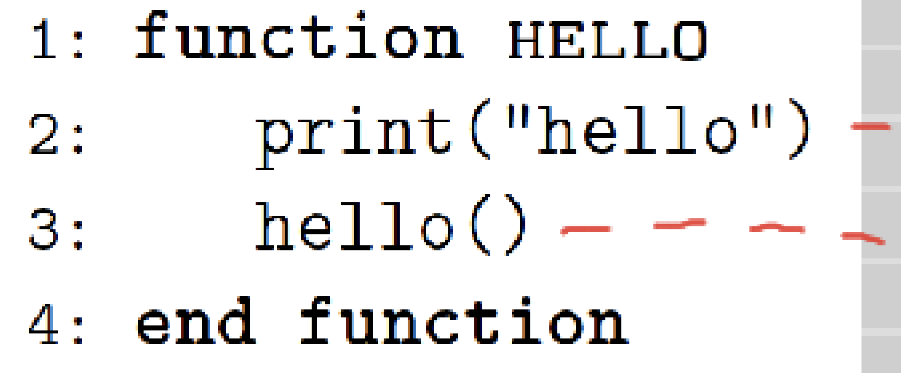
Understanding Recursion

Definition: An algorithm that calls itself



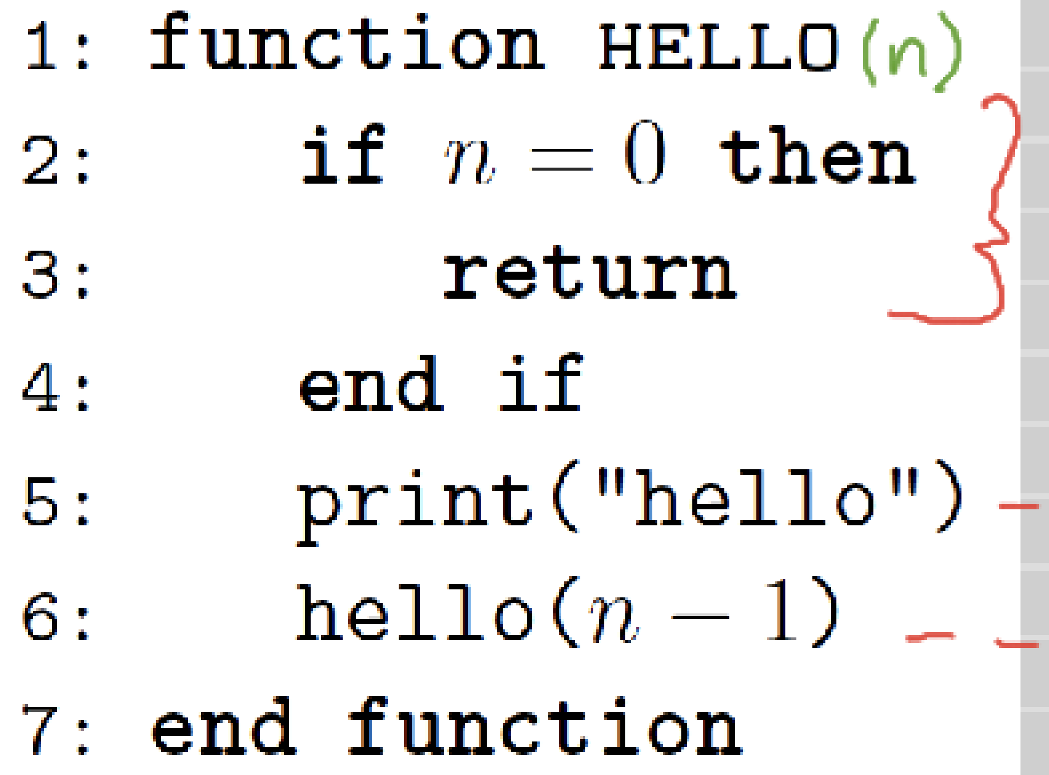
```
function HELLO
    print("hello")
    hello()
end function
```

instruction that prints hello

recursive call

*Problem: this algo doesn't have a condition to stop its execution.

Here is a well built recursive algo:

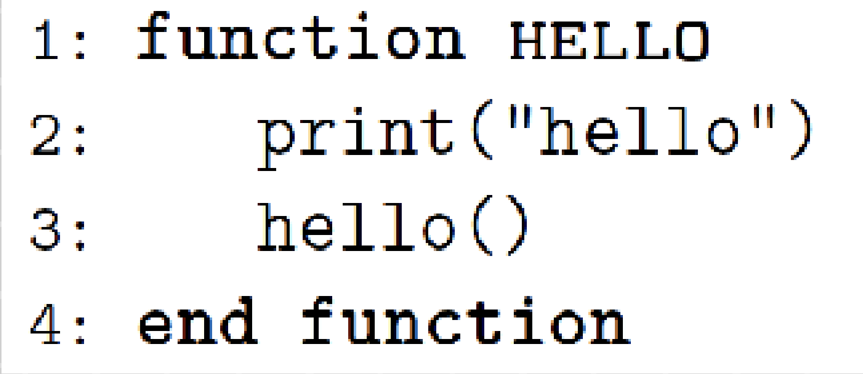


```
function HELLO(n)
    if $n = 0$ then
        return
    end if
    print("hello")
    hello($n - 1$)
end function
```

This is the base case. It is a condition that determines when the function stops execution

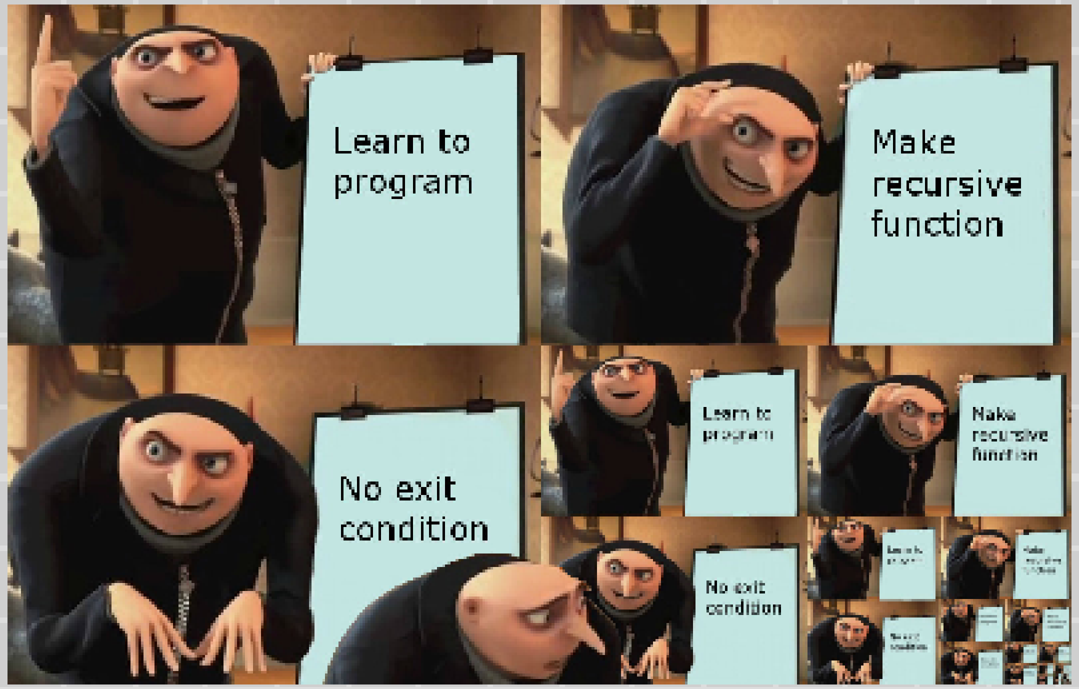
instruction to print

recursive call that brings us closer to the base case if input is a positive integer

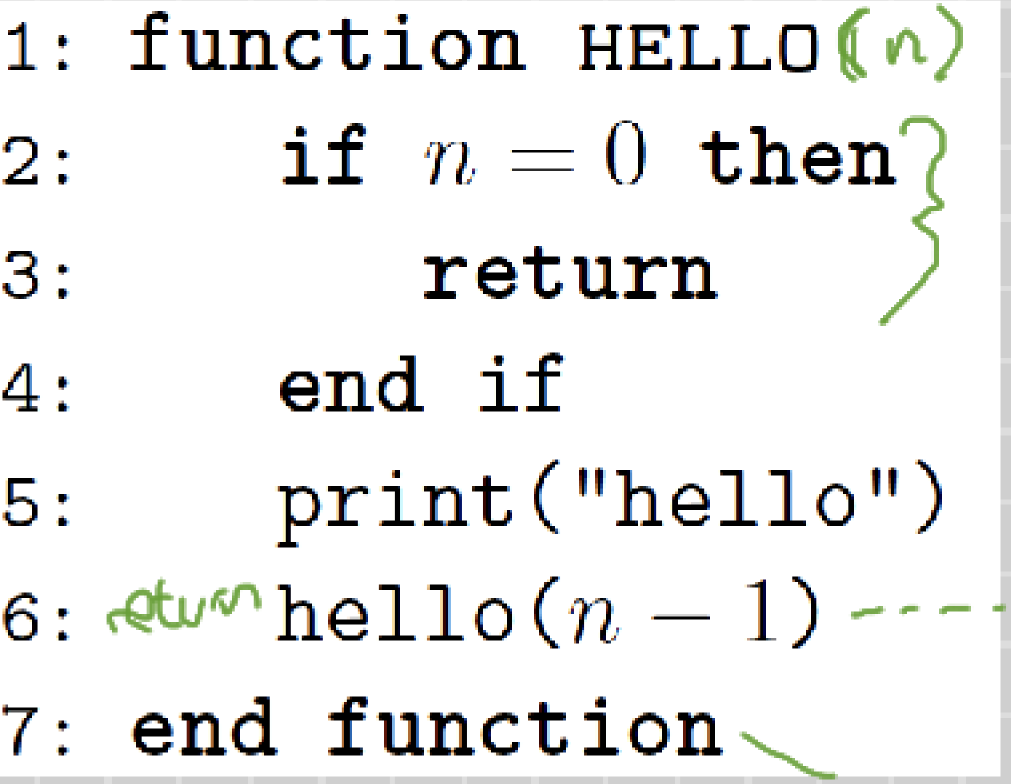


```
function HELLO
    print("hello")
    hello()
end function
```

we can get infinite recursion if either we dont have at least one base case or function calls itself with input that is not approaching base case or both.



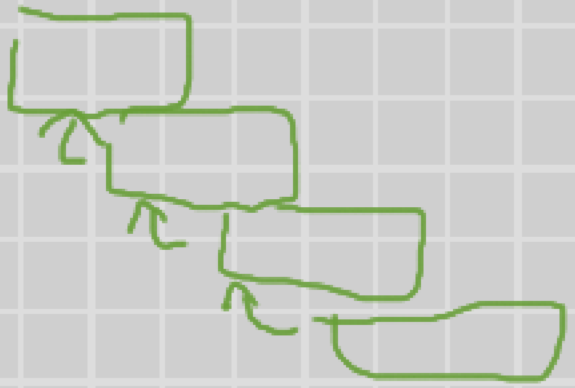
A well constructed recursive algo is one that executes a finite number of times

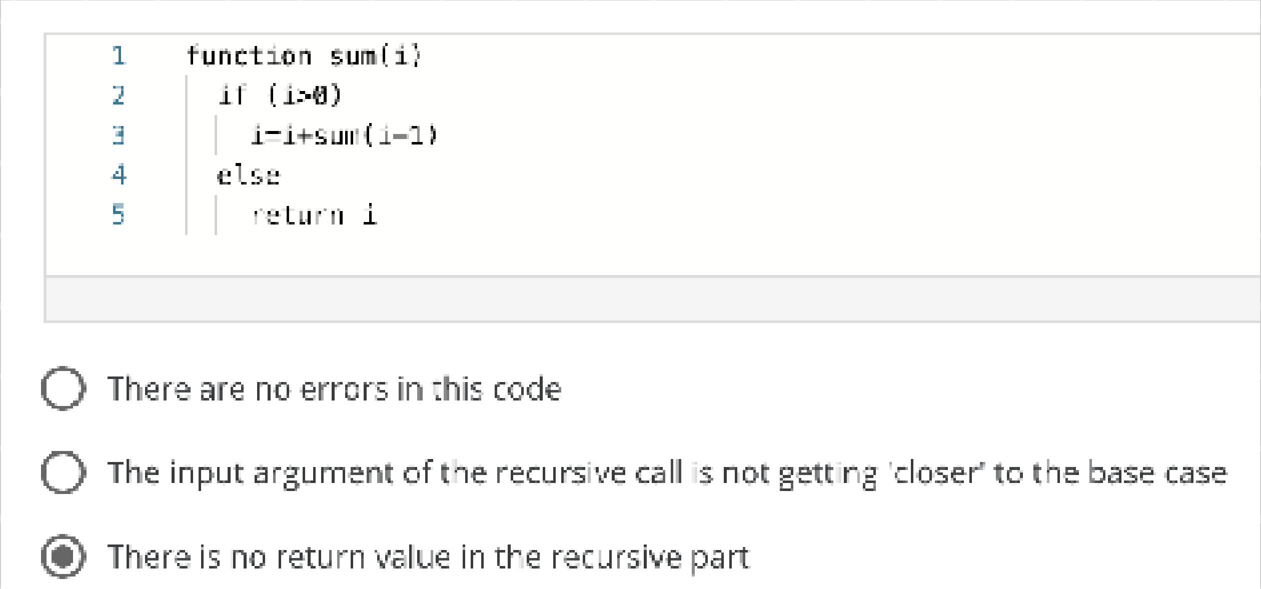


```
function HELLO(n)
    if $n = 0$ then
        return
    end if
    print("hello")
    return hello($n - 1$)
end function
```

To avoid infinite recursion we must include a base case

To avoid infinite recursion we must also include a recursive call with an input arg ument that gets closer to the base case





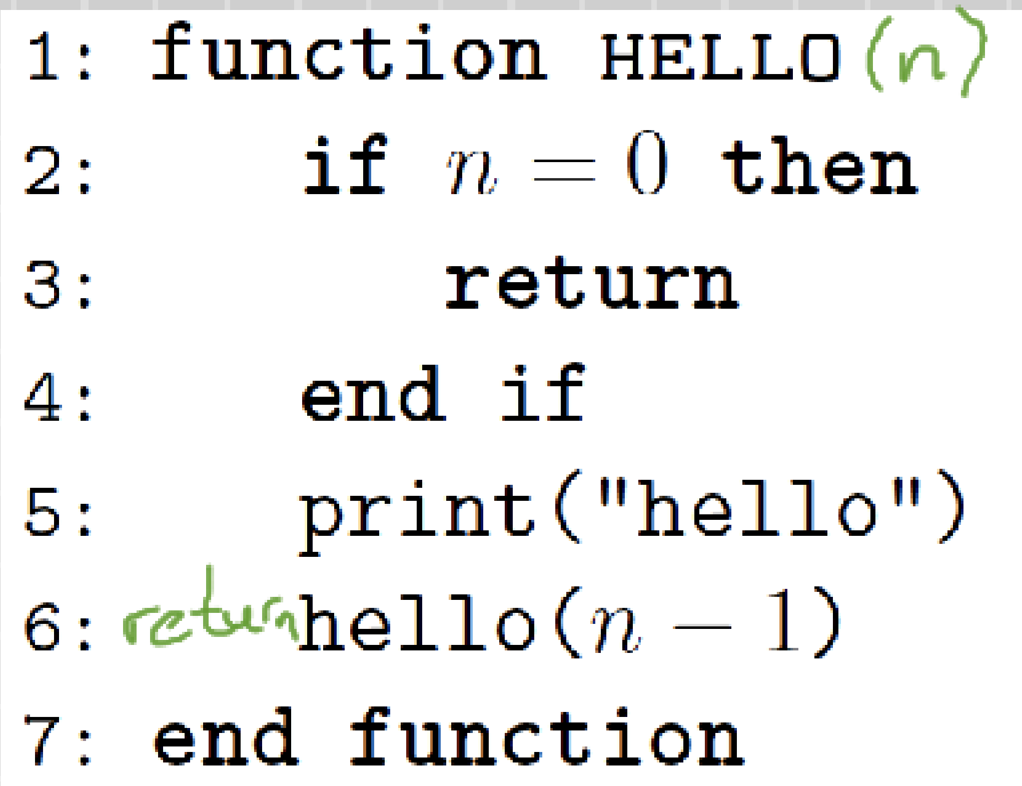
```
function sum(i)
  if (i>0)
    i=i+sum(i-1)
  else
    return i
```

○ There are no errors in this code

○ The input argument of the recursive call is not getting 'closer' to the base case

● There is no return value in the recursive part

Tracing a Recursive algorithm



```
function HELLO(n)
    if $n = 0$ then
        return
    end if
    print("hello")
    return hello($n - 1$)
end function
```

1) Call Hello(2)

2) Check base case. 2==0 is false.

3) Print hello

4) Recursive call: Hello(2-1): Hello(1)

↳ Original is paused

↳ 1st recursive call

1st Recursive call

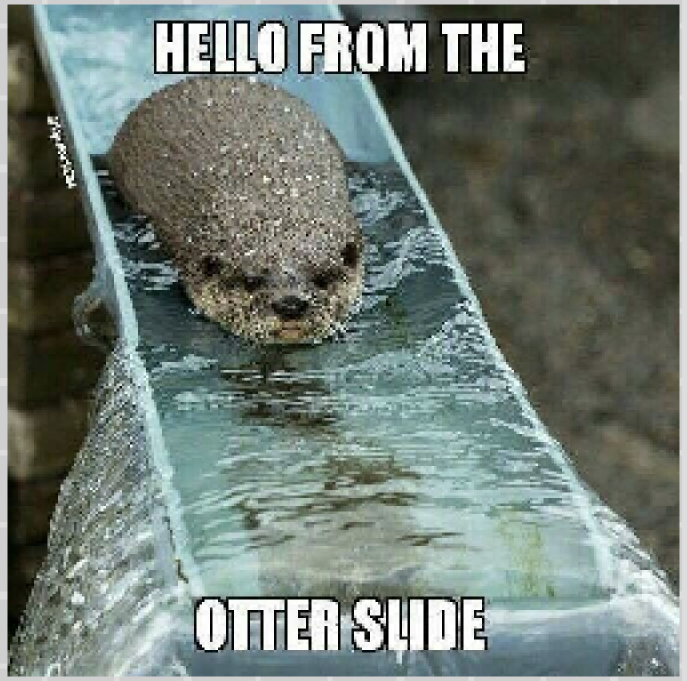
5) check base case. 1==0 is false

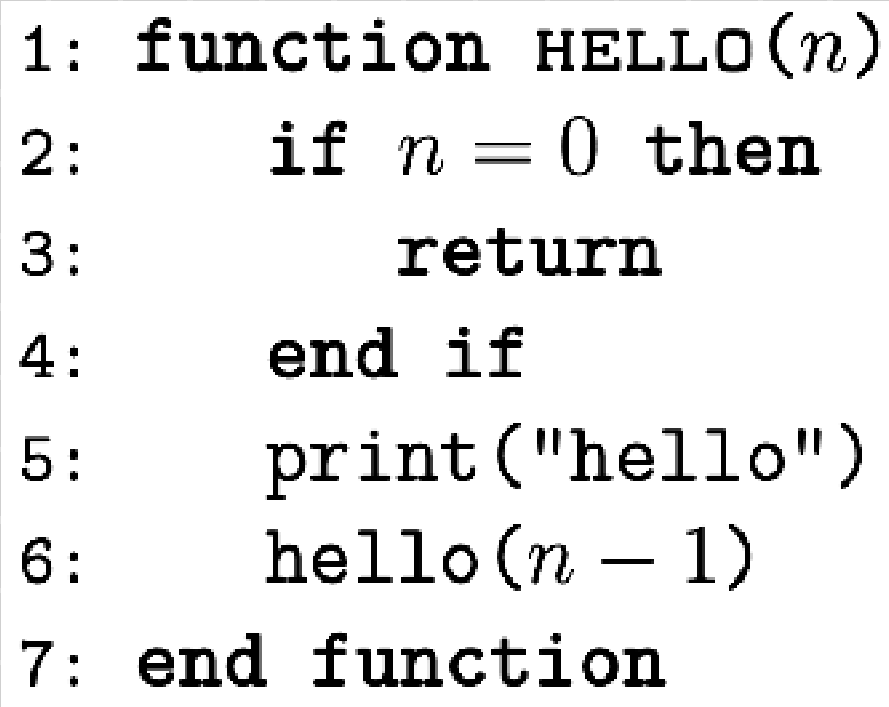
6) Print hello

7) Recursive call: Hello(1-1) : Hello(0)

↳ 1st recursive call paused

↳ 2nd recursive call created





```
function HELLO($n$)
    if $n = 0$ then
        return
    end if
    print("hello")
    hello($n - 1$)
end function
```

2nd recursive call

8) Check base case. 0 == 0 is True.

9) Return

Move back up the recursive call chain!

10) Return output/value to the function that activated the recursive call. So, we return to the first recursive call. The second recursive function call is removed from memory.

11) Return output/value to the function that activated the recursive call. So, we return to the original function call. The first recursive function call is removed from memory.

1) F3([12,3,5], 3)

5

2) if(F3([12,3,5],2) < A[3-1]

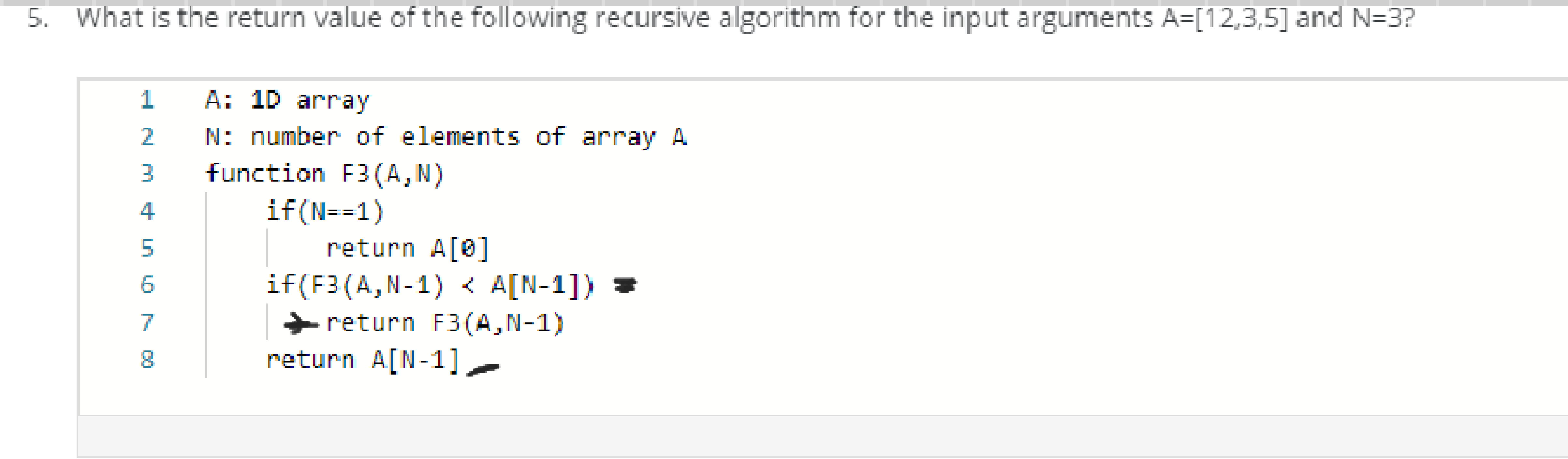
3

3) if (F3([12,3,5],1) < A[1]

4) return A[0]

↳12

We've reached the base case



5. What is the return value of the following recursive algorithm for the input arguments A=[12,3,5] and N=3?

```
1   A: 1D array
2   N: number of elements of array A
3   function F3(A,N)
4       if(N==1)
5           return A[0]
6       if(F3(A,N-1) < A[N-1])
7           return F3(A,N-1)
8       return A[N-1]
```

5) Return 12 to the comparison in step 3. Comparing 12<3, Fals

6) Move to line 8. Return A[1] = 3 ←

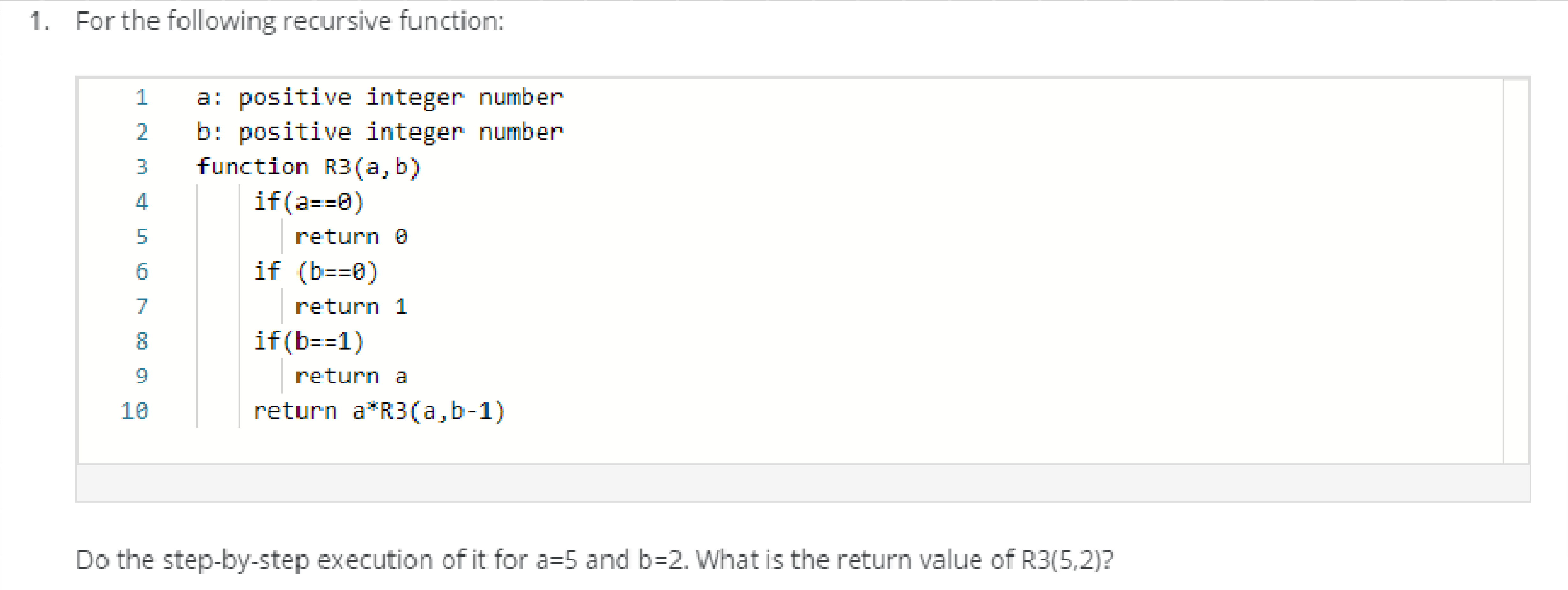
7) Return 3 to step 2. Compare 3 < 5, TRUE.

8) Return F3(A,

9)

10)

11)



1. For the following recursive function:

```
a: positive integer number
b: positive integer number
function R3(a,b)
    if(a==0)
        return 0
    if (b==0)
        return 1
    if(b==1)
        return a
    return a*R3(a,b-1)
```

Do the step-by-step execution of it for a=5 and b=2. What is the return value of R3(5,2)?

1) R3(5,2)

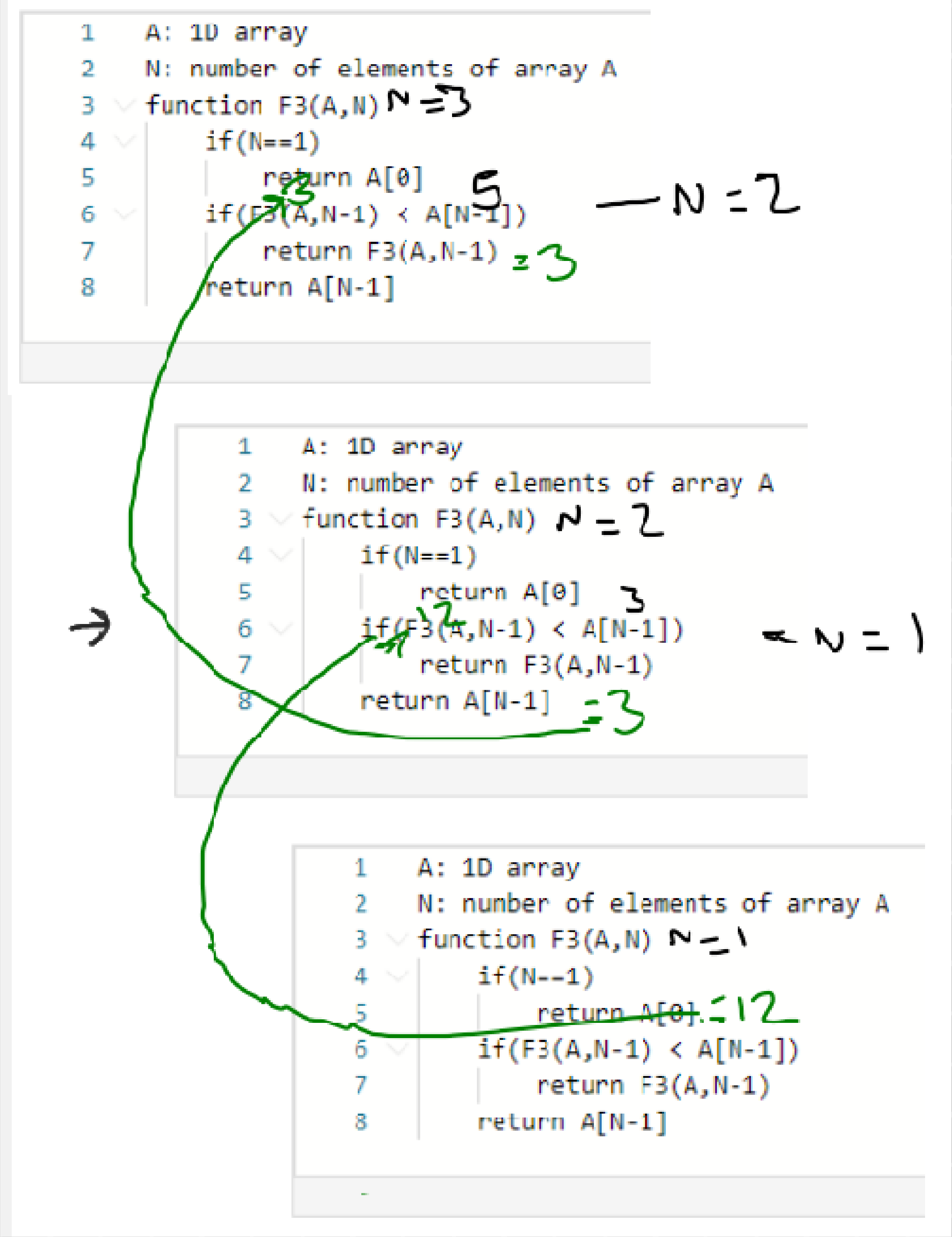
2) 5*R3(5,1)

3) return a (5)

4) $5*5 = \underline{\underline{25}}$

What task does this code perform?

a^b



```
1    A: 1D array
2    N: number of elements of array A
3    function F3(A,N)
4        if(N==1)
5            return A[0]
6        if(F3(A,N-1) < A[N-1])
7            return F3(A,N-1)
8        return A[N-1]
```

```
1    A: 1D array
2    N: number of elements of array A
3    function F3(A,N)
4        if(N==1)
5            return A[0]
6        if(F3(A,N-1) < A[N-1])
7            return F3(A,N-1)
8        return A[N-1]
```

```
1    A: 1D array
2    N: number of elements of array A
3    function F3(A,N)
4        if(N--1)
5            return A[0]
6        if(F3(A,N-1) < A[N-1])
7            return F3(A,N-1)
8        return A[N-1]
```

1) F3(A, 3)   N=3

2) Check base case. False

1st recursive call on line 6.

3) F3(A, 2) < A[2]   N=3

4) Check base case. False

2nd Recursive ~~call~~ call on line 6

5) F3(A, 1) < A[1]

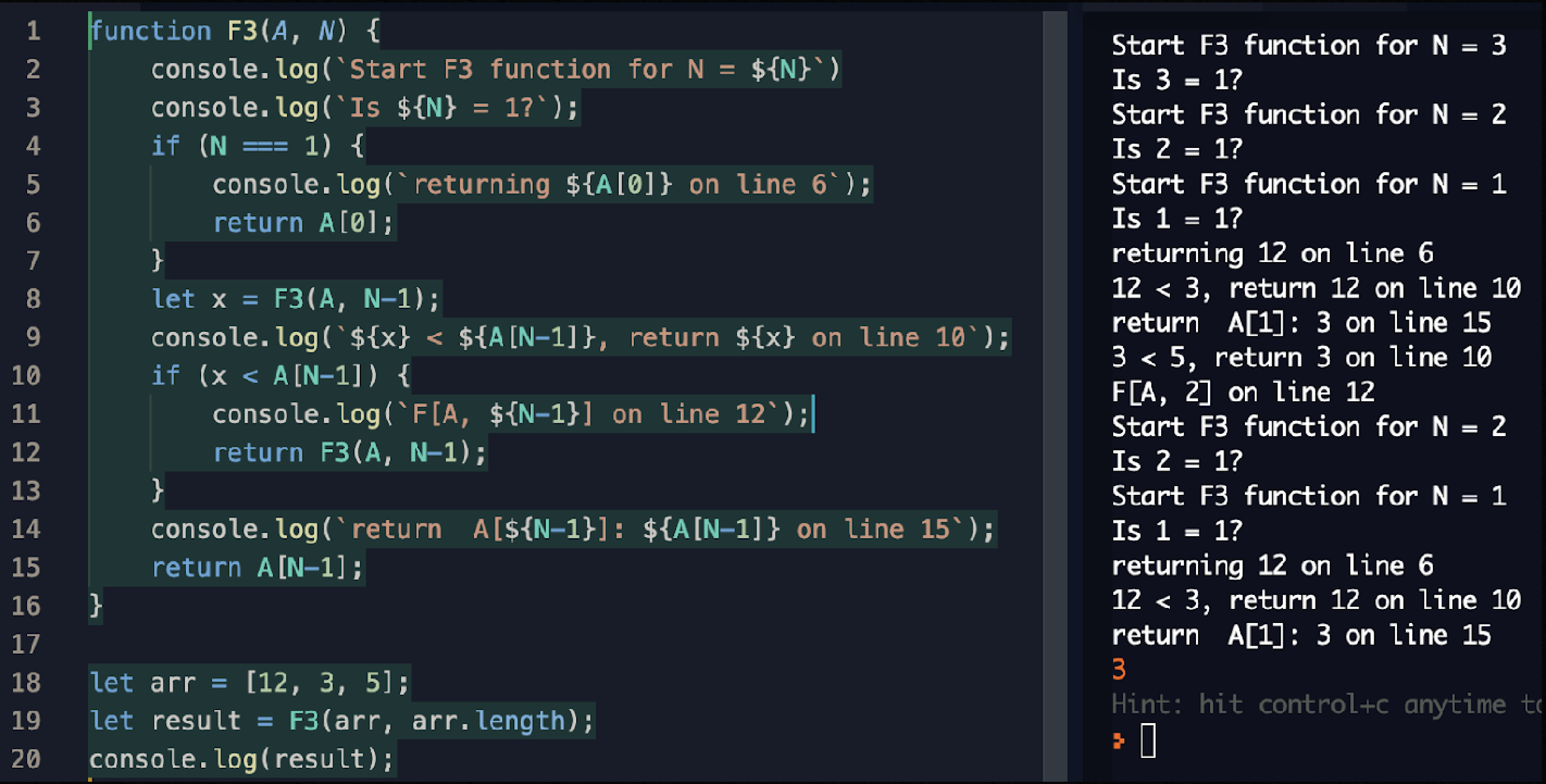
6) check ~~base~~ case. N==1. TRUE

7) Return A[0] (12).

Go back up recursive call

8) Return 12 to the calling function from step 5. 12 < , FALSE.

9)



```
1  function F3(A, N) {
2      console.log(`Start F3 function for N = ${N}`)
3      console.log(`Is ${N} = 1?`);
4      if (N === 1) {
5          console.log(`returning ${A[0]} on line 6`);
6          return A[0];
7      }
8      let x = F3(A, N-1);
9      console.log(`${x} < ${A[N-1]}, return ${x} on line 10`);
10     if (x < A[N-1]) {
11         console.log(`F[A, ${N-1}] on line 12`);
12         return F3(A, N-1);
13     }
14     console.log(`return  A[${N-1}]: ${A[N-1]} on line 15`);
15     return A[N-1];
16 }
17
18 let arr = [12, 3, 5];
19 let result = F3(arr, arr.length);
20 console.log(result);
```

```
Start F3 function for N = 3
Is 3 = 1?
Start F3 function for N = 2
Is 2 = 1?
Start F3 function for N = 1
Is 1 = 1?
returning 12 on line 6
12 < 3, return 12 on line 10
return  A[1]: 3 on line 15
3 < 5, return 3 on line 10
F[A, 2] on line 12
Start F3 function for N = 2
Is 2 = 1?
Start F3 function for N = 1
Is 1 = 1?
returning 12 on line 6
12 < 3, return 12 on line 10
return  A[1]: 3 on line 15
3
Hint: hit control+c anytime to
```

Iteration to Recursion

* Iterative algos: uses loops to repeat a set of instructions

Write an algo to print the number N to 1

```
N: integer input
function CountDown(N)
    for (i=n ; i≥1; i--)
        print(i)
```

* Recursive algos: repeates instructions by calling itself with an argument that gets closer to a base case.

```
N: integer input
function rCountDown(N)
    if (N<1)       }
        return     }  base case
    print(N)
    rCountDown(N-1)   }  recursive call
```

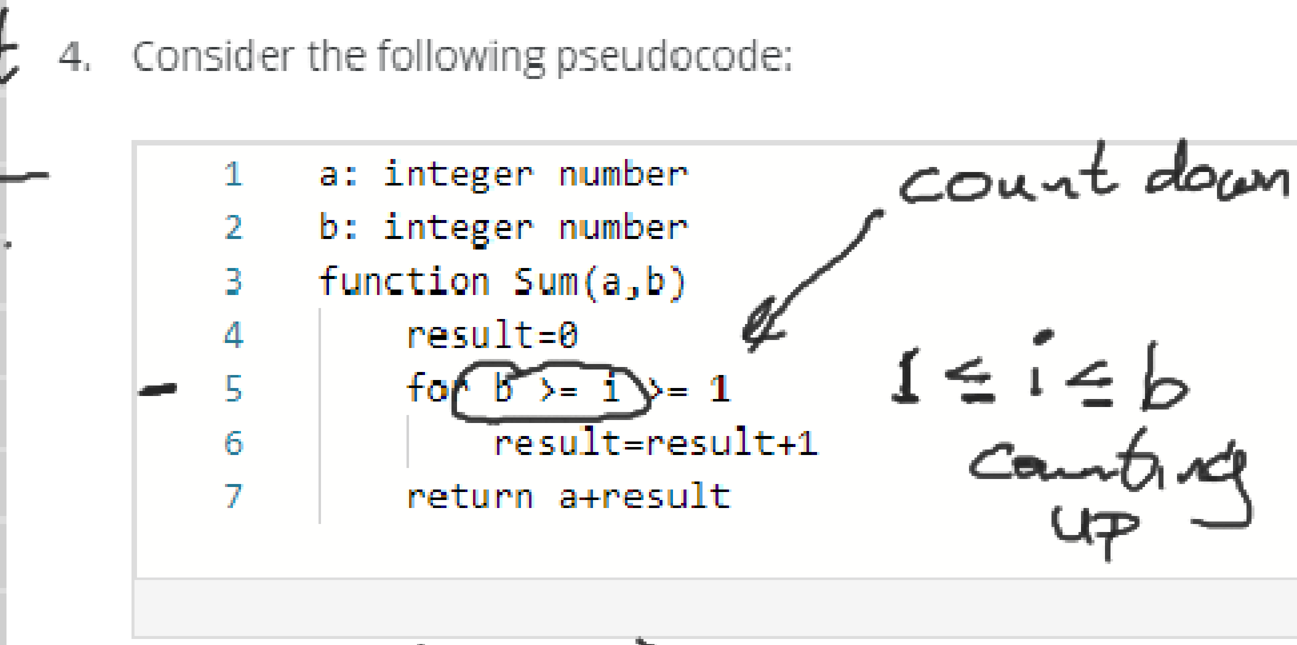
1) Both algo types must determine an intial condition.
   a) Iterative approach: uses a control variable (i). It is initialized with N, the input.
   b) Recursive approach: takes the input argument as the initial condition.

2) Both algo types must have an action that they repeat. Same in both algos.

3) Both algo types must determine when to stop execution of code.
   a) Iterative approach: *does this by verifying the condition i>= 1.
   *This condition must be true for the algo to continue executing code.
   * The control variable in the loop must be updated to get closer to the value where repetition stops.
   b) Recursive approach: *Checks a base case (N < 0) .
   * Must be true to stop execution of code.
   *Call the recursive function with an argument that approaches the base case with every new recursive call.

*Recursive function s can be more concise, however they can be difficult to understand. They use more memory because a new copy of the function is stored at each recursive call.

i	a	b	result	result=result+1	a+result
2	3	2	0	0+1=1	
				1+1=2	
1	3		1		
0					3+2=5
	6				



4. Consider the following pseudocode:

```
a: integer number
b: integer number
function Sum(a,b)
    result=0
    for b >= i >= 1
        result=result+1
    return a+result
```

count down

$1 \le i \le b$ counting up

Sum(3,2)

```
function recSum(a, b)
        if (b == 0)
            return a

    return 1 + recSum(a, b-1)
```

Trace: recSum(3,2) ← final return

1) 1 + recSum(3,1) ← 1+4=5

2) 1 + recSum(3,0) ← 1+3=4

↳ returns a

```
function recSum2(a, b)
        if (b == 0)
            return a

    return recSum2(a+1, b-1)
```

Trace: recSum2(3,2) ← paused a=5

1) recSum2(4,1) ←
↳ check base case. Call function. Paused ← a

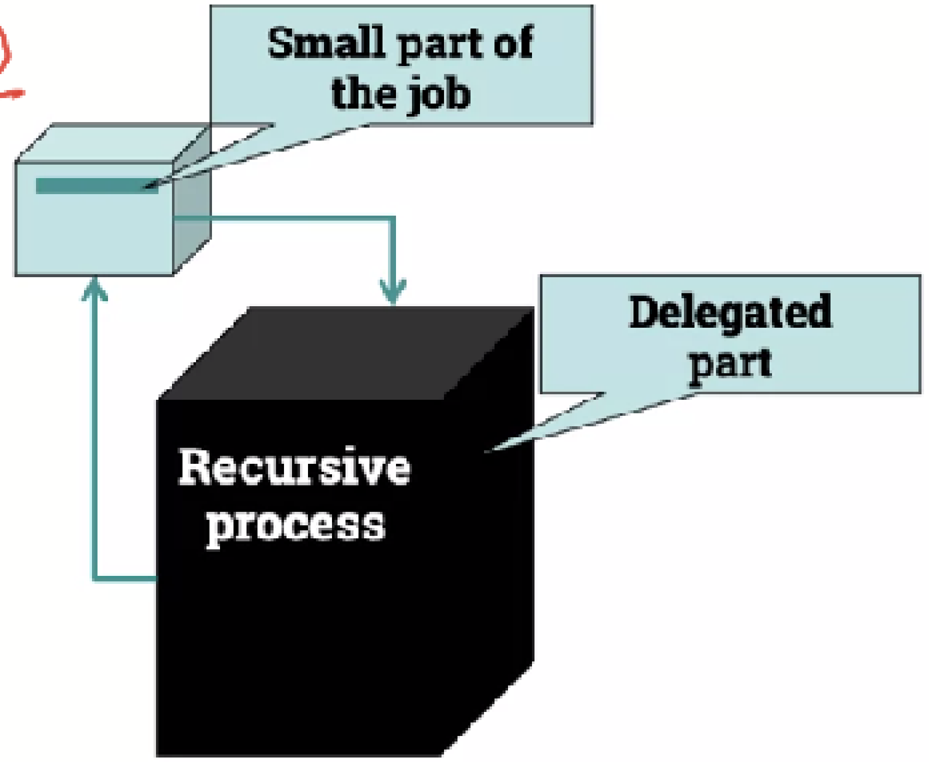
2) recSum(5,0)
↳ check base case. return a

Writing a Recursive algorithm

An approach to writing a recursive algorithm is to complete a small part and then delegate the rest of the work to someone else (someone else == recursive function call)

We can ignore details of the recursive processing.

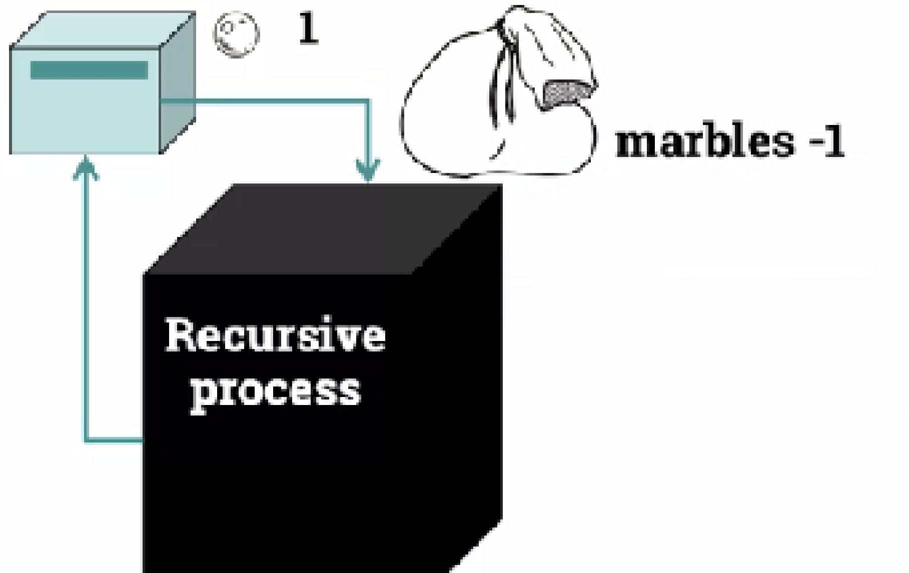
Bag of marbles. Determine how many marbles are in the bag.



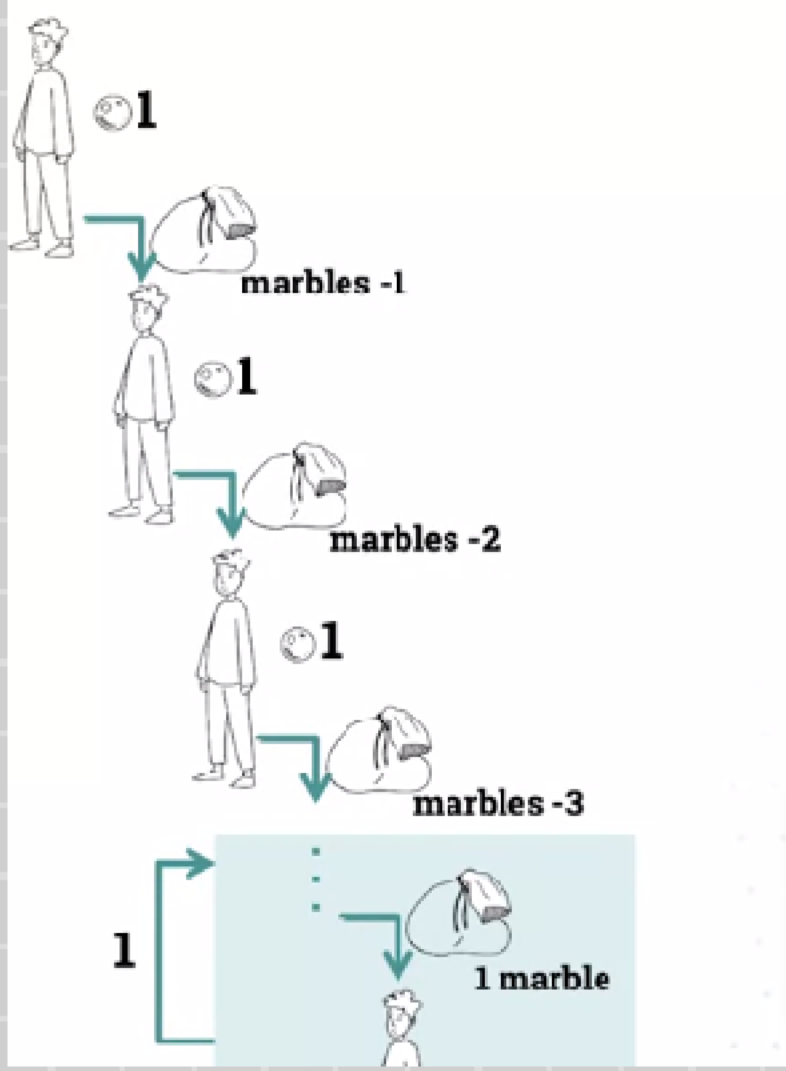
1) The small part we do is taking one marble and passing the bag to someone else, to do the same thing.

2) Delegated task for next person, is taking 1 marble from the bag that already has totMarbs-1.

3) The process is repeated until there is an empty bag. This represents the base case.



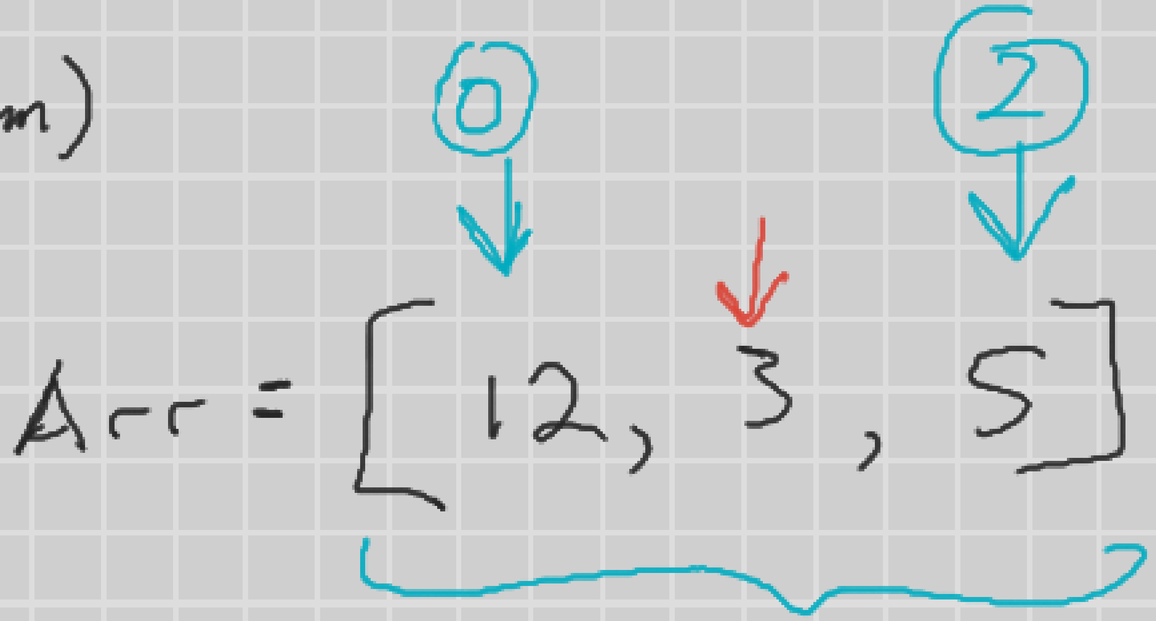
4)



Once we reach the base case, we no longer pass the bag around. We go back to the last person and ask how many marbles they counted, and keep going up the chain of people, and add the results of each person.

Given a Linear Search function

```
function LinSearch(Array, N (size of arr), search item)
        for 0 ≤ i < N
            if (Array[i] == item)
                return TRUE
        return FALSE
```

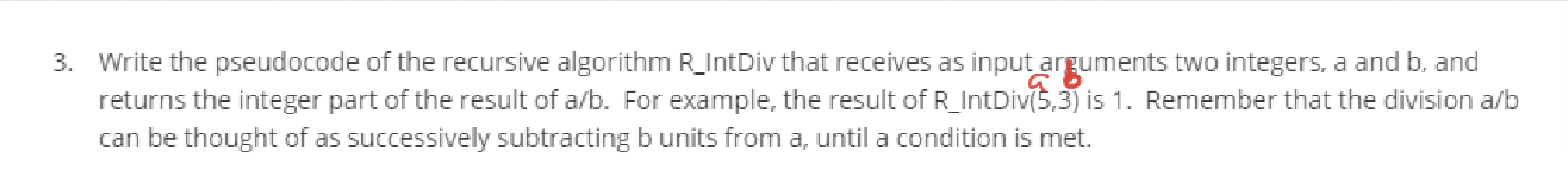


N = 3
indexes from 0-2

when N = 1 then we check A[0]

```
function recLinSearch(A, N, item)
        if (N == 0)
            return FALSE
        if (A[N-1] == item)
            return TRUE
        return recLinSearch(A, N-1, item)
```

smaller version of same problem



3. Write the pseudocode of the recursive algorithm R_IntDiv that receives as input arguments two integers, a and b, and returns the integer part of the result of a/b. For example, the result of R_IntDiv($\overset{a}{5}$,$\overset{b}{3}$) is 1. Remember that the division a/b can be thought of as successively subtracting b units from a, until a condition is met.

```
function RIntDiv(a, b)
    → if (a < b)
        return 0        * because a<b, there no b's in a
    if b==0
        return -1
    return 1 + RIntDiv(a-b, b)
```

Denis explains that we may not need to include this line, in C++ for example, a divsion by 0 exception would be thrown.

Trace RIntDiv(3,0):

1) 1 + RIntDiv(3,0)

Mon May 10th we write C++ code and analyse.