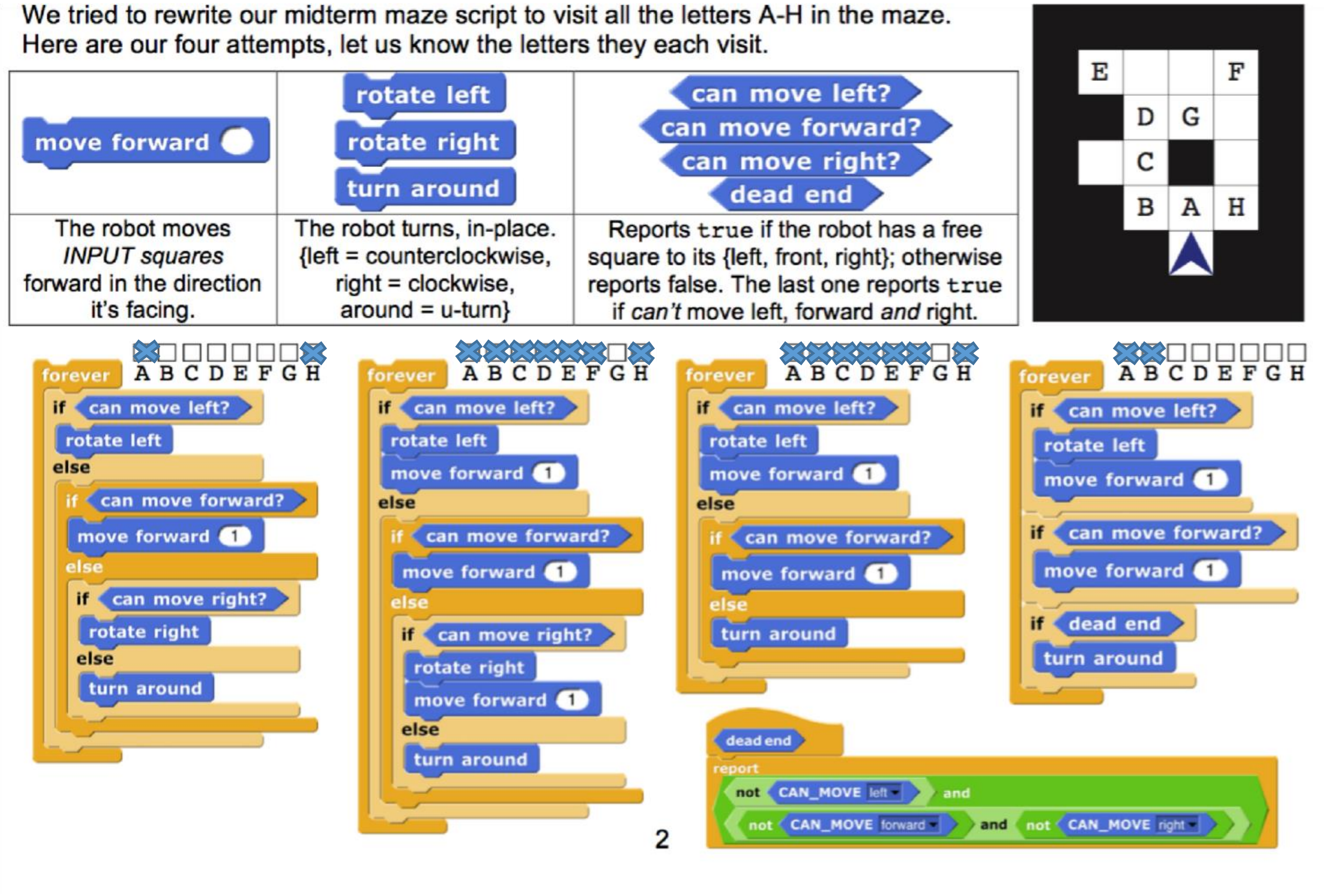
Discussion 14: Final Review SOLUTIONS

Drawing/Movement in Snap

Question 1: Mr. Robot



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

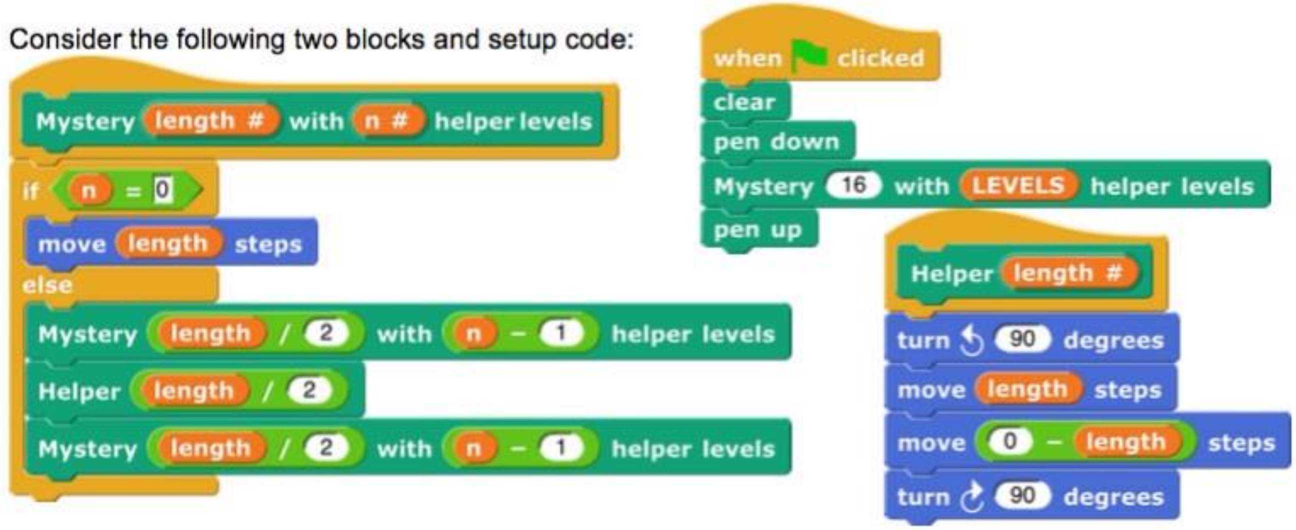
```
forever
  if <can move left?>
    rotate left
    move forward (1)
  else
    if <can move forward?>
      move forward (1)
    else
      turn around
```

☒☒☐☐☐☐☐☐
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 2: Magical Mystery Tour



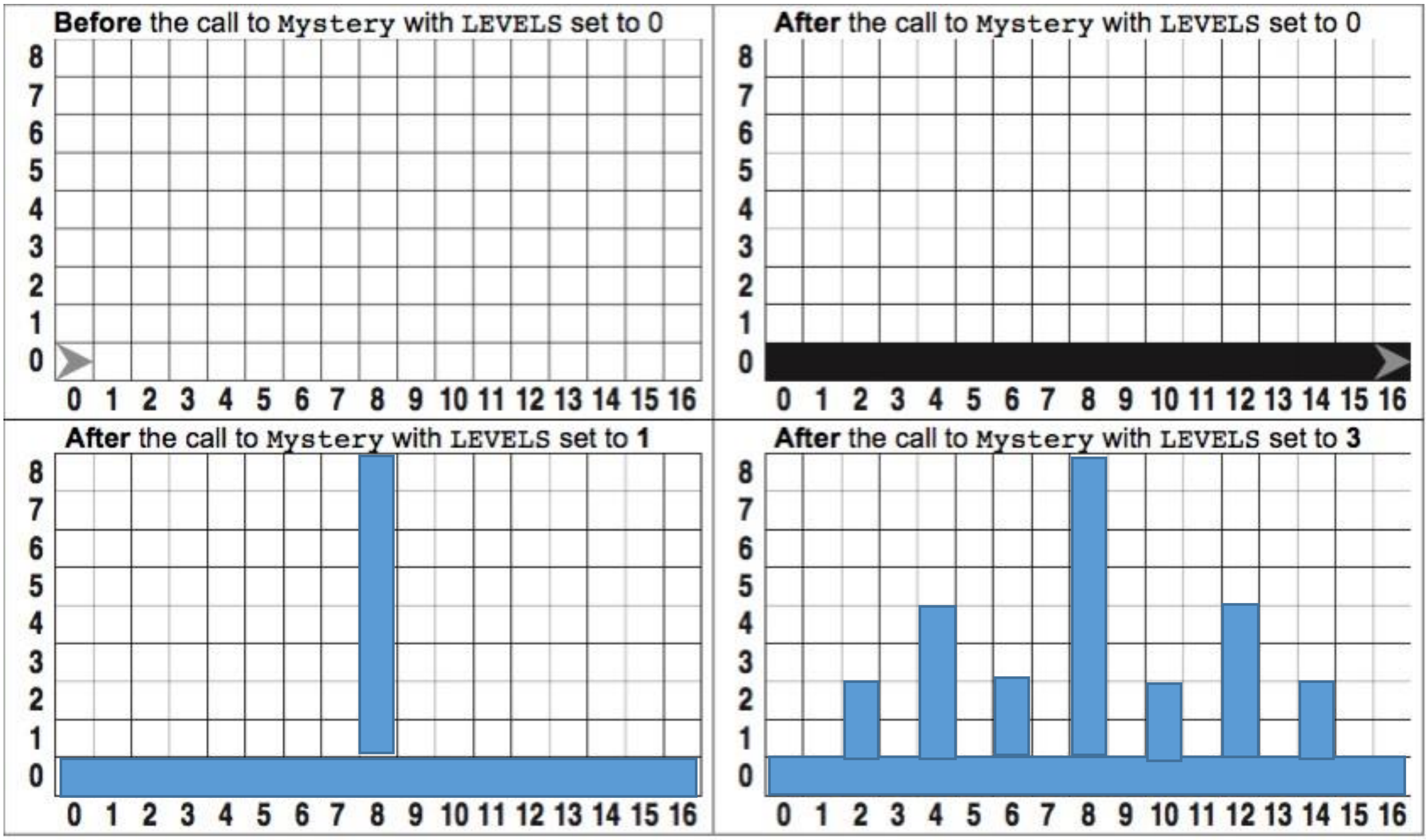
Consider the following two blocks and setup code:

```
Mystery (length #) with (n #) helper levels
if <(n) = [0]>
  move (length) steps
else
  Mystery ((length) / (2)) with ((n) - (1)) helper levels
  Helper ((length) / (2))
  Mystery ((length) / (2)) with ((n) - (1)) helper levels
```

```
when green flag clicked
clear
pen down
Mystery (16) with (LEVELS) helper levels
pen up
```

```
Helper (length #)
turn ↺ (90) degrees
move (length) steps
move ((0) - (length)) steps
turn ↻ (90) degrees
```

a. Now, given that the sprite starts out in the bottom left corner facing right, and that the pen is in the middle of the sprite, shade in the pixels that will be colored after calls to Mystery with levels set to 1 and levels set to 3. You may use the top left grid for scratch work. Levels = 0 has been given to you.



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

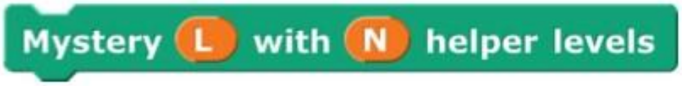
After the call to `Mystery` with `LEVELS` set to 0

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

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

b.

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)



Mystery (L) with (N) helper levels

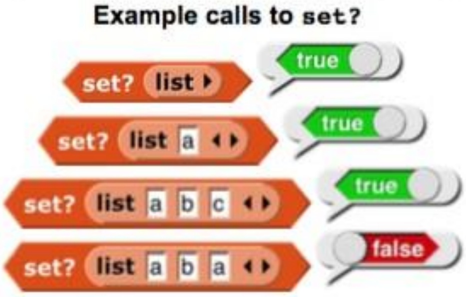
○	○	○	○	○	☒	○	○	○	○	○
L	½*L	N	½*N	L*N	½*L*N	L^N	½*L^N	N^L	½*N^L	None of these

Recursion

Question 1: Ready, Set, Go!

In this problem, we have created three different blocks to see if a given list is a set, that is, it has no duplicates. For each of the blocks below, select one of the following answer choices:

Example calls to `set?`



set? (list) → true

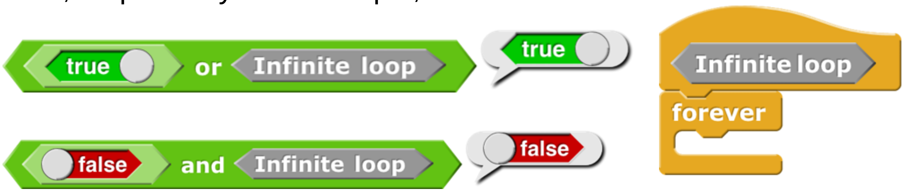
set? (list a) → true

set? (list a b c) → true

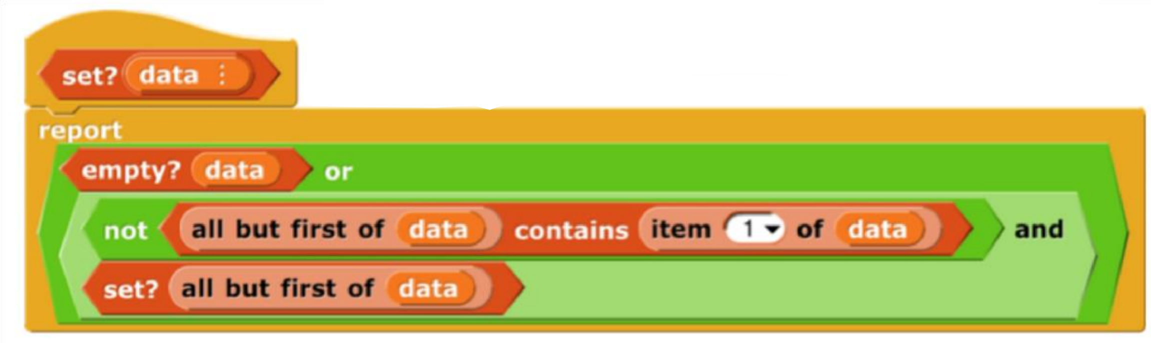
set? (list a b a) → false

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

a. For this subpart, note that the *or* and *and* blocks don't even look at their right input if the left one is true or false, respectively. For example,



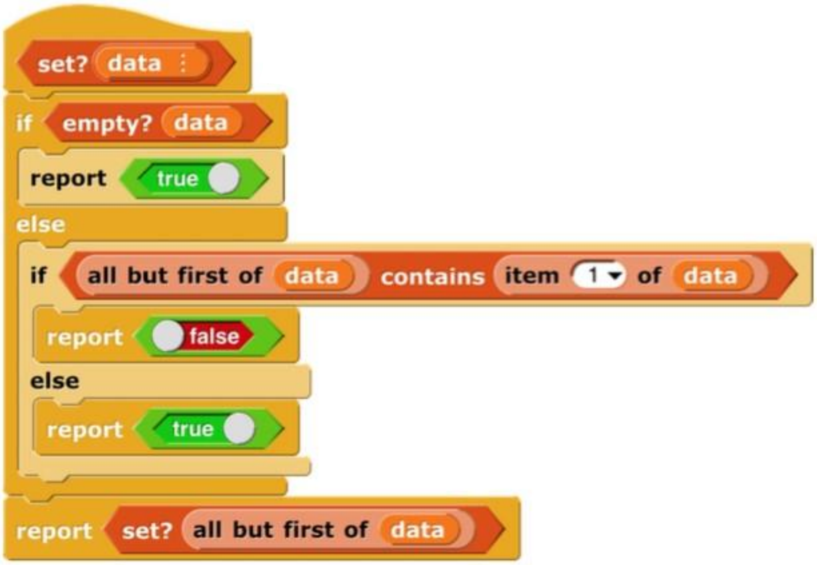
```
(true) or (Infinite loop)  ->  true
(false) and (Infinite loop)  ->  false

Infinite loop
forever
```



```
set? data :
report
  (empty? data) or
  ((not (all but first of data) contains (item 1 of data)) and
   (set? (all but first of data)))
```

[✓ A]
O B
O C
O D
O F
O G
O H

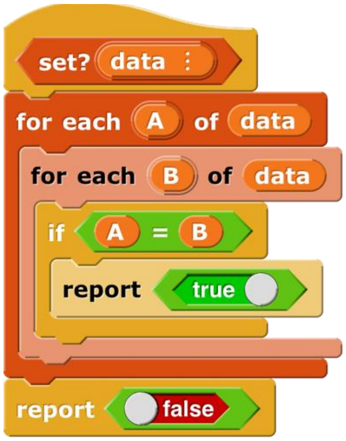
b.



```
set? data :
if empty? data
  report true
else
  if (all but first of data) contains (item 1 of data)
    report false
  else
    report true
report set? (all but first of data)
```

O A
O B
O C
O D
O F
[✓ G]
O H

c.

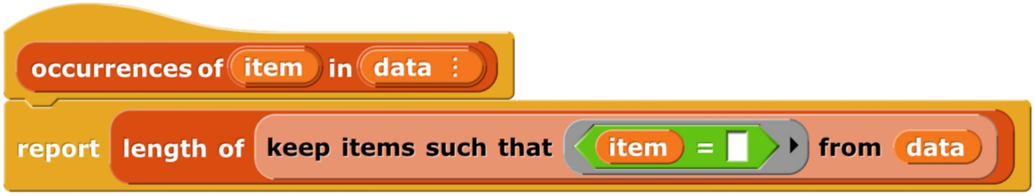


```
set? data :
for each A of data
  for each B of data
    if A = B
      report true
report false
```

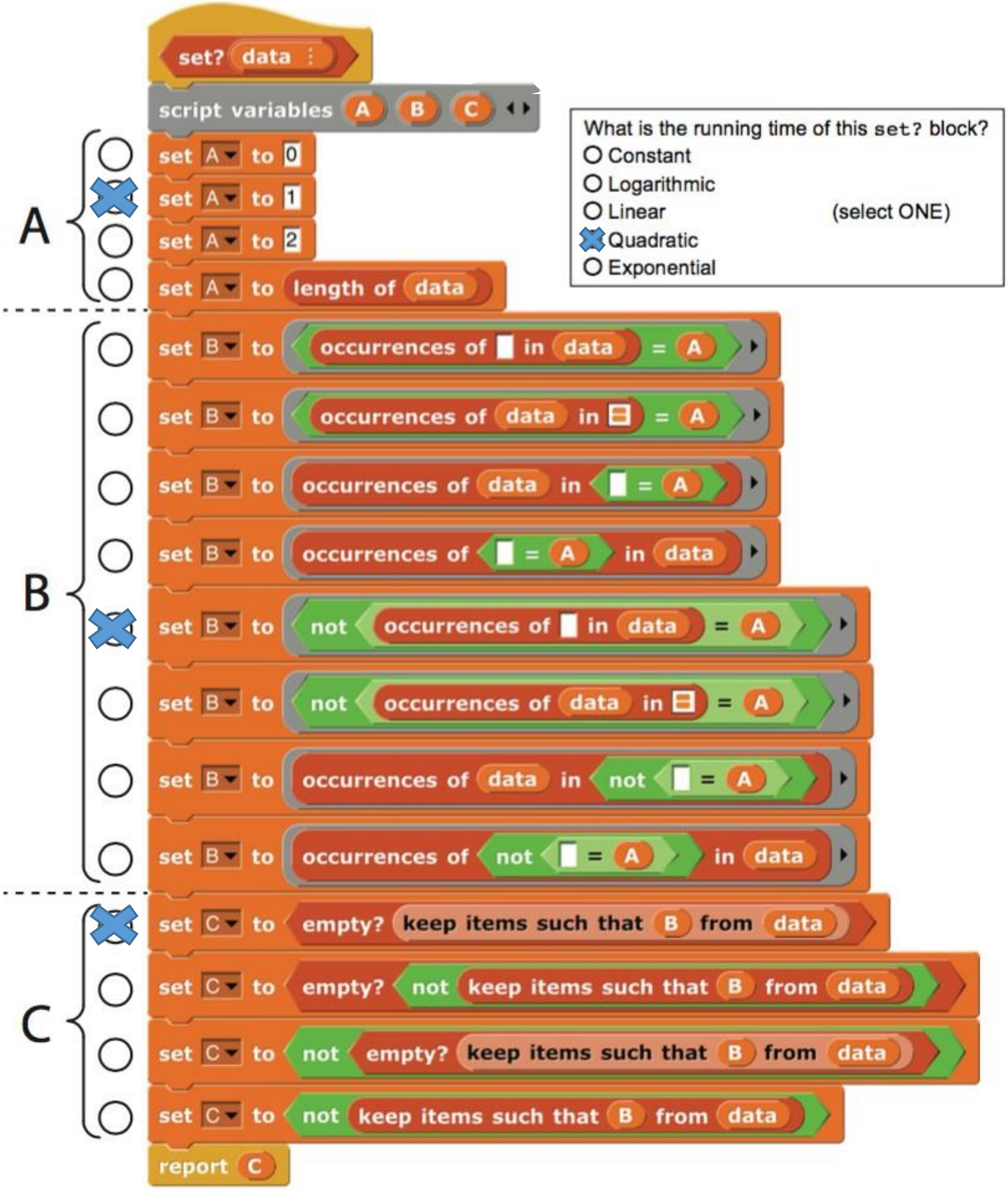
O A
O B
O C
O D
[✓ F]
O G
O H

Question 2: Constructing the *set* block

How could we construct the *set* block using the following *occurrences of* block? Note that you may only choose one option from each section A-C.



```
occurrences of (item) in (data ⋮)
report (length of (keep items such that ((item) = [ ]) from (data)))
```



```
set? (data ⋮)
script variables (A) (B) (C) ◂▸

A
  ○ set A to 0
  ☒ set A to 1
  ○ set A to 2
  ○ set A to (length of (data))
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
B
  ○ set B to ((occurrences of [ ] in (data)) = (A))
  ○ set B to ((occurrences of (data) in ⊟) = (A))
  ○ set B to (occurrences of (data) in ([ ] = (A)))
  ○ set B to (occurrences of ([ ] = (A)) in (data))
  ☒ set B to (not ((occurrences of [ ] in (data)) = (A)))
  ○ set B to (not ((occurrences of (data) in ⊟) = (A)))
  ○ set B to (occurrences of (data) in (not ([ ] = (A))))
  ○ set B to (occurrences of (not ([ ] = (A))) in (data))
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
  ☒ set C to (empty? (keep items such that (B) from (data)))
  ○ set C to (empty? (not (keep items such that (B) from (data))))
  ○ set C to (not (empty? (keep items such that (B) from (data))))
  ○ set C to (not (keep items such that (B) from (data)))
report (C)
```

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

○ Constant
○ Logarithmic
○ Linear (select ONE)
☒ Quadratic
○ Exponential

Python

Question 1: Syntax
Write the output of the following lines of code.

```
>>> [‘cal’, ‘berkeley’, ‘stanford’][1][2]
```

```
‘r’
```

```
>>> [x*10 for x in range(3) if x != 1]
```

```
[0, 20]
```

Question 2: Reversing a Dictionary
We want to write a dictionary reverser that takes in a dictionary and returns a new dictionary with the original values as the new keys and the original keys as a list of values.

```
>>> dictionary_reverser({1:3, 2:3, 8:9})
{3: [1, 2], 9: [8]}
```

Write this function by filling in the blanks in the skeleton code below.

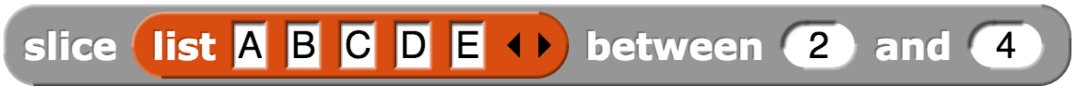
```
 def dictionary_reverser(dict):
      r = {}
      for k in dict:
           if dict[k] in r:
                r[dict[k]].append(k)
           else:
                r[dict[k]] = [k]

      return r
```

Online Final Questions

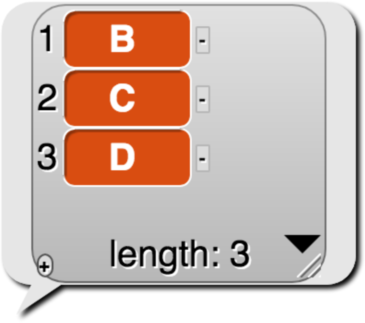
Note: You should complete all of the below questions either on a separate sheet of paper or on your computer. There is not sufficient space to write the solutions here.

Question 1: Slicing in Snap!

You want to replicate Python's list "slice" in Snap*!*. However, it should follow Snap*!*'s convention to index lists starting from 1 and include the rightmost element. You don't have to handle the case when the inputs are blank or do any error checking. That is, assume the left number ≤ the right number, and that both numbers are between 1 and the list length. If the numbers are equal, it returns a list of the element at that index.



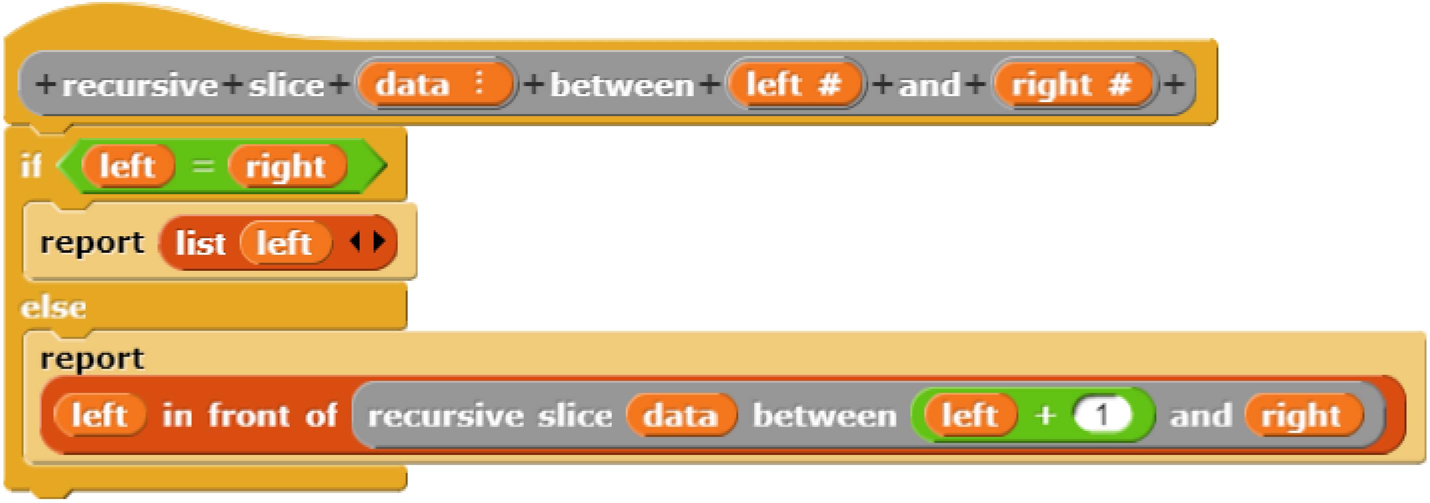
```
slice (list A B C D E) between (2) and (4)
```



1	B
2	C
3	D

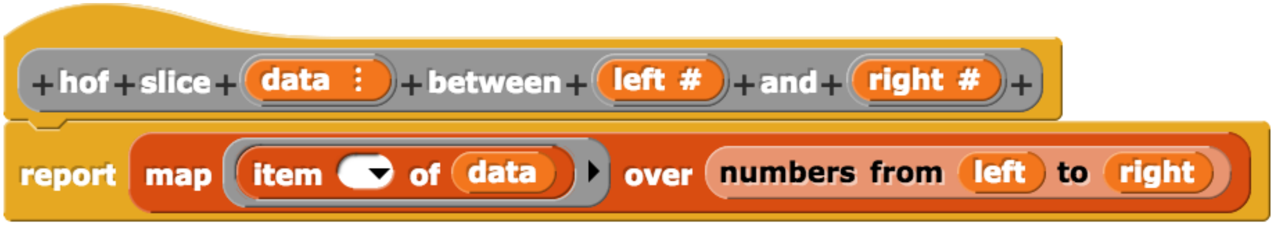
length: 3

a. Write it recursively. You may not use any iteration (`repeat, repeat until, for, for each`) or higher-order functions in this solution.



```
+recursive+slice+(data ⋮)+between+(left #)+and+(right #)+
if <(left) = (right)>
    report (list (left))
else
    report ((left) in front of (recursive slice (data) between ((left) + (1)) and (right)))
```

b. Write it using higher-order functions (<u>only map, keep and combine</u>). One helper you might find handy is the "`numbers between () and ()`" block.



```
+hof+slice+(data ⋮)+between+(left #)+and+(right #)+
report (map ((item ( ) of (data))) over (numbers from (left) to (right)))
```

Question 2: Strings and Dictionaries in Python

Write a function that returns the *first duplicate word* of an essay whose words are all in lowercase (with no punctuation). If there are no duplicates, return the empty string. You *must* use a dictionary in your solution; if you forget any commands, remember there's **help(*type*)** and **dir(*type*)**, as in **help(dict)** or **dir(str)**. To split a string into a list of words, you might find string's **split** command helpful.

```
>>>first_duplicate("ask not what your country can do for you ask what")
“ask”
>>>first_duplicate("cs ten is the best class at cal")
“”
```

```
def first_duplicate(essay):
      dict = {}
      for word in essay.split():
            if word in dict:
                  return word
            else:
                  dict[word] = 1
      return ‘’
```