

```

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
32
33 // SPDX-License-Identifier: Unlicensed
34 interface IERC20 {
35
36     function totalSupply() external view return
37     s (uint256);
38
39     /**
40      * @dev Returns the amount of tokens owned
41      by `account`.
42
43      */
44     function balanceOf(address account) externa
45     l view returns (uint256);
46
47     /**
48      * @dev Moves `amount` tokens from the call
49     er's account to `recipient`.
50
51      *
52      * Returns a boolean value indicating wheth

```

```

1 /**
2  *Submitted for verification at BscScan.com on
3  2021-04-13
4  */
5 /**
6  _____
7  .--  --.
8  ./  ()  .-. \.
9  /  o  .  ( ) \
10 /  .  ' - ' \
11 | ( )  .  0  .  |
12 |  _____  |
13
14 |  o  ( )  |
15 |  .--.  0  |
16 |  .  |  |  |  |
17 \  `_._'  o  .  /  Moonboys will take
18 you and everyone you know to the moon
19
20 \  _____  /
21
22 \  o  ( )  /'
23 `--__  __--'
24
25 _____
26
27
28
29
30 Moonboys is a deflationary token it is a Fork
31 of the SafeMoon token
32
33
34 to begin there is a 25% burn which is constant
35 ly growing due to the redistribution feature
36
37 on every transaction there is a 10% tax, 4% ad
38 ded to liquidity and 6% distributed to holders
39 to ensure your balance is constantly growing a
40 nd bots get annihilated :)
41
42
43 */
44
45 pragma solidity ^0.6.12;
46
47 // SPDX-License-Identifier: Unlicensed
48 interface IERC20 {
49
50     function totalSupply() external view return
51     s (uint256);
52
53     /**
54      * @dev Returns the amount of tokens owned
55      by `account`.
56
57      */
58     function balanceOf(address account) externa
59     l view returns (uint256);
60
61     /**
62      * @dev Moves `amount` tokens from the call
63     er's account to `recipient`.
64
65      *
66      * Returns a boolean value indicating wheth

```

```

er the operation succeeded.
37     *
38     * Emits a {Transfer} event.
39     */
40     function transfer(address recipient, uint25
6 amount) external returns (bool);
41
42     /**
43     * @dev Returns the remaining number of tok
ens that `spender` will be
44     * allowed to spend on behalf of `owner` th
rough {transferFrom}. This is
45     * zero by default.
46     *
47     * This value changes when {approve} or {tr
ansferFrom} are called.
48     */
49     function allowance(address owner, address s
pender) 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 wheth
er the operation succeeded.
55     *
56     * IMPORTANT: Beware that changing an allow
ance with this method brings the risk
57     * that someone may use both the old and th
e new allowance by unfortunate
58     * transaction ordering. One possible solut
ion to mitigate this race
59     * condition is to first reduce the spende
r'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 a
mount) external returns (bool);
66
67     /**
68     * @dev Moves `amount` tokens from `sender`
to `recipient` using the
69     * allowance mechanism. `amount` is then de
ducted from the caller's
70     * allowance.
71     *
72     * Returns a boolean value indicating wheth
er the operation succeeded.
73     *
74     * Emits a {Transfer} event.
75     */
76     function transferFrom(address sender, addre
ss recipient, uint256 amount) external returns
(bool);
77
78     /**
79     * @dev Emitted when `value` tokens are mov
ed from one account (`from`) to
80     * another (`to`).
81     *
82     * Note that `value` may be zero.

```

```

er the operation succeeded.
44     *
45     * Emits a {Transfer} event.
46     */
47     function transfer(address recipient, uint25
6 amount) external returns (bool);
48
49     /**
50     * @dev Returns the remaining number of tok
ens that `spender` will be
51     * allowed to spend on behalf of `owner` th
rough {transferFrom}. This is
52     * zero by default.
53     *
54     * This value changes when {approve} or {tr
ansferFrom} are called.
55     */
56     function allowance(address owner, address s
pender) external view returns (uint256);
57
58     /**
59     * @dev Sets `amount` as the allowance of `
spender` over the caller's tokens.
60     *
61     * Returns a boolean value indicating wheth
er the operation succeeded.
62     *
63     * IMPORTANT: Beware that changing an allow
ance with this method brings the risk
64     * that someone may use both the old and th
e new allowance by unfortunate
65     * transaction ordering. One possible solut
ion to mitigate this race
66     * condition is to first reduce the spende
r's allowance to 0 and set the
67     * desired value afterwards:
68     * https://github.com/ethereum/EIPs/issues/
20#issuecomment-263524729
69     *
70     * Emits an {Approval} event.
71     */
72     function approve(address spender, uint256 a
mount) external returns (bool);
73
74     /**
75     * @dev Moves `amount` tokens from `sender`
to `recipient` using the
76     * allowance mechanism. `amount` is then de
ducted from the caller's
77     * allowance.
78     *
79     * Returns a boolean value indicating wheth
er the operation succeeded.
80     *
81     * Emits a {Transfer} event.
82     */
83     function transferFrom(address sender, addre
ss recipient, uint256 amount) external returns
(bool);
84
85     /**
86     * @dev Emitted when `value` tokens are mov
ed from one account (`from`) to
87     * another (`to`).
88     *
89     * 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);
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      *

```

```

90     */
91     event Transfer(address indexed from, address indexed to, uint256 value);
92
93     /**
94      * @dev Emitted when the allowance of a `spender` for an `owner` is set by
95      * a call to {approve}. `value` is the new allowance.
96      */
97     event Approval(address indexed owner, address indexed spender, uint256 value);
98 }
99
100
101
102 /**
103  * @dev Wrappers over Solidity's arithmetic operations with added overflow
104  * checks.
105  *
106  * Arithmetic operations in Solidity wrap on overflow. This can easily result
107  * in bugs, because programmers usually assume that an overflow raises an
108  * error, which is the standard behavior in high level programming languages.
109  * `SafeMath` restores this intuition by reverting the transaction when an
110  * operation overflows.
111  *
112  * Using this library instead of the unchecked operations eliminates an entire
113  * class of bugs, so it's recommended to use it always.
114  */
115
116 library SafeMath {
117     /**
118      * @dev Returns the addition of two unsigned integers, reverting on
119      * overflow.
120      *
121      * Counterpart to Solidity's `+` operator.
122      *
123      * Requirements:
124      *
125      * - Addition cannot overflow.
126      */
127     function add(uint256 a, uint256 b) internal pure returns (uint256) {
128         uint256 c = a + b;
129         require(c >= a, "SafeMath: addition overflow");
130
131         return c;
132     }
133
134     /**
135      * @dev Returns the subtraction of two unsigned integers, reverting on
136      * overflow (when the result is negative).
137      *
138      * Counterpart to Solidity's `-` operator.
139      *
140      * Requirements:
141      *

```

```

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 unsi
    gned integers, reverting with custom message on
143     * 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 m
    emory errorMessage) internal pure returns (uint
    256) {
152         require(b <= a, errorMessage);
153         uint256 c = a - b;
154
155         return c;
156     }
157
158     /**
159     * @dev Returns the multiplication of two u
    nsigned 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 th
    an requiring 'a' not being zero, but the
170         // benefit is lost if 'b' is also teste
    d.
171         // See: https://github.com/OpenZeppeli
    n/openzeppelin-contracts/pull/522
172         if (a == 0) {
173             return 0;
174         }
175
176         uint256 c = a * b;
177         require(c / a == b, "SafeMath: multipli
    cation 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     *
188     * Note: this function uses a
189     * 'revert' opcode (which leaves remaining

```

```

142     * - Subtraction cannot overflow.
143     */
144     function sub(uint256 a, uint256 b) internal
    pure returns (uint256) {
145         return sub(a, b, "SafeMath: subtraction
    overflow");
146     }
147
148     /**
149     * @dev Returns the subtraction of two unsi
    gned integers, reverting with custom message on
150     * overflow (when the result is negative).
151     *
152     * Counterpart to Solidity's '-' operator.
153     *
154     * Requirements:
155     *
156     * - Subtraction cannot overflow.
157     */
158     function sub(uint256 a, uint256 b, string m
    emory errorMessage) internal pure returns (uint
    256) {
159         require(b <= a, errorMessage);
160         uint256 c = a - b;
161
162         return c;
163     }
164
165     /**
166     * @dev Returns the multiplication of two u
    nsigned integers, reverting on
167     * overflow.
168     *
169     * Counterpart to Solidity's '*' operator.
170     *
171     * Requirements:
172     *
173     * - Multiplication cannot overflow.
174     */
175     function mul(uint256 a, uint256 b) internal
    pure returns (uint256) {
176         // Gas optimization: this is cheaper th
    an requiring 'a' not being zero, but the
177         // benefit is lost if 'b' is also teste
    d.
178         // See: https://github.com/OpenZeppeli
    n/openzeppelin-contracts/pull/522
179         if (a == 0) {
180             return 0;
181         }
182
183         uint256 c = a * b;
184         require(c / a == b, "SafeMath: multipli
    cation overflow");
185
186         return c;
187     }
188
189     /**
190     * @dev Returns the integer division of two
    unsigned integers. Reverts on
191     * division by zero. The result is rounded
    towards zero.
192     *
193     * Counterpart to Solidity's '/' operator.
194     *
195     * Note: this function uses a
196     * 'revert' opcode (which leaves remaining

```

```

    gas untouched) while Solidity
188     * uses an invalid opcode to revert (consuming all remaining gas).
189     *
190     * Requirements:
191     *
192     * - The divisor cannot be zero.
193     */
194     function div(uint256 a, uint256 b) internal pure returns (uint256) {
195         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 (consuming all remaining gas).
205     *
206     * Requirements:
207     *
208     * - The divisor cannot be zero.
209     */
210     function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
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 two unsigned integers. (unsigned integer modulo),
220     * Reverts when dividing by zero.
221     *
222     * Counterpart to Solidity's `%` operator. This function uses a `revert`
223     * opcode (which leaves remaining gas untouched) 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 zero");
232     }
233

```

```

    gas untouched) while Solidity
195     * uses an invalid opcode to revert (consuming all remaining gas).
196     *
197     * Requirements:
198     *
199     * - The divisor cannot be zero.
200     */
201     function div(uint256 a, uint256 b) internal pure returns (uint256) {
202         return div(a, b, "SafeMath: division by zero");
203     }
204
205     /**
206     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
207     * division by zero. The result is rounded towards zero.
208     *
209     * Counterpart to Solidity's `/` operator. Note: this function uses a
210     * `revert` opcode (which leaves remaining gas untouched) while Solidity
211     * uses an invalid opcode to revert (consuming all remaining gas).
212     *
213     * Requirements:
214     *
215     * - The divisor cannot be zero.
216     */
217     function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
218         require(b > 0, errorMessage);
219         uint256 c = a / b;
220         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
221
222         return c;
223     }
224
225     /**
226     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
227     * Reverts when dividing by zero.
228     *
229     * Counterpart to Solidity's `%` operator. This function uses a `revert`
230     * opcode (which leaves remaining gas untouched) while Solidity uses an
231     * invalid opcode to revert (consuming all remaining gas).
232     *
233     * Requirements:
234     *
235     * - The divisor cannot be zero.
236     */
237     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
238         return mod(a, b, "SafeMath: modulo by zero");
239     }
240

```

```

234     /**
235      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
236      * Reverts with custom message when dividing by zero.
237      *
238      * Counterpart to Solidity's `%` operator. This function uses a `revert`
239      * opcode (which leaves remaining gas untouched) 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 memory errorMessage) internal pure returns (uint256) {
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 returns (bytes memory) {
258         this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
259         return msg.data;
260     }
261 }
262
263
264 /**
265  * @dev Collection of functions related to the address type
266  */
267 library Address {
268     /**
269      * @dev Returns true if `account` is a contract.
270      *
271      * [IMPORTANT]
272      * =====
273      * It is unsafe to assume that an address for which this function returns
274      * false is an externally-owned account (EOA) and not a contract.
275      *
276      * Among others, `isContract` will return false 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 created

```

```

241     /**
242      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
243      * Reverts with custom message when dividing by zero.
244      *
245      * Counterpart to Solidity's `%` operator. This function uses a `revert`
246      * opcode (which leaves remaining gas untouched) while Solidity uses an
247      * invalid opcode to revert (consuming all remaining gas).
248      *
249      * Requirements:
250      *
251      * - The divisor cannot be zero.
252      */
253     function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
254         require(b != 0, errorMessage);
255         return a % b;
256     }
257 }
258
259 abstract contract Context {
260     function _msgSender() internal view virtual returns (address payable) {
261         return msg.sender;
262     }
263
264     function _msgData() internal view virtual returns (bytes memory) {
265         this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
266         return msg.data;
267     }
268 }
269
270
271 /**
272  * @dev Collection of functions related to the address type
273  */
274 library Address {
275     /**
276      * @dev Returns true if `account` is a contract.
277      *
278      * [IMPORTANT]
279      * =====
280      * It is unsafe to assume that an address for which this function returns
281      * false is an externally-owned account (EOA) and not a contract.
282      *
283      * Among others, `isContract` will return false for the following
284      * types of addresses:
285      *
286      * - an externally-owned account
287      * - a contract in construction
288      * - an address where a contract will be created

```

```

282     * - an address where a contract lived, but
    t was destroyed
283     * ====
284     */
285     function isContract(address account) internal
    view returns (bool) {
286         // According to EIP-1052, 0x0 is the value
    returned for not-yet created accounts
287         // and 0xc5d2460186f7233c927e7db2dcc703
    c0e500b653ca82273b7bfad8045d85a470 is returned
288         // for accounts without code, i.e. `keccak256('')`
289         bytes32 codehash;
290         bytes32 accountHash = 0xc5d2460186f7233
    c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
    70;
291         // solhint-disable-next-line no-inline-
    assembly
292         assembly { codehash := extcodehash(account) }
293         return (codehash != accountHash && code
    hash != 0x0);
294     }
295
296     /**
297     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
298     * `recipient`, forwarding all available gas and reverting on errors.
299     *
300     * https://eips.ethereum.org/EIPS/eip-1884 [EIP1884] increases the gas cost
301     * of certain opcodes, possibly making contracts go over the 2300 gas limit
302     * imposed by `transfer`, making them unable to receive funds via
303     * `transfer`. {sendValue} removes this limitation.
304     *
305     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
306     *
307     * IMPORTANT: because control is transferred to `recipient`, care must be
308     * taken to not create reentrancy vulnerabilities. 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-interactions pattern].
311     */
312     function sendValue(address payable recipient, uint256 amount) internal {
313         require(address(this).balance >= amount, "Address: insufficient balance");
314
315         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
316         (bool success, ) = recipient.call{ value: amount }("");
317         require(success, "Address: unable to send value, recipient may have reverted");
318     }
319

```

```

289     * - an address where a contract lived, but
    t was destroyed
290     * ====
291     */
292     function isContract(address account) internal
    view returns (bool) {
293         // According to EIP-1052, 0x0 is the value
    returned for not-yet created accounts
294         // and 0xc5d2460186f7233c927e7db2dcc703
    c0e500b653ca82273b7bfad8045d85a470 is returned
295         // for accounts without code, i.e. `keccak256('')`
296         bytes32 codehash;
297         bytes32 accountHash = 0xc5d2460186f7233
    c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
    70;
298         // solhint-disable-next-line no-inline-
    assembly
299         assembly { codehash := extcodehash(account) }
300         return (codehash != accountHash && code
    hash != 0x0);
301     }
302
303     /**
304     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
305     * `recipient`, forwarding all available gas and reverting on errors.
306     *
307     * https://eips.ethereum.org/EIPS/eip-1884 [EIP1884] increases the gas cost
308     * of certain opcodes, possibly making contracts go over the 2300 gas limit
309     * imposed by `transfer`, making them unable to receive funds via
310     * `transfer`. {sendValue} removes this limitation.
311     *
312     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
313     *
314     * IMPORTANT: because control is transferred to `recipient`, care must be
315     * taken to not create reentrancy vulnerabilities. Consider using
316     * {ReentrancyGuard} or the
317     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
318     */
319     function sendValue(address payable recipient, uint256 amount) internal {
320         require(address(this).balance >= amount, "Address: insufficient balance");
321
322         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
323         (bool success, ) = recipient.call{ value: amount }("");
324         require(success, "Address: unable to send value, recipient may have reverted");
325     }
326

```

```

320     /**
321      * @dev Performs a Solidity function call using 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 reason, it is bubbled up by this
326      * function (like regular Solidity function calls).
327      *
328      * Returns the raw returned data. To convert to the expected return value,
329      * use https://solidity.readthedocs.io/en/latest/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, "Address: 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 reason when `target` reverts.
345      *
346      * _Available since v3.1._
347      */
348     function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
349         return _functionCallWithValue(target, data, 0, errorMessage);
350     }
351
352     /**
353      * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
354      * but also transferring `value` wei to `target`.
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 target, bytes memory data, uint256 value) internal returns (bytes memory) {

```

```

327     /**
328      * @dev Performs a Solidity function call using a low level `call`. A
329      * plain `call` is an unsafe replacement for a function call: use this
330      * function instead.
331      *
332      * If `target` reverts with a revert reason, it is bubbled up by this
333      * function (like regular Solidity function calls).
334      *
335      * Returns the raw returned data. To convert to the expected return value,
336      * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions [abi.decode].
337      *
338      * Requirements:
339      *
340      * - `target` must be a contract.
341      * - calling `target` with `data` must not revert.
342      *
343      * _Available since v3.1._
344      */
345     function functionCall(address target, bytes memory data) internal returns (bytes memory) {
346         return functionCall(target, data, "Address: low-level call failed");
347     }
348
349     /**
350      * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
351      * `errorMessage` as a fallback revert reason when `target` reverts.
352      *
353      * _Available since v3.1._
354      */
355     function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
356         return _functionCallWithValue(target, data, 0, errorMessage);
357     }
358
359     /**
360      * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
361      * but also transferring `value` wei to `target`.
362      *
363      * Requirements:
364      *
365      * - the calling contract must have an ETH balance of at least `value`.
366      * - the called Solidity function must be `payable`.
367      *
368      * _Available since v3.1._
369      */
370     function functionCallWithValue(address target, 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.
370     *
371     * _Available since v3.1._
372     */
373     function functionCallWithValue(address targ
et, bytes memory data, uint256 value, string me
memory errorMessage) internal returns (bytes memo
ry) {
374         require(address(this).balance >= value,
"Address: insufficient balance for call");
375         return _functionCallWithValue(target, d
ata, value, errorMessage);
376     }
377
378     function _functionCallWithValue(address targ
et, bytes memory data, uint256 weiValue, strin
g memory errorMessage) private returns (bytes m
emory) {
379         require(isContract(target), "Address: c
all 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 bubbl
e it up if present
387             if (returndata.length > 0) {
388                 // The easiest way to bubble th
e revert reason is using memory via assembly
389
390                 // solhint-disable-next-line no
-inline-assembly
391                 assembly {
392                     let returndata_size := mloa
d(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 g
ranted exclusive access to
405 * specific functions.
406 *
407

```

```

371         return functionCallWithValue(target, da
ta, value, "Address: low-level call with value
failed");
372     }
373
374     /**
375     * @dev Same as {xref-Address-functionCallW
ithValue-address-bytes-uint256-}[`functionCallW
ithValue`], but
376     * with `errorMessage` as a fallback revert
reason when `target` reverts.
377     *
378     * _Available since v3.1._
379     */
380     function functionCallWithValue(address targ
et, bytes memory data, uint256 value, string me
memory errorMessage) internal returns (bytes memo
ry) {
381         require(address(this).balance >= value,
"Address: insufficient balance for call");
382         return _functionCallWithValue(target, d
ata, value, errorMessage);
383     }
384
385     function _functionCallWithValue(address targ
et, bytes memory data, uint256 weiValue, strin
g memory errorMessage) private returns (bytes m
emory) {
386         require(isContract(target), "Address: c
all to non-contract");
387
388         // solhint-disable-next-line avoid-low-
level-calls
389         (bool success, bytes memory returndata)
= target.call{ value: weiValue }(data);
390         if (success) {
391             return returndata;
392         } else {
393             // Look for revert reason and bubbl
e it up if present
394             if (returndata.length > 0) {
395                 // The easiest way to bubble th
e revert reason is using memory via assembly
396
397                 // solhint-disable-next-line no
-inline-assembly
398                 assembly {
399                     let returndata_size := mloa
d(returndata)
400                     revert(add(32, returndata),
returndata_size)
401                 }
402             } else {
403                 revert(errorMessage);
404             }
405         }
406     }
407 }
408
409 /**
410 * @dev Contract module which provides a basic
access control mechanism, where
411 * there is an account (an owner) that can be g
ranted exclusive access to
412 * specific functions.
413 *
414

```

```

    * 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 */
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 the deployer as the initial owner.
423      */
424     constructor () internal {
425         address msgSender = _msgSender();
426         _owner = msgSender;
427         emit OwnershipTransferred(address(0), msgSender);
428     }
429
430     /**
431      * @dev Returns the address of the current owner.
432      */
433     function owner() public view returns (address) {
434         return _owner;
435     }
436
437     /**
438      * @dev Throws if called by any account other than the owner.
439      */
440     modifier onlyOwner() {
441         require(_owner == _msgSender(), "Ownable: 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 the 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, address(0));
454         _owner = address(0);
455     }
456

```

```

    * By default, the owner account will be the one that deploys the contract. This
415 * can later be changed with {transferOwnership}.
416 *
417 * This module is used through inheritance. It will make available the modifier
418 * `onlyOwner`, which can be applied to your functions to restrict their use to
419 * the owner.
420 */
421 contract Ownable is Context {
422     address private _owner;
423     address private _previousOwner;
424     uint256 private _lockTime;
425
426     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
427
428     /**
429      * @dev Initializes the contract setting the deployer as the initial owner.
430      */
431     constructor () internal {
432         address msgSender = _msgSender();
433         _owner = msgSender;
434         emit OwnershipTransferred(address(0), msgSender);
435     }
436
437     /**
438      * @dev Returns the address of the current owner.
439      */
440     function owner() public view returns (address) {
441         return _owner;
442     }
443
444     /**
445      * @dev Throws if called by any account other than the owner.
446      */
447     modifier onlyOwner() {
448         require(_owner == _msgSender(), "Ownable: caller is not the owner");
449         _;
450     }
451
452     /**
453      * @dev Leaves the contract without owner. It will not be possible to call
454      * `onlyOwner` functions anymore. Can only be called by the current owner.
455      *
456      * NOTE: Renouncing ownership will leave the contract without an owner,
457      * thereby removing any functionality that is only available to the owner.
458      */
459     function renounceOwnership() public virtual onlyOwner {
460         emit OwnershipTransferred(_owner, address(0));
461         _owner = address(0);
462     }
463

```

```

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 newOwner) public virtual onlyOwner {
462       require(newOwner != address(0), "Ownable: new owner is the zero address");
463       emit OwnershipTransferred(_owner, newOwner);
464       _owner = newOwner;
465   }
466
467   function getUnlockTime() public view returns (uint256) {
468       return _lockTime;
469   }
470
471   //Locks the contract for owner for the amount 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, address(0));
477   }
478
479   //Unlocks the contract for owner when _lockTime is exceeded
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, _previousOwner);
484       _owner = _previousOwner;
485   }
486 }
487
488 // pragma solidity >=0.5.0;
489
490 interface IUniswapV2Factory {
491     event PairCreated(address indexed token0, address indexed token1, address pair, uint);
492
493     function feeTo() external view returns (address);
494     function feeToSetter() external view returns (address);
495
496     function getPair(address tokenA, address tokenB) external view returns (address pair);
497     function allPairs(uint) external view returns (address pair);
498     function allPairsLength() external view returns (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 }

```

```

464  /**
465   * @dev Transfers ownership of the contract
to a new account (`newOwner`).
466   * Can only be called by the current owner.
467   */
468   function transferOwnership(address newOwner) public virtual onlyOwner {
469       require(newOwner != address(0), "Ownable: new owner is the zero address");
470       emit OwnershipTransferred(_owner, newOwner);
471       _owner = newOwner;
472   }
473
474   function getUnlockTime() public view returns (uint256) {
475       return _lockTime;
476   }
477
478   //Locks the contract for owner for the amount of time provided
479   function lock(uint256 time) public virtual onlyOwner {
480       _previousOwner = _owner;
481       _owner = address(0);
482       _lockTime = now + time;
483       emit OwnershipTransferred(_owner, address(0));
484   }
485
486   //Unlocks the contract for owner when _lockTime is exceeded
487   function unlock() public virtual {
488       require(_previousOwner == msg.sender, "You don't have permission to unlock");
489       require(now > _lockTime, "Contract is locked");
490       emit OwnershipTransferred(_owner, _previousOwner);
491       _owner = _previousOwner;
492   }
493 }
494
495 // pragma solidity >=0.5.0;
496
497 interface IUniswapV2Factory {
498     event PairCreated(address indexed token0, address indexed token1, address pair, uint);
499
500     function feeTo() external view returns (address);
501     function feeToSetter() external view returns (address);
502
503     function getPair(address tokenA, address tokenB) external view returns (address pair);
504     function allPairs(uint) external view returns (address pair);
505     function allPairsLength() external view returns (uint);
506
507     function createPair(address tokenA, address tokenB) external returns (address pair);
508
509     function setFeeTo(address) external;
510     function setFeeToSetter(address) external;
511 }

```

```

505
506
507 // pragma solidity >=0.5.0;
508
509 interface IUniswapV2Pair {
510     event Approval(address indexed owner, address indexed spender, uint value);
511     event Transfer(address indexed from, address indexed to, uint value);
512
513     function name() external pure returns (string memory);
514     function symbol() external pure returns (string memory);
515     function decimals() external pure returns (uint8);
516     function totalSupply() external view returns (uint);
517     function balanceOf(address owner) external view returns (uint);
518     function allowance(address owner, address spender) external view returns (uint);
519
520     function approve(address spender, uint value) external returns (bool);
521     function transfer(address to, uint value) external returns (bool);
522     function transferFrom(address from, address to, uint value) external returns (bool);
523
524     function DOMAIN_SEPARATOR() external view returns (bytes32);
525     function PERMIT_TYPEHASH() external pure returns (bytes32);
526     function nonces(address owner) external view returns (uint);
527
528     function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
529
530     event Mint(address indexed sender, uint amount0, uint amount1);
531     event Burn(address indexed sender, uint amount0, 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 reserve1);
541
542     function MINIMUM_LIQUIDITY() external pure returns (uint);
543     function factory() external view returns (address);
544     function token0() external view returns (address);
545     function token1() external view returns (address);
546     function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
547

```

```

512
513
514 // pragma solidity >=0.5.0;
515
516 interface IUniswapV2Pair {
517     event Approval(address indexed owner, address indexed spender, uint value);
518     event Transfer(address indexed from, address indexed to, uint value);
519
520     function name() external pure returns (string memory);
521     function symbol() external pure returns (string memory);
522     function decimals() external pure returns (uint8);
523     function totalSupply() external view returns (uint);
524     function balanceOf(address owner) external view returns (uint);
525     function allowance(address owner, address spender) external view returns (uint);
526
527     function approve(address spender, uint value) external returns (bool);
528     function transfer(address to, uint value) external returns (bool);
529     function transferFrom(address from, address to, uint value) external returns (bool);
530
531     function DOMAIN_SEPARATOR() external view returns (bytes32);
532     function PERMIT_TYPEHASH() external pure returns (bytes32);
533     function nonces(address owner) external view returns (uint);
534
535     function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
536
537     event Mint(address indexed sender, uint amount0, uint amount1);
538     event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
539     event Swap(
540         address indexed sender,
541         uint amount0In,
542         uint amount1In,
543         uint amount0Out,
544         uint amount1Out,
545         address indexed to
546     );
547     event Sync(uint112 reserve0, uint112 reserve1);
548
549     function MINIMUM_LIQUIDITY() external pure returns (uint);
550     function factory() external view returns (address);
551     function token0() external view returns (address);
552     function token1() external view returns (address);
553     function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
554

```

```

        function price0CumulativeLast() external view returns (uint);
548     function price1CumulativeLast() external view returns (uint);
549     function kLast() external view returns (uint);
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 amount1Out, 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, uint liquidity);
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, uint liquidity);

```

```

        function price0CumulativeLast() external view returns (uint);
555     function price1CumulativeLast() external view returns (uint);
556     function kLast() external view returns (uint);
557
558     function mint(address to) external returns (uint liquidity);
559     function burn(address to) external returns (uint amount0, uint amount1);
560     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
561     function skim(address to) external;
562     function sync() external;
563
564     function initialize(address, address) external;
565 }
566
567 // pragma solidity >=0.6.2;
568
569 interface IUniswapV2Router01 {
570     function factory() external pure returns (address);
571     function WETH() external pure returns (address);
572
573     function addLiquidity(
574         address tokenA,
575         address tokenB,
576         uint amountADesired,
577         uint amountBDesired,
578         uint amountAMin,
579         uint amountBMin,
580         address to,
581         uint deadline
582     ) external returns (uint amountA, uint amountB, uint liquidity);
583     function addLiquidityETH(
584         address token,
585         uint amountTokenDesired,
586         uint amountTokenMin,
587         uint amountETHMin,
588         address to,
589         uint deadline
590     ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
591     function removeLiquidity(
592         address tokenA,
593         address tokenB,
594         uint liquidity,
595         uint amountAMin,
596         uint amountBMin,
597         address to,
598         uint deadline
599     ) external returns (uint amountA, uint amountB, uint liquidity);
600     function removeLiquidityETH(
601         address token,
602         uint liquidity,
603         uint amountTokenMin,
604         uint amountETHMin,
605         address to,
606         uint deadline
607     ) external returns (uint amountToken, uint amountETH, uint liquidity);

```

```

        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, by
tes32 s
610     ) external returns (uint amountA, uint amou
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

```

```

        amountETH);
608     function removeLiquidityWithPermit(
609         address tokenA,
610         address tokenB,
611         uint liquidity,
612         uint amountAMin,
613         uint amountBMin,
614         address to,
615         uint deadline,
616         bool approveMax, uint8 v, bytes32 r, by
tes32 s
617     ) external returns (uint amountA, uint amou
ntB);
618     function removeLiquidityETHWithPermit(
619         address token,
620         uint liquidity,
621         uint amountTokenMin,
622         uint amountETHMin,
623         address to,
624         uint deadline,
625         bool approveMax, uint8 v, bytes32 r, by
tes32 s
626     ) external returns (uint amountToken, uint
amountETH);
627     function swapExactTokensForTokens(
628         uint amountIn,
629         uint amountOutMin,
630         address[] calldata path,
631         address to,
632         uint deadline
633     ) external returns (uint[] memory amounts);
634     function swapTokensForExactTokens(
635         uint amountOut,
636         uint amountInMax,
637         address[] calldata path,
638         address to,
639         uint deadline
640     ) external returns (uint[] memory amounts);
641     function swapExactETHForTokens(uint amountO
utMin, address[] calldata path, address to, uin
t deadline)
642         external
643         payable
644         returns (uint[] memory amounts);
645     function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
address to, uint deadline)
646         external
647         returns (uint[] memory amounts);
648     function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,
address to, uint deadline)
649         external
650         returns (uint[] memory amounts);
651     function swapETHForExactTokens(uint amountO
ut, address[] calldata path, address to, uint d
eadline)
652         external
653         payable
654         returns (uint[] memory amounts);
655
656     function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
657     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
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

```

```

        eserveIn, uint reserveOut) external pure return
s (uint amountOut);
658     function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
659     function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
660     function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
661 }
662
663
664
665 // pragma solidity >=0.6.2;
666
667 interface IUniswapV2Router02 is IUniswapV2Route
r01 {
668     function removeLiquidityETHSupportingFeeOnT
ransferTokens(
669         address token,
670         uint liquidity,
671         uint amountTokenMin,
672         uint amountETHMin,
673         address to,
674         uint deadline
675     ) external returns (uint amountETH);
676     function removeLiquidityETHWithPermitSupport
ingFeeOnTransferTokens(
677         address token,
678         uint liquidity,
679         uint amountTokenMin,
680         uint amountETHMin,
681         address to,
682         uint deadline,
683         bool approveMax, uint8 v, bytes32 r, by
tes32 s
684     ) external returns (uint amountETH);
685
686     function swapExactTokensForTokensSupporting
FeeOnTransferTokens(
687         uint amountIn,
688         uint amountOutMin,
689         address[] calldata path,
690         address to,
691         uint deadline
692     ) external;
693     function swapExactETHForTokensSupportingFee
OnTransferTokens(
694         uint amountOutMin,
695         address[] calldata path,
696         address to,
697         uint deadline
698     ) external payable;
699     function swapExactTokensForETHSupportingFee
OnTransferTokens(
700         uint amountIn,
701         uint amountOutMin,
702         address[] calldata path,
703         address to,
704         uint deadline
705     ) external;
706 }
707
708
709 contract MoonBoys is Context, IERC20, Ownable {

```

```

{
703     using SafeMath for uint256;
704     using Address for address;
705
706     mapping (address => uint256) private _rOwned;
707     mapping (address => uint256) private _tOwned;
708     mapping (address => mapping (address => uint256)) private _allowances;
709
710     mapping (address => bool) private _isExcludedFromFee;
711
712     mapping (address => bool) private _isExcluded;
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 % _tTotal));
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 = _liquidityFee;
729
730     IUniswapV2Router02 public immutable uniswapV2Router;
731     address public immutable uniswapV2Pair;
732     bool inSwapAndLiquify;
733     bool public swapAndLiquifyEnabled = true;
734
735     uint256 public _maxTxAmount = 5000000 * 10**6 * 10**9;
736     uint256 private numTokensSellToAddToLiquidity = 500000 * 10**6 * 10**9;
737
738     event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
739     event SwapAndLiquifyEnabledUpdated(bool enabled);
740     event SwapAndLiquify(
741         uint256 tokensSwapped,
742         uint256 ethReceived,
743         uint256 tokensIntoLiquidity
744     );
745
746     modifier lockTheSwap {
747         inSwapAndLiquify = true;
748         _;
749         inSwapAndLiquify = false;
750     }
751
752     constructor () public {
753         _rOwned[_msgSender()] = _rTotal;

```

```

710     using SafeMath for uint256;
711     using Address for address;
712
713     mapping (address => uint256) private _rOwned;
714     mapping (address => uint256) private _tOwned;
715     mapping (address => mapping (address => uint256)) private _allowances;
716
717     mapping (address => bool) private _isExcludedFromFee;
718
719     mapping (address => bool) private _isExcluded;
720     address[] private _excluded;
721
722     uint256 private constant MAX = ~uint256(0);
723     uint256 private _tTotal = 1000000000 * 10**6 * 10**9;
724     uint256 private _rTotal = (MAX - (MAX % _tTotal));
725     uint256 private _tFeeTotal;
726
727     string private _name = "MoonBoys";
728     string private _symbol = "MBS";
729     uint8 private _decimals = 9;
730
731     uint256 public _taxFee = 6;
732     uint256 private _previousTaxFee = _taxFee;
733
734     uint256 public _liquidityFee = 4;
735     uint256 private _previousLiquidityFee = _liquidityFee;
736
737     IUniswapV2Router02 public immutable uniswapV2Router;
738     address public immutable uniswapV2Pair;
739     bool inSwapAndLiquify;
740     bool public swapAndLiquifyEnabled = true;
741
742     uint256 public _maxTxAmount = 5000000 * 10**6 * 10**9;
743     uint256 private numTokensSellToAddToLiquidity = 500000 * 10**6 * 10**9;
744
745     event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
746     event SwapAndLiquifyEnabledUpdated(bool enabled);
747     event SwapAndLiquify(
748         uint256 tokensSwapped,
749         uint256 ethReceived,
750         uint256 tokensIntoLiquidity
751     );
752
753     modifier lockTheSwap {
754         inSwapAndLiquify = true;
755         _;
756         inSwapAndLiquify = false;
757     }
758
759     constructor () public {
760         _rOwned[_msgSender()] = _rTotal;

```



```

754
755     IUniswapV2Router02 _uniswapV2Router = I
    UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
    9e13aDd608C7F);
756     // Create a uniswap pair for this new
    token
757     uniswapV2Pair = IUniswapV2Factory(_unis
    wapV2Router.factory())
758     .createPair(address(this), _uniswap
    V2Router.WETH());
759
760     // set the rest of the contract variabl
    es
761     uniswapV2Router = _uniswapV2Router;
762
763     //exclude owner and this contract from
    fee
764     _isExcludedFromFee[owner()] = true;
765     _isExcludedFromFee[address(this)] = tru
    e;
766
767     emit Transfer(address(0), _msgSender(),
    _tTotal);
768 }
769
770     function name() public view returns (string
    memory) {
771         return _name;
772     }
773
774     function symbol() public view returns (stri
    ng memory) {
775         return _symbol;
776     }
777
778     function decimals() public view returns (ui
    nt8) {
779         return _decimals;
780     }
781
782     function totalSupply() public view override
    returns (uint256) {
783         return _tTotal;
784     }
785
786     function balanceOf(address account) public
    view override returns (uint256) {
787         if (_isExcluded[account]) return _tOwne
    d[account];
788         return tokenFromReflection(_rOwned[acco
    unt]);
789     }
790
791     function transfer(address recipient, uint25
    6 amount) public override returns (bool) {
792         _transfer(_msgSender(), recipient, amou
    nt);
793         return true;
794     }
795
796     function allowance(address owner, address s
    pender) public view override returns (uint256)
    {
797         return _allowances[owner][spender];
798     }
799
800     function approve(address spender, uint256 a

```

```

762
763     IUniswapV2Router02 _uniswapV2Router = I
    UniswapV2Router02(0x05fF2B0DB69458A0750badebc4f
    9e13aDd608C7F);
764     // Create a uniswap pair for this new
    token
765     uniswapV2Pair = IUniswapV2Factory(_unis
    wapV2Router.factory())
766     .createPair(address(this), _uniswap
    V2Router.WETH());
767
768     // set the rest of the contract variabl
    es
769     uniswapV2Router = _uniswapV2Router;
770
771     //exclude owner and this contract from
    fee
772     _isExcludedFromFee[owner()] = true;
773     _isExcludedFromFee[address(this)] = tru
    e;
774
775     emit Transfer(address(0), _msgSender(),
    _tTotal);
776 }
777
778     function name() public view returns (string
    memory) {
779         return _name;
780     }
781
782     function symbol() public view returns (stri
    ng memory) {
783         return _symbol;
784     }
785
786     function decimals() public view returns (ui
    nt8) {
787         return _decimals;
788     }
789
790     function totalSupply() public view override
    returns (uint256) {
791         return _tTotal;
792     }
793
794     function balanceOf(address account) public
    view override returns (uint256) {
795         if (_isExcluded[account]) return _tOwne
    d[account];
796         return tokenFromReflection(_rOwned[acco
    unt]);
797     }
798
799     function transfer(address recipient, uint25
    6 amount) public override returns (bool) {
800         _transfer(_msgSender(), recipient, amou
    nt);
801         return true;
802     }
803
804     function allowance(address owner, address s
    pender) public view override returns (uint256)
    {
805         return _allowances[owner][spender];
806     }
807
808     function approve(address spender, uint256 a

```

```

    mount) public override returns (bool) {
801         _approve(_msgSender(), spender, amount
            t);
802         return true;
803     }
804
805     function transferFrom(address sender, address
        recipient, uint256 amount) public override r
        eturns (bool) {
806         _transfer(sender, recipient, amount);
807         _approve(sender, _msgSender(), _allowan
            ces[sender][_msgSender()].sub(amount, "ERC20: t
            ransfer amount exceeds allowance"));
808         return true;
809     }
810
811     function increaseAllowance(address spender,
        uint256 addedValue) public virtual returns (boo
        l) {
812         _approve(_msgSender(), spender, _allowa
            nces[_msgSender()][spender].add(addedValue));
813         return true;
814     }
815
816     function decreaseAllowance(address spender,
        uint256 subtractedValue) public virtual returns
        (bool) {
817         _approve(_msgSender(), spender, _allowa
            nces[_msgSender()][spender].sub(subtractedValu
            e, "ERC20: decreased allowance below zero"));
818         return true;
819     }
820
821     function isExcludedFromReward(address accou
        nt) public view returns (bool) {
822         return _isExcluded[account];
823     }
824
825     function totalFees() public view returns (u
        int256) {
826         return _tFeeTotal;
827     }
828
829     function deliver(uint256 tAmount) public {
830         address sender = _msgSender();
831         require(!_isExcluded[sender], "Excluded
            addresses cannot call this function");
832         (uint256 rAmount,,,,) = _getValues(tAm
            ount);
833         _rOwned[sender] = _rOwned[sender].sub(r
            Amount);
834         _rTotal = _rTotal.sub(rAmount);
835         _tFeeTotal = _tFeeTotal.add(tAmount);
836     }
837
838     function reflectionFromToken(uint256 tAmoun
        t, bool deductTransferFee) public view returns
        (uint256) {
839         require(tAmount <= _tTotal, "Amount mus
            t be less than supply");
840         if (!deductTransferFee) {
841             (uint256 rAmount,,,,) = _getValues
                (tAmount);
842             return rAmount;
843         } else {
844             (uint256 rTransferAmount,,,,) = _g
                etValues(tAmount);

```

```

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

```

```

845         return rTransferAmount;
846     }
847 }
848
849 function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
850     require(rAmount <= _rTotal, "Amount must be less than total reflections");
851     uint256 currentRate = _getRate();
852     return rAmount.div(currentRate);
853 }
854
855 function excludeFromReward(address account) public onlyOwner() {
856     // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router. ');
857     require(!_isExcluded[account], "Account is already excluded");
858     if(_rOwned[account] > 0) {
859         _tOwned[account] = tokenFromReflection(_rOwned[account]);
860     }
861     _isExcluded[account] = true;
862     _excluded.push(account);
863 }
864
865 function includeInReward(address account) external onlyOwner() {
866     require(_isExcluded[account], "Account is already excluded");
867     for (uint256 i = 0; i < _excluded.length; i++) {
868         if (_excluded[i] == account) {
869             _excluded[i] = _excluded[_excluded.length - 1];
870             _tOwned[account] = 0;
871             _isExcluded[account] = false;
872             _excluded.pop();
873             break;
874         }
875     }
876 }
877
878 function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
879     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
880     _tOwned[sender] = _tOwned[sender].sub(tAmount);
881     _rOwned[sender] = _rOwned[sender].sub(rAmount);
882     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
883     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
884     _takeLiquidity(tLiquidity);
885     _reflectFee(rFee, tFee);
886     emit Transfer(sender, recipient, tTransferAmount);
887 }

```

```

853         return rTransferAmount;
854     }
855 }
856
857 function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
858     require(rAmount <= _rTotal, "Amount must be less than total reflections");
859     uint256 currentRate = _getRate();
860     return rAmount.div(currentRate);
861 }
862
863 function excludeFromReward(address account) public onlyOwner() {
864     // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router. ');
865     require(!_isExcluded[account], "Account is already excluded");
866     if(_rOwned[account] > 0) {
867         _tOwned[account] = tokenFromReflection(_rOwned[account]);
868     }
869     _isExcluded[account] = true;
870     _excluded.push(account);
871 }
872
873 // hi if you're reading this message me on tg just wanted to see if anyone actually reads it aha @nomessages9
874
875 function includeInReward(address account) external onlyOwner() {
876     require(_isExcluded[account], "Account is already excluded");
877     for (uint256 i = 0; i < _excluded.length; i++) {
878         if (_excluded[i] == account) {
879             _excluded[i] = _excluded[_excluded.length - 1];
880             _tOwned[account] = 0;
881             _isExcluded[account] = false;
882             _excluded.pop();
883             break;
884         }
885     }
886
887 function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
888     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
889     _tOwned[sender] = _tOwned[sender].sub(tAmount);
890     _rOwned[sender] = _rOwned[sender].sub(rAmount);
891     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
892     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
893     _takeLiquidity(tLiquidity);
894     _reflectFee(rFee, tFee);
895     emit Transfer(sender, recipient, tTransferAmount);
896 }

```

```

887
888     function excludeFromFee(address account
t) public onlyOwner {
889         _isExcludedFromFee[account] = true;
890     }
891
892     function includeInFee(address account) publ
ic onlyOwner {
893         _isExcludedFromFee[account] = false;
894     }
895
896     function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
897         _taxFee = taxFee;
898     }
899
900     function setLiquidityFeePercent(uint256 liq
uidityFee) external onlyOwner() {
901         _liquidityFee = liquidityFee;
902     }
903
904     function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
905         _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
906             10**2
907         );
908     }
909
910     function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
911         swapAndLiquifyEnabled = _enabled;
912         emit SwapAndLiquifyEnabledUpdated(_enab
led);
913     }
914
915     //to recieve ETH from uniswapV2Router when
swaping
916     receive() external payable {}
917
918     function _reflectFee(uint256 rFee, uint256
tFee) private {
919         _rTotal = _rTotal.sub(rFee);
920         _tFeeTotal = _tFeeTotal.add(tFee);
921     }
922
923     function _getValues(uint256 tAmount) privat
e view returns (uint256, uint256, uint256, uint
256, uint256, uint256) {
924         (uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity) = _getTValues(tAmount);
925         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
926         return (rAmount, rTransferAmount, rFee,
tTransferAmount, tFee, tLiquidity);
927     }
928
929     function _getTValues(uint256 tAmount) priva
te view returns (uint256, uint256, uint256) {
930         uint256 tFee = calculateTaxFee(tAmoun
t);
931         uint256 tLiquidity = calculateLiquidity
Fee(tAmount);
932         uint256 tTransferAmount = tAmount.sub(t
Fee).sub(tLiquidity);

```

```

896
897     function excludeFromFee(address account
t) public onlyOwner {
898         _isExcludedFromFee[account] = true;
899     }
900
901     function includeInFee(address account) publ
ic onlyOwner {
902         _isExcludedFromFee[account] = false;
903     }
904
905     function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
906         _taxFee = taxFee;
907     }
908
909     function setLiquidityFeePercent(uint256 liq
uidityFee) external onlyOwner() {
910         _liquidityFee = liquidityFee;
911     }
912
913     function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
914         _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
915             10**2
916         );
917     }
918
919     function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
920         swapAndLiquifyEnabled = _enabled;
921         emit SwapAndLiquifyEnabledUpdated(_enab
led);
922     }
923
924     //to recieve ETH from uniswapV2Router when
swaping
925     receive() external payable {}
926
927     function _reflectFee(uint256 rFee, uint256
tFee) private {
928         _rTotal = _rTotal.sub(rFee);
929         _tFeeTotal = _tFeeTotal.add(tFee);
930     }
931
932     function _getValues(uint256 tAmount) privat
e view returns (uint256, uint256, uint256, uint
256, uint256, uint256) {
933         (uint256 tTransferAmount, uint256 tFee,
uint256 tLiquidity) = _getTValues(tAmount);
934         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
935         return (rAmount, rTransferAmount, rFee,
tTransferAmount, tFee, tLiquidity);
936     }
937
938     function _getTValues(uint256 tAmount) priva
te view returns (uint256, uint256, uint256) {
939         uint256 tFee = calculateTaxFee(tAmoun
t);
940         uint256 tLiquidity = calculateLiquidity
Fee(tAmount);
941         uint256 tTransferAmount = tAmount.sub(t
Fee).sub(tLiquidity);

```

```

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

```

```

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

```

```

976         return _amount.mul(_liquidityFee).div(
977             10**2
978         );
979     }
980
981     function removeAllFee() private {
982         if(_taxFee == 0 && _liquidityFee == 0)
983             return;
984         _previousTaxFee = _taxFee;
985         _previousLiquidityFee = _liquidityFee;
986
987         _taxFee = 0;
988         _liquidityFee = 0;
989     }
990
991     function restoreAllFee() private {
992         _taxFee = _previousTaxFee;
993         _liquidityFee = _previousLiquidityFee;
994     }
995
996     function isExcludedFromFee(address account)
997         public view returns(bool) {
998         return _isExcludedFromFee[account];
999     }
1000
1001     function _approve(address owner, address sp
1002     ender, uint256 amount) private {
1003         require(owner != address(0), "ERC20: ap
1004         prove from the zero address");
1005         require(spender != address(0), "ERC20:
1006         approve to the zero address");
1007
1008         _allowances[owner][spender] = amount;
1009         emit Approval(owner, spender, amount);
1010     }
1011
1012     function _transfer(
1013         address from,
1014         address to,
1015         uint256 amount
1016     ) private {
1017         require(from != address(0), "ERC20: tra
1018         nsfer from the zero address");
1019         require(to != address(0), "ERC20: trans
1020         fer to the zero address");
1021         require(amount > 0, "Transfer amount mu
1022         st be greater than zero");
1023         if(from != owner() && to != owner())
1024             require(amount <= _maxTxAmount, "Tr
1025             ansfer amount exceeds the maxTxAmount.");
1026
1027         // is the token balance of this contrac
1028         t address over the min number of
1029         // tokens that we need to initiate a sw
1030         ap + liquidity lock?
1031         // also, don't get caught in a circular
1032         liquidity event.
1033         // also, don't swap & liquify if sender
1034         is uniswap pair.
1035         uint256 contractTokenBalance = balanceO
1036         f(address(this));
1037
1038         if(contractTokenBalance >= _maxTxAmoun
1039         t)
1040         {

```

```

985         return _amount.mul(_liquidityFee).div(
986             10**2
987         );
988     }
989
990     function removeAllFee() private {
991         if(_taxFee == 0 && _liquidityFee == 0)
992             return;
993         _previousTaxFee = _taxFee;
994         _previousLiquidityFee = _liquidityFee;
995
996         _taxFee = 0;
997         _liquidityFee = 0;
998     }
999
1000     function restoreAllFee() private {
1001         _taxFee = _previousTaxFee;
1002         _liquidityFee = _previousLiquidityFee;
1003     }
1004
1005     function isExcludedFromFee(address account)
1006         public view returns(bool) {
1007         return _isExcludedFromFee[account];
1008     }
1009
1010     function _approve(address owner, address sp
1011     ender, uint256 amount) private {
1012         require(owner != address(0), "ERC20: ap
1013         prove from the zero address");
1014         require(spender != address(0), "ERC20:
1015         approve to the zero address");
1016
1017         _allowances[owner][spender] = amount;
1018         emit Approval(owner, spender, amount);
1019     }
1020
1021     function _transfer(
1022         address from,
1023         address to,
1024         uint256 amount
1025     ) private {
1026         require(from != address(0), "ERC20: tra
1027         nsfer from the zero address");
1028         require(to != address(0), "ERC20: trans
1029         fer to the zero address");
1030         require(amount > 0, "Transfer amount mu
1031         st be greater than zero");
1032         if(from != owner() && to != owner())
1033             require(amount <= _maxTxAmount, "Tr
1034             ansfer amount exceeds the maxTxAmount.");
1035
1036         // is the token balance of this contrac
1037         t address over the min number of
1038         // tokens that we need to initiate a sw
1039         ap + liquidity lock?
1040         // also, don't get caught in a circular
1041         liquidity event.
1042         // also, don't swap & liquify if sender
1043         is uniswap pair.
1044         uint256 contractTokenBalance = balanceO
1045         f(address(this));
1046
1047         if(contractTokenBalance >= _maxTxAmoun
1048         t)
1049         {

```

```

1027         contractTokenBalance = _maxTxAmount;
1028     }
1029
1030     bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1031     if (
1032         overMinTokenBalance &&
1033         !inSwapAndLiquify &&
1034         from != uniswapV2Pair &&
1035         swapAndLiquifyEnabled
1036     ) {
1037         contractTokenBalance = numTokensSellToAddToLiquidity;
1038         //add liquidity
1039         swapAndLiquify(contractTokenBalance);
1040     }
1041
1042     //indicates if fee should be deducted from transfer
1043     bool takeFee = true;
1044
1045     //if any account belongs to _isExcludedFromFee account then remove the fee
1046     if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1047         takeFee = false;
1048     }
1049
1050     //transfer amount, it will take tax, burn, liquidity fee
1051     _tokenTransfer(from,to,amount,takeFee);
1052 }
1053
1054 function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1055     // split the contract balance into halves
1056     uint256 half = contractTokenBalance.div(2);
1057     uint256 otherHalf = contractTokenBalance.sub(half);
1058
1059     // capture the contract's current ETH balance.
1060     // this is so that we can capture exactly the amount of ETH that the
1061     // swap creates, and not make the liquidity event include any ETH that
1062     // has been manually sent to the contract
1063     uint256 initialBalance = address(this).balance;
1064
1065     // swap tokens for ETH
1066     swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
1067
1068     // how much ETH did we just swap into?
1069     uint256 newBalance = address(this).balance.sub(initialBalance);
1070
1071     // add liquidity to uniswap
1072     addLiquidity(otherHalf, newBalance);
1073

```

```

1036         contractTokenBalance = _maxTxAmount;
1037     }
1038
1039     bool overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity;
1040     if (
1041         overMinTokenBalance &&
1042         !inSwapAndLiquify &&
1043         from != uniswapV2Pair &&
1044         swapAndLiquifyEnabled
1045     ) {
1046         contractTokenBalance = numTokensSellToAddToLiquidity;
1047         //add liquidity
1048         swapAndLiquify(contractTokenBalance);
1049     }
1050
1051     //indicates if fee should be deducted from transfer
1052     bool takeFee = true;
1053
1054     //if any account belongs to _isExcludedFromFee account then remove the fee
1055     if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
1056         takeFee = false;
1057     }
1058
1059     //transfer amount, it will take tax, burn, liquidity fee
1060     _tokenTransfer(from,to,amount,takeFee);
1061 }
1062
1063 function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
1064     // split the contract balance into halves
1065     uint256 half = contractTokenBalance.div(2);
1066     uint256 otherHalf = contractTokenBalance.sub(half);
1067
1068     // capture the contract's current ETH balance.
1069     // this is so that we can capture exactly the amount of ETH that the
1070     // swap creates, and not make the liquidity event include any ETH that
1071     // has been manually sent to the contract
1072     uint256 initialBalance = address(this).balance;
1073
1074     // swap tokens for ETH
1075     swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
1076
1077     // how much ETH did we just swap into?
1078     uint256 newBalance = address(this).balance.sub(initialBalance);
1079
1080     // add liquidity to uniswap
1081     addLiquidity(otherHalf, newBalance);
1082

```

```

1074         emit SwapAndLiquify(half, newBalance, o
therHalf);
1075     }
1076
1077     function swapTokensForEth(uint256 tokenAmou
nt) private {
1078         // generate the uniswap pair path of to
ken -> weth
1079         address[] memory path = new address[]
(2);
1080         path[0] = address(this);
1081         path[1] = uniswapV2Router.WETH();
1082
1083         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1084
1085         // make the swap
1086         uniswapV2Router.swapExactTokensForETHSu
pportingFeeOnTransferTokens(
1087             tokenAmount,
1088             0, // accept any amount of ETH
1089             path,
1090             address(this),
1091             block.timestamp
1092         );
1093     }
1094
1095     function addLiquidity(uint256 tokenAmount,
uint256 ethAmount) private {
1096         // approve token transfer to cover all
possible scenarios
1097         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1098
1099         // add the liquidity
1100         uniswapV2Router.addLiquidityETH{value:
ethAmount}(
1101             address(this),
1102             tokenAmount,
1103             0, // slippage is unavoidable
1104             0, // slippage is unavoidable
1105             owner(),
1106             block.timestamp
1107         );
1108     }
1109
1110     //this method is responsible for taking all
fee, if takeFee is true
1111     function _tokenTransfer(address sender, add
ress recipient, uint256 amount, bool takeFee) pr
ivate {
1112         if(!takeFee)
1113             removeAllFee();
1114
1115         if (_isExcluded[sender] && !_isExcluded
[recipient]) {
1116             _transferFromExcluded(sender, recip
ient, amount);
1117         } else if (!_isExcluded[sender] && _isE
xcluded[recipient]) {
1118             _transferToExcluded(sender, recipie
nt, amount);
1119         } else if (!_isExcluded[sender] && !_is
Excluded[recipient]) {
1120             _transferStandard(sender, recipien
t, amount);

```

```

1083         emit SwapAndLiquify(half, newBalance, o
therHalf);
1084     }
1085
1086     function swapTokensForEth(uint256 tokenAmou
nt) private {
1087         // generate the uniswap pair path of to
ken -> weth
1088         address[] memory path = new address[]
(2);
1089         path[0] = address(this);
1090         path[1] = uniswapV2Router.WETH();
1091
1092         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1093
1094         // make the swap
1095         uniswapV2Router.swapExactTokensForETHSu
pportingFeeOnTransferTokens(
1096             tokenAmount,
1097             0, // accept any amount of ETH
1098             path,
1099             address(this),
1100             block.timestamp
1101         );
1102     }
1103
1104     function addLiquidity(uint256 tokenAmount,
uint256 ethAmount) private {
1105         // approve token transfer to cover all
possible scenarios
1106         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1107
1108         // add the liquidity
1109         uniswapV2Router.addLiquidityETH{value:
ethAmount}(
1110             address(this),
1111             tokenAmount,
1112             0, // slippage is unavoidable
1113             0, // slippage is unavoidable
1114             owner(),
1115             block.timestamp
1116         );
1117     }
1118
1119     //this method is responsible for taking all
fee, if takeFee is true
1120     function _tokenTransfer(address sender, add
ress recipient, uint256 amount, bool takeFee) pr
ivate {
1121         if(!takeFee)
1122             removeAllFee();
1123
1124         if (_isExcluded[sender] && !_isExcluded
[recipient]) {
1125             _transferFromExcluded(sender, recip
ient, amount);
1126         } else if (!_isExcluded[sender] && _isE
xcluded[recipient]) {
1127             _transferToExcluded(sender, recipie
nt, amount);
1128         } else if (!_isExcluded[sender] && !_is
Excluded[recipient]) {
1129             _transferStandard(sender, recipien
t, amount);

```



```

1121     } else if (_isExcluded[sender] && _isEx
cluded[recipient]) {
1122         _transferBothExcluded(sender, recip
ient, amount);
1123     } else {
1124         _transferStandard(sender, recipien
t, amount);
1125     }
1126
1127     if(!takeFee)
1128         restoreAllFee();
1129 }
1130
1131     function _transferStandard(address sender,
address recipient, uint256 tAmount) private {
1132         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1133         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1134         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1135         _takeLiquidity(tLiquidity);
1136         _reflectFee(rFee, tFee);
1137         emit Transfer(sender, recipient, tTrans
ferAmount);
1138     }
1139
1140     function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private
{
1141         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1142         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1143         _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
1144         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1145         _takeLiquidity(tLiquidity);
1146         _reflectFee(rFee, tFee);
1147         emit Transfer(sender, recipient, tTrans
ferAmount);
1148     }
1149
1150     function _transferFromExcluded(address send
er, address recipient, uint256 tAmount) private
{
1151         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1152         _tOwned[sender] = _tOwned[sender].sub(t
Amount);
1153         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1154         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1155         _takeLiquidity(tLiquidity);
1156         _reflectFee(rFee, tFee);
1157         emit Transfer(sender, recipient, tTrans
ferAmount);
1158     }

```

```

1130     } else if (_isExcluded[sender] && _isEx
cluded[recipient]) {
1131         _transferBothExcluded(sender, recip
ient, amount);
1132     } else {
1133         _transferStandard(sender, recipien
t, amount);
1134     }
1135
1136     if(!takeFee)
1137         restoreAllFee();
1138     }
1139
1140     function _transferStandard(address sender,
address recipient, uint256 tAmount) private {
1141         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1142         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1143         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1144         _takeLiquidity(tLiquidity);
1145         _reflectFee(rFee, tFee);
1146         emit Transfer(sender, recipient, tTrans
ferAmount);
1147     }
1148
1149     function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private
{
1150         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1151         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1152         _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
1153         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1154         _takeLiquidity(tLiquidity);
1155         _reflectFee(rFee, tFee);
1156         emit Transfer(sender, recipient, tTrans
ferAmount);
1157     }
1158
1159     function _transferFromExcluded(address send
er, address recipient, uint256 tAmount) private
{
1160         (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1161         _tOwned[sender] = _tOwned[sender].sub(t
Amount);
1162         _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1163         _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1164         _takeLiquidity(tLiquidity);
1165         _reflectFee(rFee, tFee);
1166         emit Transfer(sender, recipient, tTrans
ferAmount);
1167     }

```

1159  
1160  
1161  
1162  
1163 }

1168  
1169  
1170  
1171  
1172 }