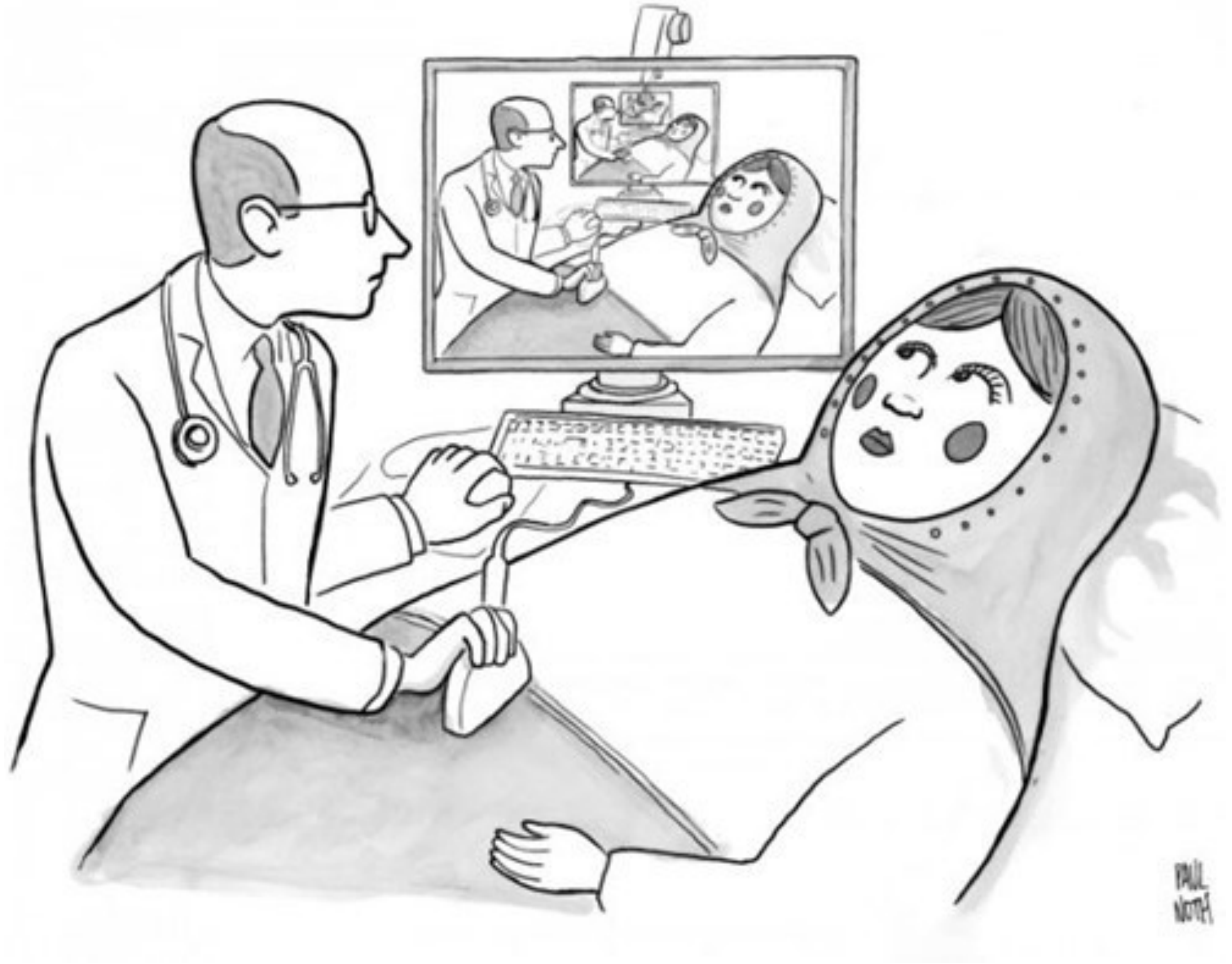
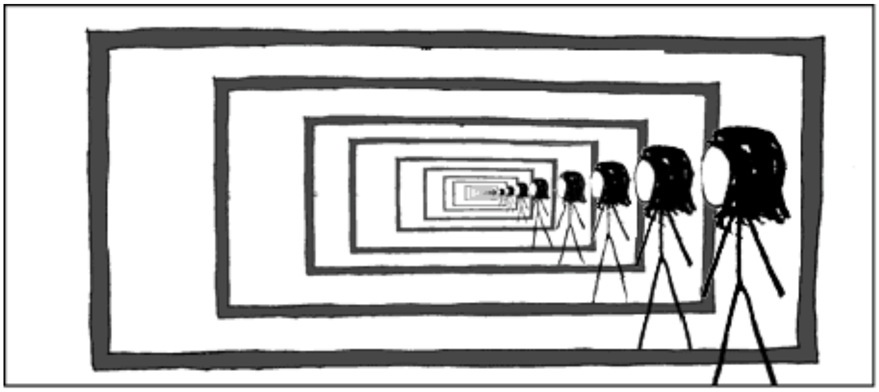
UC Berkeley's CS10 Fall 2019 Final Exam Answers: Prof. Dan Garcia

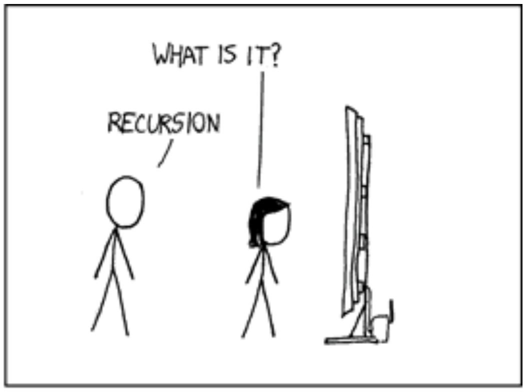
Your Name (first last) _______________ *SID* _______________ *Lab TA's Name* _______________

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

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







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

Question 1: Which of the following was in the *Human-Computer Interaction* lecture? (select ONE)

○ *Cars have crashed* because of poor interfaces – the driver thought a display was in one mode (but it wasn't)!

Nope, it was a plane that was mentioned having a dial that was read by the pilot as in the wrong mode

○ *The fridge/freezer cooling control interface was an example of good design*: two independent, intuitive knobs.

Nope, it was an example of terrible design, the diagram implies they were two separate dials but in fact they're not!

○ The iPhone was *one of the rare devices that didn't require a prototype, a testament to Steve Jobs' genius*!

Nope, they even showed a picture of the iPhone prototype (a combination of computer, phone, speakers, …)!

○ *User interface code is typically 75% of all the code* in "real" programs.

Nope, it is 50%

● None of these

Question 2: Which of the following was in the *Saving the World with Computing* lecture? (select ONE)

○ The *best data structure to model the earth is using lat/long lines* so we get skinny triangles near the poles.

Nope, just the opposite – skinny triangles don't do well with numerical simulations (equilateral ones do)

○ Faster computers *don't give us any more resolution of our hurricane models, they just render them faster*.

Nope, they actually DO allow for us to process higher-resolution simulations (as well as rendering them faster), and we showed a difference between a low-resolution and high-resolution in class

● Scientists are using Machine Learning detection (like finding a cat in an image) with hurricane simulations.

Yep, these allow our scientists to automatically detect hurricanes in the simulations

○ We use abstraction in CS10, but *scientists don't with the Community Climate Code*, since details matter!

Nope, they certainly use abstraction – each developer works on their own module and doesn't need to know how the others are written, only how they plug into the module they're working on

○ None of these

Question 3: Which of the following was in the *Limits of Computing* lecture? (select ONE)

○ To say a problem is "in NP" means that you can randomly guess a solution, and verify it in linear time.

Nope, we verify in *polynomial* time

○ Finding a polynomial-time solution to just one NP-Complete problem means *P ≠ NP!*

Nope, it would imply P = NP

○ A program exists that can tell if another program, run on some input, will eventually stop.

Nope, that's called the halting problem, and we proved in class why it was impossible to write

○ The greedy approach to solving the "knapsack problem" will always produce the optimal answer, as we saw.

Nope, we saw a case when greedy wasn't optimal (100kg knapsack, Gold: 51 kg @ $52, Silver: 50 kg @ $50); the greedy case takes only one gold and gets $52, but the optimal case is two silver to get $100.

● None of these

Question 4: Which of the following was in the *Artificial Intelligence (AI)* lecture? (select ONE)

○ It's easier for a robot to open a door than to win a game of chess.

Nope, we saw exactly the opposite – a video of many robots failing to open a door.

○ Simulated Neural Networks used to be all the rage, but limited results meant the field has abandoned it.

Nope, neural networks are ALL the rage now.

● AI systems are now able to generate (not just recognize) images that never existed before.

Yep, in fact we showed an image of a person generated from other images.

○ One of the benefits of using AI systems in law enforcement is that they don't contain racial/gender biases.

Nope, AI systems are as biased as the data used to train them, and unfortunately the data contains biases.

○ None of these.

Question 5: Who was one of the participants of the *Alumni Panel*? (select ONE)

○ A professional game developer at Electronic Arts who advised to take CS classes outside of the CS dept.
○ A professional software engineer at Pixar who advised you not to avoid challenge.
○ A professional backend engineer at Apple who advised you to delete your social media.
○ A professional full stack engineer at Pandora who advised taking personal finance courses.
● None of these

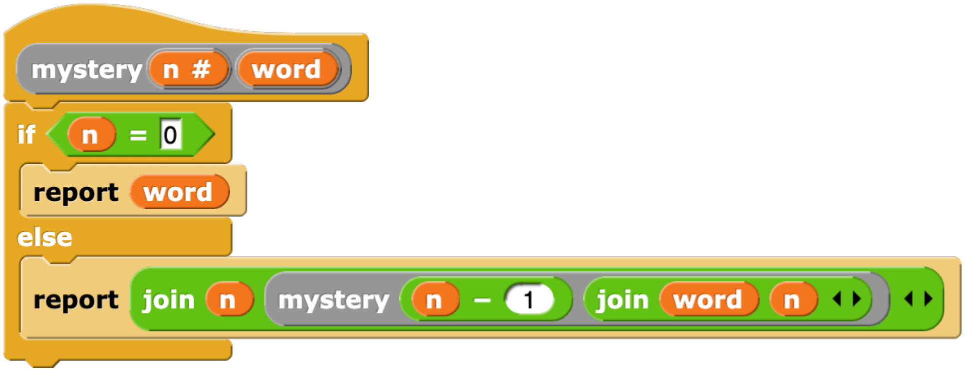
None of the participants were professional engineers/developers; we had two Cal students and a Cal lecturer.

Question 6: What is $B_{10} \times (B_{10} + 1_{10})$ represented in base B? (select ONE)

○	○	○	○	●	○	○	○	○	○	○	○	○
1_B	10_B	11_B	100_B	110_B	2_B	20_B	21_B	22_B	12_B	B_B	$B1_B$	BB_B

In base B, B is represented as 10_B, and $1_{10} = 1_B$.
Math works the same in any base, so we get $10_B \times (10_B + 1_B) = 10_B \times 11_B = 110_B$.
We can double-check this works in binary: $2_{10} \times (2_{10} + 1_{10}) = 2_{10} \times 3_{10} = 6_{10} = 110_2$ (and it checks out, yay)

Question 7: *Magical Mystery Block, Step Right This Way!* (8 pts=2+2+1+1+2) SID: ___________



```
mystery (n #) (word)
if <(n) = (0)>
    report (word)
else
    report (join (n) (mystery ((n) − (1)) (join (word) (n))))
```

a) What does (mystery 1 CAL) return?
If it is an error, write “error”. If it never returns, write “loops”.

1CAL1

If we trace it, it’s

```
mystery(1, CAL)
join(1, mystery(0, join(CAL,1) ) )
join(1, mystery(0, CAL1) )
join(1, CAL1)
1CAL1
```

b) What does (mystery 2 CAL) return?
If it is an error, write “error”. If it never returns, write “loops”.

21CAL21

If we trace it, it’s

```
join(2, mystery(1, join(CAL,2) ) )
join(2, mystery(1, CAL2) )
join(2, join(1, mystery(0, join(CAL2,1) ) ) )
join(2, join(1, mystery(0, CAL21) ) )
join(2, join(1, CAL21) )
join(2, 1CAL21)
21CAL21
```

c) What are the first and last five **characters** of the return value of (mystery 800 CAL)?
If it is an error, write “error”. If it never returns, write “loops”.

8	0	0	7	9	…	5	4	3	2	1

If we extrapolate, it’ll be all the numbers counting down in order in front of CAL and in back of CAL, no spaces.

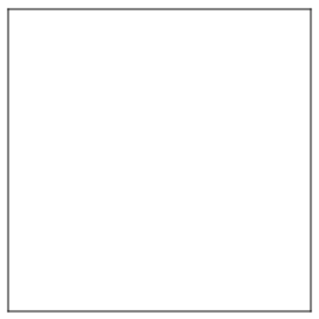
`800799798 … 987654321CAL800799798 … 987654321`

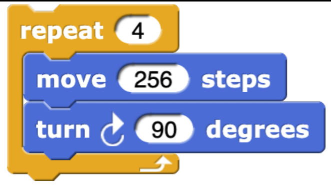
d) Give one value for the first argument of (mystery ( ) CAL) that will cause it never to return (even if we waited a very, very long time)? If that is impossible, write IMPOSSIBLE. Our base case only checks for = 0, so if we give it any negative number it’ll run forever, with more and more negative numbers.

-1

Question 8: *Box within a box within a box…* (8 pts) SID: _______________

N = 1

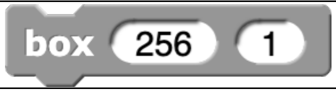




```
repeat 4
    move 256 steps
    turn ↻ 90 degrees
```

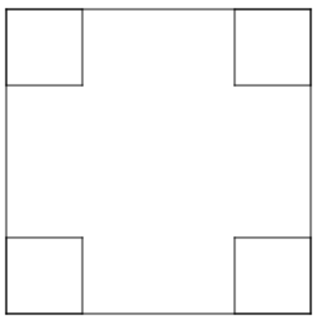
In "text your friend" format, this would be:

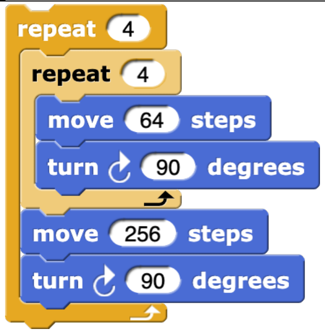
```
repeat(4)
--> move(256)steps
--> turn right(90)
```



```
box 256 1
```

N = 2

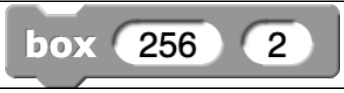




```
repeat 4
    repeat 4
        move 64 steps
        turn ↻ 90 degrees
    move 256 steps
    turn ↻ 90 degrees
```

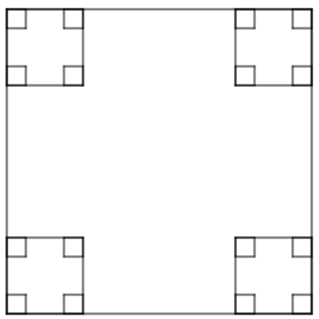
In "text your friend" format, this would be:

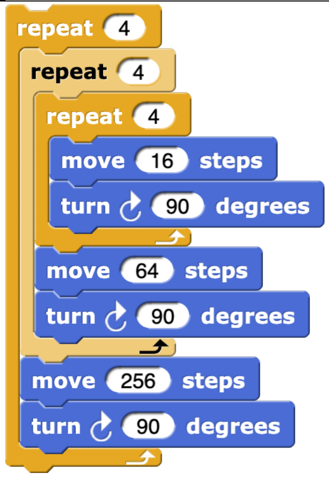
```
repeat(4)
--> repeat(4)
--> --> move(64)steps
--> --> turn right(90)
--> move(256)steps
--> turn right(90)
```



```
box 256 2
```

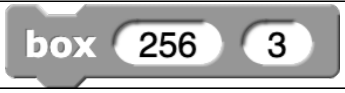
N = 3





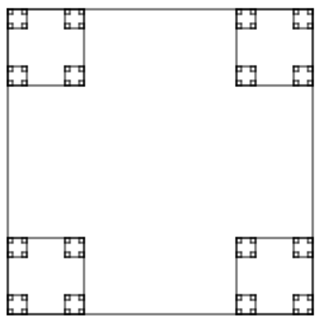
```
repeat 4
    repeat 4
        repeat 4
            move 16 steps
            turn ↻ 90 degrees
        move 64 steps
        turn ↻ 90 degrees
    move 256 steps
    turn ↻ 90 degrees
```

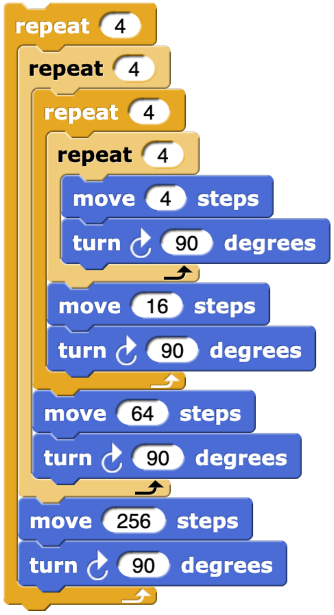
Hopefully you see what the "text your friend" pattern would be…



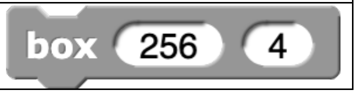
```
box 256 3
```

N = 4





```
repeat 4
    repeat 4
        repeat 4
            repeat 4
                move 4 steps
                turn ↻ 90 degrees
            move 16 steps
            turn ↻ 90 degrees
        move 64 steps
        turn ↻ 90 degrees
    move 256 steps
    turn ↻ 90 degrees
```

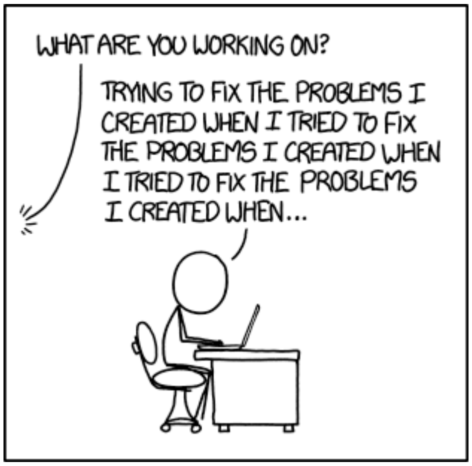


```
box 256 4
```

You load a Snap*!* file that has these four code snippets, and realize these are all the same pattern!

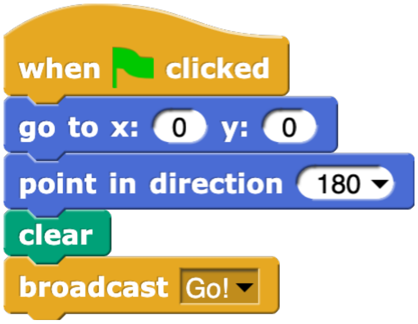
Write **box**, a single recursive program that does the same thing, and if called as shown above would generate the same drawings. Use the "text your friend" style we show above for **N=1,2**. Our **if** statement has no **else**, since here our base case is to do *nothing* (so we decided to remove it entirely). This is a little easier, actually, because you can just focus on the recursive case…

```
box(steps, N)
----> if N ≥ 1
----> ----> repeat(4)
----> ----> ----> box(steps/4, N-1)
----> ----> ----> move(steps)steps
----> ----> ----> turn right(90)
```

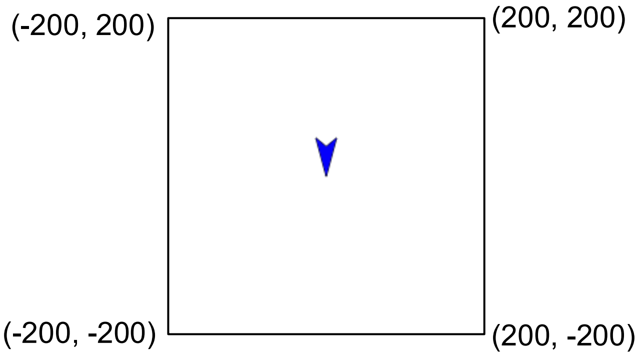

WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

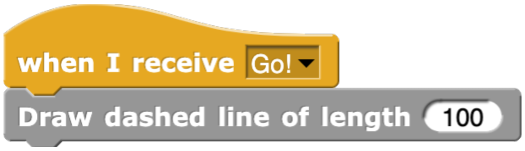
Question 9: *I should have made a left turn at Albequerque…* (8 pts)

SID: _______________

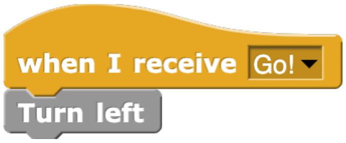


```
when [flag] clicked
go to x: (0) y: (0)
point in direction (180 ▾)
clear
broadcast [Go! ▾]
```

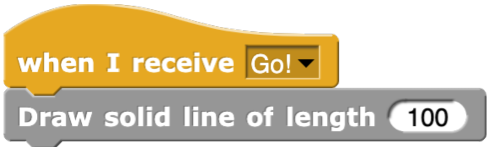




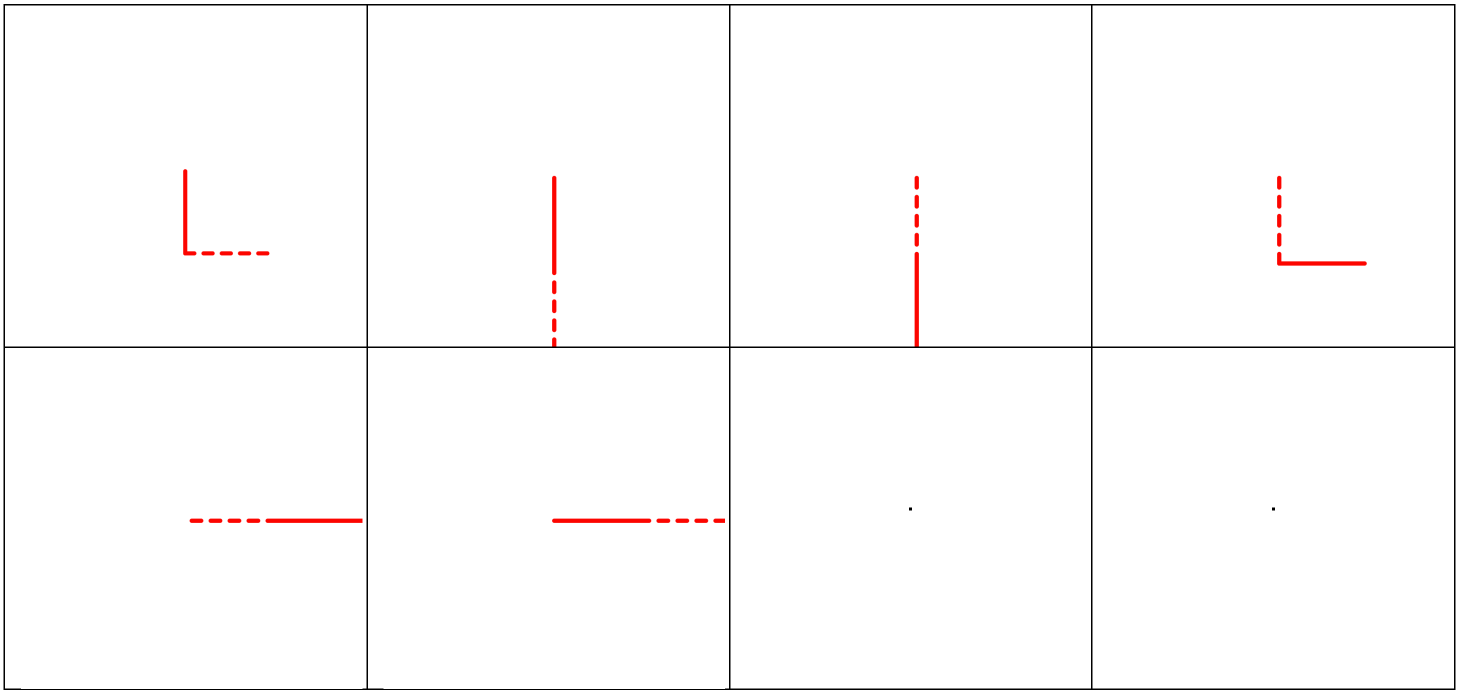
```
when I receive [Go! ▾]
Draw dashed line of length (100)
```



```
when I receive [Go! ▾]
Turn left
```



```
when I receive [Go! ▾]
Draw solid line of length (100)
```



a) *How many different images* can be drawn when we click the green flag? *You can use the area above as scratch space.* Three quick details:

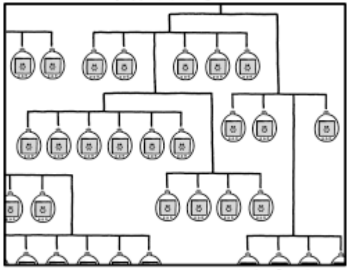
- The sprite starts in the center of the stage *facing down*.
- Each individual block is *atomic*, which means it can't be interrupted mid-way.
- The order Snap*!* chooses when there are multiple "when I receive" recipients is random.

○	○	○	○	○	○	●	○	○
0	1	2	3	4	5	6	7	8

b) What is the name we give to this kind of problem?

Race condition

MY HOBBY:



RUNNING A MASSIVE DISTRIBUTED COMPUTING PROJECT THAT SIMULATES TRILLIONS AND TRILLIONS OF TAMAGOTCHIS AND KEEPS THEM ALL CONSTANTLY FED AND HAPPY

Question 10: *Penn and Teller would be proud…* (5 pts) SID: _______________

You hold a deck of cards in your hands (facing up) and repeat the following steps, until you have no cards left:

1) put the top card (from the deck in your hand) on the table
2) put the top card (from the deck in your hand) on the bottom of the deck (no change if it's only one card).

For example, if the cards were (top-to-bottom) A B C D, the table shows what would happen…

Cards in hand (top-to-bottom)	**Table**	**Comment**
A B C D		Starting setup
B C D	A	Put the top card on the table
C D B	A	Put the top card on the bottom
D B	A C	Put the top card on the table
B D	A C	Put the top card on the bottom
D	A C B	Put the top card on the table
D	A C B	No change, only one card
	A C B D	Put the top card on the table

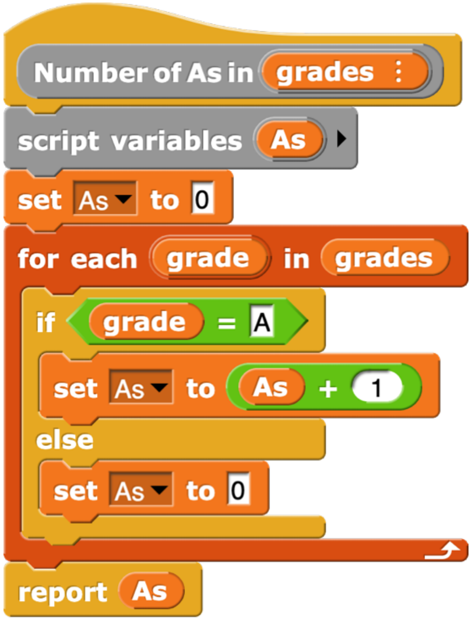
If we have cards numbered 1-10, what should the initial order of them be (here listed top-to-bottom) so that *the cards on the table end up in order (1-10)*? We've done the first two for you. *Hint: it's NOT 1,6,2,7,3,8,4,9,5,10*

1	**6**	**2**	**10**	**3**	**7**	**4**	**9**	**5**	**8**

One way to do this is to label the cards ABCDEFGHIJ and lay them out

Cards in hand (top-to-bottom)	**Table**
A B C D E F G H I J	
B C D E F G H I J	A
C D E F G H I J B	A
D E F G H I J B	A C
E F G H I J B D	A C
F G H I J B D	A C E
G H I J B D F	A C E
H I J B D F	A C E G
I J B D F H	A C E G
J B D F H	A C E G I
B D F H J	A C E G I
D F H J	A C E G I B
F H J D	A C E G I B
H J D	A C E G I B F
J D H	A C E G I B F
D H	A C E G I B F J
H D	A C E G I B F J
D	A C E G I B F J H
D	A C E G I B F J H
	A C E G I B F J H D 1 2 3 4 5 6 7 8 9 10 …reordering by letter… A B C D E F G H I J 1 6 2 10 3 7 4 9 5 8

Question 11: *You like ABBA? That's also my grades this term!* (6 pts)



```
Number of As in (grades :)
script variables (As)
set As to 0
for each (grade) in (grades)
    if <(grade) = A>
        set As to ((As) + (1))
    else
        set As to 0
report (As)
```

You write a block to returns the number of As in a list of **`grades`**. Assume **`grades`** is a list of letters drawn from { A, B, C, D, E, F } – i.e., there are no A+ or A- or B+ or B- grades, etc., and that **`grades`** has at least one grade in it. The block has a small bug, but sometimes it returns the correct value!

a) *In one sentence*, what is the requirement for the **`grades`** list so that the block is *guaranteed to return the correct value*. It should be the case that any list that *passes* your requirement will return the *correct* value and any list that *fails* it will return an incorrect value. For example (these are all wrong, but it's in the right spirit): "**`grades`** needs exactly 13 items" or "**`grades`** cannot have any Fs"

Since it resets the count whenever it finds a non-A, it only works if all non-As are in the front, and all As are at the end.

b) In one sentence, suggest a fix to the code above so that it will work for all inputs.

Remove the else clause entirely, or change it to "Set As to As" (so it doesn't reset when it finds a non-A)

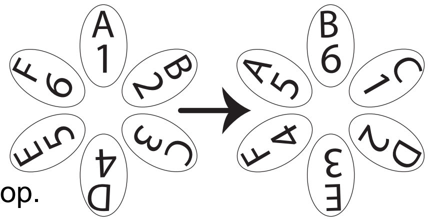
Question 12: *The Big Game? Why not the Big Dance?* (12 pts) SID: _______________

Stanford decides to expand the size of its undergrad program until it's *exactly the same size as Cal's*, let's call the number of students at each school *N*. Obviously, the SID system between the two campuses are different, so you'd like to find out if there's *any student at Cal with the same SID as a student at Stanford*. Note that the algorithm stops whenever someone says "YES" (the same SID was given to a Cal & Stanford student) or "NO".

Algorithm I – "Musical Chairs" algorithm

1. Dan records all possible pairs of Cal-Stanford students who haven't met yet (that's all of them at this point). *Don't count any of Dan's "haven't met yet" record-keeping in the stopwatch time calculation in (d).*
2. Dan finds a pair (one from each school) who still haven't met (assume he can do this in constant time), pairs them up, and updates his records. If he can't find a pair, he says "NO", and we stop.
3. The music starts, everyone *else* mills around the room and when the music stops, they find a random partner from a different school (not necessarily someone they haven't met before).
4. If anyone matches their SIDs with their partner, they say "YES", and we stop.
5. If not, we repeat 2-5.

Algorithm II – "Two Circles" algorithm



1. All the Stanford students make an inner circle facing out, Cal students form an outer circle facing in. Everyone stands up across from someone.
2. If anyone matches their SIDs with their partner, they say "YES", and we stop.
3. Otherwise, both circles move one space to the right, and everyone finds a new partner (e.g., see above).
4. Step 1-3 continues N times, and if nobody has said anything yet, they all say "NO", and we stop.

Algorithm III – "One at a Time" algorithm

1. If there are no more Cal students, all Stanford students say "NO", and we stop.

2. Otherwise, one Cal student steps forward, and one-by-one compares their SID with every student from Stanford. If any pair of the SIDs match, they both say “YES”, and we stop.
3. If the Cal student finds no Stanford SID matches, the Cal student leaves, and we go to step 1.

a) If each algorithm says “YES”, are you *guaranteed* that *at least one SID matches*? This means there are no false positives. (select ONE per row)

Algorithm	Yes	No
Musical Chairs	●	○
Two Circles	●	○
One at a Time	●	○

b) If each algorithm says “NO”, are you *guaranteed that no SID matches*? This means there are no false negatives. (select ONE per row)

Algorithm	Yes	No	Why?
Musical Chairs	●	○	
Two Circles	○	●	If N=6 as in diagram, A will only see 1,3,5.
One at a Time	●	○	

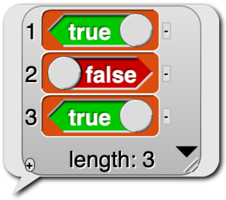
c) In the WORST case, what’s the *number of comparisons* (NOT running time)? If it’s actually between two categories, pick the bigger category. E.g., N^4 is bigger than *cubic*, so pick *exponential*. (select ONE per row)

Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential	Why?
Musical Chairs	○	○	○	○	●	○	N^2 iterations, N comparisons each
Two Circles	○	○	○	●	○	○	N iterations, N comparisons each
One at a Time	○	○	○	●	○	○	N Cal students, N comparisons each

d) If the SID comparisons between different pairs of people could happen at the same time, *how much elapsed stopwatch time* (NOT running time) would each algorithm take in the WORST case? (select ONE per row)

Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential	Why?
Musical Chairs	○	○	○	●	○	○	N^2 iterations
Two Circles	○	○	●	○	○	○	N iterations
One at a Time	○	○	○	●	○	○	N^2 comparisons, each one at a time

Question 13: *Two keeps/maps are better than one! (True? False? True False True?...)* (8 pts)

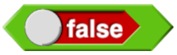


1 true
2 false
3 true
length: 3

You are given the list DATA . What do the following expressions return? Select ONE per row. A particular (**A**-**L**) choice may be the right answer for several rows (you could have multiple circles in a column).

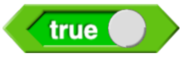
	A	B	C	D	E	F	G	H	I	J	K	L
`map (not ( )) over DATA` If you apply not to each of (T F T), you get (F T F)	○	○	○	○	○	○	●	○	○	○	○	○
`keep items (not ( )) from DATA` If you keep only the elements that are NOT true (i.e., false), you end up keeping only the middle one. The elements that don’t cause the predicate to return true aren’t present in the output, so it’s not choice (E).	○	○	●	○	○	○	○	○	○	○	○	○
`map (not (not ( ))) over DATA` This is like flipping the light switch twice, it shouldn’t change the state of the lights. So the input here is equal to the ouput.	○	○	○	○	○	○	○	○	○	○	○	●
`keep items (not (not ( ))) from DATA` If you keep only the elements that are *not not* true, that means they *are* true, so you keep only the trues.	○	○	○	○	○	○	○	○	●	○	○	○
`map (not ( )) over (map (not ( )) over DATA)` Two cascaded nots means the same thing as flipping the light switch twice, so the answer is the same as two above.	○	○	○	○	○	○	○	○	○	○	○	●
`map (not ( )) over (keep items (not ( )) from DATA)` Keep not keeps only the false elements (F), and map of not flips their state so you get (T)	○	○	○	○	○	○	○	●	○	○	○	○
`keep items (not ( )) from (map (not ( )) over DATA)` If you flip all the switches they become (F T F), and if you keep only the false ones you get (F F)	○	○	○	●	○	○	○	○	○	○	○	○
`keep items (not ( )) from (keep items (not ( )) from DATA)` If you apply the same filter twice, it doesn’t do anything the second time, so this answer is the same as only one keep.	○	○	●	○	○	○	○	○	○	○	○	○

A



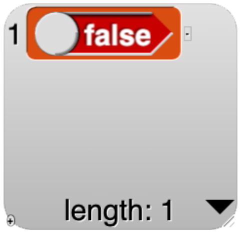
false

B



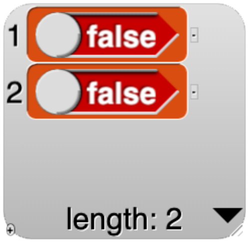
true

C



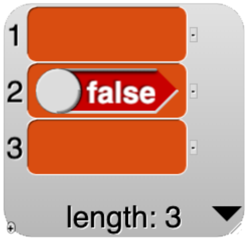
1 false
length: 1

D



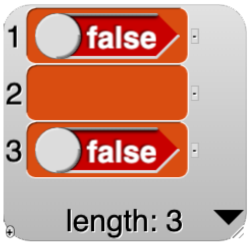
1 false
2 false
length: 2

E



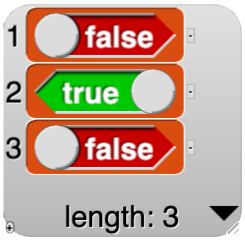
1
2 false
3
length: 3

F



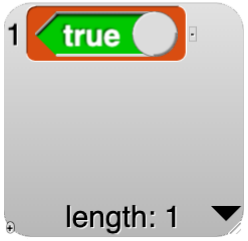
1 false
2
3 false
length: 3

G

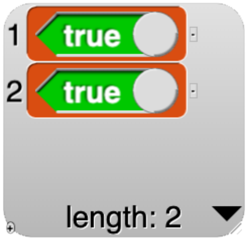


1 false
2 true
3 false
length: 3

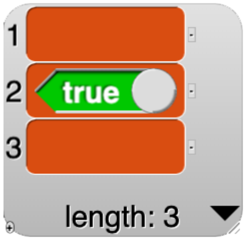
H



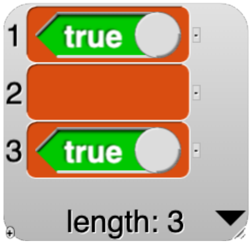
I



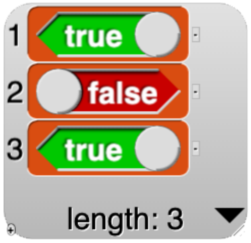
J



K



L



Question 14: *Berkeley Python Flying Circus…* (10 pts = 5 * 2 pts)

```
>>> R = range(10,20)
>>> list(R[2:4])
```

○ [2,3,4] ○ [2,3] ○ [11,12] ○ [11,12,13] ● [12,13] ○ [12,13,14] ○ None of these

```
R is the range (10, 11, 12, 13, 14, 15, 16, 17, 18, 19); it's 0-indexed so it's
[12, 13]
```

```
>>> A = ["3", "50", "700"]
>>> A[1] + "1"
```

○ [4] ○ [51] ○ ["4"] ○ ["51"] ○ 4 ○ 51 ○ "4" ○ "51" ○ "31" ● "501" ○ Error

```
Lists are 0-index so A[1] is "50" and + is string concatenation
```

```
>>> def compose(f,g): return lambda x: f(g(x))   ### Pregnant shark
>>> from functools import reduce                 ### reduce is like Snap!'s combine
>>> def double(x): return x+x
>>> def plus1(x):  return x+1
>>> def times2(x): return x*2
>>> frankenstein = reduce(compose, [double, str, plus1, times2, double])

>>> print(frankenstein( 10 )) ### What input causes 4141 to be printed?
4141
```

```
If you reverse everything you get the answer
"4141" → un-double → "41" → un-str → 41 → un-plus1 → 40 → un-times2 → 20 →
un-double → 10
```

crossmap is a really useful utility function. It takes a dyadic function (i.e., a function of two arguments), and two lists (not necessarily of equal size) and calls the function on every pair of values (one from each list) and puts them in a list. So, for example:

```
>>> def plus(a,b): return a+b
>>> crossmap(plus, [1,2,3], [10,20])
[11, 12, 13, 21, 22, 23]
>>> print(crossmap(plus,"ab","123"))
['a1', 'b1', 'a2', 'b2', 'a3', 'b3']
```

You realize this would be useful to help you with the desire to create a block that tests whether any SID in a list of Cal SIDs is the same as the SID of a student in a list of Stanford SIDs! Fill in the blanks to a **helper** function and the **any_matches** function (which takes two lists and returns **True** if the lists have a shared element).

```
>>> def helper(a,b): return a == b ## Variant: a - b

>>> def any_matches(CalSIDs, StanfordSIDs):

        return True in crossmap(helper, CalSIDs, StanfordSIDs) ## Variant: 0 in
             or Any( )
```

Question 15: *Dictionaries are too hard!…* (10 pts = 4+2+4 pts)

Boy, dictionaries sure are great! They allow us to make a connection between keys and values. We want our smart thermostat AXELA to answer us (in English) how the temperature feels. You assume the coldest it will ever be is -460º F (absolute 0) and the warmest it'll ever be is 212º F (boiling point of water). Then you'll need to store a connection between temperatures ranges to English words for how you'd feel in that range:

-460º 40º 60º 80º 212º

Cold	Cool	Nice	Hot

What we *want* is a *soft* dictionary, where we specify numeric ranges as the keys (as a tuple), and any value between the left edge (and up to but not including the right edge) should "match" the key. If we asked, "how does it feel?", if it were 45º, it'd say "Cool". If it were 99º, it'd say "Hot". Let's build this!

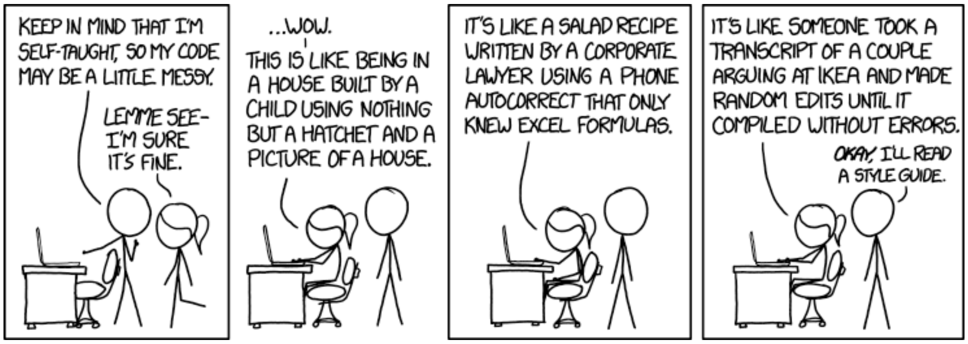
```
>>> def SD_create(): return {}         ### empty soft dictionary (SD)
>>> def SD_add(SD, low, high, value): SD[(low,high)] = value ### key is tuple
>>> sd = SD_create()
>>> SD_add(sd,-460,  40, "Cold")    ### For this entire page, assume the
>>> SD_add(sd,  40,  60, "Cool")    ### left edge value is smaller than the
>>> SD_add(sd,  60,  80, "Nice")    ### right edge value. No need to error check it
>>> SD_add(sd,  80, 212, "Hot")
>>> SD_get(sd, 45) ➔ "Cool"
>>> SD_get(sd, 99) ➔ "Hot"
```

a) Complete the definition of `SD_get` so that it works as shown above. Assume the key falls in *some* range that you have stored. Also assume that `SD_add` never had ranges overlap – the ranges in the example above don't because they are *inclusive* of the left edge but *exclusive* of the right edge, i.e., [left, right).

```
def SD_get(SD, key):

    for low_high_range in SD:
                                              ### simultaneous assignment since
        key_low, key_high = low_high_range    ### low_high_range is a tuple

        if key_low <= key < key_high

            return SD[range]

    print("ERROR: no range contains key!")
```

b) Those are some big assumptions. Let's fix the first one. If the user calls `SD_get` with a key not in some range, it should not return anything and print an error message, as shown below. *Modify the code above by adding a print statement* (in-between, above or below the lines, *indented correctly*) to do this.

```
>>> SD_get(sd, 1000)
ERROR: no range contains key!
```


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

c) Now let’s fix the last assumption. Fill in the blank of a new **SD_add** with code so that adding a range that overlaps with another range causes it to print an error, and doesn’t modify the soft dictionary.

```
>>> SD_add(sd, 70, 82, "Perfect")
ERROR: range overlaps!
```

```
def SD_add(SD, low, high, value):      ### also assume low < high
    for low_high_range in SD:
        key_low, key_high = low_high_range ### simultaneous assignment

        if not(high <= key_low or key_high <= low):
```

or, equivalently…

```
           key_low < high and low < key_high:
            print("ERROR: range overlaps!")
            return         ### Error, so just return and don’t store anything
    SD[(low,high)] = value ### key is tuple
```