```diff
  /**
  * @fileoverview microlight - syntax highlightning library
  * @version 0.0.1
  *
  * @license MIT, see http://github.com/asvd/microlight
  * @copyright 2016 asvd <heliosframework@gmail.com>
  *
  * Code structure aims at minimizing the compressed library size
  */


  (function (root, factory) {
  if (typeof define === 'function' && define.amd) {
  define(['exports'], factory);
  } else if (typeof exports !== 'undefined') {
  factory(exports);
  } else {
  factory((root.microlight = {}));
  }
  }(this, function (exports) {
  // for better compression
  var _window       = window,
  _document     = document,
  appendChild   = 'appendChild',
  test          = 'test',
  // style and color templates
  textShadow    = ';text-shadow:',
  opacity       = 'opacity:.',
  _0px_0px      = ' 0px 0px ',
  _3px_0px_5    = '3px 0px 5',
  brace         = ')',

  el,  // current microlighted element to run through

  // dynamic set of nodes to highlight
  microlighted = _document.getElementsByClassName('microlight');


- var reset = function(i) {
- for (i = 0; el = microlighted[i++];) {
- var text  = el.textContent,
+ var process = function(text, color) {
+ var el    = document.createElement('div'),
  pos   = 0,       // current position
  next1 = text[0], // next character
  chr   = 1,       // current character
  prev1,           // previous character
  prev2,           // the one before the previous
- token =          // current token content
- el.innerHTML = '',  // (and cleaning the node)
+ token = '',      // current token content

  // current token type:
  //  0: anything else (whitespaces / newlines)
  //  1: operator or brace
  //  2: closing braces (after which '/' is division not regex)
  //  3: (key)word
  //  4: regex
  //  5: string starting with "
  //  6: string starting with '
  //  7: xml comment  <!-- -->
  //  8: multiline comment /* */
  //  9: single-line comment starting with two slashes //
  // 10: single-line comment starting with hash #
  tokenType = 0,

  // kept to determine between regex and division
  lastTokenType,
  // flag determining if token is multi-character
  multichar,
  node,

  // calculating the colors for the style templates
- colorArr = /(\d*\, \d*\, \d*)(, ([.\d]*))?/g.exec(
- _window.getComputedStyle(el).color
- ),
- pxColor = 'px rgba('+colorArr[1]+',',
- alpha = colorArr[3]||1;
+ colorArr = /(\d*\, \d*\, \d*)(, ([.\d]*))?/g.exec(color),
+ pxColor  = 'px rgba('+colorArr[1]+',',
+ alpha    = colorArr[3]||1;

  // running through characters and highlighting
  while (prev2 = prev1,
  // escaping if needed (with except for comments)
  // pervious character will not be therefore
  // recognized as a token finalize condition
  prev1 = tokenType < 7 && prev1 == '\\' ? 1 : chr
  ) {
  chr = next1;
  next1=text[++pos];
  multichar = token.length > 1;

  // checking if current token should be finalized
  if (!chr  || // end of content
  // types 9-10 (single-line comments) end with a
  // newline
  (tokenType > 8 && chr == '\n') ||
  [ // finalize conditions for other token types
  // 0: whitespaces
  /\S/[test](chr),  // merged together
  // 1: operators
  1,                // consist of a single character
  // 2: braces
  1,                // consist of a single character
  // 3: (key)word
  !/[$\w]/[test](chr),
  // 4: regex
  (prev1 == '/' || prev1 == '\n') && multichar,
  // 5: string with "
  prev1 == '"' && multichar,
  // 6: string with '
  prev1 == "'" && multichar,
  // 7: xml comment
  text[pos-4]+prev2+prev1 == '-->',
  // 8: multiline comment
  prev2+prev1 == '*/'
  ][tokenType]
  ) {
  // appending the token to the result
  if (token) {
  // remapping token type into style
  // (some types are highlighted similarly)
  el[appendChild](
  node = _document.createElement('span')
  ).setAttribute('style', [
  // 0: not formatted
  '',
  // 1: keywords
  textShadow + _0px_0px+9+pxColor + alpha * .7 + '),' +
  _0px_0px+2+pxColor + alpha * .4 + brace,
  // 2: punctuation
  opacity + 6 +
  textShadow + _0px_0px+7+pxColor + alpha / 4 + '),' +
  _0px_0px+3+pxColor + alpha / 4 + brace,
  // 3: strings and regexps
  opacity + 7 +
  textShadow + _3px_0px_5+pxColor + alpha / 5 + '),-' +
  _3px_0px_5+pxColor + alpha / 5 + brace,
  // 4: comments
  'font-style:italic;'+
  opacity + 5 +
  textShadow + _3px_0px_5+pxColor + alpha / 4 + '),-' +
  _3px_0px_5+pxColor + alpha / 4 + brace
  ][
  // not formatted
  !tokenType ? 0 :
  // punctuation
  tokenType < 3 ? 2 :
  // comments
  tokenType > 6 ? 4 :
  // regex and strings
  tokenType > 3 ? 3 :
  // otherwise tokenType == 3, (key)word
  // (1 if regexp matches, 0 otherwise)
  + /^(a(bstract|lias|nd|rguments|rray|s(m|sert)?|uto)|b(ase|egin|ool(ean)?|reak|yte)|c(ase|atch|har|hecked|lass|lone|ompl|onst|ontinue)|de(bugger|cimal|clare|f(ault|er)?|init|l(egate|ete)?)|dc
  ]);

  node[appendChild](_document.createTextNode(token));
  }

  // saving the previous token type
  // (skipping whitespaces and comments)
  lastTokenType =
  (tokenType && tokenType < 7) ?
  tokenType : lastTokenType;

  // initializing a new token
  token = '';

  // determining the new token type (going up the
  // list until matching a token type start
  // condition)
  tokenType = 11;
  while (![
  1,                    //  0: whitespace
  //  1: operator or braces
  /[\/{}[(\-+*=<>:;|\\.,?!&@~]/[test](chr),
  /[\])]/[test](chr),  //  2: closing brace
  /[$\w]/[test](chr),  //  3: (key)word
  chr == '/' &&        //  4: regex
  // previous token was an
  // opening brace or an
  // operator (otherwise
  // division, not a regex)
  (lastTokenType < 2) &&
  // workaround for xml
  // closing tags
  prev1 != '<',
  chr == '"',          //  5: string with "
  chr == "'",          //  6: string with '
  //  7: xml comment
  chr+next1+text[pos+1]+text[pos+2] == '<!--',
  chr+next1 == '/*',   //  8: multiline comment
  chr+next1 == '//',   //  9: single-line comment
  chr == '#'           // 10: hash-style comment
  ][--tokenType]);
  }

  token += chr;
  }
+
+ return el.innerHTML;
  }
+
+ var reset = function(i) {
```

```
+ if (isNaN(i)) {
+ i = 0;
  }

- exports.reset = reset;
+ for (;el = microlighted[i++];) {
+ el.innerHTML = process(
+ el.textContent,
+ _window.getComputedStyle(el).color
+ );
+ }
+ };
+
+ exports.process = process;
+ exports.reset   = reset;

  if (_document.readyState == 'complete') {
  reset();
  } else {
  _window.addEventListener('load', reset, 0);
  }
  }));
```