Let's break it up!

```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

The constructor is a function that gets executed only once, when it is deployed. This constructor can call other functions of the contract, other contracts or set a storage variable, just as the one we have. Note that the received parameter has an extra modifier "memory". This is a modifier that tells the compiler where we want that data to be stored and it is not needed for simple types like uint256, bool, and byte 32 (among others). In our particular case, we are using memory, but other options are "calldata" or "storage". Examples

Let's break it up!

```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

This is our first function which is called "greet", its visibility is external, it's a view function, and it returns a string. Each of this will be explained in more detail in the following slides.

Function visibility: Just as state variables, functions have different visibility options

Private:
Can be called only by the "class" that defines it, only by the instance itself

Internal:
Can be called by the "class" that defines and its inheritors, only by the instance itself

Public:
Can be called by any EOA (externally owned account) or any other contract

External: Can be called only by other contracts (cannot be called by itself) and EOA

Functions have another classification which is its **state mutability**. This depends on how the function uses the state variables and events (which will be explained in future slides) . The possibilities are:

1. **State changing:** This does not require a keyword, and it is the one assumed by default. This classification is used if you are setting a state variable or emit an event in your function
2. **View:** This type of functions cannot set a state variable or emit an event
3. **Pure:** This type of functions implies that the function doesn't modify the state of the blockchain and it does not depend on it either.

State variables can be more complex than just a string, an integer or a boolean. Solidity has support for maps, arrays, enums, and structs.

An array example can be found here:
https://solidity-by-example.org/array/

You are encouraged to try out this example (as all of the following) on **Remix.**

Remix is an online IDE where you can code, deploy and interact with your contracts easily, you can understand how to use it to test this contracts following this guide.

Note that you will need to use the 0.8.10 compiler and copy-paste each contract manually.

A mapping example can be found here:
https://solidity-by-example.org/mapping/

You are encouraged to try it out on **Remix**.

Keep in mind that mappings return a valid (but zeroed) value for uninitialized keys. For example, it will return false for a mapping that goes from addresses to booleans when given a uninitialized address.

A structs example can be found here:
https://solidity-by-example.org/structs/

You are encouraged to try it out on **Remix**.

An Enum example can be found here:
https://solidity-by-example.org/enum/

You are encouraged to try it out on **Remix**.

These complex data structures can be composed. For example, you could have nested mappings, or an array of a particular struct as the value for a mapping.

Try to create a contract that has a setter and a getter for each of these examples in **Remix**.