

```

1 /**
2  *Submitted for verification at BscScan.com on
3  *2021-04-24
4  */
5 /**
6  #MARIOBROS COIN
7
8
9  #MARIO features:
10  7% fee auto add to the liquidity pool to loc
11  ked forever when selling
12  3% fee auto distribute to all holders
13  I created a black hole so #MARIO token will
14  deflate itself in supply with every transactio
15  n
16  25% Supply is burned at start.
17  also there is antiwhale system every buy and
18  sell
19
20  I will add 0.3 bnb as an initial liquidity,
21  and burn 25% from the start to create the blac
22  khole.
23  0.05% team token, and after that i will burn
24  the LP and renounce Ownership
25  can u make #MARIO 100000x???
26  i will give this token to the community, ple
27  ase make telegram t.me/MARIOBROS_COIN
28  */
29
30 pragma solidity ^0.6.12;
31 // SPDX-License-Identifier: Unlicensed
32 interface IERC20 {
33
34     function totalSupply() external view return
35     s (uint256);
36
37     /**
38      * @dev Returns the amount of tokens owned
39      by `account`.
40     */
41     function balanceOf(address account) externa
42     l view returns (uint256);
43
44     /**
45      * @dev Moves `amount` tokens from the call
46     er's account to `recipient`.
47
48      *
49      * Returns a boolean value indicating wheth
50     er the operation succeeded.
51
52      *
53      * Emits a {Transfer} event.
54     */
55     function transfer(address recipient, uint25
56     6 amount) external returns (bool);
57
58     /**
59      * @dev Returns the remaining number of tok
60     ens that `spender` will be
61
62      * allowed to spend on behalf of `owner` th
63     rough {transferFrom}. This is

```

```

1 /**
2  *Submitted for verification at BscScan.com on
3  *2021-04-05
4  */
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

45     * zero by default.
46     *
47     * This value changes when {approve} or {transferFrom} are called.
48     */
49     function allowance(address owner, address spender) external view returns (uint256);
50
51     /**
52     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
53     *
54     * Returns a boolean value indicating whether the operation succeeded.
55     *
56     * IMPORTANT: Beware that changing an allowance with this method brings the risk
57     * that someone may use both the old and the new allowance by unfortunate
58     * transaction ordering. One possible solution to mitigate this race
59     * condition is to first reduce the spender's allowance to 0 and set the
60     * desired value afterwards:
61     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
62     *
63     * Emits an {Approval} event.
64     */
65     function approve(address spender, uint256 amount) external returns (bool);
66
67     /**
68     * @dev Moves `amount` tokens from `sender` to `recipient` using the
69     * allowance mechanism. `amount` is then deducted from the caller's
70     * allowance.
71     *
72     * Returns a boolean value indicating whether the operation succeeded.
73     *
74     * Emits a {Transfer} event.
75     */
76     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
77
78     /**
79     * @dev Emitted when `value` tokens are moved from one account (`from`) to
80     * another (`to`).
81     *
82     * Note that `value` may be zero.
83     */
84     event Transfer(address indexed from, address indexed to, uint256 value);
85
86     /**
87     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
88     * a call to {approve}. `value` is the new allowance.
89     */
90     event Approval(address indexed owner, address indexed spender, uint256 value);

```

```

28     * zero by default.
29     *
30     * This value changes when {approve} or {transferFrom} are called.
31     */
32     function allowance(address owner, address spender) external view returns (uint256);
33
34     /**
35     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
36     *
37     * Returns a boolean value indicating whether the operation succeeded.
38     *
39     * IMPORTANT: Beware that changing an allowance with this method brings the risk
40     * that someone may use both the old and the new allowance by unfortunate
41     * transaction ordering. One possible solution to mitigate this race
42     * condition is to first reduce the spender's allowance to 0 and set the
43     * desired value afterwards:
44     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
45     *
46     * Emits an {Approval} event.
47     */
48     function approve(address spender, uint256 amount) external returns (bool);
49
50     /**
51     * @dev Moves `amount` tokens from `sender` to `recipient` using the
52     * allowance mechanism. `amount` is then deducted from the caller's
53     * allowance.
54     *
55     * Returns a boolean value indicating whether the operation succeeded.
56     *
57     * Emits a {Transfer} event.
58     */
59     function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
60
61     /**
62     * @dev Emitted when `value` tokens are moved from one account (`from`) to
63     * another (`to`).
64     *
65     * Note that `value` may be zero.
66     */
67     event Transfer(address indexed from, address indexed to, uint256 value);
68
69     /**
70     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
71     * a call to {approve}. `value` is the new allowance.
72     */
73     event Approval(address indexed owner, address indexed spender, uint256 value);

```

```

91 }
92
93
94
95 /**
96  * @dev Wrappers over Solidity's arithmetic operations with added overflow
97  * checks.
98  *
99  * Arithmetic operations in Solidity wrap on overflow. This can easily result
100 * in bugs, because programmers usually assume that an overflow raises an
101 * error, which is the standard behavior in high level programming languages.
102 * `SafeMath` restores this intuition by reverting the transaction when an
103 * operation overflows.
104 *
105 * Using this library instead of the unchecked operations eliminates an entire
106 * class of bugs, so it's recommended to use it always.
107 */
108
109 library SafeMath {
110     /**
111      * @dev Returns the addition of two unsigned integers, reverting on
112      * overflow.
113      *
114      * Counterpart to Solidity's `+` operator.
115      *
116      * Requirements:
117      *
118      * - Addition cannot overflow.
119      */
120     function add(uint256 a, uint256 b) internal pure returns (uint256) {
121         uint256 c = a + b;
122         require(c >= a, "SafeMath: addition overflow");
123
124         return c;
125     }
126
127     /**
128      * @dev Returns the subtraction of two unsigned integers, reverting on
129      * overflow (when the result is negative).
130      *
131      * Counterpart to Solidity's `-` operator.
132      *
133      * Requirements:
134      *
135      * - Subtraction cannot overflow.
136      */
137     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
138         return sub(a, b, "SafeMath: subtraction overflow");
139     }
140
141     /**
142      * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
143      * overflow (when the result is negative).

```

```

74 }
75
76
77
78 /**
79  * @dev Wrappers over Solidity's arithmetic operations with added overflow
80  * checks.
81  *
82  * Arithmetic operations in Solidity wrap on overflow. This can easily result
83  * in bugs, because programmers usually assume that an overflow raises an
84  * error, which is the standard behavior in high level programming languages.
85  * `SafeMath` restores this intuition by reverting the transaction when an
86  * operation overflows.
87  *
88  * Using this library instead of the unchecked operations eliminates an entire
89  * class of bugs, so it's recommended to use it always.
90 */
91
92 library SafeMath {
93     /**
94      * @dev Returns the addition of two unsigned integers, reverting on
95      * overflow.
96      *
97      * Counterpart to Solidity's `+` operator.
98      *
99      * Requirements:
100      *
101      * - Addition cannot overflow.
102      */
103     function add(uint256 a, uint256 b) internal pure returns (uint256) {
104         uint256 c = a + b;
105         require(c >= a, "SafeMath: addition overflow");
106
107         return c;
108     }
109
110     /**
111      * @dev Returns the subtraction of two unsigned integers, reverting on
112      * overflow (when the result is negative).
113      *
114      * Counterpart to Solidity's `-` operator.
115      *
116      * Requirements:
117      *
118      * - Subtraction cannot overflow.
119      */
120     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
121         return sub(a, b, "SafeMath: subtraction overflow");
122     }
123
124     /**
125      * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
126      * overflow (when the result is negative).

```

```

144     *
145     * Counterpart to Solidity's '-' operator.
146     *
147     * Requirements:
148     *
149     * - Subtraction cannot overflow.
150     */
151     function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
152         require(b <= a, errorMessage);
153         uint256 c = a - b;
154
155         return c;
156     }
157
158     /**
159     * @dev Returns the multiplication of two unsigned integers, reverting on
160     * overflow.
161     *
162     * Counterpart to Solidity's '*' operator.
163     *
164     * Requirements:
165     *
166     * - Multiplication cannot overflow.
167     */
168     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
169         // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
170         // benefit is lost if 'b' is also tested.
171         // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
172         if (a == 0) {
173             return 0;
174         }
175
176         uint256 c = a * b;
177         require(c / a == b, "SafeMath: multiplication overflow");
178
179         return c;
180     }
181
182     /**
183     * @dev Returns the integer division of two unsigned integers. Reverts on
184     * division by zero. The result is rounded towards zero.
185     *
186     * Counterpart to Solidity's '/' operator.
187     * Note: this function uses a
188     * 'revert' opcode (which leaves remaining gas untouched) while Solidity
189     * uses an invalid opcode to revert (consuming all remaining gas).
190     *
191     * Requirements:
192     *
193     * - The divisor cannot be zero.
194     */
195     function div(uint256 a, uint256 b) internal pure returns (uint256) {
196         return div(a, b, "SafeMath: division by

```

```

127     *
128     * Counterpart to Solidity's '-' operator.
129     *
130     * Requirements:
131     *
132     * - Subtraction cannot overflow.
133     */
134     function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
135         require(b <= a, errorMessage);
136         uint256 c = a - b;
137
138         return c;
139     }
140
141     /**
142     * @dev Returns the multiplication of two unsigned integers, reverting on
143     * overflow.
144     *
145     * Counterpart to Solidity's '*' operator.
146     *
147     * Requirements:
148     *
149     * - Multiplication cannot overflow.
150     */
151     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
152         // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
153         // benefit is lost if 'b' is also tested.
154         // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
155         if (a == 0) {
156             return 0;
157         }
158
159         uint256 c = a * b;
160         require(c / a == b, "SafeMath: multiplication overflow");
161
162         return c;
163     }
164
165     /**
166     * @dev Returns the integer division of two unsigned integers. Reverts on
167     * division by zero. The result is rounded towards zero.
168     *
169     * Counterpart to Solidity's '/' operator.
170     * Note: this function uses a
171     * 'revert' opcode (which leaves remaining gas untouched) while Solidity
172     * uses an invalid opcode to revert (consuming all remaining gas).
173     *
174     * Requirements:
175     *
176     * - The divisor cannot be zero.
177     */
178     function div(uint256 a, uint256 b) internal pure returns (uint256) {
179         return div(a, b, "SafeMath: division by

```

```

zero");
196     }
197
198     /**
199     * @dev Returns the integer division of two
    unsigned integers. Reverts with custom message
    on
200     * division by zero. The result is rounded
    towards zero.
201     *
202     * Counterpart to Solidity's `/` operator.
    Note: this function uses a
203     * `revert` opcode (which leaves remaining
    gas untouched) while Solidity
204     * uses an invalid opcode to revert (consum
    ing all remaining gas).
205     *
206     * Requirements:
207     *
208     * - The divisor cannot be zero.
209     */
210     function div(uint256 a, uint256 b, string m
    emory errorMessage) internal pure returns (uint
    256) {
211         require(b > 0, errorMessage);
212         uint256 c = a / b;
213         // assert(a == b * c + a % b); // There
    is no case in which this doesn't hold
214
215         return c;
216     }
217
218     /**
219     * @dev Returns the remainder of dividing t
    wo unsigned integers. (unsigned integer modul
    o),
220     * Reverts when dividing by zero.
221     *
222     * Counterpart to Solidity's `%` operator.
    This function uses a `revert`
223     * opcode (which leaves remaining gas untou
    ched) while Solidity uses an
224     * invalid opcode to revert (consuming all
    remaining gas).
225     *
226     * Requirements:
227     *
228     * - The divisor cannot be zero.
229     */
230     function mod(uint256 a, uint256 b) internal
    pure returns (uint256) {
231         return mod(a, b, "SafeMath: modulo by z
    ero");
232     }
233
234     /**
235     * @dev Returns the remainder of dividing t
    wo unsigned integers. (unsigned integer modul
    o),
236     * Reverts with custom message when dividin
    g by zero.
237     *
238     * Counterpart to Solidity's `%` operator.
    This function uses a `revert`
239     * opcode (which leaves remaining gas untou
    ched) while Solidity uses an

```

```

zero");
179     }
180
181     /**
182     * @dev Returns the integer division of two
    unsigned integers. Reverts with custom message
    on
183     * division by zero. The result is rounded
    towards zero.
184     *
185     * Counterpart to Solidity's `/` operator.
    Note: this function uses a
186     * `revert` opcode (which leaves remaining
    gas untouched) while Solidity
187     * uses an invalid opcode to revert (consum
    ing all remaining gas).
188     *
189     * Requirements:
190     *
191     * - The divisor cannot be zero.
192     */
193     function div(uint256 a, uint256 b, string m
    emory errorMessage) internal pure returns (uint
    256) {
194         require(b > 0, errorMessage);
195         uint256 c = a / b;
196         // assert(a == b * c + a % b); // There
    is no case in which this doesn't hold
197
198         return c;
199     }
200
201     /**
202     * @dev Returns the remainder of dividing t
    wo unsigned integers. (unsigned integer modul
    o),
203     * Reverts when dividing by zero.
204     *
205     * Counterpart to Solidity's `%` operator.
    This function uses a `revert`
206     * opcode (which leaves remaining gas untou
    ched) while Solidity uses an
207     * invalid opcode to revert (consuming all
    remaining gas).
208     *
209     * Requirements:
210     *
211     * - The divisor cannot be zero.
212     */
213     function mod(uint256 a, uint256 b) internal
    pure returns (uint256) {
214         return mod(a, b, "SafeMath: modulo by z
    ero");
215     }
216
217     /**
218     * @dev Returns the remainder of dividing t
    wo unsigned integers. (unsigned integer modul
    o),
219     * Reverts with custom message when dividin
    g by zero.
220     *
221     * Counterpart to Solidity's `%` operator.
    This function uses a `revert`
222     * opcode (which leaves remaining gas untou
    ched) while Solidity uses an

```

```

240     * invalid opcode to revert (consuming all
      remaining gas).
241     *
242     * Requirements:
243     *
244     * - The divisor cannot be zero.
245     */
246     function mod(uint256 a, uint256 b, string m
      emory errorMessage) internal pure returns (uint
      256) {
247         require(b != 0, errorMessage);
248         return a % b;
249     }
250 }
251
252 abstract contract Context {
253     function _msgSender() internal view virtual
      returns (address payable) {
254         return msg.sender;
255     }
256
257     function _msgData() internal view virtual r
      eturns (bytes memory) {
258         this; // silence state mutability warni
      ng without generating bytecode - see https://gi
      thub.com/ethereum/solidity/issues/2691
259         return msg.data;
260     }
261 }
262
263 /**
264  * @dev Collection of functions related to the
      address type
265  */
266
267 library Address {
268     /**
269     * @dev Returns true if `account` is a cont
      ract.
270     *
271     * [IMPORTANT]
272     * ====
273     * It is unsafe to assume that an address f
      or which this function returns
274     * false is an externally-owned account (EO
      A) and not a contract.
275     *
276     * Among others, `isContract` will return f
      alse for the following
277     * types of addresses:
278     *
279     * - an externally-owned account
280     * - a contract in construction
281     * - an address where a contract will be c
      reated
282     * - an address where a contract lived, bu
      t was destroyed
283     * ====
284     */
285     function isContract(address account) intern
      al view returns (bool) {
286         // According to EIP-1052, 0x0 is the va
      lue returned for not-yet created accounts
287         // and 0xc5d2460186f7233c927e7db2dcc703
      c0e500b653ca82273b7bfad8045d85a470 is returned
288         // for accounts without code, i.e. `kec

```

```

223     * invalid opcode to revert (consuming all
      remaining gas).
224     *
225     * Requirements:
226     *
227     * - The divisor cannot be zero.
228     */
229     function mod(uint256 a, uint256 b, string m
      emory errorMessage) internal pure returns (uint
      256) {
230         require(b != 0, errorMessage);
231         return a % b;
232     }
233 }
234
235 abstract contract Context {
236     function _msgSender() internal view virtual
      returns (address payable) {
237         return msg.sender;
238     }
239
240     function _msgData() internal view virtual r
      eturns (bytes memory) {
241         this; // silence state mutability warni
      ng without generating bytecode - see https://gi
      thub.com/ethereum/solidity/issues/2691
242         return msg.data;
243     }
244 }
245
246 /**
247  * @dev Collection of functions related to the
      address type
248  */
249
250 library Address {
251     /**
252     * @dev Returns true if `account` is a cont
      ract.
253     *
254     * [IMPORTANT]
255     * ====
256     * It is unsafe to assume that an address f
      or which this function returns
257     * false is an externally-owned account (EO
      A) and not a contract.
258     *
259     * Among others, `isContract` will return f
      alse for the following
260     * types of addresses:
261     *
262     * - an externally-owned account
263     * - a contract in construction
264     * - an address where a contract will be c
      reated
265     * - an address where a contract lived, bu
      t was destroyed
266     * ====
267     */
268     function isContract(address account) intern
      al view returns (bool) {
269         // According to EIP-1052, 0x0 is the va
      lue returned for not-yet created accounts
270         // and 0xc5d2460186f7233c927e7db2dcc703
      c0e500b653ca82273b7bfad8045d85a470 is returned
271         // for accounts without code, i.e. `kec

```

```

cak256('')`
289     bytes32 codehash;
290     bytes32 accountHash = 0xc5d2460186f7233
c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
70;
291     // solhint-disable-next-line no-inline-
assembly
292     assembly { codehash := extcodehash(acco
unt) }
293     return (codehash != accountHash && code
hash != 0x0);
294 }
295
296 /**
297  * @dev Replacement for Solidity's `transfe
r`: sends `amount` wei to
298  * `recipient`, forwarding all available ga
s and reverting on errors.
299  *
300  * https://eips.ethereum.org/EIPS/eip-1884
[EIP1884] increases the gas cost
301  * of certain opcodes, possibly making cont
racts go over the 2300 gas limit
302  * imposed by `transfer`, making them unabl
e to receive funds via
303  * `transfer`. {sendValue} removes this lim
itation.
304  *
305  * https://diligence.consensys.net/posts/20
19/09/stop-using-soliditys-transfer-now/[Learn
more].
306  *
307  * IMPORTANT: because control is transferre
d to `recipient`, care must be
308  * taken to not create reentrancy vulnerabi
lities. Consider using
309  * {ReentrancyGuard} or the
310  * https://solidity.readthedocs.io/en/v0.5.
11/security-considerations.html#use-the-checks-
effects-interactions-pattern[checks-effects-int
eractions pattern].
311  */
312     function sendValue(address payable recipien
t, uint256 amount) internal {
313         require(address(this).balance >= amoun
t, "Address: insufficient balance");
314
315         // solhint-disable-next-line avoid-low-
level-calls, avoid-call-value
316         (bool success, ) = recipient.call{ valu
e: amount }("");
317         require(success, "Address: unable to se
nd value, recipient may have reverted");
318     }
319
320 /**
321  * @dev Performs a Solidity function call u
sing a low level `call`. A
322  * plain `call` is an unsafe replacement for
a function call: use this
323  * function instead.
324  *
325  * If `target` reverts with a revert reaso
n, it is bubbled up by this
326  * function (like regular Solidity function
calls).

```

```

cak256('')`
272     bytes32 codehash;
273     bytes32 accountHash = 0xc5d2460186f7233
c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
70;
274     // solhint-disable-next-line no-inline-
assembly
275     assembly { codehash := extcodehash(acco
unt) }
276     return (codehash != accountHash && code
hash != 0x0);
277 }
278
279 /**
280  * @dev Replacement for Solidity's `transfe
r`: sends `amount` wei to
281  * `recipient`, forwarding all available ga
s and reverting on errors.
282  *
283  * https://eips.ethereum.org/EIPS/eip-1884
[EIP1884] increases the gas cost
284  * of certain opcodes, possibly making cont
racts go over the 2300 gas limit
285  * imposed by `transfer`, making them unabl
e to receive funds via
286  * `transfer`. {sendValue} removes this lim
itation.
287  *
288  * https://diligence.consensys.net/posts/20
19/09/stop-using-soliditys-transfer-now/[Learn
more].
289  *
290  * IMPORTANT: because control is transferre
d to `recipient`, care must be
291  * taken to not create reentrancy vulnerabi
lities. Consider using
292  * {ReentrancyGuard} or the
293  * https://solidity.readthedocs.io/en/v0.5.
11/security-considerations.html#use-the-checks-
effects-interactions-pattern[checks-effects-int
eractions pattern].
294  */
295     function sendValue(address payable recipien
t, uint256 amount) internal {
296         require(address(this).balance >= amoun
t, "Address: insufficient balance");
297
298         // solhint-disable-next-line avoid-low-
level-calls, avoid-call-value
299         (bool success, ) = recipient.call{ valu
e: amount }("");
300         require(success, "Address: unable to se
nd value, recipient may have reverted");
301     }
302
303 /**
304  * @dev Performs a Solidity function call u
sing a low level `call`. A
305  * plain `call` is an unsafe replacement for
a function call: use this
306  * function instead.
307  *
308  * If `target` reverts with a revert reaso
n, it is bubbled up by this
309  * function (like regular Solidity function
calls).

```

```

327     *
328     * Returns the raw returned data. To conver
t to the expected return value,
329     * use https://solidity.readthedocs.io/en/l
atest/units-and-global-variables.html?highlight
=abi.decode#abi-encoding-and-decoding-functions
[abi.decode`].
330     *
331     * Requirements:
332     *
333     * - `target` must be a contract.
334     * - calling `target` with `data` must not
revert.
335     *
336     * _Available since v3.1._
337     */
338     function functionCall(address target, bytes
memory data) internal returns (bytes memory) {
339         return functionCall(target, data, "Addres
s: low-level call failed");
340     }
341
342     /**
343     * @dev Same as {xref-Address-functionCall-
address-bytes-}[`functionCall`], but with
344     * `errorMessage` as a fallback revert reas
on when `target` reverts.
345     *
346     * _Available since v3.1._
347     */
348     function functionCall(address target, bytes
memory data, string memory errorMessage) intern
al returns (bytes memory) {
349         return _functionCallWithValue(target, d
ata, 0, errorMessage);
350     }
351
352     /**
353     * @dev Same as {xref-Address-functionCall-
address-bytes-}[`functionCall`],
354     * but also transferring `value` wei to `ta
rget`.
355     *
356     * Requirements:
357     *
358     * - the calling contract must have an ETH
balance of at least `value`.
359     * - the called Solidity function must be `
payable`.
360     *
361     * _Available since v3.1._
362     */
363     function functionCallWithValue(address targ
et, bytes memory data, uint256 value) internal
returns (bytes memory) {
364         return functionCallWithValue(target, da
ta, value, "Address: low-level call with value
failed");
365     }
366
367     /**
368     * @dev Same as {xref-Address-functionCallW
ithValue-address-bytes-uint256-}[`functionCallW
ithValue`], but
369     * with `errorMessage` as a fallback revert
reason when `target` reverts.

```

```

310     *
311     * Returns the raw returned data. To conver
t to the expected return value,
312     * use https://solidity.readthedocs.io/en/l
atest/units-and-global-variables.html?highlight
=abi.decode#abi-encoding-and-decoding-functions
[abi.decode`].
313     *
314     * Requirements:
315     *
316     * - `target` must be a contract.
317     * - calling `target` with `data` must not
revert.
318     *
319     * _Available since v3.1._
320     */
321     function functionCall(address target, bytes
memory data) internal returns (bytes memory) {
322         return functionCall(target, data, "Addres
s: low-level call failed");
323     }
324
325     /**
326     * @dev Same as {xref-Address-functionCall-
address-bytes-}[`functionCall`], but with
327     * `errorMessage` as a fallback revert reas
on when `target` reverts.
328     *
329     * _Available since v3.1._
330     */
331     function functionCall(address target, bytes
memory data, string memory errorMessage) intern
al returns (bytes memory) {
332         return _functionCallWithValue(target, d
ata, 0, errorMessage);
333     }
334
335     /**
336     * @dev Same as {xref-Address-functionCall-
address-bytes-}[`functionCall`],
337     * but also transferring `value` wei to `ta
rget`.
338     *
339     * Requirements:
340     *
341     * - the calling contract must have an ETH
balance of at least `value`.
342     * - the called Solidity function must be `
payable`.
343     *
344     * _Available since v3.1._
345     */
346     function functionCallWithValue(address targ
et, bytes memory data, uint256 value) internal
returns (bytes memory) {
347         return functionCallWithValue(target, da
ta, value, "Address: low-level call with value
failed");
348     }
349
350     /**
351     * @dev Same as {xref-Address-functionCallW
ithValue-address-bytes-uint256-}[`functionCallW
ithValue`], but
352     * with `errorMessage` as a fallback revert
reason when `target` reverts.

```



```

370     *
371     * _Available since v3.1._
372     */
373     function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
374         require(address(this).balance >= value, "Address: insufficient balance for call");
375         return _functionCallWithValue(target, data, value, errorMessage);
376     }
377
378     function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
379         require(isContract(target), "Address: call to non-contract");
380
381         // solhint-disable-next-line avoid-low-level-calls
382         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
383         if (success) {
384             return returndata;
385         } else {
386             // Look for revert reason and bubble it up if present
387             if (returndata.length > 0) {
388                 // The easiest way to bubble the revert reason is using memory via assembly
389
390                 // solhint-disable-next-line no-inline-assembly
391                 assembly {
392                     let returndata_size := mload(returndata)
393                     revert(add(32, returndata), returndata_size)
394                 }
395             } else {
396                 revert(errorMessage);
397             }
398         }
399     }
400 }
401
402 /**
403  * @dev Contract module which provides a basic access control mechanism, where
404  * there is an account (an owner) that can be granted exclusive access to
405  * specific functions.
406  *
407  * By default, the owner account will be the one that deploys the contract. This
408  * can later be changed with {transferOwnership}.
409  *
410  * This module is used through inheritance. It will make available the modifier
411  * `onlyOwner`, which can be applied to your functions to restrict their use to
412  * the owner.
413  */

```

```

353     *
354     * _Available since v3.1._
355     */
356     function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
357         require(address(this).balance >= value, "Address: insufficient balance for call");
358         return _functionCallWithValue(target, data, value, errorMessage);
359     }
360
361     function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
362         require(isContract(target), "Address: call to non-contract");
363
364         // solhint-disable-next-line avoid-low-level-calls
365         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
366         if (success) {
367             return returndata;
368         } else {
369             // Look for revert reason and bubble it up if present
370             if (returndata.length > 0) {
371                 // The easiest way to bubble the revert reason is using memory via assembly
372
373                 // solhint-disable-next-line no-inline-assembly
374                 assembly {
375                     let returndata_size := mload(returndata)
376                     revert(add(32, returndata), returndata_size)
377                 }
378             } else {
379                 revert(errorMessage);
380             }
381         }
382     }
383 }
384
385 /**
386  * @dev Contract module which provides a basic access control mechanism, where
387  * there is an account (an owner) that can be granted exclusive access to
388  * specific functions.
389  *
390  * By default, the owner account will be the one that deploys the contract. This
391  * can later be changed with {transferOwnership}.
392  *
393  * This module is used through inheritance. It will make available the modifier
394  * `onlyOwner`, which can be applied to your functions to restrict their use to
395  * the owner.
396  */

```

```

414 contract Ownable is Context {
415     address private _owner;
416     address private _previousOwner;
417     uint256 private _lockTime;
418
419     event OwnershipTransferred(address indexed
previousOwner, address indexed newOwner);
420
421     /**
422      * @dev Initializes the contract setting th
e deployer as the initial owner.
423      */
424     constructor () internal {
425         address msgSender = _msgSender();
426         _owner = msgSender;
427         emit OwnershipTransferred(address(0), m
sgSender);
428     }
429
430     /**
431      * @dev Returns the address of the current
owner.
432      */
433     function owner() public view returns (addre
ss) {
434         return _owner;
435     }
436
437     /**
438      * @dev Throws if called by any account oth
er than the owner.
439      */
440     modifier onlyOwner() {
441         require(_owner == _msgSender(), "Ownabl
e: caller is not the owner");
442         _;
443     }
444
445     /**
446      * @dev Leaves the contract without owner.
It will not be possible to call
447      * `onlyOwner` functions anymore. Can only
be called by the current owner.
448      *
449      * NOTE: Renouncing ownership will leave th
e contract without an owner,
450      * thereby removing any functionality that
is only available to the owner.
451      */
452     function renounceOwnership() public virtual
onlyOwner {
453         emit OwnershipTransferred(_owner, addre
ss(0));
454         _owner = address(0);
455     }
456
457     /**
458      * @dev Transfers ownership of the contract
to a new account (`newOwner`).
459      * Can only be called by the current owner.
460      */
461     function transferOwnership(address newOwne
r) public virtual onlyOwner {
462         require(newOwner != address(0), "Ownabl
e: new owner is the zero address");
463         emit OwnershipTransferred(_owner, newOw
ner);

```

```

397 contract Ownable is Context {
398     address private _owner;
399     address private _previousOwner;
400     uint256 private _lockTime;
401
402     event OwnershipTransferred(address indexed
previousOwner, address indexed newOwner);
403
404     /**
405      * @dev Initializes the contract setting th
e deployer as the initial owner.
406      */
407     constructor () internal {
408         address msgSender = _msgSender();
409         _owner = msgSender;
410         emit OwnershipTransferred(address(0), m
sgSender);
411     }
412
413     /**
414      * @dev Returns the address of the current
owner.
415      */
416     function owner() public view returns (addre
ss) {
417         return _owner;
418     }
419
420     /**
421      * @dev Throws if called by any account oth
er than the owner.
422      */
423     modifier onlyOwner() {
424         require(_owner == _msgSender(), "Ownabl
e: caller is not the owner");
425         _;
426     }
427
428     /**
429      * @dev Leaves the contract without owner.
It will not be possible to call
430      * `onlyOwner` functions anymore. Can only
be called by the current owner.
431      *
432      * NOTE: Renouncing ownership will leave th
e contract without an owner,
433      * thereby removing any functionality that
is only available to the owner.
434      */
435     function renounceOwnership() public virtual
onlyOwner {
436         emit OwnershipTransferred(_owner, addre
ss(0));
437         _owner = address(0);
438     }
439
440     /**
441      * @dev Transfers ownership of the contract
to a new account (`newOwner`).
442      * Can only be called by the current owner.
443      */
444     function transferOwnership(address newOwne
r) public virtual onlyOwner {
445         require(newOwner != address(0), "Ownabl
e: new owner is the zero address");
446         emit OwnershipTransferred(_owner, newOw
ner);

```

```

464     _owner = newOwner;
465 }
466
467 function geUnlockTime() public view returns
(uint256) {
468     return _lockTime;
469 }
470
471 //Locks the contract for owner for the amou
nt of time provided
472 function lock(uint256 time) public virtual
onlyOwner {
473     _previousOwner = _owner;
474     _owner = address(0);
475     _lockTime = now + time;
476     emit OwnershipTransferred(_owner, addre
ss(0));
477 }
478
479 //Unlocks the contract for owner when _lock
Time is exceeds
480 function unlock() public virtual {
481     require(_previousOwner == msg.sender,
"You don't have permission to unlock");
482     require(now > _lockTime , "Contract is
locked until 7 days");
483     emit OwnershipTransferred(_owner, _prev
iousOwner);
484     _owner = _previousOwner;
485 }
486 }
487
488 // pragma solidity >=0.5.0;
489
490 interface IUniswapV2Factory {
491     event PairCreated(address indexed token0, a
ddress indexed token1, address pair, uint);
492
493     function feeTo() external view returns (add
ress);
494     function feeToSetter() external view return
s (address);
495
496     function getPair(address tokenA, address to
kenB) external view returns (address pair);
497     function allPairs(uint) external view retur
ns (address pair);
498     function allPairsLength() external view ret
urns (uint);
499
500     function createPair(address tokenA, address
tokenB) external returns (address pair);
501
502     function setFeeTo(address) external;
503     function setFeeToSetter(address) external;
504 }
505
506
507 // pragma solidity >=0.5.0;
508
509 interface IUniswapV2Pair {
510     event Approval(address indexed owner, addre
ss indexed spender, uint value);
511     event Transfer(address indexed from, addres
s indexed to, uint value);
512
513     function name() external pure returns (stri
ng memory);

```

```

447     _owner = newOwner;
448 }
449
450 function geUnlockTime() public view returns
(uint256) {
451     return _lockTime;
452 }
453
454 //Locks the contract for owner for the amou
nt of time provided
455 function lock(uint256 time) public virtual
onlyOwner {
456     _previousOwner = _owner;
457     _owner = address(0);
458     _lockTime = now + time;
459     emit OwnershipTransferred(_owner, addre
ss(0));
460 }
461
462 //Unlocks the contract for owner when _lock
Time is exceeds
463 function unlock() public virtual {
464     require(_previousOwner == msg.sender,
"You don't have permission to unlock");
465     require(now > _lockTime , "Contract is
locked until 7 days");
466     emit OwnershipTransferred(_owner, _prev
iousOwner);
467     _owner = _previousOwner;
468 }
469 }
470
471 // pragma solidity >=0.5.0;
472
473 interface IUniswapV2Factory {
474     event PairCreated(address indexed token0, a
ddress indexed token1, address pair, uint);
475
476     function feeTo() external view returns (add
ress);
477     function feeToSetter() external view return
s (address);
478
479     function getPair(address tokenA, address to
kenB) external view returns (address pair);
480     function allPairs(uint) external view retur
ns (address pair);
481     function allPairsLength() external view ret
urns (uint);
482
483     function createPair(address tokenA, address
tokenB) external returns (address pair);
484
485     function setFeeTo(address) external;
486     function setFeeToSetter(address) external;
487 }
488
489
490 // pragma solidity >=0.5.0;
491
492 interface IUniswapV2Pair {
493     event Approval(address indexed owner, addre
ss indexed spender, uint value);
494     event Transfer(address indexed from, addres
s indexed to, uint value);
495
496     function name() external pure returns (stri
ng memory);

```

```

514     function symbol() external pure returns (st
ring memory);
515     function decimals() external pure returns
(uint8);
516     function totalSupply() external view return
s (uint);
517     function balanceOf(address owner) external
view returns (uint);
518     function allowance(address owner, address s
pender) external view returns (uint);
519
520     function approve(address spender, uint valu
e) external returns (bool);
521     function transfer(address to, uint value) e
xternal returns (bool);
522     function transferFrom(address from, address
to, uint value) external returns (bool);
523
524     function DOMAIN_SEPARATOR() external view r
eturns (bytes32);
525     function PERMIT_TYPEHASH() external pure re
turns (bytes32);
526     function nonces(address owner) external vie
w returns (uint);
527
528     function permit(address owner, address spen
der, uint value, uint deadline, uint8 v, bytes3
2 r, bytes32 s) external;
529
530     event Mint(address indexed sender, uint amo
unt0, uint amount1);
531     event Burn(address indexed sender, uint amo
unt0, uint amount1, address indexed to);
532     event Swap(
533         address indexed sender,
534         uint amount0In,
535         uint amount1In,
536         uint amount0Out,
537         uint amount1Out,
538         address indexed to
539     );
540     event Sync(uint112 reserve0, uint112 reserv
e1);
541
542     function MINIMUM_LIQUIDITY() external pure
returns (uint);
543     function factory() external view returns (a
ddress);
544     function token0() external view returns (ad
dress);
545     function token1() external view returns (ad
dress);
546     function getReserves() external view return
s (uint112 reserve0, uint112 reserve1, uint32 b
lockTimestampLast);
547     function price0CumulativeLast() external vi
ew returns (uint);
548     function price1CumulativeLast() external vi
ew returns (uint);
549     function kLast() external view returns (uin
t);
550
551     function mint(address to) external returns
(uint liquidity);
552     function burn(address to) external returns
(uint amount0, uint amount1);
553     function swap(uint amount0Out, uint amount1

```

```

497     function symbol() external pure returns (st
ring memory);
498     function decimals() external pure returns
(uint8);
499     function totalSupply() external view return
s (uint);
500     function balanceOf(address owner) external
view returns (uint);
501     function allowance(address owner, address s
pender) external view returns (uint);
502
503     function approve(address spender, uint valu
e) external returns (bool);
504     function transfer(address to, uint value) e
xternal returns (bool);
505     function transferFrom(address from, address
to, uint value) external returns (bool);
506
507     function DOMAIN_SEPARATOR() external view r
eturns (bytes32);
508     function PERMIT_TYPEHASH() external pure re
turns (bytes32);
509     function nonces(address owner) external vie
w returns (uint);
510
511     function permit(address owner, address spen
der, uint value, uint deadline, uint8 v, bytes3
2 r, bytes32 s) external;
512
513     event Mint(address indexed sender, uint amo
unt0, uint amount1);
514     event Burn(address indexed sender, uint amo
unt0, uint amount1, address indexed to);
515     event Swap(
516         address indexed sender,
517         uint amount0In,
518         uint amount1In,
519         uint amount0Out,
520         uint amount1Out,
521         address indexed to
522     );
523     event Sync(uint112 reserve0, uint112 reserv
e1);
524
525     function MINIMUM_LIQUIDITY() external pure
returns (uint);
526     function factory() external view returns (a
ddress);
527     function token0() external view returns (ad
dress);
528     function token1() external view returns (ad
dress);
529     function getReserves() external view return
s (uint112 reserve0, uint112 reserve1, uint32 b
lockTimestampLast);
530     function price0CumulativeLast() external vi
ew returns (uint);
531     function price1CumulativeLast() external vi
ew returns (uint);
532     function kLast() external view returns (uin
t);
533
534     function mint(address to) external returns
(uint liquidity);
535     function burn(address to) external returns
(uint amount0, uint amount1);
536     function swap(uint amount0Out, uint amount1

```

```

        Out, address to, bytes calldata data) external;
554     function skim(address to) external;
555     function sync() external;
556
557     function initialize(address, address) external;
558 }
559
560 // pragma solidity >=0.6.2;
561
562 interface IUniswapV2Router01 {
563     function factory() external pure returns (address);
564     function WETH() external pure returns (address);
565
566     function addLiquidity(
567         address tokenA,
568         address tokenB,
569         uint amountADesired,
570         uint amountBDesired,
571         uint amountAMin,
572         uint amountBMin,
573         address to,
574         uint deadline
575     ) external returns (uint amountA, uint amountB, uint liquidity);
576     function addLiquidityETH(
577         address token,
578         uint amountTokenDesired,
579         uint amountTokenMin,
580         uint amountETHMin,
581         address to,
582         uint deadline
583     ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
584     function removeLiquidity(
585         address tokenA,
586         address tokenB,
587         uint liquidity,
588         uint amountAMin,
589         uint amountBMin,
590         address to,
591         uint deadline
592     ) external returns (uint amountA, uint amountB);
593     function removeLiquidityETH(
594         address token,
595         uint liquidity,
596         uint amountTokenMin,
597         uint amountETHMin,
598         address to,
599         uint deadline
600     ) external returns (uint amountToken, uint amountETH);
601     function removeLiquidityWithPermit(
602         address tokenA,
603         address tokenB,
604         uint liquidity,
605         uint amountAMin,
606         uint amountBMin,
607         address to,
608         uint deadline,
609         bool approveMax, uint8 v, bytes32 r, bytes32 s
610     ) external returns (uint amountA, uint amountB);

```

```

        Out, address to, bytes calldata data) external;
537     function skim(address to) external;
538     function sync() external;
539
540     function initialize(address, address) external;
541 }
542
543 // pragma solidity >=0.6.2;
544
545 interface IUniswapV2Router01 {
546     function factory() external pure returns (address);
547     function WETH() external pure returns (address);
548
549     function addLiquidity(
550         address tokenA,
551         address tokenB,
552         uint amountADesired,
553         uint amountBDesired,
554         uint amountAMin,
555         uint amountBMin,
556         address to,
557         uint deadline
558     ) external returns (uint amountA, uint amountB, uint liquidity);
559     function addLiquidityETH(
560         address token,
561         uint amountTokenDesired,
562         uint amountTokenMin,
563         uint amountETHMin,
564         address to,
565         uint deadline
566     ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
567     function removeLiquidity(
568         address tokenA,
569         address tokenB,
570         uint liquidity,
571         uint amountAMin,
572         uint amountBMin,
573         address to,
574         uint deadline
575     ) external returns (uint amountA, uint amountB);
576     function removeLiquidityETH(
577         address token,
578         uint liquidity,
579         uint amountTokenMin,
580         uint amountETHMin,
581         address to,
582         uint deadline
583     ) external returns (uint amountToken, uint amountETH);
584     function removeLiquidityWithPermit(
585         address tokenA,
586         address tokenB,
587         uint liquidity,
588         uint amountAMin,
589         uint amountBMin,
590         address to,
591         uint deadline,
592         bool approveMax, uint8 v, bytes32 r, bytes32 s
593     ) external returns (uint amountA, uint amountB);

```

```

        ntB);
611     function removeLiquidityETHWithPermit(
612         address token,
613         uint liquidity,
614         uint amountTokenMin,
615         uint amountETHMin,
616         address to,
617         uint deadline,
618         bool approveMax, uint8 v, bytes32 r, by
tes32 s
619     ) external returns (uint amountToken, uint
amountETH);
620     function swapExactTokensForTokens(
621         uint amountIn,
622         uint amountOutMin,
623         address[] calldata path,
624         address to,
625         uint deadline
626     ) external returns (uint[] memory amounts);
627     function swapTokensForExactTokens(
628         uint amountOut,
629         uint amountInMax,
630         address[] calldata path,
631         address to,
632         uint deadline
633     ) external returns (uint[] memory amounts);
634     function swapExactETHForTokens(uint amountO
utMin, address[] calldata path, address to, uin
t deadline)
635         external
636         payable
637         returns (uint[] memory amounts);
638     function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
address to, uint deadline)
639         external
640         returns (uint[] memory amounts);
641     function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,
address to, uint deadline)
642         external
643         returns (uint[] memory amounts);
644     function swapETHForExactTokens(uint amountO
ut, address[] calldata path, address to, uint d
eadline)
645         external
646         payable
647         returns (uint[] memory amounts);
648
649     function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
650     function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
651     function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
652     function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
653     function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
654 }
655

```

```

        ntB);
594     function removeLiquidityETHWithPermit(
595         address token,
596         uint liquidity,
597         uint amountTokenMin,
598         uint amountETHMin,
599         address to,
600         uint deadline,
601         bool approveMax, uint8 v, bytes32 r, by
tes32 s
602     ) external returns (uint amountToken, uint
amountETH);
603     function swapExactTokensForTokens(
604         uint amountIn,
605         uint amountOutMin,
606         address[] calldata path,
607         address to,
608         uint deadline
609     ) external returns (uint[] memory amounts);
610     function swapTokensForExactTokens(
611         uint amountOut,
612         uint amountInMax,
613         address[] calldata path,
614         address to,
615         uint deadline
616     ) external returns (uint[] memory amounts);
617     function swapExactETHForTokens(uint amountO
utMin, address[] calldata path, address to, uin
t deadline)
618         external
619         payable
620         returns (uint[] memory amounts);
621     function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
address to, uint deadline)
622         external
623         returns (uint[] memory amounts);
624     function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,
address to, uint deadline)
625         external
626         returns (uint[] memory amounts);
627     function swapETHForExactTokens(uint amountO
ut, address[] calldata path, address to, uint d
eadline)
628         external
629         payable
630         returns (uint[] memory amounts);
631
632     function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
633     function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
634     function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
635     function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
636     function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
637 }
638

```

```

656
657
658 // pragma solidity >=0.6.2;
659
660 interface IUniswapV2Router02 is IUniswapV2Route
r01 {
661     function removeLiquidityETHSupportingFeeOnT
ransferTokens(
662         address token,
663         uint liquidity,
664         uint amountTokenMin,
665         uint amountETHMin,
666         address to,
667         uint deadline
668     ) external returns (uint amountETH);
669     function removeLiquidityETHWithPermitSupport
ingFeeOnTransferTokens(
670         address token,
671         uint liquidity,
672         uint amountTokenMin,
673         uint amountETHMin,
674         address to,
675         uint deadline,
676         bool approveMax, uint8 v, bytes32 r, by
tes32 s
677     ) external returns (uint amountETH);
678
679     function swapExactTokensForTokensSupporting
FeeOnTransferTokens(
680         uint amountIn,
681         uint amountOutMin,
682         address[] calldata path,
683         address to,
684         uint deadline
685     ) external;
686     function swapExactETHForTokensSupportingFee
OnTransferTokens(
687         uint amountOutMin,
688         address[] calldata path,
689         address to,
690         uint deadline
691     ) external payable;
692     function swapExactTokensForETHSupportingFee
OnTransferTokens(
693         uint amountIn,
694         uint amountOutMin,
695         address[] calldata path,
696         address to,
697         uint deadline
698     ) external;
699 }
700
701
702 contract MARIOBROS is Context, IERC20, Ownable
{
703     using SafeMath for uint256;
704     using Address for address;
705
706     mapping (address => uint256) private _rOwne
d;
707     mapping (address => uint256) private _tOwne
d;
708     mapping (address => mapping (address => uin
t256)) private _allowances;
709
710     mapping (address => bool) private _isExclud
edFromFee;

```

```

639
640
641 // pragma solidity >=0.6.2;
642
643 interface IUniswapV2Router02 is IUniswapV2Route
r01 {
644     function removeLiquidityETHSupportingFeeOnT
ransferTokens(
645         address token,
646         uint liquidity,
647         uint amountTokenMin,
648         uint amountETHMin,
649         address to,
650         uint deadline
651     ) external returns (uint amountETH);
652     function removeLiquidityETHWithPermitSupport
ingFeeOnTransferTokens(
653         address token,
654         uint liquidity,
655         uint amountTokenMin,
656         uint amountETHMin,
657         address to,
658         uint deadline,
659         bool approveMax, uint8 v, bytes32 r, by
tes32 s
660     ) external returns (uint amountETH);
661
662     function swapExactTokensForTokensSupporting
FeeOnTransferTokens(
663         uint amountIn,
664         uint amountOutMin,
665         address[] calldata path,
666         address to,
667         uint deadline
668     ) external;
669     function swapExactETHForTokensSupportingFee
OnTransferTokens(
670         uint amountOutMin,
671         address[] calldata path,
672         address to,
673         uint deadline
674     ) external payable;
675     function swapExactTokensForETHSupportingFee
OnTransferTokens(
676         uint amountIn,
677         uint amountOutMin,
678         address[] calldata path,
679         address to,
680         uint deadline
681     ) external;
682 }
683
684
685 contract SafeLight is Context, IERC20, Ownable
{
686     using SafeMath for uint256;
687     using Address for address;
688
689     mapping (address => uint256) private _rOwne
d;
690     mapping (address => uint256) private _tOwne
d;
691     mapping (address => mapping (address => uin
t256)) private _allowances;
692
693     mapping (address => bool) private _isExclud
edFromFee;

```

```

711
712     mapping (address => bool) private _isExclud
ed;
713     address[] private _excluded;
714
715     uint256 private constant MAX = ~uint256(0);
716     uint256 private _tTotal = 1000000000 * 10**
6 * 10**9;
717     uint256 private _rTotal = (MAX - (MAX % _tT
otal));
718     uint256 private _tFeeTotal;
719
720     string private _name = "MARIOBROS";
721     string private _symbol = "MARIO";
722     uint8 private _decimals = 9;
723
724     uint256 public _taxFee = 3;
725     uint256 private _previousTaxFee = _taxFee;
726
727     uint256 public _liquidityFee = 7;
728     uint256 private _previousLiquidityFee = _li
quidityFee;
729
730     IUniswapV2Router02 public immutable uniswap
V2Router;
731     address public immutable uniswapV2Pair;
732
733     bool inSwapAndLiquify;
734     bool public swapAndLiquifyEnabled = true;
735
736     uint256 public _maxTxAmount = 5000000 * 10*
*6 * 10**9;
737     uint256 private numTokensSellToAddToLiquidi
ty = 500000 * 10**6 * 10**9;
738
739     event MinTokensBeforeSwapUpdated(uint256 mi
nTokensBeforeSwap);
740     event SwapAndLiquifyEnabledUpdated(bool ena
bled);
741     event SwapAndLiquify(
742         uint256 tokensSwapped,
743         uint256 ethReceived,
744         uint256 tokensIntoLiquidity
745     );
746
747     modifier lockTheSwap {
748         inSwapAndLiquify = true;
749         _;
750         inSwapAndLiquify = false;
751     }
752
753     constructor () public {
754         _rOwned[_msgSender()] = _rTotal;
755
756         IUniswapV2Router02 _uniswapV2Router = I
UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
9e13aDd608C7F);
757         // Create a uniswap pair for this new
token
758         uniswapV2Pair = IUniswapV2Factory(_unis
wapV2Router.factory())
759             .createPair(address(this), _uniswap
V2Router.WETH());
760
761         // set the rest of the contract variabl
es

```

```

694
695     mapping (address => bool) private _isExclud
ed;
696     address[] private _excluded;
697
698     uint256 private constant MAX = ~uint256(0);
699     uint256 private _tTotal = 1000000000 * 10**
6 * 10**9;
700     uint256 private _rTotal = (MAX - (MAX % _tT
otal));
701     uint256 private _tFeeTotal;
702
703     string private _name = "SafeLight";
704     string private _symbol = "SAFELIGHT";
705     uint8 private _decimals = 9;
706
707     uint256 public _taxFee = 5;
708     uint256 private _previousTaxFee = _taxFee;
709
710     uint256 public _liquidityFee = 5;
711     uint256 private _previousLiquidityFee = _li
quidityFee;
712
713     IUniswapV2Router02 public immutable uniswap
V2Router;
714     address public immutable uniswapV2Pair;
715
716     bool inSwapAndLiquify;
717     bool public swapAndLiquifyEnabled = true;
718
719     uint256 public _maxTxAmount = 5000000 * 10*
*6 * 10**9;
720     uint256 private numTokensSellToAddToLiquidi
ty = 500000 * 10**6 * 10**9;
721
722     event MinTokensBeforeSwapUpdated(uint256 mi
nTokensBeforeSwap);
723     event SwapAndLiquifyEnabledUpdated(bool ena
bled);
724     event SwapAndLiquify(
725         uint256 tokensSwapped,
726         uint256 ethReceived,
727         uint256 tokensIntoLiquidity
728     );
729
730     modifier lockTheSwap {
731         inSwapAndLiquify = true;
732         _;
733         inSwapAndLiquify = false;
734     }
735
736     constructor () public {
737         _rOwned[_msgSender()] = _rTotal;
738
739         IUniswapV2Router02 _uniswapV2Router = I
UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
9e13aDd608C7F);
740         // Create a uniswap pair for this new
token
741         uniswapV2Pair = IUniswapV2Factory(_unis
wapV2Router.factory())
742             .createPair(address(this), _uniswap
V2Router.WETH());
743
744         // set the rest of the contract variabl
es

```



```

762         uniswapV2Router = _uniswapV2Router;
763
764         //exclude owner and this contract from
        fee
765         _isExcludedFromFee[owner()] = true;
766         _isExcludedFromFee[address(this)] = true;
767     e;
768     emit Transfer(address(0), _msgSender(),
        _tTotal);
769 }
770
771     function name() public view returns (string
        memory) {
772         return _name;
773     }
774
775     function symbol() public view returns (string
        memory) {
776         return _symbol;
777     }
778
779     function decimals() public view returns (uint8) {
780         return _decimals;
781     }
782
783     function totalSupply() public view override
        returns (uint256) {
784         return _tTotal;
785     }
786
787     function balanceOf(address account) public
        view override returns (uint256) {
788         if (_isExcluded[account]) return _tOwned[account];
789         return tokenFromReflection(_rOwned[account]);
790     }
791
792     function transfer(address recipient, uint256
        amount) public override returns (bool) {
793         _transfer(_msgSender(), recipient, amount);
794         return true;
795     }
796
797     function allowance(address owner, address
        spender) public view override returns (uint256)
        {
798         return _allowances[owner][spender];
799     }
800
801     function approve(address spender, uint256
        amount) public override returns (bool) {
802         _approve(_msgSender(), spender, amount);
803         return true;
804     }
805
806     function transferFrom(address sender, address
        recipient, uint256 amount) public override
        returns (bool) {
807         _transfer(sender, recipient, amount);
808         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
809

```

```

745         uniswapV2Router = _uniswapV2Router;
746
747         //exclude owner and this contract from
        fee
748         _isExcludedFromFee[owner()] = true;
749         _isExcludedFromFee[address(this)] = true;
750     e;
751     emit Transfer(address(0), _msgSender(),
        _tTotal);
752 }
753
754     function name() public view returns (string
        memory) {
755         return _name;
756     }
757
758     function symbol() public view returns (string
        memory) {
759         return _symbol;
760     }
761
762     function decimals() public view returns (uint8) {
763         return _decimals;
764     }
765
766     function totalSupply() public view override
        returns (uint256) {
767         return _tTotal;
768     }
769
770     function balanceOf(address account) public
        view override returns (uint256) {
771         if (_isExcluded[account]) return _tOwned[account];
772         return tokenFromReflection(_rOwned[account]);
773     }
774
775     function transfer(address recipient, uint256
        amount) public override returns (bool) {
776         _transfer(_msgSender(), recipient, amount);
777         return true;
778     }
779
780     function allowance(address owner, address
        spender) public view override returns (uint256)
        {
781         return _allowances[owner][spender];
782     }
783
784     function approve(address spender, uint256
        amount) public override returns (bool) {
785         _approve(_msgSender(), spender, amount);
786         return true;
787     }
788
789     function transferFrom(address sender, address
        recipient, uint256 amount) public override
        returns (bool) {
790         _transfer(sender, recipient, amount);
791         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
792

```

```

        return true;
810     }
811
812     function increaseAllowance(address spender,
uint256 addedValue) public virtual returns (boo
l) {
813         _approve(_msgSender(), spender, _allowa
nces[_msgSender()][spender].add(addedValue));
814         return true;
815     }
816
817     function decreaseAllowance(address spender,
uint256 subtractedValue) public virtual returns
(bool) {
818         _approve(_msgSender(), spender, _allowa
nces[_msgSender()][spender].sub(subtractedValu
e, "ERC20: decreased allowance below zero"));
819         return true;
820     }
821
822     function isExcludedFromReward(address accou
nt) public view returns (bool) {
823         return _isExcluded[account];
824     }
825
826     function totalFees() public view returns (u
int256) {
827         return _tFeeTotal;
828     }
829
830     function deliver(uint256 tAmount) public {
831         address sender = _msgSender();
832         require(!_isExcluded[sender], "Excluded
addresses cannot call this function");
833         (uint256 rAmount,,,,) = _getValues(tAm
ount);
834         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
835         _rTotal = _rTotal.sub(rAmount);
836         _tFeeTotal = _tFeeTotal.add(tAmount);
837     }
838
839     function reflectionFromToken(uint256 tAmoun
t, bool deductTransferFee) public view returns
(uint256) {
840         require(tAmount <= _tTotal, "Amount mus
t be less than supply");
841         if (!deductTransferFee) {
842             (uint256 rAmount,,,,) = _getValues
(tAmount);
843             return rAmount;
844         } else {
845             (,uint256 rTransferAmount,,,,) = _g
etValues(tAmount);
846             return rTransferAmount;
847         }
848     }
849
850     function tokenFromReflection(uint256 rAmoun
t) public view returns(uint256) {
851         require(rAmount <= _rTotal, "Amount mus
t be less than total reflections");
852         uint256 currentRate = _getRate();
853         return rAmount.div(currentRate);
854     }
855
856     function excludeFromReward(address account)

```

```

        return true;
793     }
794
795     function increaseAllowance(address spender,
uint256 addedValue) public virtual returns (boo
l) {
796         _approve(_msgSender(), spender, _allowa
nces[_msgSender()][spender].add(addedValue));
797         return true;
798     }
799
800     function decreaseAllowance(address spender,
uint256 subtractedValue) public virtual returns
(bool) {
801         _approve(_msgSender(), spender, _allowa
nces[_msgSender()][spender].sub(subtractedValu
e, "ERC20: decreased allowance below zero"));
802         return true;
803     }
804
805     function isExcludedFromReward(address accou
nt) public view returns (bool) {
806         return _isExcluded[account];
807     }
808
809     function totalFees() public view returns (u
int256) {
810         return _tFeeTotal;
811     }
812
813     function deliver(uint256 tAmount) public {
814         address sender = _msgSender();
815         require(!_isExcluded[sender], "Excluded
addresses cannot call this function");
816         (uint256 rAmount,,,,) = _getValues(tAm
ount);
817         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
818         _rTotal = _rTotal.sub(rAmount);
819         _tFeeTotal = _tFeeTotal.add(tAmount);
820     }
821
822     function reflectionFromToken(uint256 tAmoun
t, bool deductTransferFee) public view returns
(uint256) {
823         require(tAmount <= _tTotal, "Amount mus
t be less than supply");
824         if (!deductTransferFee) {
825             (uint256 rAmount,,,,) = _getValues
(tAmount);
826             return rAmount;
827         } else {
828             (,uint256 rTransferAmount,,,,) = _g
etValues(tAmount);
829             return rTransferAmount;
830         }
831     }
832
833     function tokenFromReflection(uint256 rAmoun
t) public view returns(uint256) {
834         require(rAmount <= _rTotal, "Amount mus
t be less than total reflections");
835         uint256 currentRate = _getRate();
836         return rAmount.div(currentRate);
837     }
838
839     function excludeFromReward(address account)

```

```

    public onlyOwner() {
857         // require(account != 0x7a250d5630B4cF5
39739dF2C5dAcB4c659F2488D, 'We can not exclude
        Uniswap router.');
```

858 require(!_isExcluded[account], "Account
is already excluded");

```

859         if(_rOwned[account] > 0) {
860             _tOwned[account] = tokenFromReflect
            ion(_rOwned[account]);
861         }
862         _isExcluded[account] = true;
863         _excluded.push(account);
864     }
865
866     function includeInReward(address account) e
xternal onlyOwner() {
867         require(!_isExcluded[account], "Account
        is already excluded");
868         for (uint256 i = 0; i < _excluded.length
        h; i++) {
869             if (_excluded[i] == account) {
870                 _excluded[i] = _excluded[_exclu
                ded.length - 1];
871                 _tOwned[account] = 0;
872                 _isExcluded[account] = false;
873                 _excluded.pop();
874                 break;
875             }
876         }
877     }
878
879     function _transferBothExcluded(address
        sender, address recipient, uint256 tAmount) pr
        ivate {
879         (uint256 rAmount, uint256 rTransferAmou
        nt, uint256 rFee, uint256 tTransferAmount, uint
        256 tFee, uint256 tLiquidity) = _getValues(tAmo
        unt);
880         _tOwned[sender] = _tOwned[sender].sub(t
        Amount);
881         _rOwned[sender] = _rOwned[sender].sub(r
        Amount);
882         _tOwned[recipient] = _tOwned[recipien
        t].add(tTransferAmount);
883         _rOwned[recipient] = _rOwned[recipien
        t].add(rTransferAmount);
884         _takeLiquidity(tLiquidity);
885         _reflectFee(rFee, tFee);
886         emit Transfer(sender, recipient, tTrans
        ferAmount);
887     }
888
889     function excludeFromFee(address accoun
        t) public onlyOwner {
890         _isExcludedFromFee[account] = true;
891     }
892
893     function includeInFee(address account) publ
        ic onlyOwner {
894         _isExcludedFromFee[account] = false;
895     }
896
897     function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
898         _taxFee = taxFee;
899     }
900
```

```

    public onlyOwner() {
840         // require(account != 0x7a250d5630B4cF5
39739dF2C5dAcB4c659F2488D, 'We can not exclude
        Uniswap router.');
```

841 require(!_isExcluded[account], "Account
is already excluded");

```

842         if(_rOwned[account] > 0) {
843             _tOwned[account] = tokenFromReflect
            ion(_rOwned[account]);
844         }
845         _isExcluded[account] = true;
846         _excluded.push(account);
847     }
848
849     function includeInReward(address account) e
xternal onlyOwner() {
850         require(!_isExcluded[account], "Account
        is already excluded");
851         for (uint256 i = 0; i < _excluded.length
        h; i++) {
852             if (_excluded[i] == account) {
853                 _excluded[i] = _excluded[_exclu
                ded.length - 1];
854                 _tOwned[account] = 0;
855                 _isExcluded[account] = false;
856                 _excluded.pop();
857                 break;
858             }
859         }
860     }
861
862     function _transferBothExcluded(address
        sender, address recipient, uint256 tAmount) pr
        ivate {
862         (uint256 rAmount, uint256 rTransferAmou
        nt, uint256 rFee, uint256 tTransferAmount, uint
        256 tFee, uint256 tLiquidity) = _getValues(tAmo
        unt);
863         _tOwned[sender] = _tOwned[sender].sub(t
        Amount);
864         _rOwned[sender] = _rOwned[sender].sub(r
        Amount);
865         _tOwned[recipient] = _tOwned[recipien
        t].add(tTransferAmount);
866         _rOwned[recipient] = _rOwned[recipien
        t].add(rTransferAmount);
867         _takeLiquidity(tLiquidity);
868         _reflectFee(rFee, tFee);
869         emit Transfer(sender, recipient, tTrans
        ferAmount);
870     }
871
872     function excludeFromFee(address accoun
        t) public onlyOwner {
873         _isExcludedFromFee[account] = true;
874     }
875
876     function includeInFee(address account) publ
        ic onlyOwner {
877         _isExcludedFromFee[account] = false;
878     }
879
880     function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
881         _taxFee = taxFee;
882     }
883
```

```

901     function setLiquidityFeePercent(uint256 liq
liquidityFee) external onlyOwner() {
902         _liquidityFee = liquidityFee;
903     }
904
905     function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
906         _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
907             10**2
908         );
909     }
910
911     function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
912         swapAndLiquifyEnabled = _enabled;
913         emit SwapAndLiquifyEnabledUpdated(_enab
led);
914     }
915
916     //to recieve ETH from uniswapV2Router when
swaping
917     receive() external payable {}
918
919     function _reflectFee(uint256 rFee, uint256
tFee) private {
920         _rTotal = _rTotal.sub(rFee);
921         _tFeeTotal = _tFeeTotal.add(tFee);
922     }
923
924     function _getValues(uint256 tAmount) privat
e view returns (uint256, uint256, uint256, uint
256, uint256, uint256) {
925         (uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity) = _getTValues(tAmount);
926         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
927         return (rAmount, rTransferAmount, rFee,
tTransferAmount, tFee, tLiquidity);
928     }
929
930     function _getTValues(uint256 tAmount) priva
te view returns (uint256, uint256, uint256) {
931         uint256 tFee = calculateTaxFee(tAmoun
t);
932         uint256 tLiquidity = calculateLiquidity
Fee(tAmount);
933         uint256 tTransferAmount = tAmount.sub(t
Fee).sub(tLiquidity);
934         return (tTransferAmount, tFee, tLiquidi
ty);
935     }
936
937     function _getRValues(uint256 tAmount, uint2
56 tFee, uint256 tLiquidity, uint256 currentRat
e) private pure returns (uint256, uint256, uint
256) {
938         uint256 rAmount = tAmount.mul(currentRa
te);
939         uint256 rFee = tFee.mul(currentRate);
940         uint256 rLiquidity = tLiquidity.mul(cur
rentRate);
941         uint256 rTransferAmount = rAmount.sub(r
Fee).sub(rLiquidity);
942

```

```

884     function setLiquidityFeePercent(uint256 liq
liquidityFee) external onlyOwner() {
885         _liquidityFee = liquidityFee;
886     }
887
888     function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
889         _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
890             10**2
891         );
892     }
893
894     function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
895         swapAndLiquifyEnabled = _enabled;
896         emit SwapAndLiquifyEnabledUpdated(_enab
led);
897     }
898
899     //to recieve ETH from uniswapV2Router when
swaping
900     receive() external payable {}
901
902     function _reflectFee(uint256 rFee, uint256
tFee) private {
903         _rTotal = _rTotal.sub(rFee);
904         _tFeeTotal = _tFeeTotal.add(tFee);
905     }
906
907     function _getValues(uint256 tAmount) privat
e view returns (uint256, uint256, uint256, uint
256, uint256, uint256) {
908         (uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity) = _getTValues(tAmount);
909         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
910         return (rAmount, rTransferAmount, rFee,
tTransferAmount, tFee, tLiquidity);
911     }
912
913     function _getTValues(uint256 tAmount) priva
te view returns (uint256, uint256, uint256) {
914         uint256 tFee = calculateTaxFee(tAmoun
t);
915         uint256 tLiquidity = calculateLiquidity
Fee(tAmount);
916         uint256 tTransferAmount = tAmount.sub(t
Fee).sub(tLiquidity);
917         return (tTransferAmount, tFee, tLiquidi
ty);
918     }
919
920     function _getRValues(uint256 tAmount, uint2
56 tFee, uint256 tLiquidity, uint256 currentRat
e) private pure returns (uint256, uint256, uint
256) {
921         uint256 rAmount = tAmount.mul(currentRa
te);
922         uint256 rFee = tFee.mul(currentRate);
923         uint256 rLiquidity = tLiquidity.mul(cur
rentRate);
924         uint256 rTransferAmount = rAmount.sub(r
Fee).sub(rLiquidity);
925

```

```

        return (rAmount, rTransferAmount, rFee);
943     }
944
945     function _getRate() private view returns(uint256) {
946         (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
947         return rSupply.div(tSupply);
948     }
949
950     function _getCurrentSupply() private view returns(uint256, uint256) {
951         uint256 rSupply = _rTotal;
952         uint256 tSupply = _tTotal;
953         for (uint256 i = 0; i < _excluded.length; i++) {
954             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
955             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
956             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
957         }
958         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
959         return (rSupply, tSupply);
960     }
961
962     function _takeLiquidity(uint256 tLiquidity) private {
963         uint256 currentRate = _getRate();
964         uint256 rLiquidity = tLiquidity.mul(currentRate);
965         _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
966         if(_isExcluded[address(this)])
967             _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
968     }
969
970     function calculateTaxFee(uint256 _amount) private view returns (uint256) {
971         return _amount.mul(_taxFee).div(
972             10**2
973         );
974     }
975
976     function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
977         return _amount.mul(_liquidityFee).div(
978             10**2
979         );
980     }
981
982     function removeAllFee() private {
983         if(_taxFee == 0 && _liquidityFee == 0)
984             return;
985         _previousTaxFee = _taxFee;
986         _previousLiquidityFee = _liquidityFee;
987
988         _taxFee = 0;
989         _liquidityFee = 0;
990     }

```

```

        return (rAmount, rTransferAmount, rFee);
926     }
927
928     function _getRate() private view returns(uint256) {
929         (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
930         return rSupply.div(tSupply);
931     }
932
933     function _getCurrentSupply() private view returns(uint256, uint256) {
934         uint256 rSupply = _rTotal;
935         uint256 tSupply = _tTotal;
936         for (uint256 i = 0; i < _excluded.length; i++) {
937             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
938             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
939             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
940         }
941         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
942         return (rSupply, tSupply);
943     }
944
945     function _takeLiquidity(uint256 tLiquidity) private {
946         uint256 currentRate = _getRate();
947         uint256 rLiquidity = tLiquidity.mul(currentRate);
948         _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
949         if(_isExcluded[address(this)])
950             _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
951     }
952
953     function calculateTaxFee(uint256 _amount) private view returns (uint256) {
954         return _amount.mul(_taxFee).div(
955             10**2
956         );
957     }
958
959     function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
960         return _amount.mul(_liquidityFee).div(
961             10**2
962         );
963     }
964
965     function removeAllFee() private {
966         if(_taxFee == 0 && _liquidityFee == 0)
967             return;
968         _previousTaxFee = _taxFee;
969         _previousLiquidityFee = _liquidityFee;
970
971         _taxFee = 0;
972         _liquidityFee = 0;
973     }

```

```

991
992     function restoreAllFee() private {
993         _taxFee = _previousTaxFee;
994         _liquidityFee = _previousLiquidityFee;
995     }
996
997     function isExcludedFromFee(address account)
998     public view returns(bool) {
999         return _isExcludedFromFee[account];
1000     }
1001
1002     function _approve(address owner, address sp
1003     ender, uint256 amount) private {
1004         require(owner != address(0), "ERC20: ap
1005         prove from the zero address");
1006         require(spender != address(0), "ERC20:
1007         approve to the zero address");
1008
1009         _allowances[owner][spender] = amount;
1010         emit Approval(owner, spender, amount);
1011     }
1012
1013     function _transfer(
1014     address from,
1015     address to,
1016     uint256 amount
1017     ) private {
1018         require(from != address(0), "ERC20: tra
1019         nsfer from the zero address");
1020         require(to != address(0), "ERC20: trans
1021         fer to the zero address");
1022         require(amount > 0, "Transfer amount mu
1023         st be greater than zero");
1024         if(from != owner() && to != owner())
1025             require(amount <= _maxTxAmount, "Tr
1026             ansfer amount exceeds the maxTxAmount.");
1027
1028         // is the token balance of this contrac
1029         t address over the min number of
1030         // tokens that we need to initiate a sw
1031         ap + liquidity lock?
1032         // also, don't get caught in a circular
1033         liquidity event.
1034         // also, don't swap & liquify if sender
1035         is uniswap pair.
1036         uint256 contractTokenBalance = balanceO
1037         f(address(this));
1038
1039         if(contractTokenBalance >= _maxTxAmoun
1040         t)
1041         {
1042             contractTokenBalance = _maxTxAmoun
1043             t;
1044         }
1045
1046         bool overMinTokenBalance = contractToke
1047         nBalance >= numTokensSellToAddToLiquidity;
1048         if (
1049             overMinTokenBalance &&
1050             !inSwapAndLiquify &&
1051             from != uniswapV2Pair &&
1052             swapAndLiquifyEnabled
1053         ) {
1054             contractTokenBalance = numTokensSel
1055             lToAddToLiquidity;
1056             //add liquidity

```

```

974
975     function restoreAllFee() private {
976         _taxFee = _previousTaxFee;
977         _liquidityFee = _previousLiquidityFee;
978     }
979
980     function isExcludedFromFee(address account)
981     public view returns(bool) {
982         return _isExcludedFromFee[account];
983     }
984
985     function _approve(address owner, address sp
986     ender, uint256 amount) private {
987         require(owner != address(0), "ERC20: ap
988         prove from the zero address");
989         require(spender != address(0), "ERC20:
990         approve to the zero address");
991
992         _allowances[owner][spender] = amount;
993         emit Approval(owner, spender, amount);
994     }
995
996     function _transfer(
997     address from,
998     address to,
999     uint256 amount
1000     ) private {
1001         require(from != address(0), "ERC20: tra
1002         nsfer from the zero address");
1003         require(to != address(0), "ERC20: trans
1004         fer to the zero address");
1005         require(amount > 0, "Transfer amount mu
1006         st be greater than zero");
1007         if(from != owner() && to != owner())
1008             require(amount <= _maxTxAmount, "Tr
1009             ansfer amount exceeds the maxTxAmount.");
1010
1011         // is the token balance of this contrac
1012         t address over the min number of
1013         // tokens that we need to initiate a sw
1014         ap + liquidity lock?
1015         // also, don't get caught in a circular
1016         liquidity event.
1017         // also, don't swap & liquify if sender
1018         is uniswap pair.
1019         uint256 contractTokenBalance = balanceO
1020         f(address(this));
1021
1022         if(contractTokenBalance >= _maxTxAmoun
1023         t)
1024         {
1025             contractTokenBalance = _maxTxAmoun
1026             t;
1027         }
1028
1029         bool overMinTokenBalance = contractToke
1030         nBalance >= numTokensSellToAddToLiquidity;
1031         if (
1032             overMinTokenBalance &&
1033             !inSwapAndLiquify &&
1034             from != uniswapV2Pair &&
1035             swapAndLiquifyEnabled
1036         ) {
1037             contractTokenBalance = numTokensSel
1038             lToAddToLiquidity;
1039             //add liquidity

```

```

1040         swapAndLiquify(contractTokenBalance);
1041     }
1042
1043     //indicates if fee should be deducted from transfer
1044     bool takeFee = true;
1045
1046     //if any account belongs to _isExcludedFromFee account then remove the fee
1047     if(!_isExcludedFromFee[from] || !_isExcludedFromFee[to]){
1048         takeFee = false;
1049     }
1050
1051     //transfer amount, it will take tax, burn, liquidity fee
1052     _tokenTransfer(from,to,amount,takeFee);
1053 }
1054
1055 function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1056     // split the contract balance into halves
1057     uint256 half = contractTokenBalance.div(2);
1058     uint256 otherHalf = contractTokenBalance.sub(half);
1059
1060     // capture the contract's current ETH balance.
1061     // this is so that we can capture exactly the amount of ETH that the
1062     // swap creates, and not make the liquidity event include any ETH that
1063     // has been manually sent to the contract
1064     uint256 initialBalance = address(this).balance;
1065
1066     // swap tokens for ETH
1067     swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
1068
1069     // how much ETH did we just swap into?
1070     uint256 newBalance = address(this).balance.sub(initialBalance);
1071
1072     // add liquidity to uniswap
1073     addLiquidity(otherHalf, newBalance);
1074
1075     emit SwapAndLiquify(half, newBalance, otherHalf);
1076 }
1077
1078 function swapTokensForEth(uint256 tokenAmount) private {
1079     // generate the uniswap pair path of token -> weth
1080     address[] memory path = new address[](2);
1081     path[0] = address(this);
1082     path[1] = uniswapV2Router.WETH();
1083
1084     _approve(address(this), address(uniswapV2Router), tokenAmount);

```

```

1023         swapAndLiquify(contractTokenBalance);
1024     }
1025
1026     //indicates if fee should be deducted from transfer
1027     bool takeFee = true;
1028
1029     //if any account belongs to _isExcludedFromFee account then remove the fee
1030     if(!_isExcludedFromFee[from] || !_isExcludedFromFee[to]){
1031         takeFee = false;
1032     }
1033
1034     //transfer amount, it will take tax, burn, liquidity fee
1035     _tokenTransfer(from,to,amount,takeFee);
1036 }
1037
1038 function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1039     // split the contract balance into halves
1040     uint256 half = contractTokenBalance.div(2);
1041     uint256 otherHalf = contractTokenBalance.sub(half);
1042
1043     // capture the contract's current ETH balance.
1044     // this is so that we can capture exactly the amount of ETH that the
1045     // swap creates, and not make the liquidity event include any ETH that
1046     // has been manually sent to the contract
1047     uint256 initialBalance = address(this).balance;
1048
1049     // swap tokens for ETH
1050     swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
1051
1052     // how much ETH did we just swap into?
1053     uint256 newBalance = address(this).balance.sub(initialBalance);
1054
1055     // add liquidity to uniswap
1056     addLiquidity(otherHalf, newBalance);
1057
1058     emit SwapAndLiquify(half, newBalance, otherHalf);
1059 }
1060
1061 function swapTokensForEth(uint256 tokenAmount) private {
1062     // generate the uniswap pair path of token -> weth
1063     address[] memory path = new address[](2);
1064     path[0] = address(this);
1065     path[1] = uniswapV2Router.WETH();
1066
1067     _approve(address(this), address(uniswapV2Router), tokenAmount);

```

```

1085
1086         // make the swap
1087         uniswapV2Router.swapExactTokensForETHSu
        pportingFeeOnTransferTokens(
1088             tokenAmount,
1089             0, // accept any amount of ETH
1090             path,
1091             address(this),
1092             block.timestamp
1093         );
1094     }
1095
1096     function addLiquidity(uint256 tokenAmount,
        uint256 ethAmount) private {
1097         // approve token transfer to cover all
        possible scenarios
1098         _approve(address(this), address(uniswap
        V2Router), tokenAmount);
1099
1100         // add the liquidity
1101         uniswapV2Router.addLiquidityETH{value:
        ethAmount}(
1102             address(this),
1103             tokenAmount,
1104             0, // slippage is unavoidable
1105             0, // slippage is unavoidable
1106             owner(),
1107             block.timestamp
1108         );
1109     }
1110
1111     //this method is responsible for taking all
        fee, if takeFee is true
1112     function _tokenTransfer(address sender, add
        ress recipient, uint256 amount, bool takeFee) pr
        ivate {
1113         if(!takeFee)
1114             removeAllFee();
1115
1116         if (_isExcluded[sender] && !_isExcluded
        [recipient]) {
1117             _transferFromExcluded(sender, recip
        ient, amount);
1118         } else if (!_isExcluded[sender] && _isE
        xcluded[recipient]) {
1119             _transferToExcluded(sender, recipie
        nt, amount);
1120         } else if (!_isExcluded[sender] && !_is
        Excluded[recipient]) {
1121             _transferStandard(sender, recipien
        t, amount);
1122         } else if (_isExcluded[sender] && _isEx
        cluded[recipient]) {
1123             _transferBothExcluded(sender, recip
        ient, amount);
1124         } else {
1125             _transferStandard(sender, recipien
        t, amount);
1126         }
1127
1128         if(!takeFee)
1129             restoreAllFee();
1130     }
1131
1132     function _transferStandard(address sender,
        address recipient, uint256 tAmount) private {
1133

```

```

1068
1069         // make the swap
1070         uniswapV2Router.swapExactTokensForETHSu
        pportingFeeOnTransferTokens(
1071             tokenAmount,
1072             0, // accept any amount of ETH
1073             path,
1074             address(this),
1075             block.timestamp
1076         );
1077     }
1078
1079     function addLiquidity(uint256 tokenAmount,
        uint256 ethAmount) private {
1080         // approve token transfer to cover all
        possible scenarios
1081         _approve(address(this), address(uniswap
        V2Router), tokenAmount);
1082
1083         // add the liquidity
1084         uniswapV2Router.addLiquidityETH{value:
        ethAmount}(
1085             address(this),
1086             tokenAmount,
1087             0, // slippage is unavoidable
1088             0, // slippage is unavoidable
1089             owner(),
1090             block.timestamp
1091         );
1092     }
1093
1094     //this method is responsible for taking all
        fee, if takeFee is true
1095     function _tokenTransfer(address sender, add
        ress recipient, uint256 amount, bool takeFee) pr
        ivate {
1096         if(!takeFee)
1097             removeAllFee();
1098
1099         if (_isExcluded[sender] && !_isExcluded
        [recipient]) {
1100             _transferFromExcluded(sender, recip
        ient, amount);
1101         } else if (!_isExcluded[sender] && _isE
        xcluded[recipient]) {
1102             _transferToExcluded(sender, recipie
        nt, amount);
1103         } else if (!_isExcluded[sender] && !_is
        Excluded[recipient]) {
1104             _transferStandard(sender, recipien
        t, amount);
1105         } else if (_isExcluded[sender] && _isEx
        cluded[recipient]) {
1106             _transferBothExcluded(sender, recip
        ient, amount);
1107         } else {
1108             _transferStandard(sender, recipien
        t, amount);
1109         }
1110
1111         if(!takeFee)
1112             restoreAllFee();
1113     }
1114
1115     function _transferStandard(address sender,
        address recipient, uint256 tAmount) private {
1116

```



```

        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1134     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1135     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1136     _takeLiquidity(tLiquidity);
1137     _reflectFee(rFee, tFee);
1138     emit Transfer(sender, recipient, tTransferAmount);
1139 }
1140
1141 function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
1142     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1143     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1144     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1145     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1146     _takeLiquidity(tLiquidity);
1147     _reflectFee(rFee, tFee);
1148     emit Transfer(sender, recipient, tTransferAmount);
1149 }
1150
1151 function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
1152     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1153     _tOwned[sender] = _tOwned[sender].sub(tAmount);
1154     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1155     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1156     _takeLiquidity(tLiquidity);
1157     _reflectFee(rFee, tFee);
1158     emit Transfer(sender, recipient, tTransferAmount);
1159 }
1160
1161
1162
1163
1164 }

```

```

        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1117     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1118     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1119     _takeLiquidity(tLiquidity);
1120     _reflectFee(rFee, tFee);
1121     emit Transfer(sender, recipient, tTransferAmount);
1122 }
1123
1124 function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
1125     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1126     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1127     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1128     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1129     _takeLiquidity(tLiquidity);
1130     _reflectFee(rFee, tFee);
1131     emit Transfer(sender, recipient, tTransferAmount);
1132 }
1133
1134 function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
1135     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1136     _tOwned[sender] = _tOwned[sender].sub(tAmount);
1137     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1138     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1139     _takeLiquidity(tLiquidity);
1140     _reflectFee(rFee, tFee);
1141     emit Transfer(sender, recipient, tTransferAmount);
1142 }
1143
1144
1145
1146
1147 }

```