**Advanced EVM - Recap**

**Advanced EVM - Encoding Recap**

Before looking at how we can apply all our new encoding knowledge to call our own functions, let's recap some of the things we've gone over so far, there's been a lot.

**Concatenation**

At a high-level, we learnt that abi.encodePacked can be used to concatenate strings.

string memory someString = string(abi.encodePacked("Hi Mom! ", "Miss you!"))

❗ **PROTIP** Remember: In newer versions of Solidity, you can use string.concat("Hi Mom! ", "Miss you!")

**Binary and Opcodes**

We learnt that when a contract is compiled, it's actually compiled into an ABI (application binary interface) and a binary or bytecode format.

Any transaction we send to the blockchain is ultimately compiled down to this bytecode. For contract creation transactions, the data field of the transaction *is* this bytecode.

Any system capable of reading the operations contained within this bytecode is said to be EVM Compatible.

**Encoding**

We also learnt that we can use the encoding functionality of the EVM to encode basically anything. Basic encoding is accomplished with abi.encode, but we've a few options available to us.

abi.encode will result in a padded return value, however the EVM offers a way to save space/gas by packing our encodings through abi.encodePacked.

The EVM also affords us the ability to decode and multi-encode, really giving us flexibility to work with our data.

**Low-level Calls**

Lastly we touched on a couple of Solidity's available low-level calls such as call, staticcall.

The data passed to these functions allows us to make *any* arbitrary call to an address we want - this is what we'll cover in more detail next.

**Wrap Up**

Alright, great work so far. Now's a great time to take a break before we make the final push to the end of this section.

In the next lesson we'll see how these concepts work in practice as we dive into function selector encoding and sending encoded function calls.

Let's go!