}

**Solidity Addresses**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Contract {
    address public owner;

    constructor() {
        owner = msg.sender;  // Store the address that deploys the contract
    }
}
```

**Receive Function**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Contract {
    // This function allows the contract to receive ether without any calldata
    receive() external payable {
        // You can implement custom logic here to handle the received ether
    }

    // Add a function to view the contract's balance
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

**Transferring Funds**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Contract {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    // Function to receive tips and forward them to the owner
    function tip() public payable {
        (bool sent, ) = owner.call{ value: msg.value }("");
        require(sent, "Transfer failed");
    }
}
```

**Reverting Transactions**

```
pragma solidity ^0.8.0;

contract Contract {
    uint public x;

    // Payable constructor requiring a 1 ether deposit
    constructor() payable {
        // Ensure that at least 1 ether is sent to the contract
        require(msg.value >= 1 ether, "You must send at least 1 ether");

        // If 1 ether is received, set x to the amount sent (in wei)
        x = msg.value;
    }
}
```

**Restricting by Address**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Contract {
    address public owner;

    // Error for unauthorized withdrawal
    error NotOwner(address caller);

    // Payable constructor to accept ether during deployment
    constructor() payable {
        owner = msg.sender;  // Set the owner to the address that deploys the contract
    }

    // Function to withdraw all funds to the owner's address
    function withdraw() external {
        // Ensure only the owner can withdraw
        if(msg.sender != owner) {
            revert NotOwner(msg.sender);
        }

        // Transfer all ether to the owner
        payable(owner).transfer(address(this).balance);
    }

    // Function to receive ether
    receive() external payable {}
}
```

**Function Modifiers**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract OwnerRestricted {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

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

    uint public value;

    function setValue(uint _value) external onlyOwner {
        value = _value;
    }

    function changeOwner(address newOwner) external onlyOwner {
        owner = newOwner;
    }
}
```

**Call Function**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

// Interface for the Hero contract
interface IHero {
    function alert() external;
}

// Sidekick contract that calls alert on the Hero
contract Sidekick {
    function sendAlert(address hero) external {
        IHero(hero).alert();
    }
}
```

**Function Signature**

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;



contract Sidekick {

    function sendAlert(address hero) external {

        // keccak256("alert()") =
0xc633fd39ca3d8c7765be031c2c3d638a2f360d08e4fd3e69f15516f8d4e3c34e

        // first 4 bytes: 0xc633fd39

        (bool success, ) = hero.call(abi.encodeWithSelector(0xc633fd39));

        require(success, "Alert failed");

    }

}
```

**Setup ESCROW**

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;


contract Escrow {

    address public depositor;

    address public beneficiary;

    address public arbiter;

}
```

**Constructor Storage**

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.20;

contract Escrow {
    address public depositor;
    address public beneficiary;
    address public arbiter;

    constructor(address _arbiter, address _beneficiary) {
        depositor = msg.sender;
        arbiter = _arbiter;
        beneficiary = _beneficiary;
    }
}
```

## Funding

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Escrow {
    address public depositor;
    address public beneficiary;
    address public arbiter;

    constructor(address _arbiter, address _beneficiary) payable {
        depositor = msg.sender;
        arbiter = _arbiter;
        beneficiary = _beneficiary;
    }
}
```

## Approval

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Escrow {
    address public depositor;
    address public beneficiary;
    address public arbiter;

    // Constructor to initialize the contract with the arbiter and beneficiary addresses
    constructor(address _arbiter, address _beneficiary) payable {
        depositor = msg.sender;
        arbiter = _arbiter;
        beneficiary = _beneficiary;
    }

    // Function to approve the transfer of funds from the contract to the beneficiary
    function approve() external {
        require(msg.sender == arbiter, "Only the arbiter can approve the transfer.");
        require(address(this).balance > 0, "Contract has no funds to transfer.");

        payable(beneficiary).transfer(address(this).balance);
    }
}
```

**Security**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Escrow {
    address public depositor;
    address public beneficiary;
    address public arbiter;

    // Constructor to initialize the contract with the arbiter and beneficiary addresses
    constructor(address _arbiter, address _beneficiary) payable {
        depositor = msg.sender;
        arbiter = _arbiter;
        beneficiary = _beneficiary;
    }

    // Function to approve the transfer of funds from the contract to the beneficiary
    function approve() external {
        // Ensure only the arbiter can approve the transfer
        require(msg.sender == arbiter, "Only the arbiter can approve the transfer.");

        // Ensure the contract has a positive balance to transfer
        require(address(this).balance > 0, "Contract has no funds to transfer.");

        // Transfer the contract's balance to the beneficiary
        payable(beneficiary).transfer(address(this).balance);
    }
}
```