UC Berkeley's CS10 Spring 2019 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) →*

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: What is $101_2 + 21_3$? (select ONE)

○	○	○	○	○	○	○	○	○	○	○	○	○
122_{16}	8_{16}	9_{16}	A_{16}	B_{16}	C_{16}	D_{16}	E_{16}	F_{16}	10_{16}	11_{16}	12_{16}	13_{16}

Question 2: Which of the following was in the *Human-Computer Interaction* lecture? (select ONE)
○ Solid planning by good design teams always beats "enlightened trial and error"
○ The digital pen was invented after 2000.
○ The HCI design cycle is "Design", "Prototype", "Evaluate" (and repeat!).
○ With the rise of cloud computing, we have moved from Ubiquitous computing to Mainframe-based computing.
○ None of these

Question 3: Which of the following was in the *Saving the World with Computing* lecture? (select ONE)
○ After using higher resolution models, some said earlier "doomsday" global warming trends were too extreme.
○ One of the projects at the lab was to use computers to study Ebola outbreaks and how to contain them.
○ The data structures to model the earth don't affect the computation; "big fat" or "long skinny" triangles are ok.
○ Genome assembly is fairly easy and straightforward; it's just splicing the data from the read strips together.
○ None of these

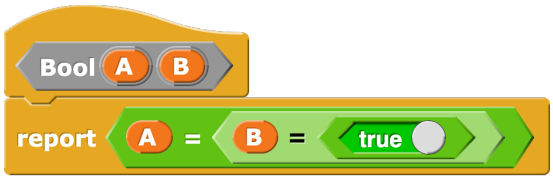
Question 4: Which of the following was in the *Limits of Computing* lecture? (select ONE)
○ The "knapsack problem" is all about geometry – how to fit the most different-shaped boxes in a knapsack.
○ The "subset sum problem" is solvable in linear time – you just add up the subset of the numbers you want.
○ Alan Turing proved all problems are decidable; people used to believe it wasn't.
○ They recently proved that P=NP, sharing this amazing discovery was the main point of the lecture.
○ None of these

Question 5: Based on the *Artificial Intelligence (AI)* lecture, which one is easiest for a computer? (select ONE)
○ Detecting a stop sign from an image captured by a camera.
○ Translating an essay from one language to another.
○ Having a conversation in English with a person.
○ Winning a game of Chess.
○ Opening a door.

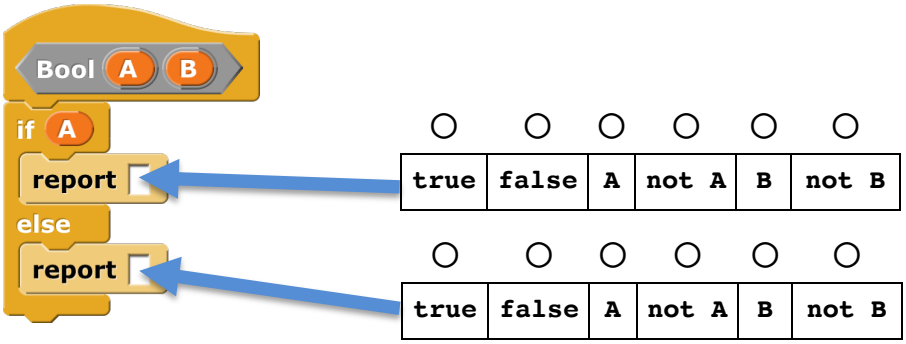
Question 6: Which one of these WAS NOT one of the participants of the *Alumni Panel*? (select ONE)
○ An indie game developer who advised to take CS classes outside of CS dept.
○ A software engineer at YouTube who advised you not to avoid challenge.
○ A backend engineer at Lisnr who advised you to delete your social media.
○ A data visualization consultant who advised taking personal finance courses.
○ None of these, they were all on the panel!

SID: _____

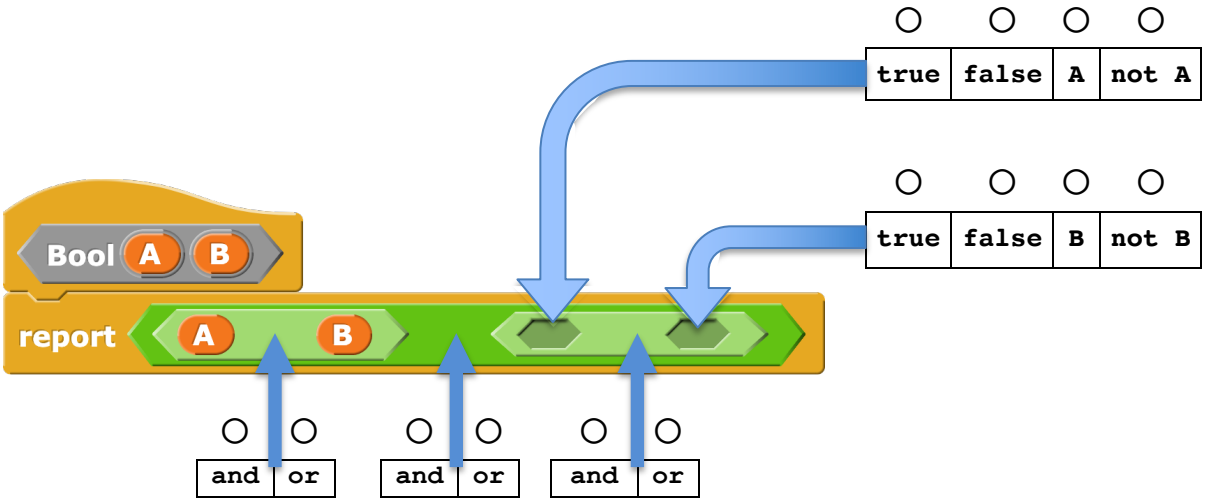
(The **Bool** block on the right is used for Questions 7 & 8; 2,3 pts)



Question 7: Fill in the blanks so the predicate is the same as the original **Bool** block. (select ONE from each)

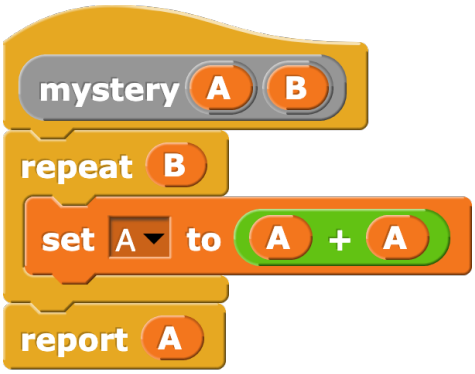


Question 8: Fill in the blanks so the predicate is the same as the original **Bool** block. (select ONE from each)



Question 9: ***Roll up for the magical `mystery` block, step right this way…*** (4 pts)

What does **`mystery`** (below) report, if B is a counting number (i.e., 1, 2, 3, …)? (select ONE)



○ $A \times B$	○ A^B	○ B^A	○ $A \times B^B$	○ $A \times B^A$	○ $A \times A^B$	○ $A \times 2^B$	○ $A \times B^2$	○ $A+B$	○ $A+B^2$	○ $A+2^B$	○ Error	○ Infinite Loop

Question 10: ***It's a mad, mad, mad, mad world…*** (4 pts)

Two people (A and B) were told to find a hidden paper bag, which initially has $10 in it. Each one was told that when they found it, they would look at the contents, then wait a random amount of time within another week before returning to the spot, burning the paper bag (and whatever was in it) to ashes, and replacing it *with an identical paper bag and contents to the one they initially saw*, except:

- **Person A:** The $ amount has been increased by 2. E.g., if the paper bag initially had $200, they would swap it with a paper bag with $202 inside. If the paper bag had $200 and €1000, they would swap it with a paper bag that had $202 and €1000 inside.

- **Person B:** They would convert half of the $ to € (assume the exchange rate is the same, $1 = €1). E.g., if the paper bag initially had $200, they would swap it with a paper bag that had $100 and €100.

What are possible contents of the paper bag at the end, *after both had done their swap*? (select ALL that apply by **filling in the box completely** for a value of dollars ($) in the column and euros (€) in the row.)

	$0	**$5**	**$6**	**$7**	**$10**	**$12**
€0						
€5						
€6						
€7						
€10						
€12						

Question 11: *Did we assign all student ID numbers correctly?* (9 pts) SID: _____

There's been a problem with the student ID system and campus is not sure everyone has a unique number! Here are 3 algorithms to find out whether all student IDs (SIDs) are unique or not. For all problems, assume the number of students (N) is a power of 2 and really big. (How big?) Really big. Also, "clock time" is the actual elapsed time if you used a clock or stopwatch to time the running of the algorithm.

Algorithm I – "Musical Chairs" algorithm

1. The music starts, everyone mills around the room and when the music stops, they find a random partner and compare SIDs.
2. If anyone matches their SIDs with their partner, they yell "NOT UNIQUE!!" and stop.
3. If not, they wait for the music to start again, and repeat 1-3.
4. This continues for exactly N rounds, and if nobody has yelled "NOT UNIQUE", they all yell "UNIQUE".

Algorithm II – "Recursive Halving" algorithm

1. All people in the room pair up and compare SIDs with their partner.
2. If anyone matches their SIDs with their partner, they yell "NOT UNIQUE!!" and stop.
3. Otherwise, there will be a smaller and a larger SID. All the smaller SID students hold hands and go find a new room together, and all the larger SID students hold hands and go find a new room.
4. Step 1-3 continues until there are N rooms occupied with one person in each room, at that point they all yell "UNIQUE" and stop.

Algorithm III – "Down the Line" algorithm

1. Everyone lines up, and the first person is designated as the "unique tester".
2. The "unique tester" person goes down the line, comparing their SID to that of each new person, 1-on-1.
3. If ever the SIDs match, they yell "NOT UNIQUE!!" and stop.
4. If they get to the end and haven't matched, they go to the back of the line and the new first person is designated as the "unique tester", and they repeat steps 2-4.
5. If that initial first person ever ends up being first again, the yell "UNIQUE" and stop.

a) If each algorithm says "UNIQUE", are you *guaranteed* all SIDs are unique? (select ONE per row)

Algorithm	Yes	No
Musical Chairs	○	○
Recursive Halving	○	○
Down the Line	○	○

b) 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
Musical Chairs	○	○	○	○	○	○
Recursive Halving	○	○	○	○	○	○
Down the Line	○	○	○	○	○	○

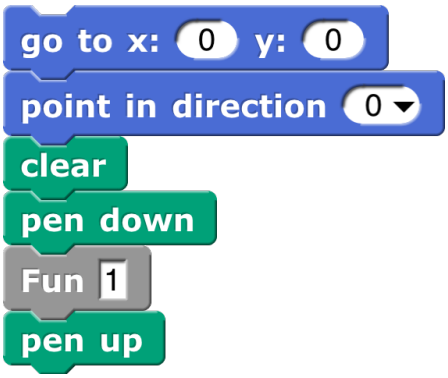
c) If the SID comparisons between different pairs of people could happen at the same time, *how much clock time* (NOT running time) would each algorithm take in the WORST case? (select ONE per row)

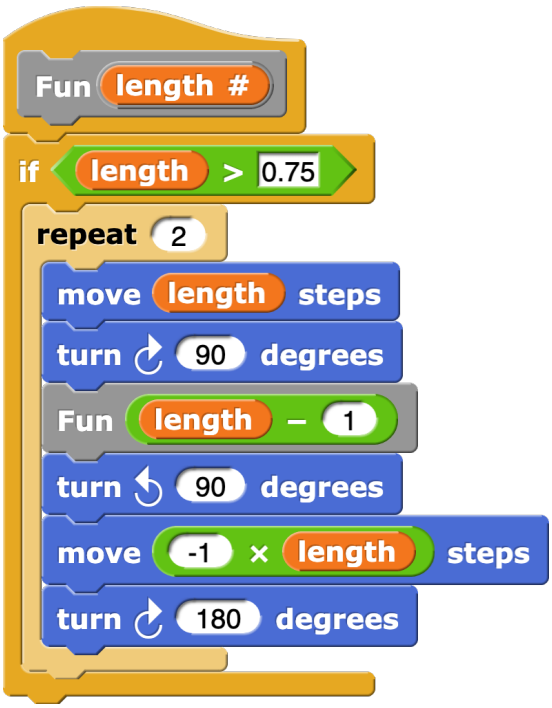
Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential
Musical Chairs	○	○	○	○	○	○
Recursive Halving	○	○	○	○	○	○
Down the Line	○	○	○	○	○	○

Question 12: *We put the `Fun` in Functional Programming…* (9 = 2+3+4 pts)

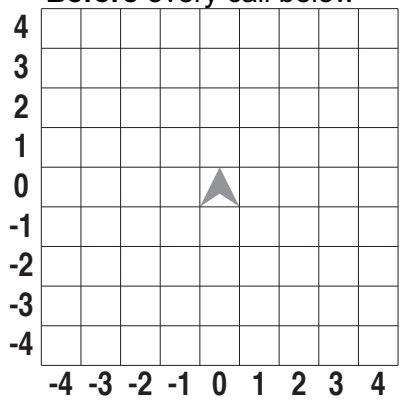
Consider the following code below. We're now to going to zoom in on pixels affected by calls to **`Fun`**; the stage is always cleared before the calls below, the sprite always starts in the center facing up, and the pen is in the *center* of the sprite.

Your job is to shade in (completely!) *all* the pixels that will be colored in after calls to **`Fun`** with n set to **1**, **2** & **3**. *Clarification: if the sprite were at (0,0) and moved 2 steps up, it would be at (0,2) and all pixels along the line from (0,0) through (0,2) would be shaded; 3 pixels in total.*

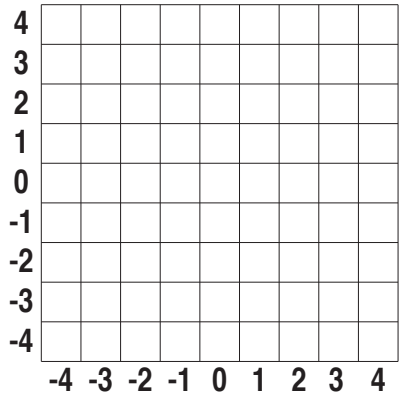




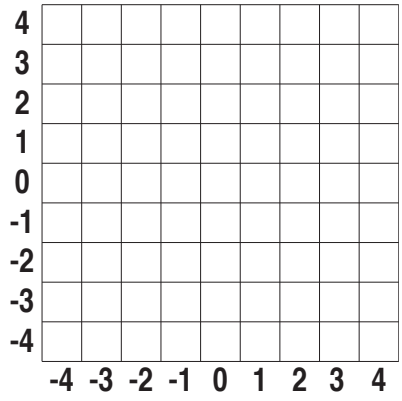
Before every call below



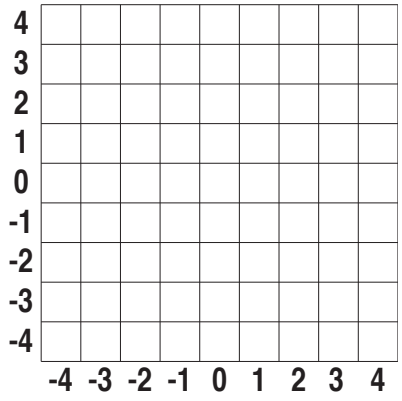
After the call to **`Fun(1)`**



After the call to **`Fun(2)`**

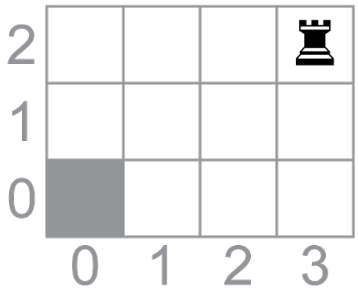


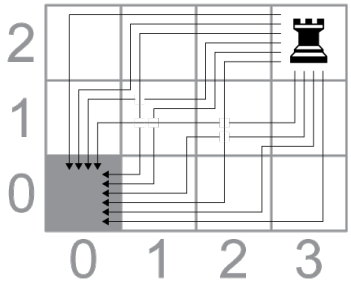
After the call to **`Fun(3)`**



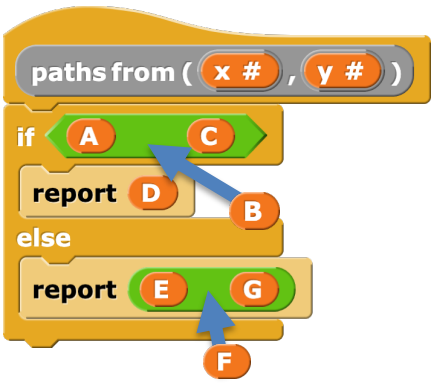
Question 13: *I just want to go home…* (14 pts = 1+1+1+1+4+2+4) SID: _____

A rook is a piece in the game of chess that can move any number of squares vertically or horizontally. We put a rook somewhere on integer coordinates in the first quadrant ($0 \le x \le \infty$, $0 \le y \le \infty$) and put a spell on it so that it can only move toward the origin (i.e., either down or left). Author `paths from ( x , y )` to calculate how many different paths there are home given an (x, y) starting point. E.g., the rook at (3, 2) could get back to (0, 0) any one of 10 ways, and the number of paths for each starting square in ($0 \le x \le 3$, $0 \le y \le 2$) is below.





2	1	3	6	10
1	1	2	3	4
0	1	1	1	1
	0	1	2	3



	A	B	C	D	E	F	G
0							
1							
2							
( ) × ( )							
( ) + ( )							
( ) or ( )							
( ) and ( )							
x < 0							
x = 0							
x > 0							
not x < 0							
not x = 0							
not x > 0							
y < 0							
y = 0							
y > 0							
not y < 0							
not y = 0							
not y > 0							
paths from (x − 1 , y − 1)							
paths from (x − 1 , y)							
paths from (x − 1 , y + 1)							
paths from (x , y − 1)							
paths from (x , y)							
paths from (x , y + 1)							
paths from (x + 1 , y − 1)							
paths from (x + 1 , y)							
paths from (x + 1 , y + 1)							

Author your solution below AND then **completely fill in the boxes** above to correspond to the solution you wrote below.

`paths from ( 3 , 2 )` 10

```
paths from(x,y):
    if _____________________________
        report _____________________________________________
    else
        report _____________________________________________
```

Question 14: ***Hey, let's call the function `f` on our `DATA`! Uh oh, on second thought…*** (7 pts, 1 each)

Ever wanted an "undo" in life? Well, finally we have it! Given a *particular* function 

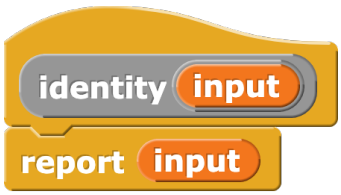
whose domain is anything, we have invented a miracle function 

that can always undo whatever 

did. I.e., this is *always true* for any 

:

Given the following function:



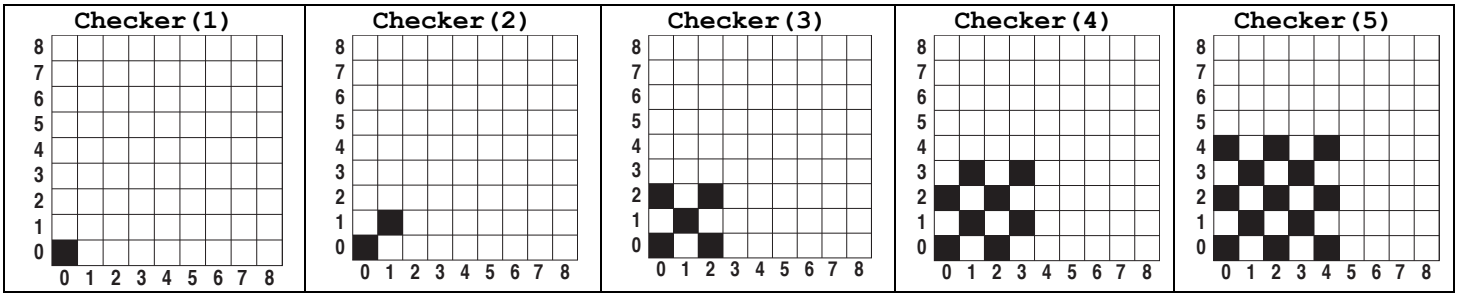
…which of the following are ALWAYS equivalent to the list 

?
We did the first one for you. (select ALL that apply)

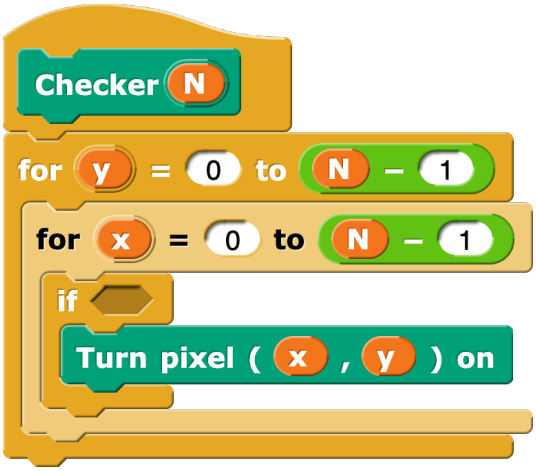
Expression	Yes, always equivalent to DATA	Not always equivalent to DATA
	●	○
	○	○
	○	○
	○	○
	○	○
	○	○
	○	○
	○	○

Question 15: *Joshua: "How about a nice game of checkers?"* (4 pts) SID: _____

You are trying to write code to generate a "N x N" checkerboard in which the bottom-left-most "origin" pixel is always turned on (black). E.g., calling it with N on values 1 through 5 should result in the following five images.



What Boolean expression should go in the `if` to accomplish this? (Select ONE)



- ○ x + y mod 2 = 0
- ○ x + y mod 2 = 1
- ○ x × y mod 2 = 0
- ○ x × y mod 2 = 1
- ○ x mod 2 = 0 and y mod 2 = 0
- ○ x mod 2 = 0 and y mod 2 = 1
- ○ x mod 2 = 1 and y mod 2 = 0
- ○ x mod 2 = 1 and y mod 2 = 1
- ○ x mod 2 = 0 or y mod 2 = 0
- ○ x mod 2 = 0 or y mod 2 = 1
- ○ x mod 2 = 1 or y mod 2 = 0
- ○ x mod 2 = 1 or y mod 2 = 1

Question 16: ***Berkeley python's flying circus…*** (19 pts = 8 * 2 + 3pts)

We recreated an interpreter script. For each, indicate what the right answer should be.

```
>>> 1 + '2'
```

○ `3` ○ `'3'` ○ `12` ○ `'12'` ○ Error ○ None of these

```
>>> [1] + ['2']
```

○ `[3]` ○ `['3']` ○ `[12]` ○ `['12']` ○ `[1,2]` ○ `['1','2']` ○ Error ○ None of these

```
>>> 'CAL'[1:2]
```

○ `'CA'` ○ `'AL'` ○ `'C'` ○ `'A'` ○ Error ○ None of these

```
>>> A = [3]
>>> B = A
>>> A.append(2)
>>> B
```

○ `[32]` ○ `[3]` ○ `[3,2]` ○ Error ○ None of these

We'd like to write a `backwards` function that would have the following behavior:

```
>>> school = 'cal'
>>> backwards(school)
'lac'
```

Is it possible to write `backwards`?

○ `yes` ○ `no`

```
>>> [n+1 for n in range(2,4) if n != 3]
```

○ `[3,5]` ○ `[4,5]` ○ `[3]` ○ `[4]` ○ Error ○ None of these

```
>>> ['n+1' for n in range(2,4) if n != 3]
```

○ `['3','5']` ○ `['4','5']` ○ `['3']` ○ `['4']` ○ Error ○ None of these

```
>>> [n+n for n in ['HA','AB'] if n != 'A']
```

○ `['HH','BB']` ○ `['HA','HA','AB','AB']` ○ `['HAHA','ABAB']` ○ Error ○ None of these

```
>>> def increm(n): return n+1
>>> def double(n): return n+n
>>> def square(n): return n*n
>>> D = {1: increm, 2: double, 3: square}
>>> [D[i](i) for i in [3,1,2]]
```

`[_____, _____, _____]`