UC Berkeley's CS10 Spring 2017 Midterm 2 : Instructor Dan Garcia

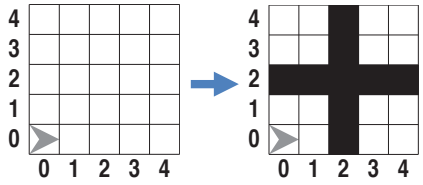
Your Name (first last) | *SID* | *Lab TA's Name*

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

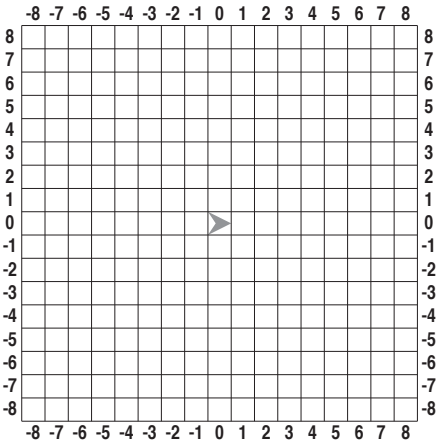
Fill in the correct circles & squares completely…like this: ● (select ONE) ■ (select ALL that apply)

Question 1: ***Magical mystery Tour, step right this way!*** (9 pts = 6+3)

Consider a block that draws a "plus sign" and keeps the sprite in the same position at the end. For example, if the sprite were at (0,0) and facing to the right (as shown), after a call to `Draw + 4` it would draw… (as shown).



a) Shade in (completely!) *all* the pixels that are filled in after `Mystery 8`.



b) If **`Draw+`** is constant time, what's the running time of **`Mystery`**? (select ONE)

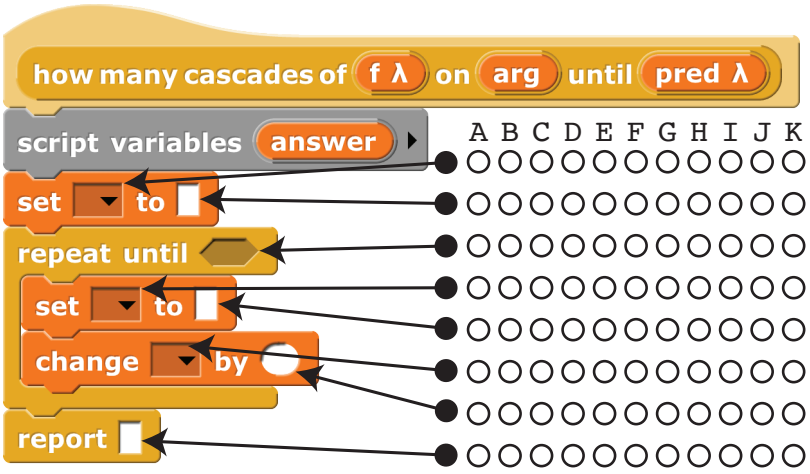
- ○ Constant
- ○ Logarithmic
- ○ Linear
- ○ Quadratic
- ○ Exponential

Question 2: ***I'm sorry, I didn't get that. Can you repeat it?…*** (6 pts)

There are times you want to compute **`f(arg)`**, or you want **`f(f(arg))`**, or even **`f(f(f(f(arg))))`**. Snap*!* provides the **`cascade`** block that takes a number **`n`**, a function **`f`**, an argument **`arg`**, and applies **`f`** to the **`arg`** over and over, **`n`** times. E.g.,

We'd like to use it as inspiration for a block that takes a function **`f`**, an argument **`arg`**, and a predicate **`pred`**, and returns the number of times **`f`** must be called on **`arg`** until it satisfies **`pred`**. E.g.,

a) Fill in ONE circle from each row to complete the block.



A	0
B	1
C	2
D	f
E	arg
F	pred
G	answer
H	call f with inputs arg
I	call f with inputs answer
J	call pred with inputs arg
K	call pred with inputs answer

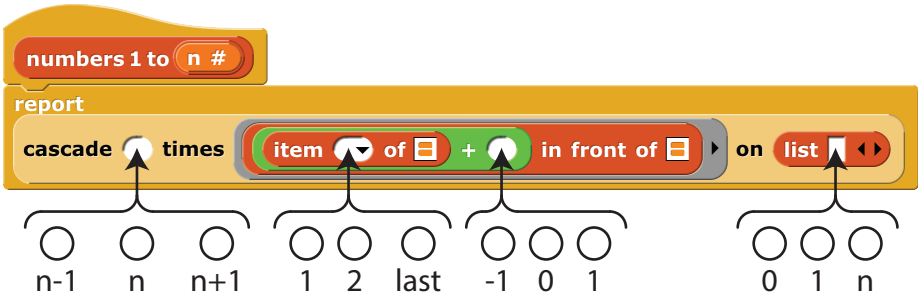
Question 2 (continued): *I'm sorry, I didn't get that. Can you repeat it?...* (11 pts = 5+3+3 pts) SID: _____

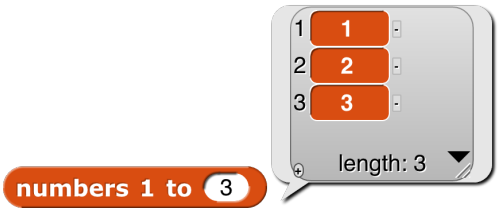
*A number is divisible by 9 if it **is** 9, or if the sum of its digits is divisible by 9.*

A recursive definition, eh? Let's see, is the number 14832 divisible by 9? Well, is it 9? No, so let's check if the sum of the digits is divisible by 9. Let's see, 1+4+8+3+2 = 18. Ok, is 18 divisible by 9? Well, is it 9? No, so let's check if the sum of the digits is divisible by 9. Let's see, 1+8 = 9. Ok, is 9 divisible by 9? Well, is it 9? Yep! Then 14832 was divisible by 9! Actually, we don't really care about 9-divisibility of a general number. We first want to know *how many recursive steps a multiple of 9 took until it got to 9*. 14832 →18→9 was 2 steps. Then we want to know, for multiples of 9 (9, 18, 27, ...), **how many steps each took through that algorithm until it was 9.**

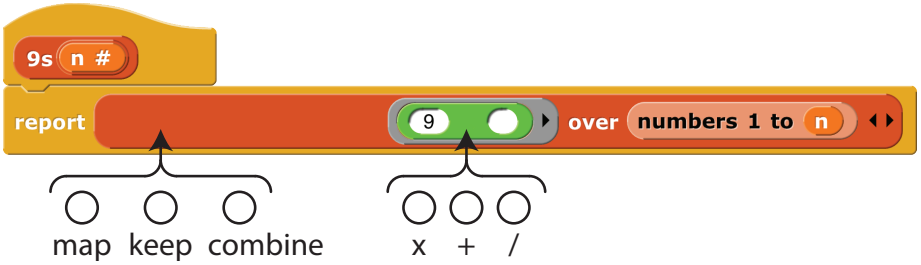
b) Using `cascade`, write 

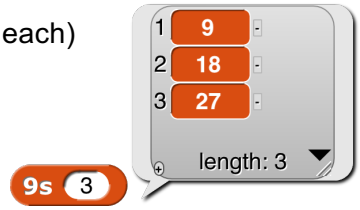
that reports a list from 1 to *n*. (Select ONE from each)



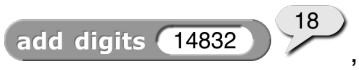


c) Write `9s`, that reports the first *n* multiples of 9 as a list. (Select ONE from each)





Now, given 

that adds the digits of a number: 

, the reporter 

defined below and the blocks above, we are able to generate the table below.

Number	9	18	27	36	45	54	63	72	81	90	99	108	117	…	14832	…
Steps until 9 for that number	0	1	1	1	1	1	1	1	1	1	2	1	1	…	2	…

d) Based on that result, what is the **smallest** number that has 3 steps until 9? _____

Example compound expression for Question 3:

Question 3: *Beethoven was a tremendous composer...* (4 pts)

The two blocks below operate on numbers; examples are shown:

Block	Description	Examples
	Reverses the numbers. (leading zeros go away)	
	Adds 2 to the number.	

Imagine a single, compound expression made up of both of these blocks intermingled and composed together (none, some, or possibly very many of each), like the example above: "`reverse(reverse(plus2(reverse())))`". If the *output* of this (possibly large) expression were "`7`", which could have been the *input*? (select ALL that apply)

☐ 68	☐ 6	☐ 9	☐ 1111