override

onlyOwner

🧱 you may start see this pattern everywhere:

```
contract Example {
  address owner = msg.sender;
  uint importantVar;

  function privilegedMethod(uint x) external onlyOwner {
    importantVar = x;
  }

  error NotTheOwner();
  modifier onlyOwner {
    if(msg.sender != owner) {
      revert NotTheOwner();
    }
```

```
      _;
    }
  }
```

Modular!

```
  contract Ownable {
    address owner = msg.sender;
    error NotTheOwner();
    modifier onlyOwner {
      if(msg.sender != owner) {
        revert NotTheOwner();
      }
      _;
    }
  }

  contract Example is Ownable {
    function privilegedMethod(uint x) external onlyOwner {
      importantVar = x;
    }
  }
```

Import Statements

Think of them like its copy/pasting the code into your file

```
  import "./Ownable.sol";

  contract Example is Ownable {
    function privilegedMethod(uint x) external onlyOwner {
      importantVar = x;
    }
  }
```

Inherit Functions

Functions will be inherited as well, like `transferOwner` :

```
  contract Ownable {
    address owner = msg.sender;
    // virtual allows this method to be overriden
    function transferOwner(address newOwner) public virtual onlyOwner {
      owner = newOwner;
    }

    error NotTheOwner();
    modifier onlyOwner {
      if(msg.sender != owner) {
        revert NotTheOwner();
      }
      _;
    }
  }
```

Override

Override methods to build on the functionality:

```
import "./Ownable.sol";

contract Example is Ownable {
  event TransferOwnership(address oldOwner, address newOwner);

  // think of virtual and override as compliments,
  // we can override this method because it is declared as virtual in
  function transferOwner(address newOwner) public override onlyOwner
    address oldOwner = owner;
    // call the function on the base or parent contract, Ownable
    super.transferOwner(newOwner);
    emit TransferOwnership(oldOwner, newOwner);
  }
}
```