UC Berkeley's CS10 Spring 2017 Final Exam : Prof. Dan Garcia

`Your Name (first last)` `SID` `Lab TA's Name`

`← Name of person on left (or aisle)` `Name of person on right (or aisle) →`

What's that Smell? Oh, it's Potpourri! (2 pts each for 1-6, low score dropped)

Fill in the correct circles & squares completely…like this: ● *(select ONE)* ■ *(select ALL that apply)*

Question 1: Free points for attending the alumni panel! What happened? (select ONE)

○	○	○	○	○	○	○
Panelist came late	Panelist left early	Standing ovation at end	Given flowers at the end	Said Dan's nickname	Said their nicknames	They ate lunch on stage

Question 2: Which of the following is *true*? (select ONE)

○ Moore's Law says that processor performance doubles every 18-24 months
○ Moore's Law ended in 2005
○ Most of the time in scientific computer code is spent doing arithmetic calculations
○ All of the above
○ None of the above

Question 3: Around 1990, what happened in the world of computing (from an HCI viewpoint)... (select ONE)

○ It began an era known as "Mainframe computing".
○ It began an era known as "Ubiquitous computing".
○ It began an era known as "Personal computing".
○ We were not able to cool our chips, so they went parallel and added cores.
○ HCI's focus changed to focus on Psychology, and the Cognitive Science of Individuals.
○ HCI's focus changed to focus on Human Factors.
○ None of the Above.

Question 4: If today someone proved that P *did not* equal NP, that'd mean for the *FIRST TIME*… (select ONE)

○ We can now *verify* a randomly-generated subset sum problem solution in polynomial time.
○ We can now solve the subset sum problem *approximately*.
○ We can now solve the subset sum problem *in polynomial time*.
○ We can now solve the subset sum problem *using a heuristic*.
○ We can now write a program that could tell whether any other program would eventually halt on its input.
○ None of the above.

Question 5: Internet standards started moving to IPv6 (and away from IPv4) because… (select ONE)

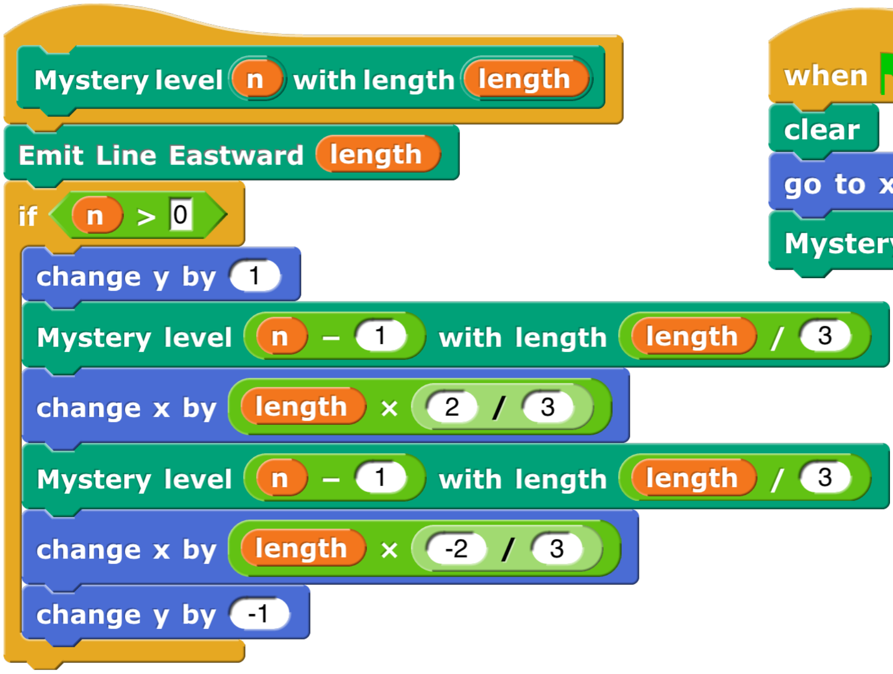
○ IPv6 ran over *fiber-optic cable*, whereas IPv4 ran on *copper cables*.
○ IPv6 allows for *wireless* connectivity, whereas IPv4 only allowed *wired* capabilities.
○ IPv6 allowed for an *end-to-end architecture*, whereas IPv4 did not.
○ IPv6 used a *hexadecimal* instead of *binary* encoding.
○ None of the above.

Question 6: Recall the knapsack problem involves calculating the most money you can carry in a weight-limited knapsack, given boxes of different values and weights (you may choose as many boxes of the same kind as you wish). You are given a bag that holds **20lbs** and three boxes: **A=$15@15lbs**, **B=$5@6lbs**, **C=$6@7lbs**. How much better does the *optimal* solution do over the *greedy* solution? (select ONE)

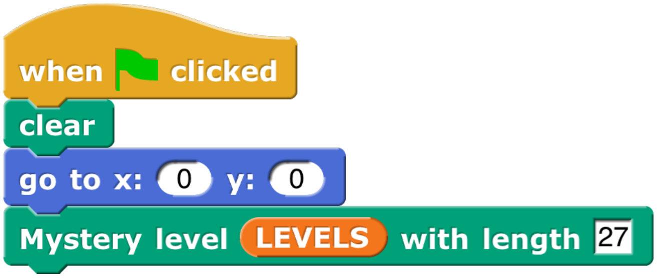
○	○	○	○	○	○	○	○	○	○	○
$0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$10

Question 7a-b: *Magical* *Mystery* *Tour, step right this way…* (6 = 2+4 pts) SID: ______________

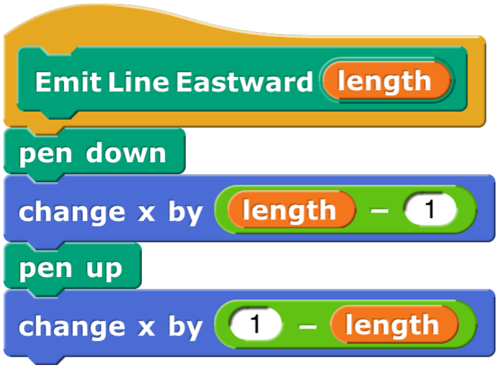
Consider the following two blocks and setup code:



```
Mystery level (n) with length (length)
Emit Line Eastward (length)
if <(n) > 0>
    change y by 1
    Mystery level ((n) − 1) with length ((length) / 3)
    change x by ((length) × (2 / 3))
    Mystery level ((n) − 1) with length ((length) / 3)
    change x by ((length) × (-2 / 3))
    change y by -1
```



```
when ⚑ clicked
clear
go to x: 0 y: 0
Mystery level (LEVELS) with length 27
```

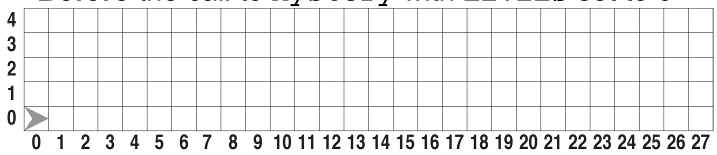


```
Emit Line Eastward (length)
pen down
change x by ((length) − 1)
pen up
change x by (1 − (length))
```

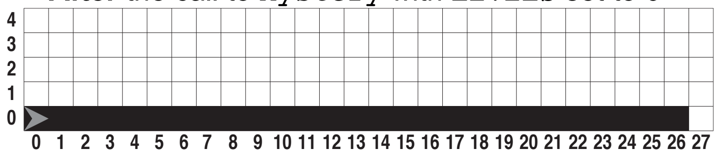
We're now to going to zoom in on pixels affected by calls to `Mystery`; the sprite always starts facing right in the lower left, and the pen is in the *center* of the sprite. The top two images are the pixels before and after a call to `Mystery` with `LEVELS` set to 0. Note the `Emit Line Eastward` command fills in exactly `length` pixels: `0` through `length-1`. Your job is to shade in (completely!) *all* the pixels that will be colored in after calls to Mystery with `LEVELS` set to **1** and **3**; don't worry about drawing the location of the sprite at the end.

*(If you need scratch space for the `LEVELS` set to **3** case, use the "Before the call to `Mystery`" pixels below)*

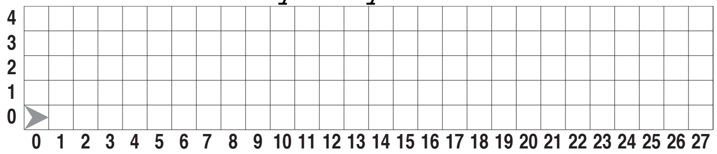
Before the call to `Mystery` with `LEVELS` set to 0



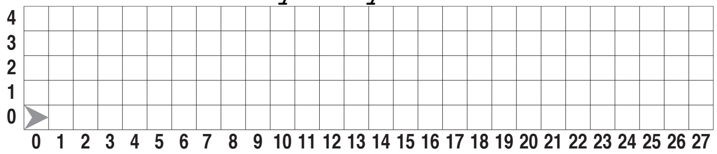
After the call to `Mystery` with `LEVELS` set to 0



After the call to `Mystery` with `LEVELS` set to **1**

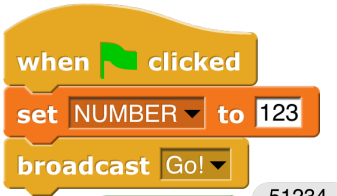


After the call to `Mystery` with `LEVELS` set to **3**

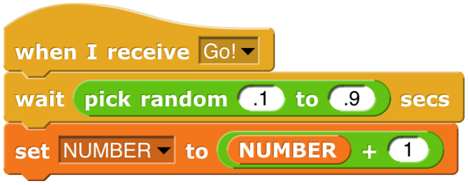


Question 8: *It's a rat race (condition) out there…* (5 pts)

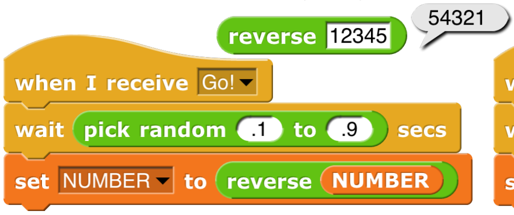
`reverse` reverses the order of the digits, and **`right`** rotates the digits to the right (the one furthest on the right moves over to the left). How many *different, unique values* of NUMBER are there at the end? (select all that apply) *Note: "**`set`**" blocks run instantaneously and "atomically" (i.e., aren't interrupted by any other script)*



```
when ⚑ clicked
set NUMBER to 123
broadcast Go!
```

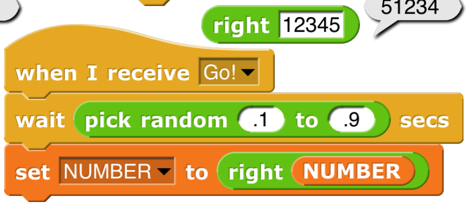


```
when I receive Go!
wait (pick random .1 to .9) secs
set NUMBER to ((NUMBER) + 1)
```



```
reverse 12345
```

```
when I receive Go!
wait (pick random .1 to .9) secs
set NUMBER to (reverse (NUMBER))
```



```
right 12345
```

```
when I receive Go!
wait (pick random .1 to .9) secs
set NUMBER to (right (NUMBER))
```

□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
123	124	125	132	133	142	213	214	231	232	312	313	321	322	412	421	423	424	431	432	433

Question 9a-c: Berkeley `Python` Flying Circus… (8 = 2+2+4 pts) SID: ______________

We've always appreciated how easily Python handles functions as data. There's no **`call`**, **`run`** or **`launch`** block, you can just receive a function as a parameter and call it directly. This gives us an idea. You remember when we talked in lecture about how **`combine`** should always work if you're given an associative dyadic (two-input) function, and how it paired its values up was below the abstraction line (i.e., you shouldn't know about it)? It could evaluate it any way it wants and the answer should still be the same. For example, if given + on a list of 5 elements A through E, it could return ((((A+B)+C)+D)+E) or ((A + ((B + C) + D)) + E) etc, and it will all work because + is an associative dyadic function. Taking that to heart, we've rewritten **`combine`** (as **`mycombine`**) below that finds a random adjacent pair of elements, calls its function **`f`** on them and puts the result back in line:

(note **`randint`** below is *inclusive* of its endpoints like pick random (1) to (3), so it might return 1,2, or 3)

```
def mycombine(f,data):
    L = data[:]                          ## make a copy of the input
    while len(L) > 1:                    ## until we're done,i.e. list==[elt]
        index = random.randint(0,len(L)-2) ## pick a random index (not the end)
        L[index]=f(L[index],L[index+1])  ## replace element with call to f
        L.pop(index+1)                   ## delete the one to the right
    return L[0]                          ## return the list's only element
```

```
def joinswap(x,y):
    return int(str(y)+str(x))
```

a) What will **`mycombine(joinswap,[1,2,3])`** return if **`randint`** always returns 0 (i.e., always picks the *leftmost* index)? (select ONE)

○	○	○	○	○	○	○	○
123	132	213	231	312	321	Error	None of these

b) What will be **`mycombine(joinswap,[1,2,3])`** return if **`randint`** always returns **`len(L)-2`** (i.e., always picks the *all-but-rightmost* index)? (select ONE)

○	○	○	○	○	○	○	○
123	132	213	231	312	321	Error	None of these

```
def minus(x,y):
    return x-y
```

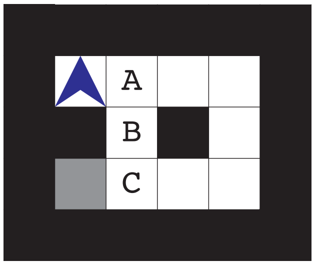
c) What are *all* the return values of **`mycombine(minus,[4,3,2,1])`**? (select all that apply)

□	□	□	□	□	□	□	□	□	□	□	□	□
-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6

FoxTrot
BILL AMEND
(with CS10 customizations)
Here's your CS10 final exam, class. Good luck.
Actually, you'll probably need super-duper luck.
What's all this stuff from Chapter 100?!
1. Create a fractal that models the shape of the North American continent.
2. Write a program to translate English to any language spoken by Mankind.
3. Write down all the moves in the solution to TicTacToe played on a 1,000 x 1,000 board.
4. On the screen above you is the Pixar movie "Up". Using Snap!, create the sequel.
People, I'm sorry – I've handed out the wrong test. If you could just pass those back to me...
PHEW!
Here's the exam you should have gotten...
That other one was for the remedial class.
The only thing worse than dreaming about your CS10 final is waking with the realization that you're only half-done!
λ

Question 10a-c: どうもありがとうミスターロボット ***Dōmo arigatō, Mr. Roboto…*** (9=3+3+3 pts)

When designing a maze, you have lots of options. You can make it *easy* or *hard* or *impossible* to get to the goal square (here, shown in grey). If we look at the maze on the right, there are three squares that haven't been set yet: **A**, **B** and **C**. They could be set as a wall ■ (which we'll call false) or empty slot □ (which we'll call true).



a) How many *total mazes* can we create? (select ONE)

○	○	○	○	○	○	○	○	○
1	2	3	4	5	6	7	8	9

b) How many mazes are *solvable*? (i.e., robot can get to the goal)

○	○	○	○	○	○	○	○	○
1	2	3	4	5	6	7	8	9

c) What expression is *true* for all the solvable mazes? (select ONE →)

```
○ (A and B) and C
○ (A or B) or C
○ (not A) or (not C)
○ (not A) and (not C)
○ A or C
○ A and C
```

Question 11: *Will the largest SID person please stand up…* (16 pts)
Consider the problem of trying to find out which student has the largest student ID number (SID) in a class of N students, with N a power of two (2, 4, 8, 16, …). Note that all SIDs are unique. There are four algorithms proposed:

> ***Algorithm I:*** *The students line up, sitting down. The first two students stand and compare their SIDs. The student with the smaller SID sits back down, the other remains standing. The next student in line stands up and this process repeats until there is only one student (with the largest SID) left standing.*
>
> ***Algorithm II:*** *All students stand up, pair up, and simultaneously compare SIDs, and the smaller of each pair sits down. Those still standing repeat the process, pairing up with another standing person, until there is only one left standing; that student has the largest SID.*
>
> ***Algorithm III:*** *All students are seated in a circle, facing inward. They write their SID on a sticky note and put it on the back of their neck, number facing out. A random student stands up and walks around to each person and compares his/her number with the number on the neck. If his/hers is larger than all others, he/she declares him/herself as the largest. If they are not, they sit down and someone else (who has not stood up before) stands up, and the process repeats until one person declares he/she is largest.*
>
> ***Algorithm IV:*** *Same as Algorithm III but after a person goes around and isn't the largest, rather than the next person being someone who hasn't stood up before, a random person (of the N total students) is chosen again (and it could be someone who has gotten up before).*

Fill in the table for the *worst-case* running time and the *worst-case* number of SID comparisons. (Select ONE for each algorithm from the top group, and one for each from the bottom).

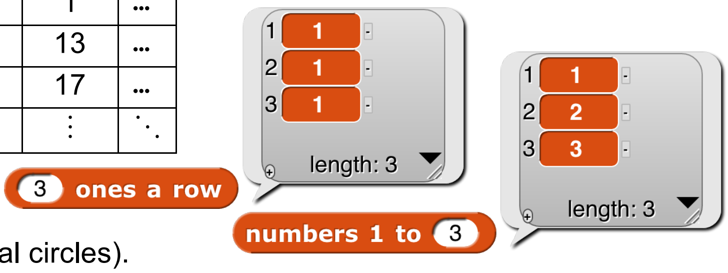
	Algorithm I	***Algorithm II***	***Algorithm III***	***Algorithm IV***
Constant running time	○	○	○	○
Logarithmic running time	○	○	○	○
Linear running time	○	○	○	○
Quadratic running time	○	○	○	○
Exponential running time	○	○	○	○
May never end, could go forever	○	○	○	○
Constant number of SID comparisons	○	○	○	○
Logarithmic number of SID comparisons	○	○	○	○
Linear number of SID comparisons	○	○	○	○
Quadratic number of SID comparisons	○	○	○	○
Exponential number of SID comparisons	○	○	○	○
Infinite number of SID comparisons	○	○	○	○

Question 12: ***You're telling a GenFib…*** (21 pts)

Recall the Fibonacci series from lecture, which starts with **TWO** 1s and subsequently every number is the sum of the previous **TWO** numbers: 1, 1, 2, 3, 5, 8, 13, etc. See how we've highlighted the word "**TWO**"? That **TWO** was a *hard-coded* value, we know as budding computer scientists to strive for generalization. The *Gen*eralized *Fib*onacci (**`Gen Fib`**) sequence starts with quantity **`ONES`** of 1 and subsequently every number is the sum of the previous **`ONES`** numbers. The Generalized Fibonacci sequence with **`ONES`** = 2 is the original sequence above.

Gen Fib	**N** = 1	**N** = 2	**N** = 3	**N** = 4	**N** = 5	**N** = 6	**N** = 7	…
ONES = 1	1	1	1	1	1	1	1	…
ONES = 2	1	1	2	3	5	8	13	…
ONES = 3	1	1	1	3	5	9	17	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

These two blocks may be helpful…

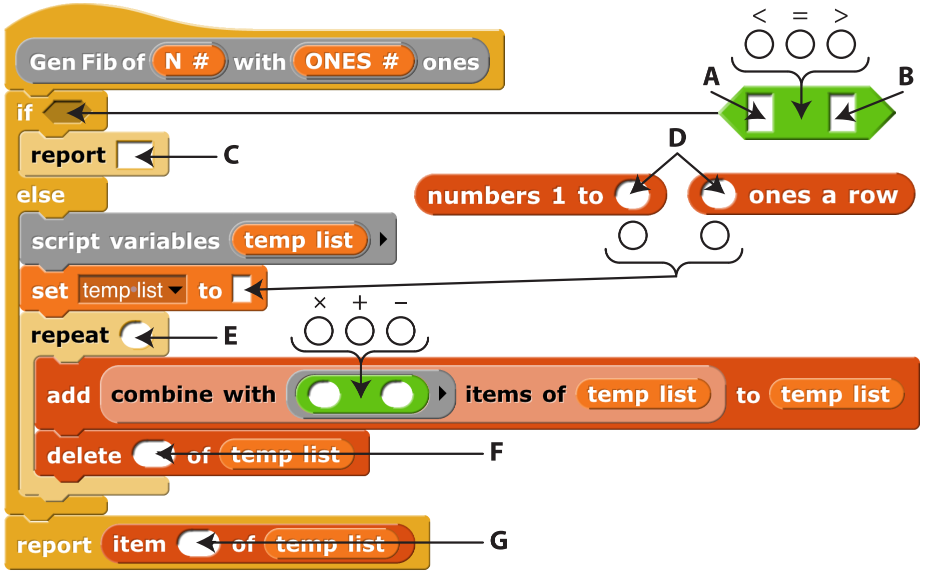


```
3 ones a row
  1  1
  2  1
  3  1
  length: 3

numbers 1 to 3
  1  1
  2  2
  3  3
  length: 3
```

Write **`Gen Fib`** *iteratively*.

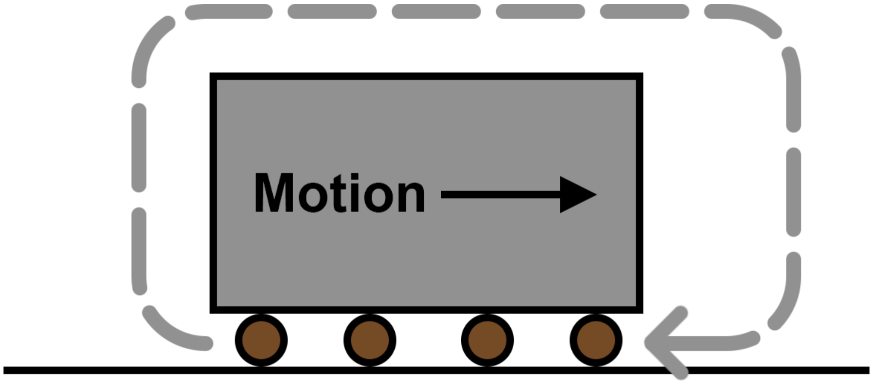
Fill in the 3 choices below and **A-G** in the table (10 total circles).

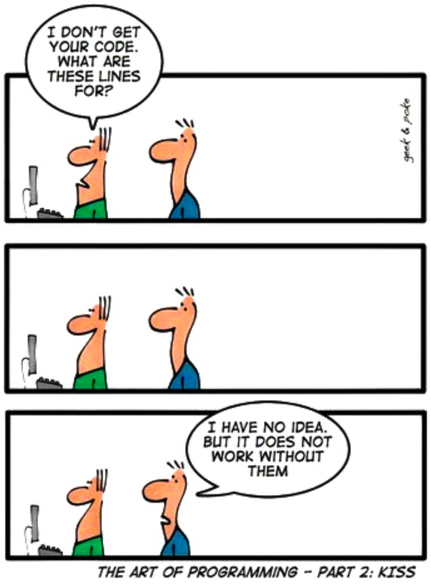


```
Gen Fib of (N #) with (ONES #) ones
if < [A] (< = >) [B] >
    report [C]
else
    script variables (temp list)
    set temp list to [D: (numbers 1 to ( ) / ( ) ones a row)]
    repeat [E]
        add (combine with (( ) (× + −) ( )) items of (temp list)) to (temp list)
        delete [F] of (temp list)
report item [G] of (temp list)
```

	A	B	C	D	E	F	G
N − 1	○	○	○	○	○	○	○
N	○	○	○	○	○	○	○
N + 1	○	○	○	○	○	○	○
0	○	○	○	○	○	○	○
1	○	○	○	○	○	○	○
last	○	○	○	○	○	○	○
ONES − 1	○	○	○	○	○	○	○
ONES	○	○	○	○	○	○	○
ONES + 1	○	○	○	○	○	○	○
ONES − N	○	○	○	○	○	○	○
N − ONES	○	○	○	○	○	○	○
ONES + N	○	○	○	○	○	○	○

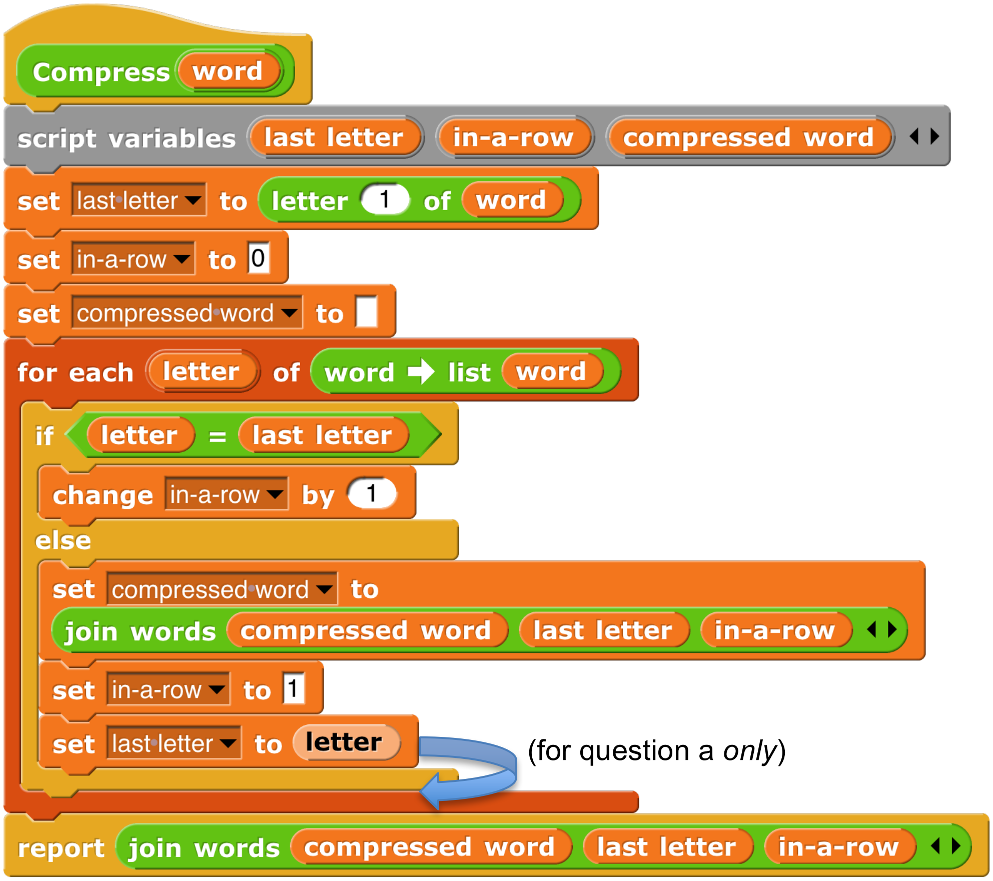
Fun Fact: When the Great Pyramids were being built in ancient Egypt, they used logs as rollers under the giant blocks of granite (instead of trying to slide the massive rocks along the ground)





I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
geek & poke
THE ART OF PROGRAMMING – PART 2: KISS

Question 13a-c: ***Oh you hurt your knee? What you need is a cold compress…*** (10 = 4+4+2pts)



```
Compress (word)
script variables (last letter) (in-a-row) (compressed word) ◂▸
set [last letter] to (letter (1) of (word))
set [in-a-row] to [0]
set [compressed word] to [ ]
for each (letter) of (word ➜ list (word))
    if <(letter) = (last letter)>
        change [in-a-row] by (1)
    else
        set [compressed word] to
            (join words (compressed word) (last letter) (in-a-row) ◂▸)
        set [in-a-row] to [1]
        set [last letter] to (letter)
report (join words (compressed word) (last letter) (in-a-row) ◂▸)
```

(for question a *only*)

The **compress** block works *perfectly* to compress a word with repeated letters into a sentence in which every set of consecutive letters is replaced by the letter and the number in-a-row after it. Here is a simple example:

Compress [a] → a 1

…and here's how we'd write it:

a	■	
b		
0		
1		■
2		
3		

Here's a musical example:

Compress [abba] → a 1 b 2 a 1

a	■				■	
b			■			
0						
1		■				■
2				■		
3						

Now, we're going to modify the code (***always*** *starting from the original working version*) and see how things change. Fill in the squares completely, just like in the examples, to show the return value of each expression.

a) Move `set [last letter] to (letter)` outside the **if** (as the last block of the **foreach**)

Compress [a]

a							
b							
0							
1							
2							
3							

Compress [abba]

a							
b							
0							
1							
2							
3							

b) Change `<(letter) = (last letter)>` to `<not <(letter) = (last letter)>>`

Compress [a]

a							
b							
0							
1							
2							
3							

Compress [abba]

a							
b							
0							
1							
2							
3							

c) Your friend suggests to compress the return even further, and instead of using **join words** to build up the compressed word, use **join** (so there will be no spaces in the output). Thoughts? (Choose ONE)

○ Great! That'll compress the return value even more and not affect the ability to uncompress it.
○ There's a problem, since you can't uniquely uncompress all numbers (a special kind of "word" with digits).
○ There's a problem, since the working compress block above would then run in an infinite loop.