Expand the following function, 'loop', to call 'cb', which is a function, 'count' times, where 'count' is an integer. This must be done using recursion, and with no loops.

```
let loop = (count, cb) => { }
```

```
let loop = (count, cb) => {
    if (count !== 0) {
        cb();
        count--;
        loop(count, cb);
    }
}
```

Function Size → C(n) where 'C' is the complexity of the callback, and 'n' is the number of 'count's. Applies to Memory Used and Time to complete the function.

#1 ↑

Phelan

DAVID LINDAHL

PROBLEM: WRITE FUNCTION CALLED LOOP, THAT TAKES COUNT & CALLBACK

- RECURSIVE: CALLS ITSELF
- CALLS CALLBACK AS MANY TIMES AS COUNT

no for loop/while :^)

EXAMPLES:

loop equals a function runs

loop (5, console.log(josh))
↓
= x5

PSUEDOCODE

let loop (counter, callback) => {

X for number of counter
call the callback

if counter > 0
call callback & counter times

loop function takes counter & callback
if counter > 0
run callback

REAL CODE

let LOOP, (COUNTER, CALLBACK) = {
if (COUNTER > 0) {
callback();
Counter--;
};
loop();

};

TEST

loop (5, console.log(dog));
↳ 5 dog

loop (0, console.log(dog));
↳ undefn.

Catherine Looper

Problem Domain

- create a recursive function called loop that accepts a count and a callback as its parameters
- let loop = (count, callback) => { }

Example: loop (2, callback)

loop (2, console.log ('Cat'));

'Cat'
'Cat'

Code:

```
let loop = (count, callback) => {
  if (count > 0) {
    callback();
    return loop(count-1,
                callback)
  };
};
```

Pseudocode

call the callback function if count still above 0

loop(3, console.log('cat'));

if

callback function() => {
return count - 1
}

OE();

#2 Problem Domain:

- a function that accepts only one function as its parameter.
- The function passed in can only execute once (regardless of how many times it's called

```
let acceptFunction = (oneExecution) => {
  let count = 1;
  let once = oneExecution = () => {
    count--;
  }
  if (count === 0) {
    once = null;
  }
  return once;
}
```

Jeff Kusowski

input count = integer
cb = any call back

output = call back is called <count> number of times

Assume count is integer > 0
no return

Recursive function of form
let loop = (count, cb) => {
}

Examples	count	cb	Output
	2	console.log('hello')	hello hello
	4	count - 1	4, 3, 2, 1

Pseudocode

loop starts
  Call back
  count--
  if count
  loop(count, cb)

Code

```
let loop = (count, cb) => {
    if (count > 0) {
        cb();
        count--;
        return loop(count, cb);
    }
}
```

```
let once = (cb) => {
    let oneTime = (cb) => {
        cb();
        oneTime = null;
    }
    return oneTime;
}
```

function 'once' returns the function passed in. That function will only execute once.

example once(

once (function)
  let oneTime = (function)

return ()

Andrew

define a function loop which takes a count and a callback as parameters and recursively loops the callback count # of times.

e.g. `loop(3, () => console.log(hello))`

Should return
// hello
// hello
// hello

define function
perform input validation
run callback
recursively call loop with (count-1) and callback as parameter, If count > 0

```
const loop = (count, callback) => {
  // input validation here
  if (count > 0) {
    callback();
    return loop((count-1), callback);
  };
};
```

Create a function that takes another function as input and returns another function. The outer function can only be run once

fun (foo) = returns newfoo

newfoo ( ) = runs

newfoo ( ) = doesn't run

define function
input validation
return function with new property, count = 1, that decrements when that function runs. that function can only run if count > 0

```
const Once = func => {
  let newfunc = {};
  newfunc.counter = 1;
  newfunc = () => {
    if(newfunc.counter > 0){
      newfunc.counter--;
      func();
    };
  };
  return newfunc;
};
```

Example:
```
let impl = Once(somefunc)
impl() // run somefunc
impl() // do nothing
```

Jacob Evans

Self-made Higher Order function Big O(n)
recursive loop counting . n of Count

```
const Looper = (ctr, callback) {
  let ctr;
  (ctr !== 0 ) ? ctr-- : null;

  Callback(ctr, Looper );

};
```

---

func = (func2 = (arg, callback) => returns New function ())

Callback

Cameron

1)
```
const loop = (count, callback) => {
    if (count <= 0) {
      return;
    }
    callback(); O(1)
    return (count - 1, callback);
};
```

```
const exampleCB = () => {
    console.log('Hello');
};
```

Big O:

Space: Recursive solution will use O(n) on callstack memory where n is equal to count

Time: O(n) where n is equal to count

```
let x = once(console.log);
x();
x();
x();
```

2)
```
const once = someFunction => {
    let hasBeenCalled = false;
    if (!hasBeenCalled) {
      hasBeenCalled = true;
      return someFunction;
    }
    return;
}
```

Big O:

Space: O(1) constant (for examples)

Time: O(1) constant

But it depends on the curried functions operations...

Kerry Nordstrom 12/8/17

Restate: Create a function that takes as parameters a count and a callback, then runs as long as the count is.

Example/Pseudo: f(5, cb) = count is 2, function is looped 5 times

Declare function that takes in count & callback
Determine if count is above zero
Instantiate callback that contains...

5

Code:

```
let loop = (count, callback) => {
  if(count < 0){
    return;
    callback();
  loop(count-1, callback);
  };
};
```

① Problem: write a function that takes in a function as its only argument and returns a new function that can be called infinitely many times, but will only execute the first time.

② Example:

```
const yum = () => {
  console.log('first')
};
let onlyOnce = once(yum)
onlyOnce(); // 'first'
onlyOnce(); //
onlyOnce(); //
```

③ Figure out:
1. Use currying & closure!
1) Double arrow
2)

④

```
const onlyOnce = (cB) => {
  let first = true;
  const go = cB => {
    if(first){
      cB();
      first = false;
    }
    return go;
  }
};
```

Q: Write a recursive function called loop

EX: count to 10.

```
num=10
Count=0
ADD 1 to(Count) if(count) < (num);
+1 to(Count)
if (count) == (num)
return (count);
```

Fredric

```
const count (loop) => {

    var   cb  =10
    var   count=0

let  loop=(count, cb)=  {

if  (count < cb; #count)

}else{
return (count),

};
```

Seth Donohue

① let count = 5;

```
const loop = function(count, callback){
    count--;
    if(count>=0){
      callback();
      loop(count, callback);
    };
```

②
```
const once = function ( functionOne) {
    let ran = true;
    return functionTWO ( x );
      if( x = ran) break;
```

```
loop = (count, callback) => {
```

example
pseudo

```
loop(count, callback){
  count--;
  if (count > 0){
    callback();
    loop(count, callback);
  }
  else {
    loop complete
  }
```

```
function loop(count, callback){
  if(count > 0) {
    count --;
    callback();
    loop(count, callback);
  } else {
    console.log('loop complete')
  }
```

(1)

```
let x = Once (() {
  let on = true
  () => { if(on)
    () => {}
    console.log 'this'  on = false
  }
```

x (log 'this')

x (log 'this') do nothing

count

def

e

p

return

=> ...

...rue(fn) {

GNicholas Caergnon

```
Let Loop = (x, cb){
    Loop.count = 0
    return cb(x);
}
```

```
ct = 5
cb(x) => {
    Loop.count++;
    if(Loop.count == x)
        return x
    cb(x);
}
```

returns 5

```
const infini = (cb) => {
    infini.counter = 0;
    if(infini.counter === 0){
        infini.counter++;
        return cb();
    }
    return new Error(....)
};
```

```
cb = (x) => {
    console.log(infini.x)
    cb(x)
```

Anthony

```
let function = (infunction) => {
    infunction (value) => {
}
```

```
loop(2, callback)

let loop = (count, callback) => {
    let count = count
    if (count < 1) {
    return count
} else {
    loop(count - 1, callback)
}
```

1)
I want to run the callback as many times the value of count is.

I want to subtract 1 from count

count.value

2)
a function that takes in another function as it's argument. the outer function must end after it runs once.

1. PROBLEM define f-n loop (count, cb) that calls cb recur. count times

2. Simple Code

loop (2, cb) ⇒ console.log('running cb 1 time')
count
console.log('running cb 2 time')

3. REVIEW

declare loop = (count, logger) => {
f-n
assign count to var.
Check if count is a number
if count till hitting condition
OR count
Run callback
call loop( )

loop (2, cb)
let count;
if(count-1 !==0)
logger(console.log
loop()
loop(1, cb)

Pedja

4. CODE

```
LET loop = (count, logger)=>{
  let count;
  let newCount=count-1;
  if(newCount !==0){
    let logger= () =>{
      console.log('calling cb f-n');
  }
  loop(newCount, logger);
```

Shannon

```
let loop = (count, callback) => {
    let n = count;
    if (n > 0){
        loop(n-1, callback);
        callback();
    };
};
```

i.e.) `loop(5, () => console.log('hi'))`

1) n = 5
   loop(4, log('hi')) 'hi'
2) n = 4
   loop(3, log('hi') 'hi'
3) n = 3
   loop(2, log('hi') 'hi'
4) n = 2
   loop(1, log('hi') 'hi'
5) n = 1
   loop(0, log('hi')
6) n = 0 - break