Recursion Techniques for JavaScript

G James Mitchell

Programmers use a variety of techniques to apply sophisticated solutions through specific computer languages. Different techniques have affordances and constraints that determine which is best for individual problems or solutions. A common technique used among programmers is recursive programming, which helps to organize code into steps that addresses pieces of a problem. This assists in creating an algorithm that works through a dataset and produces desired results. Here is how it works.

What is recursion in javascript?

Recursion refers to a function that iterates an operation until a condition is met, resulting in a loop[1]. For recursive programming to run effectively, and not produce an infinite loop, two elements are required[2]. The **recursive case** and the **base**. The base is the condition that terminates the recursion, once reached. The recursive case is the function that is recurring.

For a simple example of this technique we can create a function that counts to ten.

Ex. 1

```
1  var n = 0;
2 function countToTen(n) {
3      if (n === 10) { /*base */
4      return 'counted to ten';}
5      else { /* recursive case */
6      console.log(n + ' is not ten');
7      return countToTen(n+1);
8      }
9  }
```

To further extrapolate the meaning of this code we'll break down each line.

1. Set the beginning value of our variable before starting the recursion process.
2. Label our function and identify its value.
3. Identify the condition of our base.
4. Command a return to print ‘counted to ten’ upon reaching the base.
5. Identify the recursive case when the base is not reached.
6. Command a return to print of each step during the recursive process.
7. Command the function to call itself adding 1, for the loop to progress.

When is recursive programming used?

The recursion technique is highly effective at traversing large data sets that automate calling steps, figures, and objects in functions like binary search and quicksort[1]. Recursion is not isolated to one language and is widely used to tackle large data stacks. This makes it easy to recognize and read, which is important when building upon existing code.

By organizing data into calculated structures, programming recursive loops into algorithms provides solutions and accessibility to complex problems. This can be done by segmenting large problems into numerous smaller ones, then solving each with the recursive technique. The algorithm as a whole will necessary calculations to quickly deliver solutions, directions, or figures.

Having the flexibility to apply multiple recursive equations and functions to varying pieces of data within the same algorithm consolidates programming efforts, offering cleaner code. When programming for large data sets, it is important to have code noted and organized, so that adjustments can be made and improved upon.

Principles of recursive programming

There are a few principles you should understand to implement recursive techniques. As an algorithm takes shape, and loops are inserted to navigate data structures, complexity increases within the application. These principles should guide you to quickly locate common errors.

- Base Syntax and Logic

We already know step 3, the base, is key at delivering proper recursive programming. If your function produces errors, check the base for correct syntax and logic.

- Always Add a Step

Step 7 is imperative to move the function toward the condition. If your function is not working, check that your return command leads the result to exact termination.

- Null is Your Enemy

As your algorithm includes more diverse functions it is important to remember that a function with a null value will remove that function from performing, resulting in an error. If you are getting an error check areas where Null may be used.

Why recursion in javascript?

As mentioned, recursion is effective when applying a function to a large set of data. Rather than programming a function for each operation, recursion can be applied to calculations that include specific variables that produce desired results. This is the process of using computation to automate solutions. With

automation we expedite answers and accessibility to valuable information.

This technique can also be used to verify itself against other pieces of information, and alert users if parameters are breached. This ensures algorithms avoid corruption and deviation, maintaining accuracy.

Applying Recursion in JavaScript

Affordances

When using logic to solve problems, we are navigating a binary tree of information. An answer is either true or false. This principle is the essence of recursion, and can be strategically applied to produce creative solutions. The next example demonstrates how to apply multiple recursions in the same function to navigate this data set.

Ex. 2

```
var [a=1, b=2, c=b, d=10];
function affordance(b){
    if (c === d){ /* strategic base */
        return 'C is equal to D';
    }
    else if (b === 10){ /*base */
        return affordance(b);
    }
        else if (a < 2){ /*adjusting recursion */
            console.log ('C is not equal to D');
            return affordance(b + (a + 1));
        }
        else { /*recursive case */
            console.log ('C is not equal to D');
            return affordance(b + a);
        }
}
```

By breaking down the problem into smaller steps we can use one small adjustment to build momentum when combining both variables.

1. Set your strategic base for var C.
2. Within that recursion, set the base of the variable you'll be working with.
3. Within the next recursion use an operation to make your small adjustment.
4. Finally, command your recursive case of your two variables.

Constraints

This technique does have limitations, however. Should the recursive function become too large, the application could begin having problems when responding. The point of recursion is to break the large problem into smaller ones. Just remember each technique has its own affordances and constraints. Where one may not work, another may.

Footnotes

[1] https://www.javascripttutorial.net/javascript-recursive-function/

[2] https://github.com/JS-Challenges/recursion-prompts