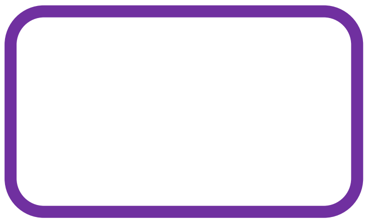
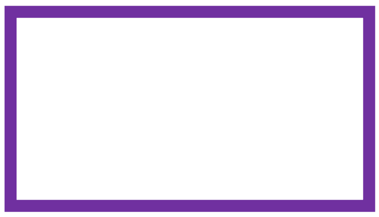
Recursion & flowcharts for Brownbag: Algorithms and Problem Solving with Graphs and Trees

Some slides will use a flowchart to represent the logic of an algorithm.

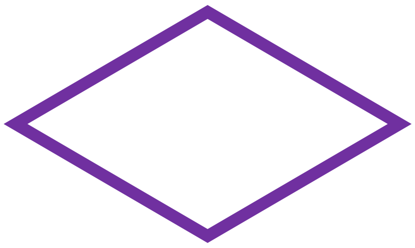
Key to flowcharts used in the presentation (colors may vary):



Start/stop



Instructions (general program logic)

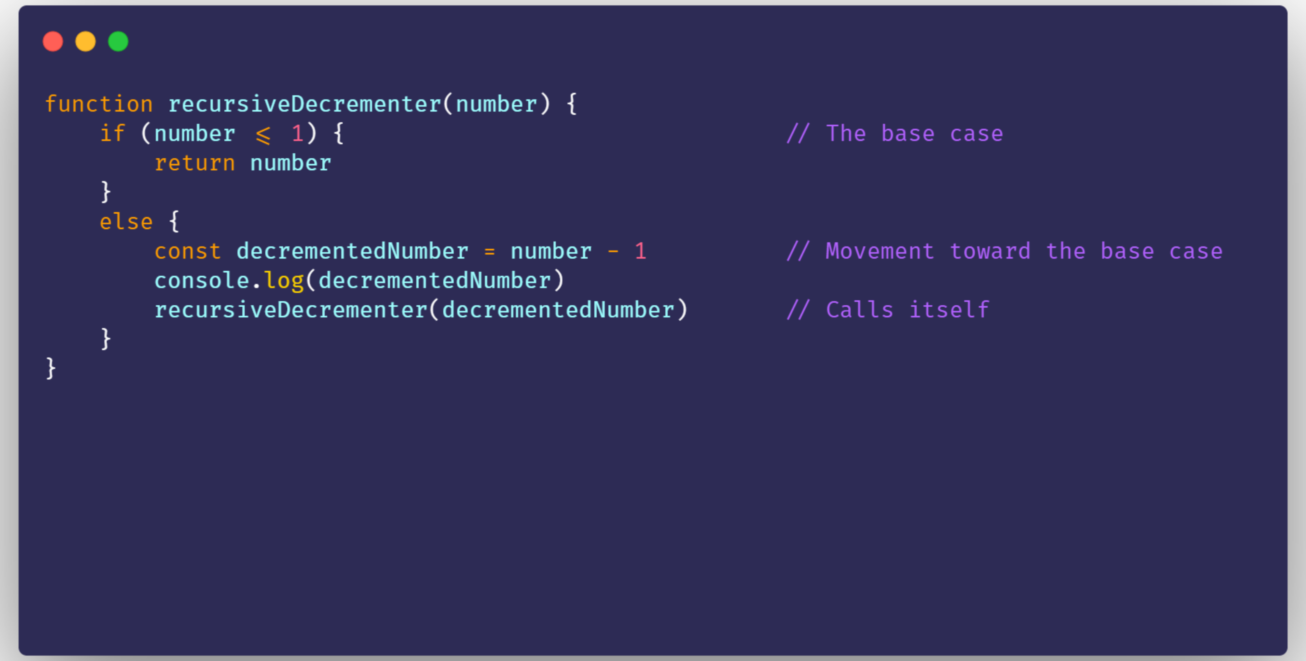


Decisions (normally represented by if or if..else statements)

Some of the algorithm examples I will be sharing use recursion. If you are not familiar with recursion, I have included a summary of basic principles below:

A recursive function is a function that calls itself, often repeatedly.

For example:



```
function recursiveDecrementer(number) {
    if (number <= 1) {                              // The base case
        return number
    }
    else {
        const decrementedNumber = number - 1        // Movement toward the base case
        console.log(decrementedNumber)
        recursiveDecrementer(decrementedNumber)     // Calls itself
    }
}
```

This function takes a number and decrements it until it is less than or equal to 1, logging the current value to console each time it runs.

All recursive functions must follow the three laws of recursion:

There must be a base case. The base case is a state when the algorithm does not call itself, thereby allowing execution to finish.

The logic must move toward the base case. If the base case is never reached, program execution continues forever (the recursion equivalent of an infinite loop).

The function must call itself recursively, meaning that it calls itself using the output of the function logic as an argument in the next call.

Note that it is very important to ensure that the base case will be reached given any input your program might receive. In the first version of the above example I used `number == 1` as the condition for the base case, meaning that if a number less than 1 or a non-integer number was passed, the base case would never be reached.