Calling super.initState() etc.

In the lectures of this course, I always call `super.initState()` and similar methods **LAST** when overriding built-in methods.

Example:

```
@override
void initState() {
  print('Do something...');
  super.initState();
}
```

Whilst it won't make a visual (or performance-related) difference, it is actually now **recommended** to call `super.initState()` (etc.) **FIRST**:

```
@override
void initState() {
  super.initState();
  print('Do something...');
}
```

In **production**, the order will actually **NOT make any difference**. The only code executed by `initState()` in the parent class checks whether "everything is working as intended". It's a debugging-only check, which will have no impact in production mode.

In case you're interested, this is the code inside of the built-in `initState()` method:

```
@protected
@mustCallSuper
void initState() {
  assert(_debugLifecycleState == _StateLifecycle.created);
}
```

`assert` is a Dart function that tests a condition and throws an error if it's not met. `_debugLifecycleState` is a property managed by Flutter to find out in which phase the state object currently is. During production, asserts aren't executed and `_debugLifecycleState` is not set.

During **development**, you'll also **not face any problems** because of changed order. Additional built-in checks will still run properly, the only minor difference is that some of your code may run before the check runs (which won't affect the check though, it'll still work properly).