```
async function loadData() {
    const response = await this.http.load(
        '/some-data'
    );
}
```

```
'use strict';

var loadData = function () {
    var _ref = _asyncToGenerator( /*#__PURE__*/
regeneratorRuntime.mark(function _callee() {
        var response;
        return regeneratorRuntime.wrap(function _callee$(_context) {
            while (1) {
                switch (_context.prev = _context.next) {
                    case 0:
                        _context.next = 2;
                        return this.httpClient.load('/some-data');

                    case 2:
                        response = _context.sent;

                    case 3:
                    case 'end':
                        return _context.stop();
                }
            }
        }, _callee, this);
    }));

    return function loadData() {
        return _ref.apply(this, arguments);
    };
}();

function _asyncToGenerator(fn) { return function () { var gen =
fn.apply(this, arguments); return new Promise(function (resolve, reject) {
function step(key, arg) { try { var info = gen[key](arg); var value =
info.value; } catch (error) { reject(error); return; } if (info.done)
{ resolve(value); } else { return Promise.resolve(value).then(function
(value) { step("next", value); }, function (err) { step("throw", err); });
} } return step("next"); }); }; }
```

```
async function loadData() {
    const response = await this.http.load(
        ‘/some-data'
    );
}
```

```
'use strict';

var loadData = function () {
    var _ref = _asyncToGenerator( /*#__PURE__*/
regeneratorRuntime.mark(function _callee() {
        var response;
        return regeneratorRuntime.wrap(function _callee$(_context) {
            while (1) {
                switch (_context.prev = _context.next) {
                    case 0:
                        _context.next = 2;
                        return this.httpClient.load('/some-data');

                    case 2:
                        response = _context.sent;

                    case 3:
                    case 'end':
                        return _context.stop();
                }
            }
        }, _callee, this);
    }));

    return function loadData() {
        return _ref.apply(this, arguments);
    };
}();

function _asyncToGenerator(fn) { return function () { var gen =
fn.apply(this, arguments); return new Promise(function (resolve, reject) {
function step(key, arg) { try { var info = gen[key](arg); var value =
info.value; } catch (error) { reject(error); return; } if (info.done)
{ resolve(value); } else { return Promise.resolve(value).then(function
(value) { step("next", value); }, function (err) { step("throw", err); });
} } return step("next"); }); }; }
```

HOW TO SHIP MODERN CODE TO MODERN BROWSERS