UC Berkeley's CS10 Fall 2016 Final Exam : 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)* →

What's that Smell? Oh, it's Potpourri! (2 pts each for 1-11, low score dropped)

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

Question 1: Who gave the "Mother of all Demos", introducing the mouse to the world? (select ONE)

○ Alan Turing	○ Bill Gates	○ Steve Jobs	○ Doug Englebart	○ Ivan Sutherland	○ Maus Klein	○ Jim Maus	○ Eric Paulos

Question 2: The internet uses an *end-to-end architecture*. That means… (select ONE)
○ They lay fiber-optic cable starting from one end to the other, rather than starting in the middle and going out.
○ The network gateways, switches and routers have all the "intelligence", e.g., encrypting and decrypting files.
○ The network requires computers and devices on its *ends* have unique IP addresses, like 128.32.169.12.
○ The network connects devices all across the world, from one "end" of the earth to the other.
○ None of the above.

Question 3: A polynomial-time solution to the knapsack problem means that for the **first time**... (select ONE)
○ We can now *verify* a randomly-generated knapsack problem solution in polynomial time.
○ We can now *verify* a randomly-generated subset sum problem solution in polynomial time.
○ We can now solve the knapsack problem *approximately*.
○ We now have a *heuristic* to solve the knapsack problem.
○ We can now solve the *subset sum* problem in polynomial time.
○ We can now solve *every exponential-time problem* in polynomial time.
○ None of the above.

Question 4: Which of the following is **true**? (select ONE)
○ Most software is written by *individuals working alone*.
○ Parallel programming is a *solved problem*.
○ *Fortran* is the most common programming language in scientific code.
○ *Performance* is the top goal in software.
○ Most software is *rewritten from scratch* every few years.
○ *None* of the above.

Question 5: Which of the following are part of the traditional "interface design cycle"? (select ALL that apply)

□ Abstract	□ Ideate	□ Refine	□ Design	□ Prototype	□ Iterate	□ Evaluate	□ Analyze	□ Debug	□ Visualize	□ Publicize

Question 6: Why did we move from IPv4 to IPv6? Because IPv4… (select ONE)

○ didn't support newer *fiber-optic cables*	○ didn't have enough *bandwidth*	○ didn't have enough *througput*	○ couldn't handle the noise	○ didn't support newer protocols.	○ couldn't travel the distance of IPv6	○ didn't have enough addresses.	○ None of these

Question 7: What did the halting problem prove? (select ONE)
○ Not all problems were *decidable*.
○ You *can* write a program to decide if another program would halt (not run forever) on its input.
○ The *traveling salesman problem* was NP-complete.
○ The *subset sum problem* was NP-complete.
○ Determining whether a program would halt on its input can be done in less than exponential time.
○ P = NP.
○ P != NP.
○ None of the above.

Question 8: What is the "Attack of the Killer Cellphones"? (select ONE) SID: _____

- ○ Cellphones have cancer-causing radiation.
- ○ Cellphones are one of the leading causes of traffic fatalities due to distracted driving.
- ○ Cellphones have mobile apps that are "killing" the brains of young people with social apps.
- ○ Parallel system architects are looking to cell phone processors to understand how to manage power better.
- ○ Computational scientists are building software so people can use their cell phones as a "volunteer cluster".
- ○ Parallel system architects are worried scientists may start using many cell phones for their computations.
- ○ None of the above.

Question 9: You plan to test an algorithm with a set of very extensive test cases. Which is true? (select ONE)

- ○ Only if it fails all test cases is the program incorrect.
- ○ Only if it fails more than half is the program incorrect.
- ○ If it passes more than half, the program is considered correct.
- ○ If it passes all of them, the program is considered correct
- ○ None of the above.

Question 10: YouTube now uses 64 instead of 32 bits to count views. How many more is that? (select ONE)

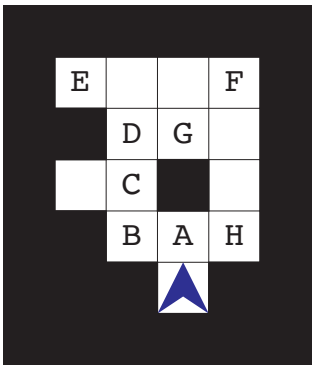
○ 2x	○ 32x	○ 64x	○ 2^2x	○ 2^{32}x	○ 2^{64}x	○ None of these

Question 11: A coin has two sides, labeled "1" and "2". Consider the goal of simulating the results of flipping the coin *five* times, and displaying the *sum* from the five flips. Which of the following code segments will produce the appropriate results? *Hint: Compare the # of ways there are of summing to 5 vs. 7…* (select ONE)

○	○	○	○
pick random 5 to 10	2 × pick random 1 to 5	5 × pick random 1 to 2	None of the Above

Question 12a-d: どうもありがとうミスターロボット ***Dōmo arigatō, Mr. Roboto 2…*** (12=3+3+3+3 pts)

We tried to rewrite our midterm maze script to visit all the letters A-H in the maze. Here are our four attempts, let us know the letters they each visit.



move forward ( )	rotate left rotate right turn around	can move left? can move forward? can move right? dead end
The robot moves *INPUT squares* forward in the direction it's facing.	The robot turns, in-place. {left = counterclockwise, right = clockwise, around = u-turn}	Reports `true` if the robot has a free square to its {left, front, right}; otherwise reports false. The last one reports `true` if *can't* move left, forward *and* right.

□□□□□□□□ A B C D E F G H

```
forever
if can move left?
  rotate left
else
  if can move forward?
    move forward 1
  else
    if can move right?
      rotate right
    else
      turn around
```

□□□□□□□□ A B C D E F G H

```
forever
if can move left?
  rotate left
  move forward 1
else
  if can move forward?
    move forward 1
  else
    if can move right?
      rotate right
      move forward 1
    else
      turn around
```

□□□□□□□□ A B C D E F G H

```
forever
if can move left?
  rotate left
  move forward 1
else
  if can move forward?
    move forward 1
  else
    turn around
```

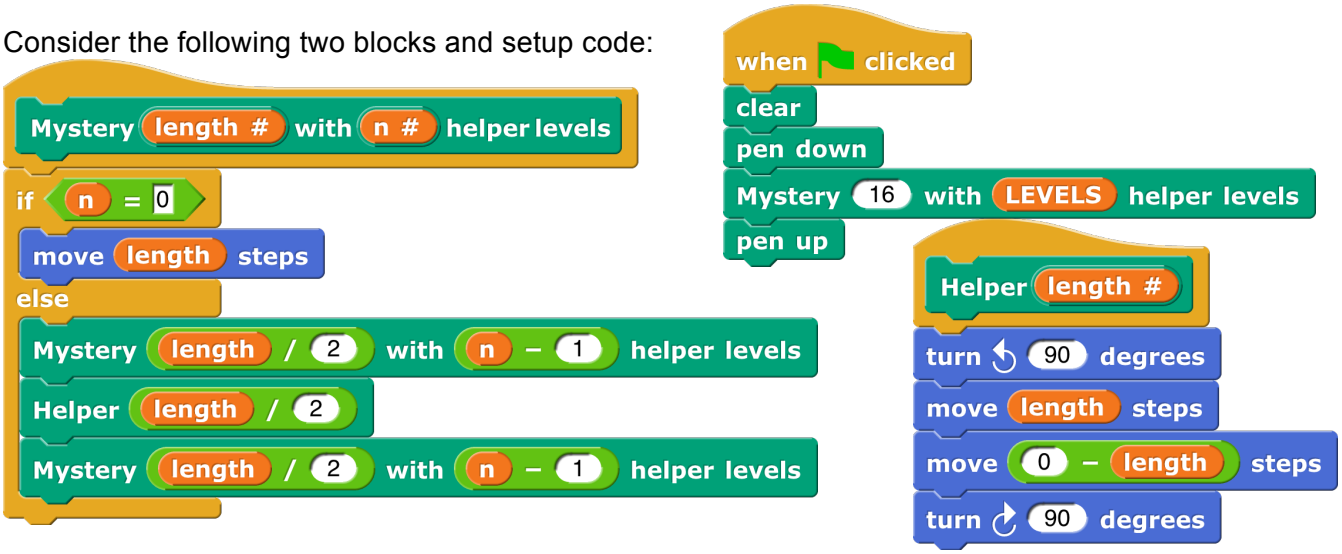
□□□□□□□□ A B C D E F G H

```
forever
if can move left?
  rotate left
  move forward 1
if can move forward?
  move forward 1
if dead end
  turn around
```

```
dead end
report
  not CAN_MOVE left  and
  not CAN_MOVE forward  and  not CAN_MOVE right
```

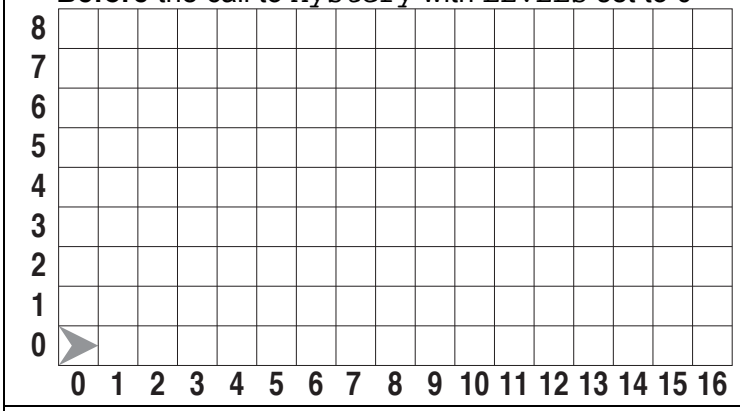
Question 13a-c: Magical `Mystery` Tour, step right this way… (13 = 5+5+3 pts) SID: _____

Consider the following two blocks and setup code:

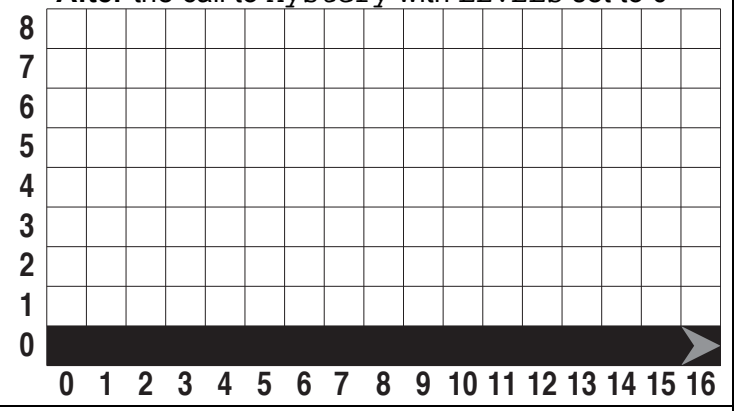


We're now to going to zoom in on pixels affected by calls to `Mystery`; the sprite always starts facing right in the lower left, and the pen is in the *center* of the sprite. The top two images are the pixels before and after a call to `Mystery` with `LEVELS` set to 0. Your job is to shade in (completely!) *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 **2** case, use the "Before the call to `Mystery`" pixels below)*

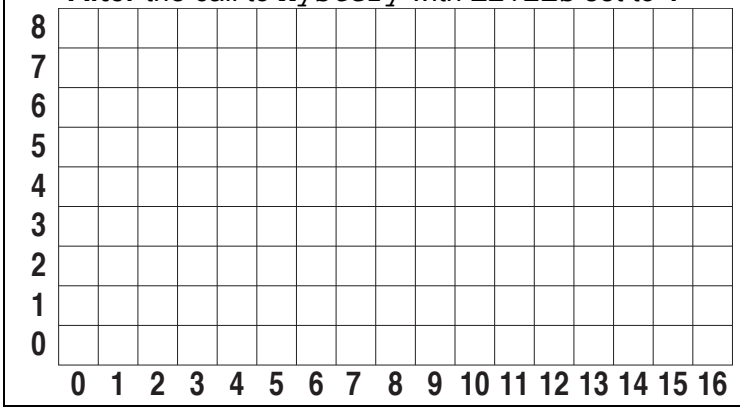
Before the call to `Mystery` with `LEVELS` set to 0



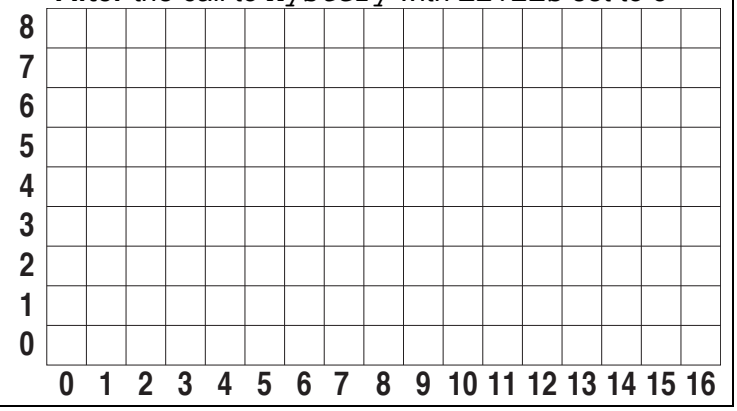
After the call to `Mystery` with `LEVELS` set to 0



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



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



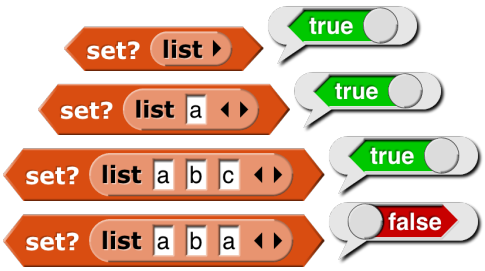
We're told that it actually costs a *dollar* to fill in all the pixels drawn by `Helper`. Which expression best captures the cost (in dollars) for this call? (select ONE)

○ L	○ ½*L	○ N	○ ½*N	○ L*N	○ ½*L*N	○ L^N	○ ½*L^N	○ N^L	○ ½*N^L	○ None of these

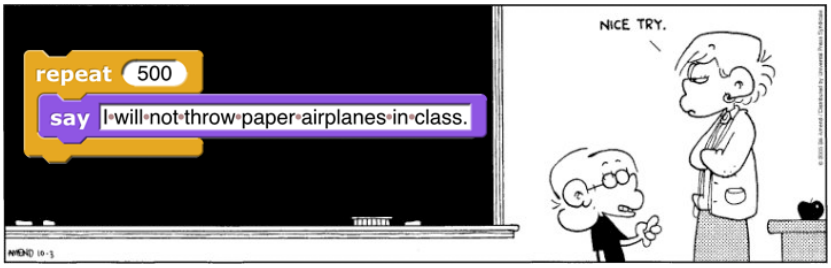
Question 14a-c: On your mark, get `set?`, go! (12 = 4+4+4 pts) SID: _____

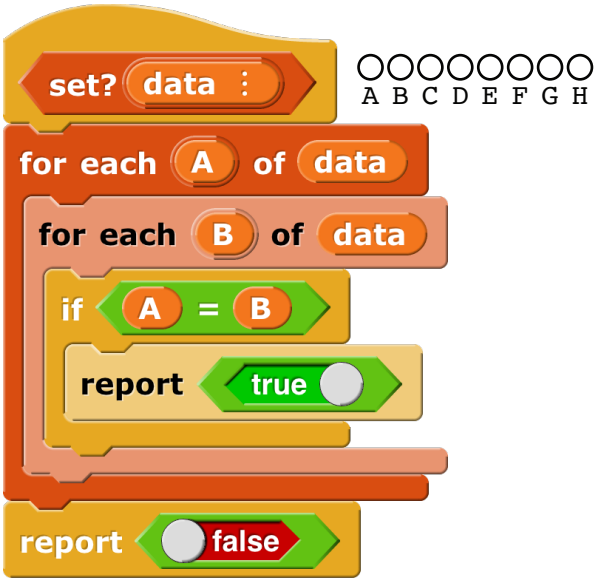
Consider the problem of wanting to determine if a list is a *set* (i.e., *every* element is unique, there are no duplicates). What's wrong (if anything) with each of the following 3 attempts? (select ONE from each legend)

Example calls to `set?`

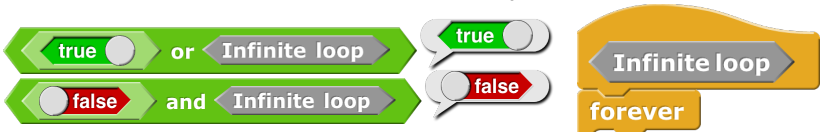


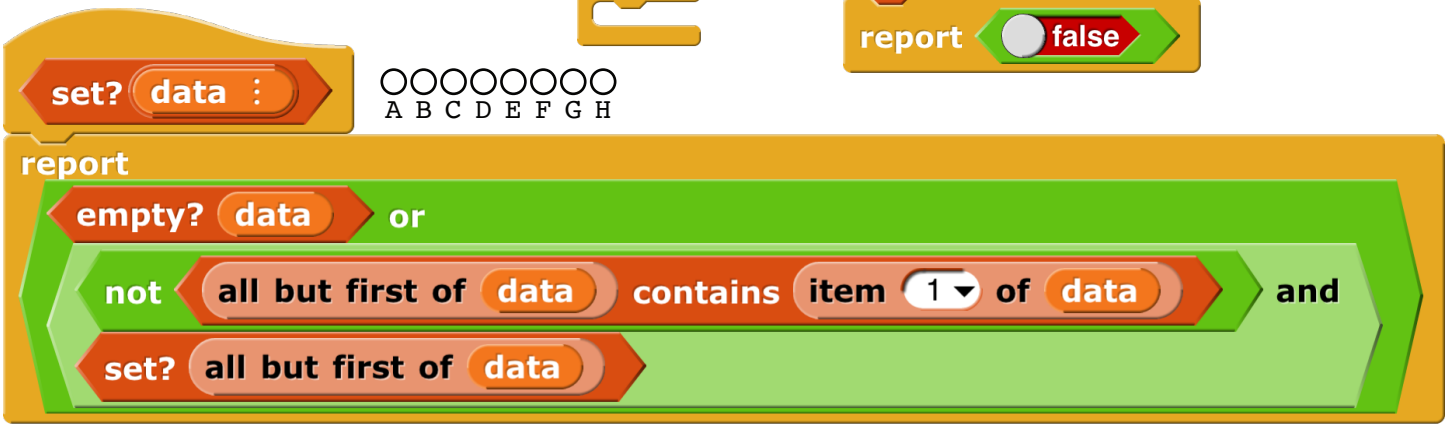
A = it works fine.
B = It will cause an error or run forever.
C = It always returns *true*.
D = It always returns *false*.
E = If it's the empty list, *true*, otherwise it always returns *false*
F = If it's the empty list, *false*, otherwise it always returns *true*
G = If it's the empty list, *true*, otherwise it only returns whether the *first* element is in the list multiple times
H = If it's the empty list, *true*, otherwise it only returns whether the *last* element is in the list multiple times





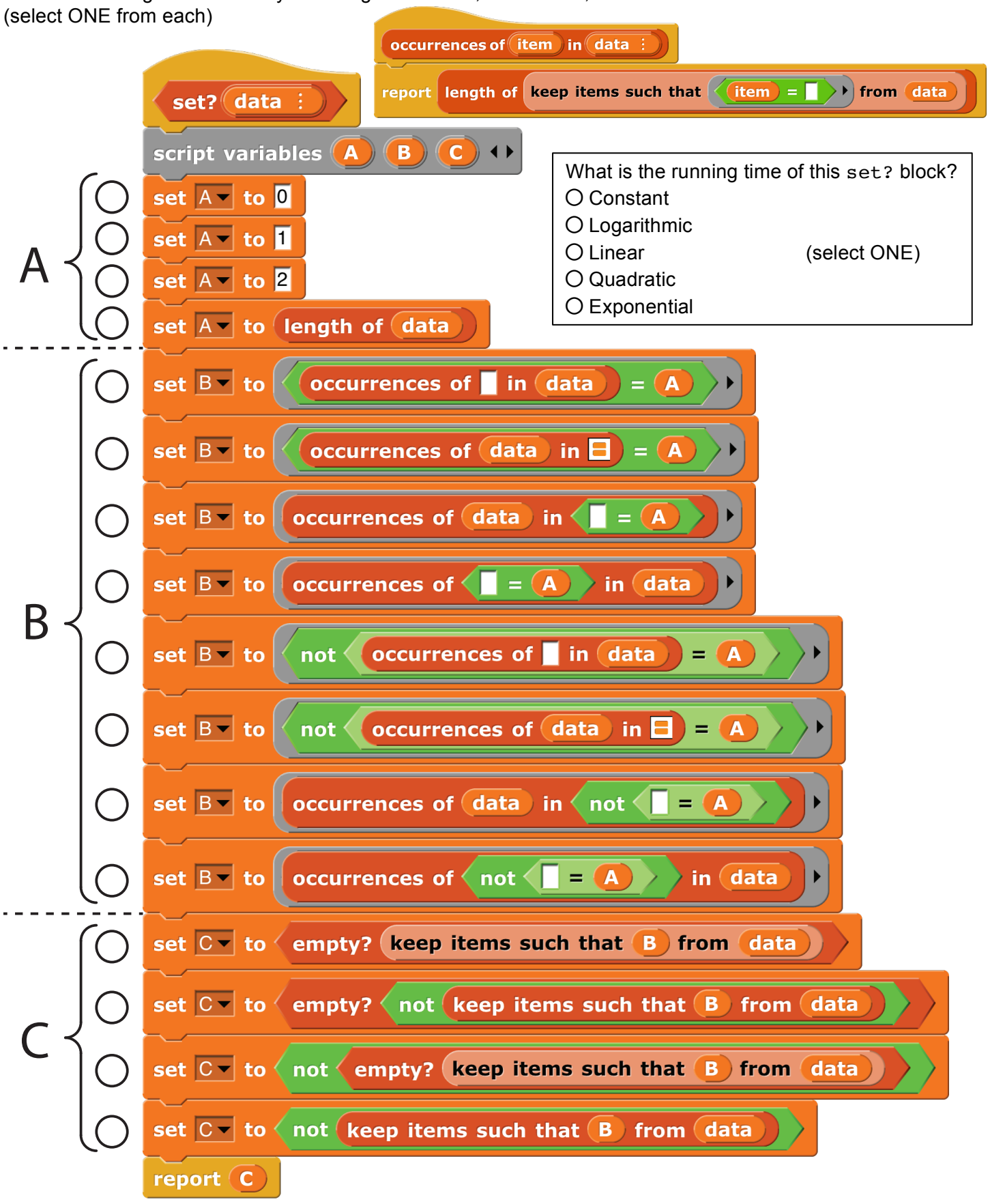
Note that **`or`** and **`and`** don't even *look* at their right input if the left input is **`true`** or **`false`**, respectively. Witness:





Question 14d-e: On your mark, get `set?`, go! (15 = 12+3 pts) SID: _____

Author a working `set?` block by choosing one from A, one from B, one from C.
(select ONE from each)



What is the running time of this `set?` block?

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

(select ONE)

Question 15a-f: Berkeley `Python` Flying Circus… (13 = 2+2+2+2+2+3 pts) SID: _____

Interpreter fun! For each, choose ONE that best matches what would display on the next line.

Note that some of the code might have bugs! We include "`Error`" and "`None of these`" in case the output is not what is expected.

```
>>> S = "Berkeley"
>>> S[1:3]
```

○ Be	○ Ber	○ erk	○ er	○ Error	○ None of these

```
>>> [N ** 2 for N in range(4) if N != 2]
```

○ [0,1,3]	○ [0,2,6]	○ [1,3,4]	○ [1,6,8]	○ [0,1,9]	○ [1,9,16]	○ Error	○ None of these

```
>>> "".join([word[0] for word in "Univ of Calif at Davis" if not(len(word) == 2)])
```

○ "UCD"	○ UCD	○ "oa"	○ oa	○ UoCaD	○ "UoCaD"	○ Error	○ None of these

```
>>> f1 = lambda x: x+x
>>> f2 = lambda y: y > "9"
>>> list(map(lambda f: f("10"), [f1, f2]))
```

○ [20, True]	○ [1010,True]	○ ["1010",True]	○ [20,False]	○ [1010,False]	○ ["1010",False]	○ Error	○ None of these

(there was some typing here, probably defining the **school** *variable)*

```
>>> school
"cal"
>>> if school = "berkeley":
...     print("go " + school)
... else:
...     print("not here")
```

○ go berkeley	○ go cal	○ not here	○ "go berkeley"	○ "go cal"	○ "not here"	○ Error	○ None of these

We're trying to make a histogram function that returns the count of every item in `data`, for example `histogram([7,8,8,8,9])` ➔ `{7:1, 8:3, 9:1}`. Check the box for every syntax or logical error you find.

```
☐ define histogram(data):
☐     D = {} ;;; empty histogram
☐     foreach item in data:
☐         if item in D:
☐             D[item] = 0
☐         else
☐             D[item] = D[item] + 1
☐     report D
```