Recursion

Referring to your inner self

Function call and stack frame (Revisited)

```
// Output?

function A(){
    console.log("A is called");
    console.log("Before B is called");
    B();
    console.log("After B is called")
}

function B(){
    console.log("B is called");
    console.log("Before C is called");
    C();
    console.log("After C is called");
}

function C(){
    console.log("C is called");
}
A();
console.log("After A is called");
```

Recursion

- Defining something in terms of itself
- When a solution to a problem is defined in terms of itself, it's called a recursive solution.

$$f(n) = n*f(n-1)$$

- When a function solves a task, in the process it can call many other functions.
  - A partial case of this is when a function calls itself.
- When a function calls itself, inside its body it's called a recursive function.
  - The process of a function calling itself is recursion.

Iteration vs Recursion (two ways of thinking)

- Any problem that can be solved using recursion can also be solved using iteration, loops.

- In typical JavaScript implementations, recursive solutions are about three times slower than its iterative version.
  - But in some situations, recursive solutions are much more elegant (shorter, simpler and clearer) than the iterative ones.

Recursion Example

```
// find sum of n natural numbers

function recursiveSum(n){
    if(n===0){
        return 0;
    }else{
        return n + recursiveSum(n-1);
    }
}

console.log(recursiveSum(10));
```

- Solve it using loop (iteratively)

Recursion depth

- The total number of nested calls (including the first one) is called *recursion depth.*
- The maximal recursion depth is limited by JavaScript engine.
  - We can rely on it being 10000, some engines allow more, but 100000 is probably out of limit for most of them.

- "too much recursion" or "Maximum call stack size exceeded" exceptions
  - too many function calls,
  - or missing a base case.
  - CTRL-C CTRL-C to break out of infinite loop in node

The Base Case and Reduction Step

- To stop recursion going into an infinite recursion (and exceed the max recursion depth)
  - Reduction step: We need to make sure that each recursive call moves us closer to a base case
  - Base case: returns without calling itself

- Recursion creates stack frames on the call stack until the base case
  - Then it comes back down through the frames

Exercise

- Write both iterative and recursive solutions to calculate factorial of an integer.

  factorial (0) = 1

  factorial (n) = n * factorial (n-1) [for n>0]

Exercise

- Write recursive solution to find Fibonacci(n) based on following definition

  fibonacci(0) = 0

  fibonacci(1) = 1

  fibonacci(n) = fibonacci(n-1) + fibonacci(n-2) [for n>1]

- This definition is a little different than the previous ones because
  - It has two base cases, not just one; in fact, you can have as many as you like.
  - In the recursive case, there are two recursive calls, not just one. There can be as many as needed.
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on

Example

```
// Find length of a String recursively, without using
length property
function findLengthRecursive(s){
    if(s==="")
        return 0;
    else
        return 1+ findLengthRecursive(s.substr(1));
}
console.log(findLengthRecursive("Hello"));
```

Exercise

- Write a recursive function to test whether a string is a palindrome
  - Base case?
  - Reductive recursive call?
  - Hint: do comparisons from outer edges inwards

Recursive traversals

- great application of recursion is recursive traversal.
  - Example: searching for a file in a directory (folder)
    - Get all the files and other directories that lie in the given directory `dir`
      - For each of these files, compare names with the target file name
      - if the same, return `true`
    - For each directory d among the directories found in `dir` ,
      - recursively search for `file`
      - Return `false`

- Practice: see recursive traversal example from javascript.info .
  - It’s okay if you don’t get it the first time.
  - our standard thinking pattern needs some adjustment.
  - key is to think of a recursive call just like a call to another function
  - think of the function call stack

Main Point

- In recursive solution a function calls itself which is similar to *self-referral during the practice of transcendental meditation. Self-referral is a way to find solutions within. Awareness aware of itself can be compared to a function calling itself.*

Assignments

- Write recursive functions to
  - calculate the power of any base
  - count the digits of a given number
  - reverse a given string.

- Write your own recursive logic for above problems. You cannot use existing methods like reverse or length property.

References

- Recursion and stack (javascript.info)