Solidity Programming

Common primitive types_

1. **bool** – true / false
2. **int / uint** – signed / unsigned integers
3. **address** – Ethereum address (20 bytes)
4. **bytes1 ... bytes32** – fixed-length byte arrays
5. **bytes / string** – dynamically sized
6. **enum** – user-defined type with finite values

```
pragma solidity ^0.8.20;

contract PrimitiveTypes {
    bool public isOpen = true;

    uint256 public totalSupply = 1000;   // unsigned integer
    int256 public temperature = -10;     // signed integer

    address public owner = msg.sender;

    bytes32 public id = keccak256("ABC");

    enum Status { Pending, Shipped, Delivered }
    Status public currentStatus = Status.Pending;
}
```

Variables in Solidity_

Variables define storage locations for data. Types by storage location:

Type	Meaning
state variables	stored on blockchain (persistent, cost gas)
local variables	exist inside functions
global variables	provided by Ethereum VM (block info, msg.sender etc.)

```
pragma solidity ^0.8.20;

contract VariablesExample {

    // State variables (stored on blockchain)
    uint public count = 0;
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function demo() public pure returns(uint) {
        // Local variable
        uint x = 5;
        return x;
    }
}
```

Functions in Solidity_

Functions perform operations or return values. Function features:

- can be **public / private / internal / external**
- can be **view** (reads state) or **pure** (no state access)
- can be **payable** (accept Ether)

```
pragma solidity ^0.8.20;

contract FunctionsExample {

    uint public number;

    // Write function (changes blockchain state)
    function setNumber(uint _num) public {
        number = _num;
    }

    // Read-only function
    function getNumber() public view returns(uint) {
        return number;
    }

    // Pure function (no state read/write)
    function add(uint a, uint b) public pure returns(uint) {
        return a + b;
    }

    // Payable function (accepts Ether)
    function deposit() public payable {}
}
```

Modifiers and Constructors_

Modifiers allow **reusable condition checks** on functions. Common use cases:

- access control (only owner can call)
- validating inputs
- locking functions

```
pragma solidity ^0.8.20;

contract ModifierExample {

    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not the owner");
        _;
    }

    uint public value;

    function setValue(uint _v) public onlyOwner {
        value = _v;
    }
}
```

A **constructor runs only once** when contract is deployed. Used to:

- initialize values
- set owner
- configure parameters

```
pragma solidity ^0.8.20;

contract ConstructorExample {

    address public owner;
    uint public startValue;

    constructor(uint _value) {
        owner = msg.sender;
        startValue = _value;
    }
}
```