# Admin

- Sixth quiz deadline next Monday at 4pm
  - Seventh quiz deadline **29th March** at 4pm

- Sudoku assignment feedback for written work **hopefully** by end of 29th March

- Primes assignment
  - Cut-off date **29th March 4pm**
  - Help with Primes Assignment next week in VCH

- New worksheet next week
  - Help in VCH

- Mock online test available on Monday

- Review Seminar next week - Q&A with second-year students

# Employability Portal

# https://learn.gold.ac.uk/course/view.php?id=18110

## Semester 2: Week 10: 22nd - 28th March 2021

**Technical Interview Skills & Preparation**

**Wednesday 17th March 2021, 2 - 3 pm**

An Employment focused online session for perhaps of particular relevance for 2nd year, 3rd year and PG students.

We are planning to bring in an external expert to look at the topic of technical interview skills and how to prepare for technical interviews, the part of the interview where candidates are expected to demonstrate their technical skills. This is of particular relevance for those looking for work, or indeed work placements and internships.

**The link to the call will be posted here shortly:**

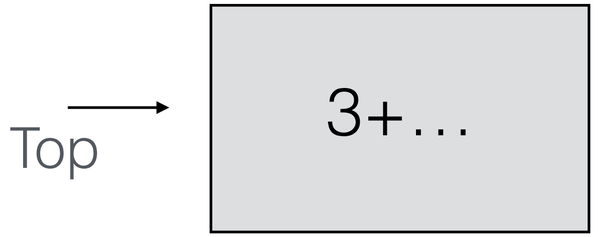
# Recursion and Call Stacks

```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

*Let's call recursiveSum(3)*

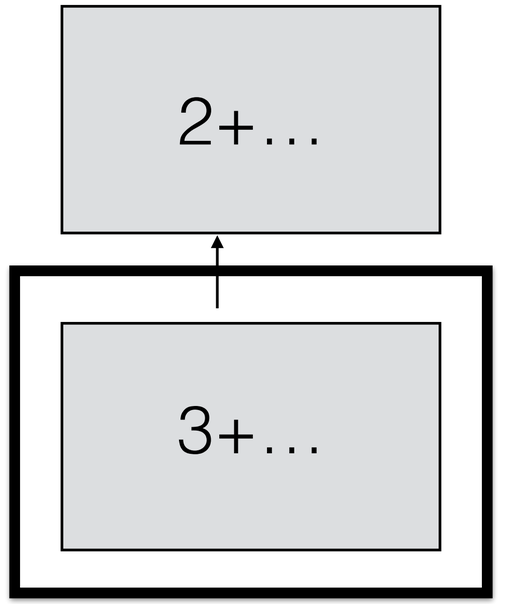
Call recursiveSum(3)



```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

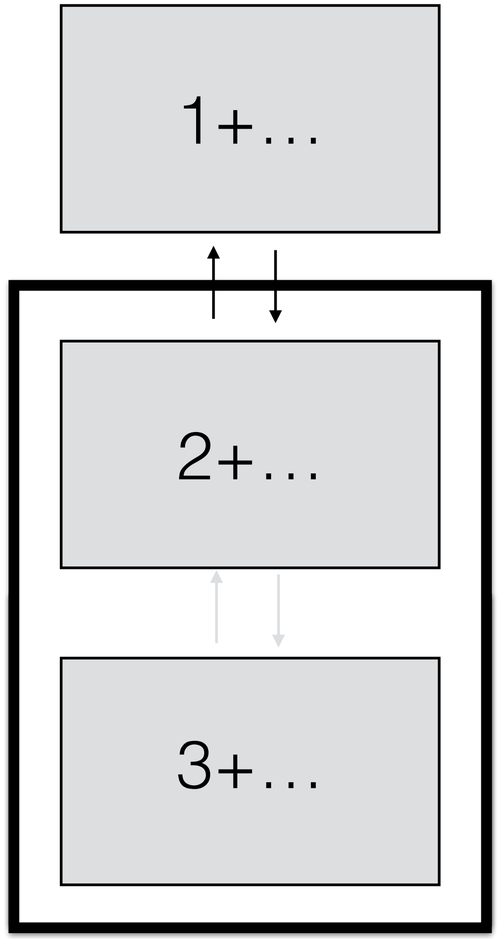
Call recursiveSum(2)



```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

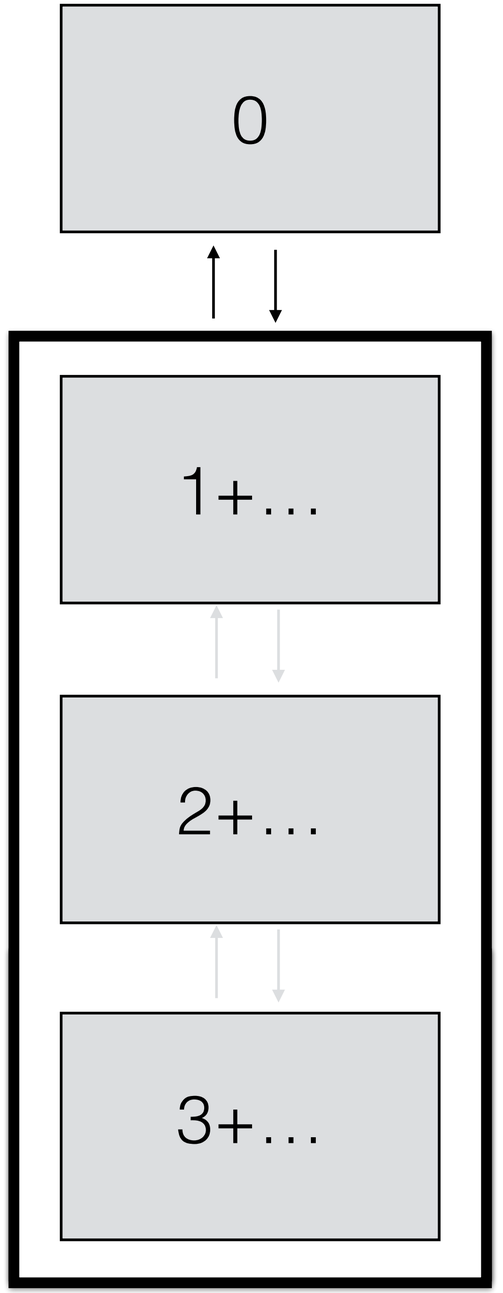
Call recursiveSum(1)



```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

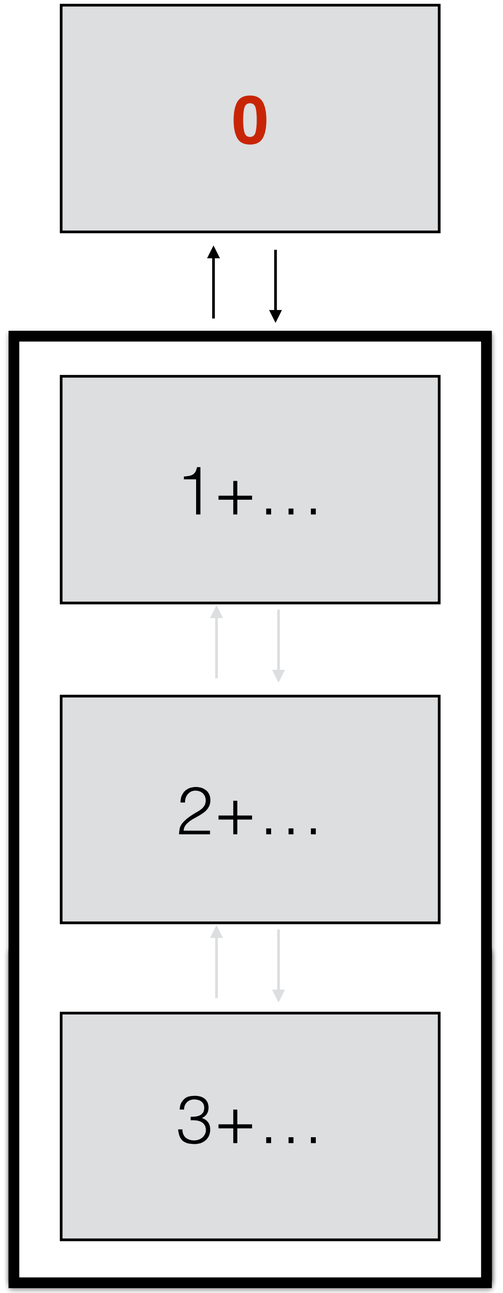
Call recursiveSum(0)



```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

Return 0, pop the stack

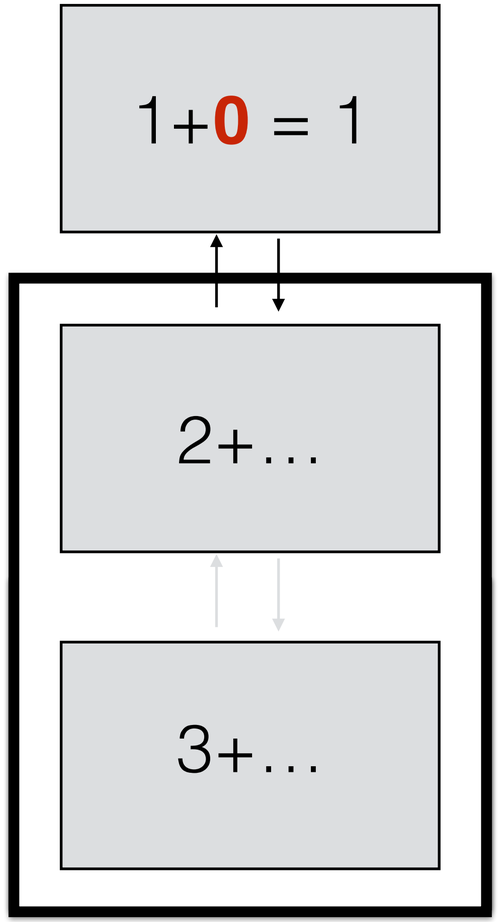


```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

Return 0, pop the stack

Return 1, pop the stack



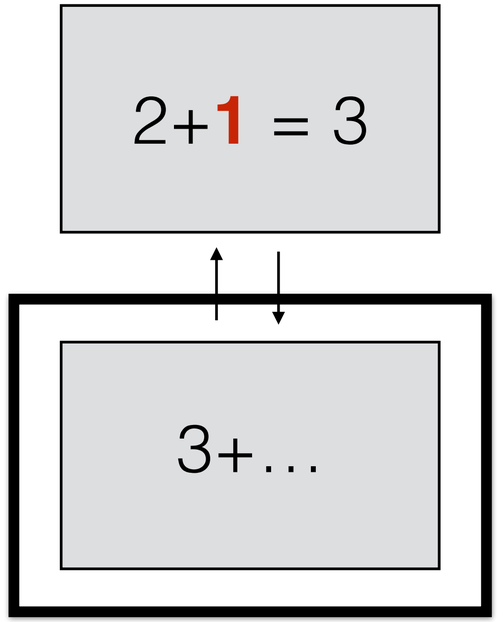
```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

Return 0, pop the stack

Return 1, pop the stack

Return 3, pop the stack



```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

Return 0, pop the stack

Return 1, pop the stack

Return 3, pop the stack

Return 6, empty the stack

3+**3** = 6

```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
    }

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

Return 0, pop the stack

Return 1, pop the stack

Return 3, pop the stack

Return 6, empty the stack

# For Review Seminar

In *stack.js* of folder *recursion*

“Simulate” the call stack actions for *recSum* (using iteration or otherwise) in *stackSum*

# Time Complexity for Recursive Programs

We can use the connection between recursion and stacks to say something about time complexity

In particular, the time complexity scales with the number of **function calls**

e.g. if each function call takes constant time and there are *n* function calls then the time complexity is *O(n)*

# Time Complexity for Recursive Programs

In particular, the time complexity scales with the number of **function calls**

This is also the number of operations on the stack

```
function stackSum(n) {
    if (n == 0) {
        return 0
    }
    var stack = new Stack();
    while (n > 0) {
        stack.push(n);
        n--;
    }
    var out = 0;
    while (stack.isEmpty() == false) {
        out += stack.peek();
        stack.pop();
    }
    return out;
}
```

Constant

# Time Complexity for Recursive Programs

In particular, the time complexity scales with the number of **function calls**

This is also the number of operations on the stack

```
function stackSum(n) {
    if (n == 0) {
        return 0
    }
    var stack = new Stack();
    while (n > 0) {
        stack.push(n);
        n--;
    }
    var out = 0;
    while (stack.isEmpty() == false) {
        out += stack.peek();
        stack.pop();
    }
    return out;
}
```

Constant

n iterations

# Time Complexity for Recursive Programs

In particular, the time complexity scales with the number of **function calls**

This is also the number of operations on the stack

```
function stackSum(n) {
    if (n == 0) {
        return 0
    }
    var stack = new Stack();
    while (n > 0) {
        stack.push(n);
        n--;
    }
    var out = 0;
    while (stack.isEmpty() == false) {
        out += stack.peek();
        stack.pop();
    }
    return out;
}
```

Constant

n iterations

n iterations

# Time Complexity for Recursive Programs

In particular, the time complexity scales with the number of **function calls**

This is also the number of operations on the stack

```
function stackSum(n) {
    if (n == 0) {
        return 0
    }
    var stack = new Stack();
    while (n > 0) {
        stack.push(n);
        n--;
    }
    var out = 0;
    while (stack.isEmpty() == false) {
        out += stack.peek();
        stack.pop();
    }
    return out;
}
```


**O(n)**

Given this connection between recursion and stacks

We can turn solution methods involving stacks into ones just using recursion

The stack is now “implicit”

# Problem 6:

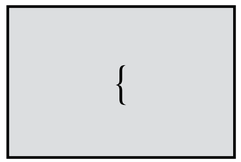
You've been given the task of helping to build a "pre-compiler" for a JavaScript teaching tool

This will conduct preliminary checks on code to look for syntax errors to avoid compile errors

**Your part in this project is to write code that checks for bracketing errors, both {} and ()**

Describe an algorithm and/or JavaScript implementation that flags an error when there is a bracketing error in the code

# Problem 6:



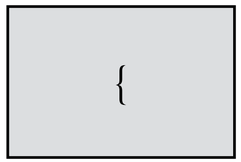

Instead of pushing this to a stack, make a function call

This has the same result of pushing data to a stack

We will pass the number of open brackets to the function and the rest of the string

When we encounter a closed bracket decrease the number of open brackets

# Problem 6:


In *recursion.js* let's look at the function *checkBrackets*

In your own time, complete the functions *findNext* and *checkBrackets* to check non-curly brackets as well

**We will go through solution in Review Seminar**

# Making Sorting Algorithms Recursive

Think about how the Insertion Sort and Bubble Sort algorithms can be made recursive

Try and implement them in the file *sort.js* in folder *recursion*

**For Review Seminar**