```
/*

This expanded Euclidean Algorithm is a JavaScript version for seeking the
Greatest Common Divisor among any quantity of integers stored in an array
named: 'ofTheseIntegers' and passed to the getTheGCD() function.

Written by Vinyasi in the summer of 2016 by extracting its code from...
http://vinyasi.info/Infinite%20Range%20of%20Golden%20Ratios/tablature_format-gcd.html

...based on research, spanning the years from 1994 to 1997, into the premise that there are
infinitely various golden ratios and golden series of integers from which golden ratios arise.

<html><head>
<title>
Testy Westy
</title>

<!--
This is an example of how to invoke the getTheGCD() function from within 'gcd.js'....
//-->

<script src="./gcd.js" type="text/javascript"></script>

<script type="text/javascript">
function displayGCD()
{
var temp = document.getElementById("ofTheseIntegers").value;
var GCDArray = temp.split(" ");

// Invoke the function...
var display = 'The Greatest Common Divisor of ' + GCDArray + ' is ' + getTheGCD(GCDArray);
alert(display);
}
</script>

</head>
<body>

<div align="center">
<form action="" method="post" onsubmit="return displayGCD()">
<h1>
Find the Greatest Common Divisor<br />of these Multiple Positive Integers:
</h1>
<input type="text" id="ofTheseIntegers" name="ofTheseIntegers" placeholder="ofTheseIntegers" value="49 21 35 77">
<br /><br />
<input type="submit" value="Submit">
</form>
</div>

</body></html>

*/

var terms = [];

function getTheGCD(terms)
{
```

```
// Quantity of integers whose GCD is to be sought...
var count = terms.length;

// Last position of an integer within 'terms'...
var last = count - 1;

// Sort the contents of 'terms'...
terms.sort ( function(a, b) { return a-b } );

// This has to be at least one greater than the largest integer in the array: 'terms'.
// Otherwise, the sort function will push all the numbers to the top of the 'terms' array
// and begin to cut them out a little at a time!
var numeric_padding = terms[last] + 1;

// Permanently save the quantity of integers...
var save_count = count;

// Temporarily save the quantity of integers...
var kount = count;

// Establish a second array, 'remains', to swap with the contents of 'terms' and
// continue to swap them back and forth to each other throughout the 'while' loop below...
var remains = [];
remains[0] = 0;

// Perform our first "shift to the right" of the contents of 'terms'...
for (var i = 0; i < (count - 1); i++)
{
        remains[i + 1] = terms[i];
}

// Uncomment out the following line, for debugging...
//for (var q = 0; q < 4; q++)

// ...and comment out the next line for debugging...
while (kount > 1)
{
// Transfer 'count' to 'kount' since neither variable will retain
// their values throughout the iterations of the 'while' loop above...
        count = kount;

// Take the modulo remainder of 'terms' and store it in 'remains'...
        for (var i = 0; i < save_count; i++)
        {
                if (i === 0)
                {
                        remains[i] = terms[i];
                }
                else
                {
                        remains[i] = terms[i] % remains[i];
                }
        }

//      if (q == 3) { break; } // for debugging

// Fill in empty slots, within 'terms', using padded values greater than
// the largest integer to help with properly sorting 'terms'...
```

```
          for (var v = 0; v < save_count; v++)
          {
                    terms[v] = numeric_padding;
          }

// Reset 'kount' to zero...
          kount = 0;

// Transfer 'remains' to 'terms' and 'kount' the number of non-zero 'terms'...
          for (var i = 0; i < count; i++)
          {
                    if (remains[i] !== 0)
                    {
                              terms[i] = remains[i];
                              kount++;
                    }
          }

          if (kount > 1)
          {
// Sort the contents of 'terms'...
                    terms.sort ( function(a, b) { return a-b } );
                    remains[0] = 0;

// Transfer 'terms' to 'remains' by shifting, or offsetting, this transfer to the right of the first array position...
                    for (var v = 1; v < save_count; v++)
                    {
                              if (terms[v] != numeric_padding)
                              {
                                        remains[v] = terms[v - 1];
                              }
                    }

//                  if (q == 5) { break; } // for debugging
//                  document.write ('*' + kount + '*'); // for debugging
          }
}

// Return the answer...
return terms[0];

}
```