



*How to develop ERC20 token on EtherCore mainnet

1. Access <https://ide.ethercore.org>

2. Copy-paste the following code to IDE

```
pragma solidity ^0.5.16;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal {}

    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable)
    { return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
        https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
```



```

* @dev Wrappers over Solidity's arithmetic operations with added overflow
* checks.
*
* Arithmetic operations in Solidity wrap on overflow. This can easily result
* in bugs, because programmers usually assume that an overflow raises an
* error, which is the standard behavior in high level programming languages.
* `SafeMath` restores this intuition by reverting the transaction when an
* operation overflows.
*
* Using this library instead of the unchecked operations eliminates an entire
* class of bugs, so it's recommended to use it always.
*/

```

```

library SafeMath {

```

```

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256)
    {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */

```

```

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {

```



```

        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's '-' operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0_
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
    returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's '*' operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

```



```

uint256 c = a * b;
require(c / a == b, "SafeMath: multiplication overflow");

return c;
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b) internal pure returns (uint256)
{ return div(a, b, "SafeMath: division by zero");
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts with custom message
on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 * - The divisor cannot be zero.
 *
 * _Available since v2.4.0._
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256)
{
    // Solidity only automatically asserts when dividing by 0
    require(b > 0, errorMessage);

```



```

uint256 c = a / b;

// assert(a == b * c + a % b); // There is no case in which this doesn't
hold return c;
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
 * Reverts when dividing by zero.
 *
 * Counterpart to Solidity's `%` operator. This function uses a `revert`
 * opcode (which leaves remaining gas untouched) while Solidity uses an
 * invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 * - The divisor cannot be zero.
 */
function mod(uint256 a, uint256 b) internal pure returns (uint256)
{ return mod(a, b, "SafeMath: modulo by zero");
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
 * Reverts with custom message when dividing by zero.
 *
 * Counterpart to Solidity's `%` operator. This function uses a `revert`
 * opcode (which leaves remaining gas untouched) while Solidity uses an
 * invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 * - The divisor cannot be zero.
 *
 * _Available since v2.4.0._
 */
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
    require(b != 0, errorMessage);
    return a % b;
}

```



```

}

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
}

```



```

/**
 * @dev Returns true if the caller is the current owner.
 */
function isOwner() public view returns (bool) {
    return _msgSender() == _owner;
}

/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public onlyOwner
{
    emit OwnershipTransferred(_owner,
    address(0)); _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public onlyOwner
{
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 */
function _transferOwnership(address newOwner) internal {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
}

/**

```



```

* @dev Interface of the ERC20 standard as defined in the EIP. Does not include
* the optional functions; to access them see {ERC20Detailed}.
*/
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate

```




```

* transaction ordering. One possible solution to mitigate this race
* condition is to first reduce the spender's allowance to 0 and set the
* desired value afterwards:
* https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
*
* Emits an {Approval} event.
*/
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 */
*
* Emits a {Transfer} event.
*/
function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);

/**
 * @dev Emitted when `value` tokens are moved from one account (`from`) to
 * another (`to`).
 *
 * Note that `value` may be zero.
 */
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Implementation of the {IERC20} interface.

```



- * This implementation is agnostic to the way tokens are created. This means
- * that a supply mechanism has to be added in a derived contract using {_mint}.
- * For a generic mechanism see {ERC20Mintable}.
- *
- * TIP: For a detailed writeup see our guide
- * <https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226>[How
- * to implement supply mechanisms].
- *
- * We have followed general OpenZeppelin guidelines: functions revert instead
- * of returning `false` on failure. This behavior is nonetheless conventional
- * and does not conflict with the expectations of ERC20 applications.
- *
- * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
- * This allows applications to reconstruct the allowance for all accounts just
- * by listening to said events. Other implementations of the EIP may not emit
- * these events, as it isn't required by the specification.
- *
- * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
- * functions have been added to mitigate the well-known issues around setting
- * allowances. See {IERC20-approve}.
- */

```
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view returns (uint256)
    {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
```



```

*/

function balanceOf(address account) public view returns (uint256)
    { return _balances[account];
    }

/**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `recipient` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
function transfer(address recipient, uint256 amount) public returns (bool)
    { _transfer(_msgSender(), recipient, amount);
      return true;
    }

/**
 * @dev See {IERC20-allowance}.
 */
function allowance(address owner, address spender) public view returns (uint256)
    { return _allowances[owner][spender];
    }

/**
 * @dev See {IERC20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(address spender, uint256 amount) public returns (bool)
    { _approve(_msgSender(), spender, amount);
      return true;
    }

/**
 * @dev See {IERC20-transferFrom}.

```



```

*
* Emits an {Approval} event indicating the updated allowance. This is not
* required by the EIP. See the note at the beginning of {ERC20};
*
* Requirements:
* - `sender` and `recipient` cannot be the zero address.
* - `sender` must have a balance of at least `amount`.
* - the caller must have allowance for `sender`'s tokens of at least
* `amount`.
*/
function transferFrom(address sender, address recipient, uint256 amount) public returns (bool)
{
    _transfer(sender, recipient, amount);
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
    return true;
}

/**
 * @dev Atomically decreases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for

```



```

* problems described in {IERC20-approve}.
*
* Emits an {Approval} event indicating the updated allowance.
*
* Requirements:
*
* - `spender` cannot be the zero address.
* - `spender` must have allowance for the caller of at least
* `subtractedValue`.
*/

function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool)
{
    _approve(_msgSender(), spender,
allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below
zero"));

    return true;
}

/**
* @dev Moves tokens `amount` from `sender` to `recipient`.
*
* This is internal function is equivalent to {transfer}, and can be used to
* e.g. implement automatic token fees, slashing mechanisms, etc.
*
* Emits a {Transfer} event.
*
* Requirements:
*
* - `sender` cannot be the zero address.
* - `recipient` cannot be the zero address.
* - `sender` must have a balance of at least `amount`.
*/

function _transfer(address sender, address recipient, uint256 amount) internal
{
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount
exceeds balance");

    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);

```



```

}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements
 *
 * - `to` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: mint to the zero address");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: burn from the zero address");

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds
balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}

```



```

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
 *
 * This is internal function is equivalent to `approve`, and can be used to
 * e.g. set automatic allowances for certain subsystems, etc.
 *
 * Emits an {Approval} event.
 *
 * Requirements:
 *
 * - `owner` cannot be the zero address.
 * - `spender` cannot be the zero address.
 */
function _approve(address owner, address spender, uint256 amount) internal
{
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
 * from the caller's allowance.
 *
 * See {_burn} and {_approve}.
 */
function _burnFrom(address account, uint256 amount) internal
{
    _burn(account, amount);
    _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20:
burn amount exceeds allowance"));
}
}

/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
    string private _name;

```



```

string private _symbol;
uint8 private _decimals;

/**
 * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
 * these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name, string memory symbol, uint8 decimals) public
{
    _name = name;
    _symbol = symbol;
    _decimals = decimals;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view returns (string memory)
{
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view returns (string memory)
{
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5,05` ( $505 / 10^{** 2}$ ).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including

```




```

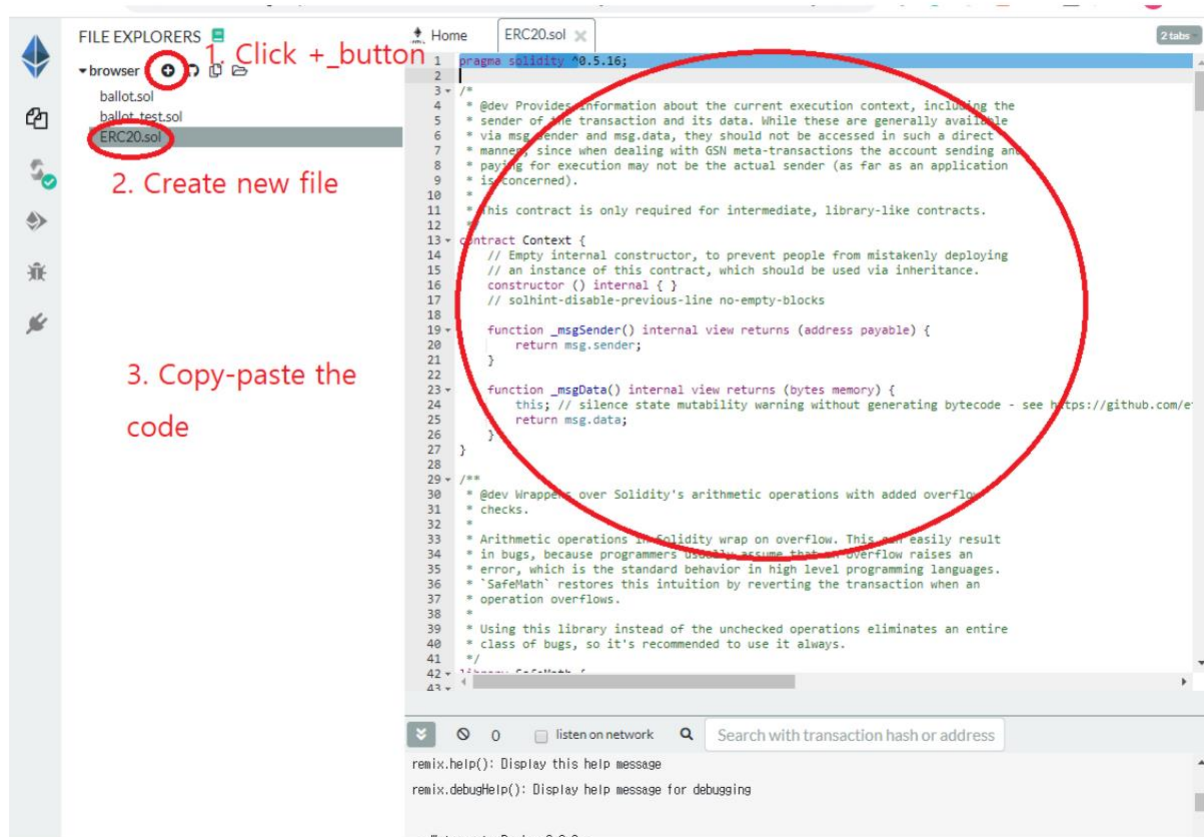
    * {IERC20-balanceOf} and {IERC20-transfer}.
    */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

contract TEST is ERC20 {
    string public constant name = "TEST token";
    string public constant symbol = "TEST";
    uint public constant decimals = 18;
    uint public constant INITIAL_SUPPLY = 1000 * (10 ** decimals);

    constructor() public {
        _mint(msg.sender, INITIAL_SUPPLY);
    }
}

```





You can modify the `string public constant name = "TEST token", string public constant symbol = "TEST", string public constant decimals = 18, string public constant INITIAL_SUPPLY` part as you would like



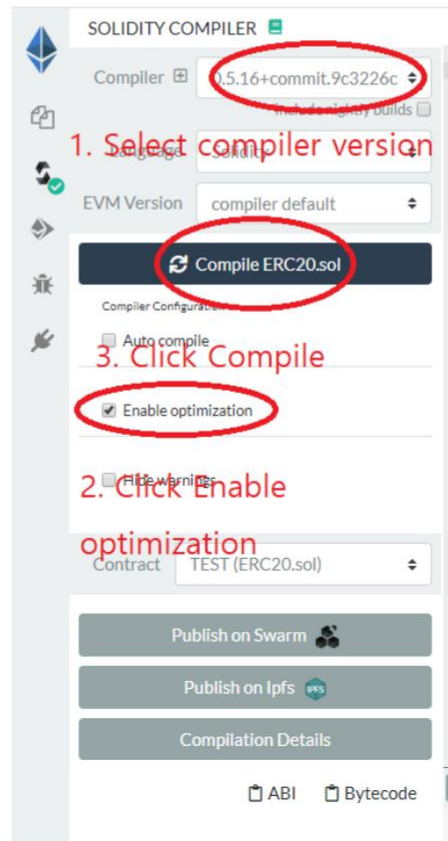
3. Click plugin icon and activate some modules

The screenshot displays the Remix IDE interface. On the left, the 'PLUGIN MANAGER' sidebar is open, showing a search bar and a list of modules. Under 'Active Modules', the 'Debugger' and 'Solidity compiler' are listed with 'Deactivate' buttons. A red circle highlights the plugin icon (a wrench) in the sidebar. Under 'Inactive Modules', the '3Box Spaces' and 'Debug Tools for Remix plugins' are listed with 'Activate' buttons. A red oval highlights the 'Deactivate' buttons for the 'Debugger' and 'Solidity compiler' in the 'Active Modules' section. Red text annotations are present: '1. Click Plugin button' points to the wrench icon, and '2. Activate the following modules (Solidity compiler, Debugger)' points to the 'Deactivate' buttons. The main editor on the right shows a Solidity contract named 'ERC20.sol' with the following code:

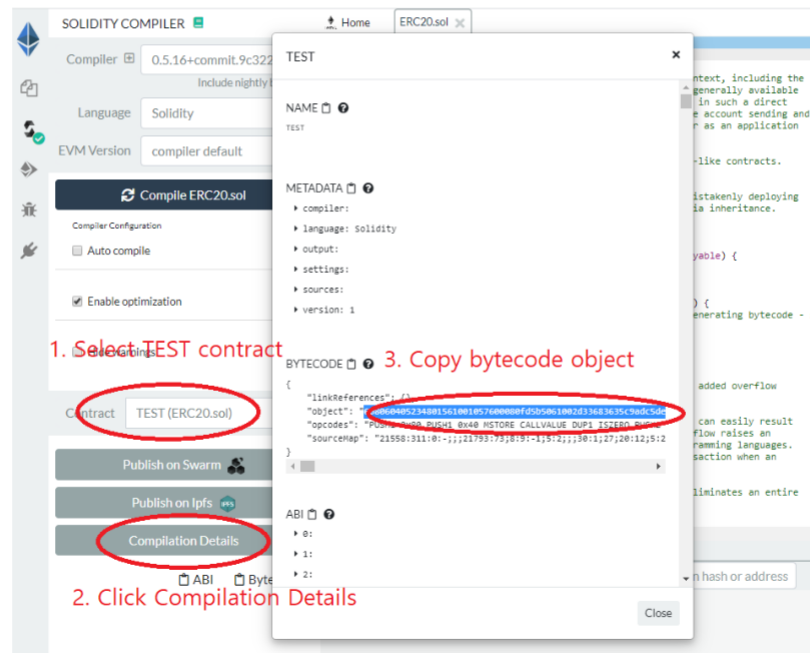
```
1 pragma solidity ^0.5.16;
2
3 /*
4  * @dev Provides information about the current execution conte
5  * sender of the transaction and its data. While these are gen
6  * via msg.sender and msg.data, they should not be accessed in
7  * manner, since when dealing with GSN meta-transactions the a
8  * paying for execution may not be the actual sender (as far a
9  * is concerned).
10  *
11  * This contract is only required for intermediate, library-li
12  */
13 contract Context {
14     // Empty internal constructor, to prevent people from mist
15     // an instance of this contract, which should be used via
16     constructor () internal { }
17     // solhint-disable-previous-line no-empty-blocks
18
19     function _msgSender() internal view returns (address payab
20         return msg.sender;
21     }
22
23     function _msgData() internal view returns (bytes memory) {
24         this; // silence state mutability warning without gene
25         return msg.data;
26     }
27 }
28
29 * @dev Wrappers over Solidity's arithmetic operations with ad
30 * checks.
31 *
32 * Arithmetic operations in Solidity wrap on overflow. This ca
33 * in bugs, because programmers usually assume that an overflo
34 * error, which is the standard behavior in high level program
35 * 'SafeMath' restores this intuition by reverting the transac
36 * operation overflows.
37 *
38 * Using this library instead of the unchecked operations elim
39 * class of bugs, so it's recommended to use it always.
40 */
41
42 * SafeMath
43
```



4. Select the proper compiler version and compile the contract



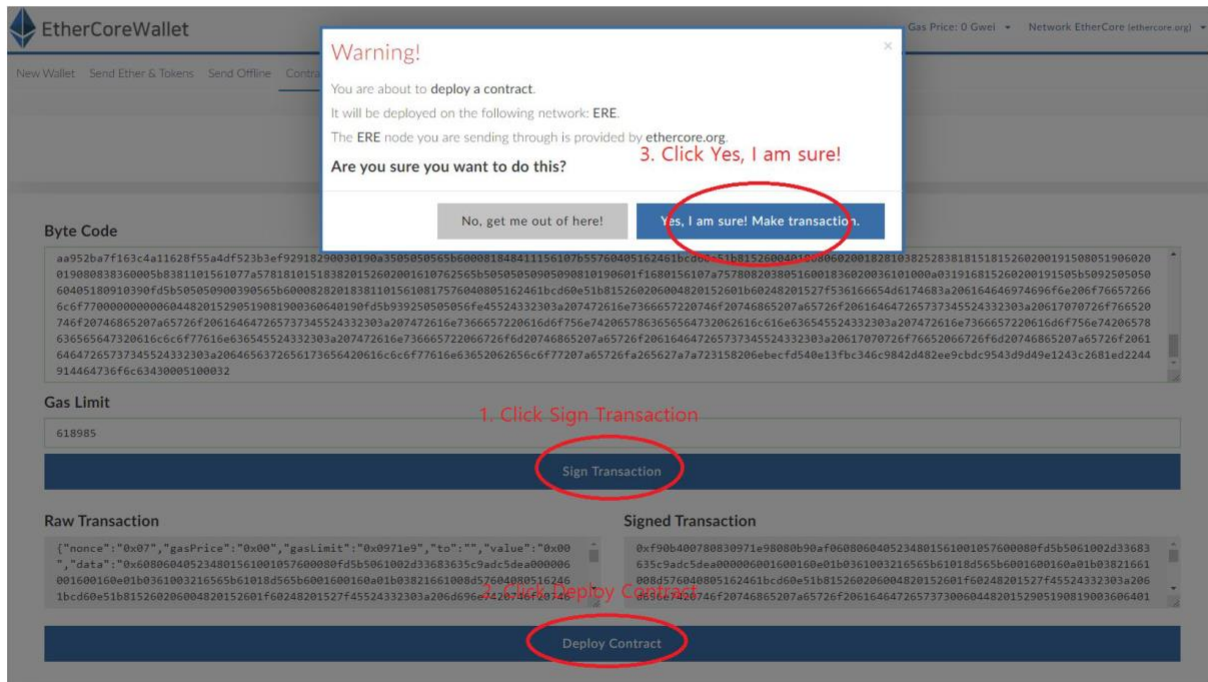
- Copy the bytecode of the compiled contract



6. Access <https://wallet.ethereum.io> and deploy the compiled contract to the mainnet


[illegible]

7. Click Deploy Contract and check the deployed contract on the explorer



8. After deploying contract, click Send Ether & Tokens on web wallet and select Add Custom Token and fill in the necessary information to interact with deployed token contract


Token Balances

 **How to See Your Tokens**

You can also view your Balances on explorer.ethercore.org

Show All Tokens Add Custom Token

Token Contract Address



Token Symbol

Decimals

Save

