Understand recursion with these 8 classical JavaScript coding challenges for beginners

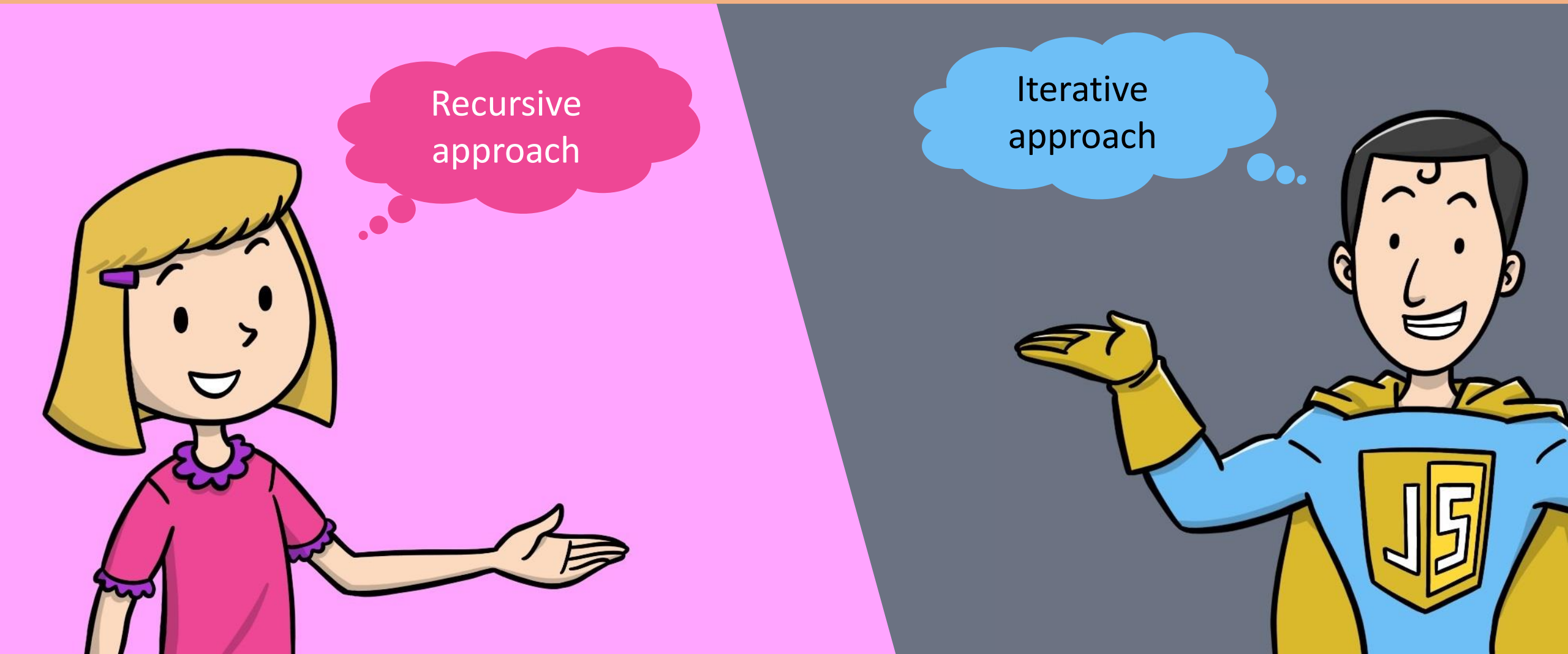


Table of contents

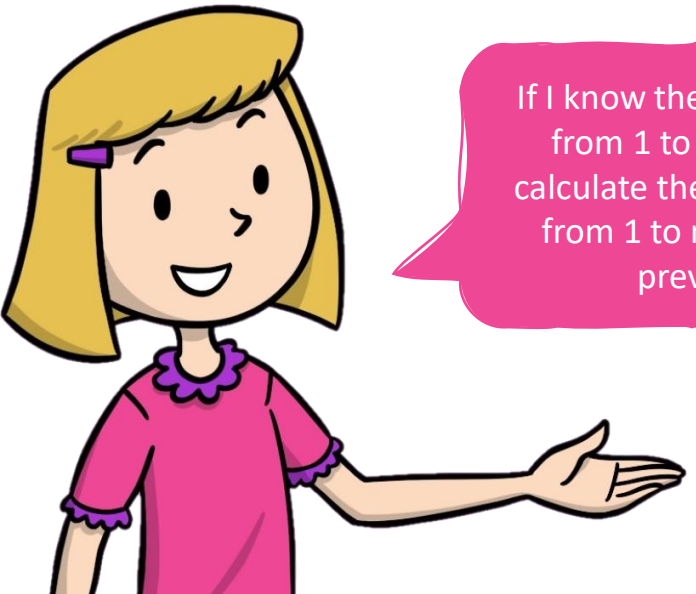
Problem 1: Calculate the sum of natural numbers from 1 to n

Recursive solution

```
var sum = addTo(10);
println(sum);

function addTo(n)
{
    if (n == 0)
        return 0;

    return n + addTo(n - 1);
}
```

Iterative solution

```
var sum = addTo(10);
println(sum);

function addTo(n)
{
    var sum = 0;

    for(var i = 1; i <= n; i++)
    {
        sum += i;
    }

    return sum;
}
```



If I know the sum of numbers from 1 to n - 1 then I can calculate the sum of numbers from 1 to n by adding n to previous sum

This is the classical solution for calculating the sum of numbers from 1 to n.

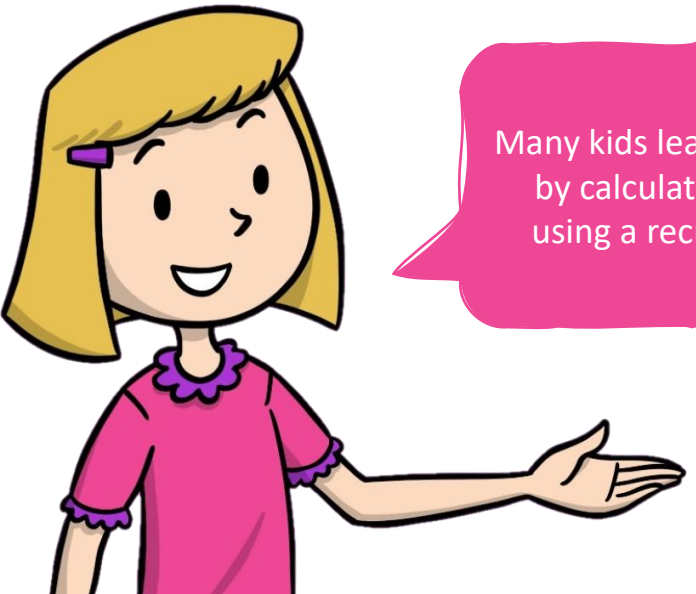
Problem 2: Calculate factorial of n. Remember n! = 1 * 2 * … * n

Recursive solution

```
var prod = factorial(10);
println(prod);

function factorial(n)
{
    if (n <= 1)
        return 1;

    return n * factorial(n - 1);
}
```

Iterative solution

```
var prod = factorial(10);
println(prod);

function factorial(n)
{
    var prod = 1;

    for(var i = 1; i <= n; i++)
    {
        prod *= i;
    }

    return prod;
}
```



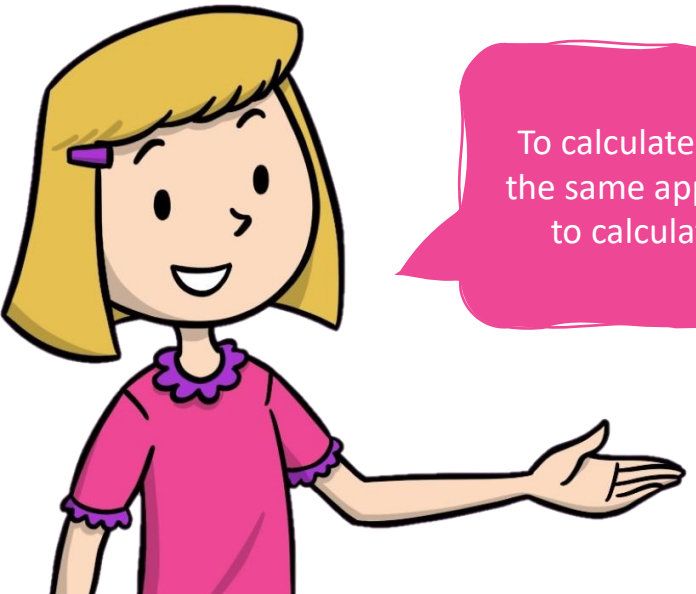
Many kids learn about recursion by calculating factorial of n using a recursive function…

… although is trivial and fast to calculate n! using the iterative approach!

Problem 3: Calculate n^m - the value of *n* to the *m* power

Recursive solution

```
println(powerNo(3, 2));

function powerNo(n, m)
{
    if (m == 0)
        return 1;

    if (m == 1)
        return n;

    return n * powerNo(n, m - 1);
}
```



To calculate n^m we will use the same approach we used to calculate n factorial

The for loop is used to repeat the multiplication operation m times.

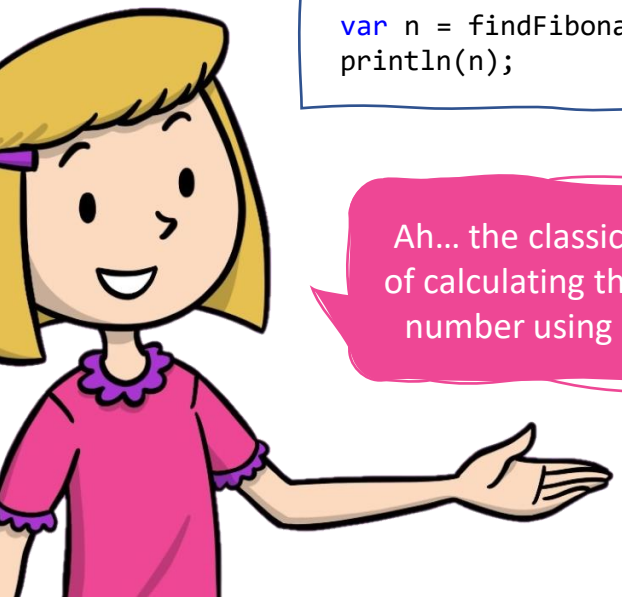
Iterative solution

```
println(powerNo(3, 2));

function powerNo(n, m)
{
    var prod = 1;

    for(var i = 1; i <= m; i++)
    {
        prod *= n;
    }

    return prod;
}
```

Problem 4: Find the n^{th} Fibonacci number. The Fibonacci series is the series of numbers in which each number is the sum of the previous two numbers.

Recursive solution

```
function findFibonacci(n)
{
    if (n == 0)
        return 0;

    if (n == 1)
        return 1;

    return findFibonacci(n - 1) +
           findFibonacci(n - 2);
}

var n = findFibonacci(10);
println(n);
```



Ah… the classical problem of calculating the Fibonacci number using recursion!

Iterative solution

```
function findFibonacci(n)
{
    var fib0 = 0;
    var fib1 = 1;

    if (n == 0)
        return fib0;

    if (n == 1)
        return fib1;

    var fib;

    for(var i = 2; i <= n; i++)
    {
        fib = fib0 + fib1;

        fib0 = fib1;
        fib1 = fib;
    }

    return fib;
}

println(findFibonacci(10));
```

This iterative solution is so FAST! I can calculate much bigger Fibonacci numbers than with the recursive solution!

Problem 5: Calculate the sum of elements of an array of numbers

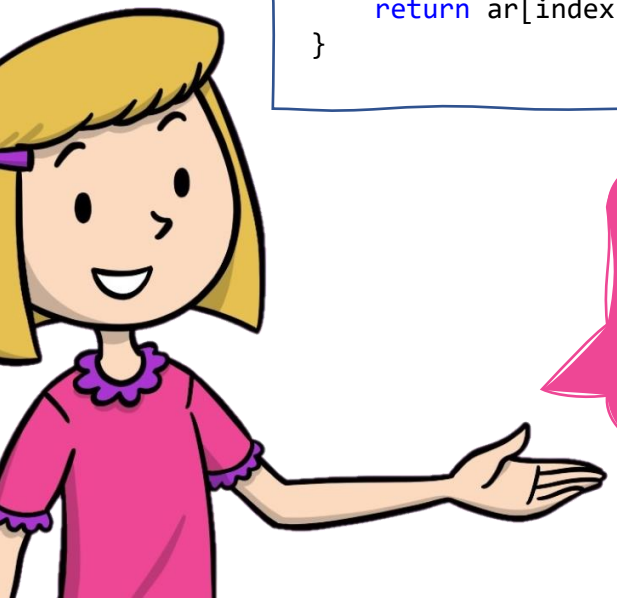
Recursive solution

```
var ar = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var n = sum(ar);
println(n);


function sum(ar)
{
    return _sum(ar, ar.length - 1);
}

function _sum(ar, index)
{
    if (index == 0)
        return ar[0];

    return ar[index] + _sum(ar, index - 1);
}
```

Iterative solution

```
var ar = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var n = sum(ar);
println(n);

function sum(ar)
{
    var sum = 0;

    for(var el of ar)
    {
        sum += el;
    }

    return sum;
}
```

If you know how to iterate an array, then it is very easy to calculate the sum of its elements

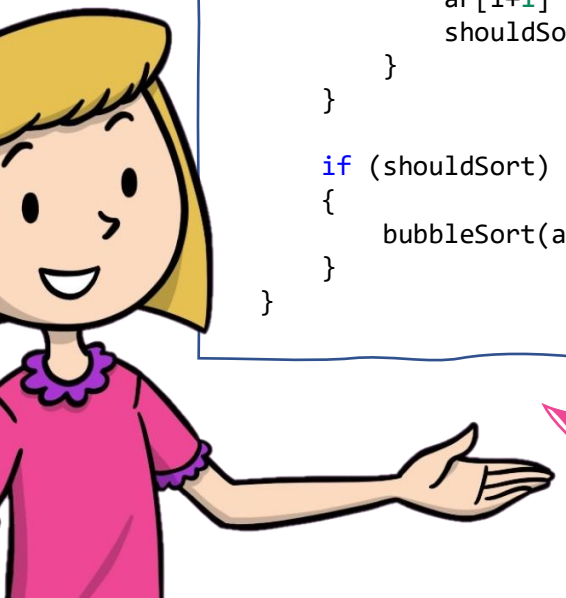


The sum of elements is calculated using recursion – without any for loop

Problem 6: Sort an array of numbers using bubble sort algorithm

Recursive solution

```
var ar = [23, 1000, 1, -1, 8, 3];
println(ar);
bubbleSort(ar);
println(ar);

function bubbleSort(ar)
{
    var shouldSort = false;

    for(var i = 0; i < ar.length - 1; i++)
    {
        var a = ar[i];
        if ( a > ar[i+1] )
        {
            ar[i] = ar[i+1];
            ar[i+1] = a;
            shouldSort = true;
        }
    }

    if (shouldSort)
    {
        bubbleSort(ar);
    }
}
```

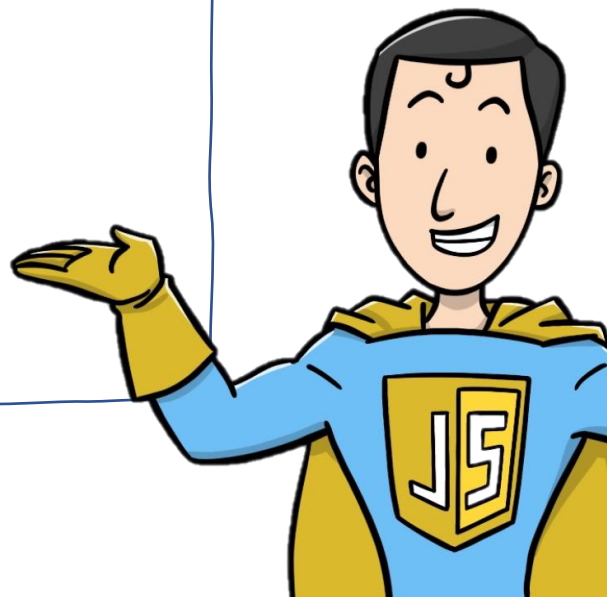


Swap array elements in the right order… then recall the same function until array is sorted.

Iterative solution

```
var ar = [23, 1000, 1, -1, 8, 3];
println(ar);
bubbleSort(ar);
println(ar);

function bubbleSort(ar)
{
    var shouldSort = true;

    while(shouldSort)
    {
        shouldSort = false;

        for(var i = 0; i < ar.length - 1; i++)
        {
            var a = ar[i];
            if ( a > ar[i+1] )
            {
                ar[i] = ar[i+1];
                ar[i+1] = a;
                shouldSort = true;
            }
        }
    }
}
```

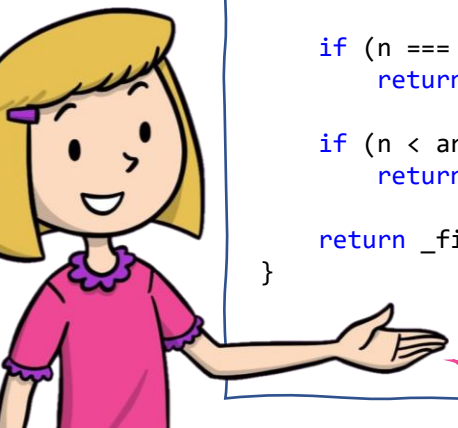
A big while loop should do it…



Problem 7: Find a number in a sorted array of numbers (binary search)

Recursive solution

```
//        0   1   2   3   4   5   6   7   8   9
var ar = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

var position = findNumber(90, ar);
println(position);

// Find number n in sorted array ar
// Returns array index if found or -1 if not found
function findNumber(n, ar)
{
    return _findNumber(n, ar, 0, ar.length - 1);
}

// Find number n in sorted array ar in between indexes
// i1 and i2 using recursive approach
function _findNumber(n, ar, i1, i2)
{
    if (i2 < i1)
        return -1;

    println("Checking interval: [" + i1 + ", " + i2 + "]");

    var mid = i1 + Math.floor((i2 - i1) / 2);

    if (n === ar[mid])
        return mid;

    if (n < ar[mid])
        return _findNumber(n, ar, i1, mid - 1);

    return _findNumber(n, ar, mid + 1, i2);
}
```



… check if the number is in the middle of the array…

Iterative solution

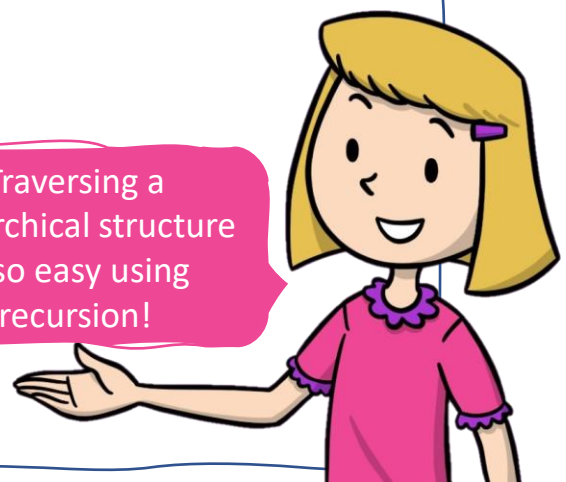
```
//        0   1   2   3   4   5   6   7   8   9
var ar = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

var position = findNumber(90, ar);
println(position);

// Find number n in sorted array ar using iterative approach
// Returns array index if found or -1 if not found
function findNumber(n, ar)
{
    var i1 = 0;
    var i2 = ar.length - 1;

    while(i1 <= i2)
    {
        println("Checking interval: [" + i1 + ", " + i2 + "]");

        var mid = i1 + Math.floor((i2 - i1) / 2);

        if (n === ar[mid])
            return mid;

        if (n < ar[mid])
        {
            i2 = mid - 1;
        }
        else
        {
            i1 = mid + 1;
        }
    }

    return -1;
}
```

The iterative solution is very similar with the recursive one

Problem 8: Find the maximum number in an array containing numbers or other array of numbers (on an unlimited number of levels)

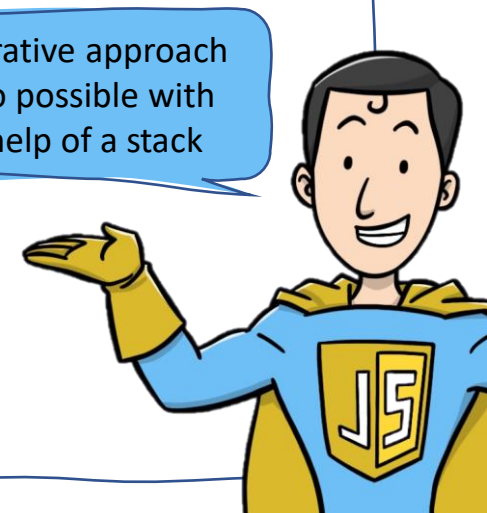
Recursive solution

```
var ar = [2, 4, 10, [12, 4, [100, 99], 4], [3, 2, 99], 0];

var max = findMax(ar);
println("Max  = ", max);

// Use recursion to find the maximum numeric value in an
// array of arrays
function findMax(ar)
{
    var max = -Infinity;

    // Cycle through all the elements of the array
    for(var i = 0; i < ar.length; i++)
    {
        var el = ar[i];

        // If an element is of type array -> invoke the same
        // function to find out the max elem. of that subarray

        if ( Array.isArray(el) )
        {
            el = findMax( el );
        }

        if ( el > max )
        {
            max = el;
        }
    }

    return max;
}
```



Iterative solution

```
// Use a stack to find the maximum numeric value in an array of arrays
function findMax(arElements)
{
    var max = -Infinity;

    // This is the stack on which will put the first array and then
    // all the other sub-arrays that we find as we traverse an array
    var arrays = [];

    arrays.push(arElements);

    // Loop as long as are arrays added to the stack for processing
    while(arrays.length > 0)
    {
        // Extract an array from the stack
        ar = arrays.pop();

        // ... and loop through its elements
        for(var i = 0; i < ar.length; i++)
        {
            var el = ar[i];

            // If an element is of type array, we'll add it to stack
            // to be processed later
            if ( Array.isArray(el) )
            {
                arrays.push(el);
                continue;
            }

            if ( el > max )
            {
                max = el;
            }
        }
    }

    return max;
}
```



These coding challenges were brought to you by codeguppy.com – the fun coding site for kids, teens and creative adults

Don’t forget to visit https://codeguppy.com for more fun projects!

For news and updates follow @codeguppy on Twitter!