UC Berkeley's CS10 Fall 2018 Final Exam ANSWERS: Prof. Dan Garcia, Instructor

Your Name (first last)

SID

Lab TA's Name

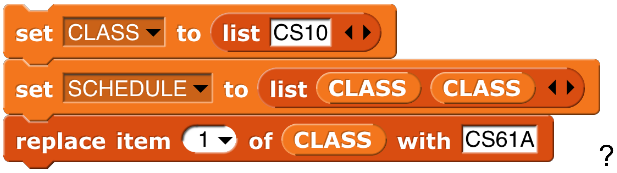
← Name of person on left (or aisle)

Name of person on right (or aisle) →

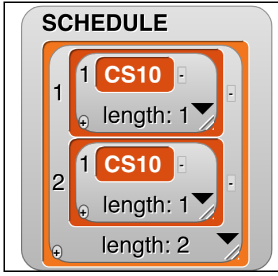
Fill in the correct circles & squares completely…like this: ● (select ONE) ■ (select ALL that apply)

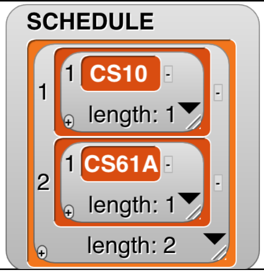
anthropocene man
DAD! THERE IS A RECURSION UNDER MY BED!
DAD! THERE IS A RECURSION UNDER MY BED!

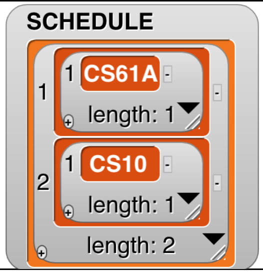
What's that Smell? Oh, it's Potpourri! (2 pts each for 1-6, low score dropped)

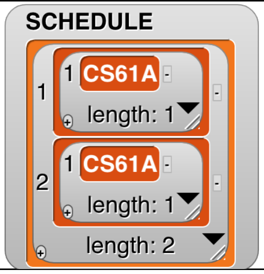
Question 1: What is **SCHEDULE** after  ? (select ONE)

```
set CLASS to list CS10
set SCHEDULE to list CLASS CLASS
replace item 1 of CLASS with CS61A
```

○ 

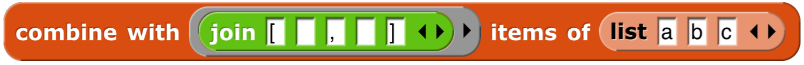
○ 

○ 

● 

○ None of these

This is because there's SHARING among the inner list elements – there's only ONE CS10 in schedule.

Question 2: What does  `combine with join [ _ , _ ] items of list a b c` report? (select ONE)

○ It's an error since you can only have + or × or `join` with two total holes in the `combine`.
○ It doesn't report anything, it mutates the values in the list.
○ It doesn't report anything, it causes an infinite loop.
● It depends on the implementation of `combine`, since it's not an associative *and* commutative function.
○ None of these

We did this one in lecture – join(_)(_) works, join( )(,)( ) works, but the outer brackets make it not associative.

Question 3: Which of the following was NOT from the *Human-Computer Interaction* lecture? (select ONE)

○ The design of ballots can affect and has affected the outcomes of elections.
● The classic design cycle is (1) gather specifications from client (2) code it up (3) deliver it to the client. Done!
○ We study interfaces because ~50% of a program's source code is typically dedicated to the user interface.
○ The Mouse was invented in 1963, well before Apple showed it to the world connected to an early Macintosh.
○ None of these

The design cycle is Design, Prototype, Evaluate REPEAT!!! (NOT done, it's almost never perfect the 1st time).

Question 4: Which of the following was NOT from the *Artificial Intelligence (AI)* lecture? (select ONE)

○ AI systems can now *transfer the "style"* of a painting to another image.
● The primary goal of reinforcement learning today is to make agents that *think rationally*.
○ China has developed a "virtual anchor" to deliver the news that is *almost indistinguishable from a real person*.
○ Neural networks would determine the probability that a photo has a cat in it, *rather than* a simple yes/no.
○ None of these

The primary goal of reinforcement learning today is to make agents that ACT rationally

Question 5: Which of the following was NOT from the *Algorithmic Bias* lecture? (select ONE)

● Google Image searches for "Grandmother" showed that almost all the results were of older white women.
○ Facial recognition algorithms were 99% accurate for white male faces, far less for darker skinned women.
○ Google translate was exhibiting gender bias: "__ is a doctor" was "he", but "__ is a nurse" was "she".
○ COMPAS was software that was written to try to address racial bias in sentencing by human judges.
○ None of these

While true, it wasn't something that Schuyler showed in class.

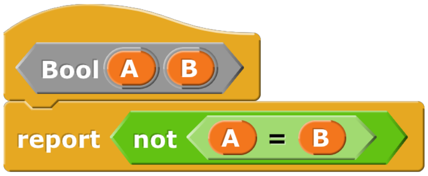
Question 6: What was one memorable moment from the alumni panel? (select ONE)

○ One of the panelists had a bloody nose and had to leave the stage.
○ One of the panelists came quite late (more than half-way into the conversation)
○ One of the panelists had an emergency phone call during the panel and had to leave the room.
○ One of the panelists gave out swag to every CS10 student.
● None of these

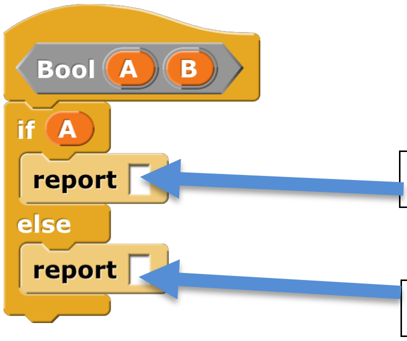
None of these happened, thankfully!.

SID: ________________

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



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



○	○	○	○	○	●
`true`	`false`	`A`	`not A`	`B`	`not B`

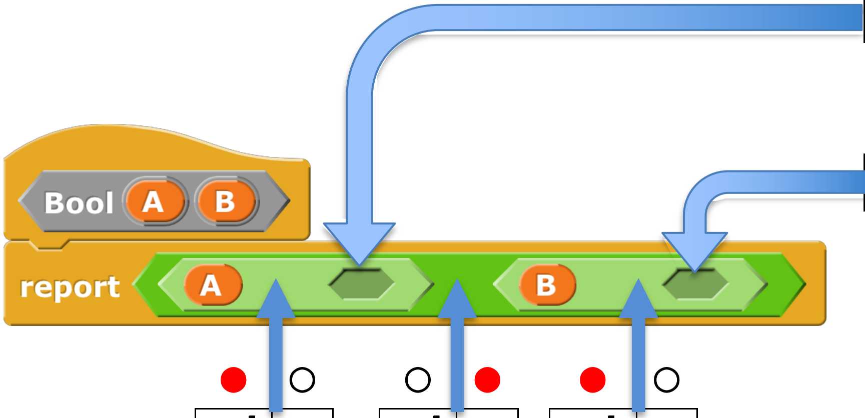
○	○	○	○	●	○
`true`	`false`	`A`	`not A`	`B`	`not B`

Bool returns true when A is different than B. So that's A=true, B=false or A=false, B=true. This if-else says when A is true, then that's case 1 when B must be false for the block to return true, so that would be not B. The then case hits when A is false, and that's case 2 when B must be true, so we just return B.

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

○	○	○	○	○	●
`true`	`false`	`A`	`not A`	`B`	`not B`

○	○	○	●	○	○
`true`	`false`	`A`	`not A`	`B`	`not B`



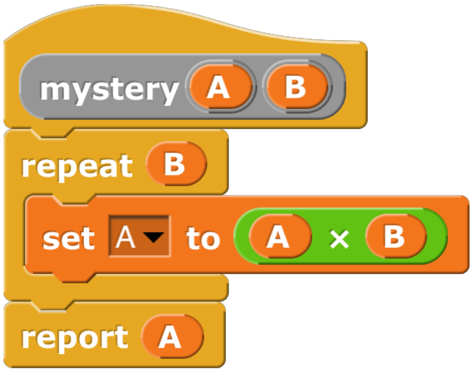
●	○
`and`	`or`

○	●
`and`	`or`

●	○
`and`	`or`

Bool returns true when A is different than B. So that's A=true, B=false or A=false, B=true. So that's when A and not B or B and not A (case 1 + case 2 above)

Question 9: What does **`mystery`** (below) report, if B is a counting number (i.e., 1, 2, …)? (select ONE, 3 pts)



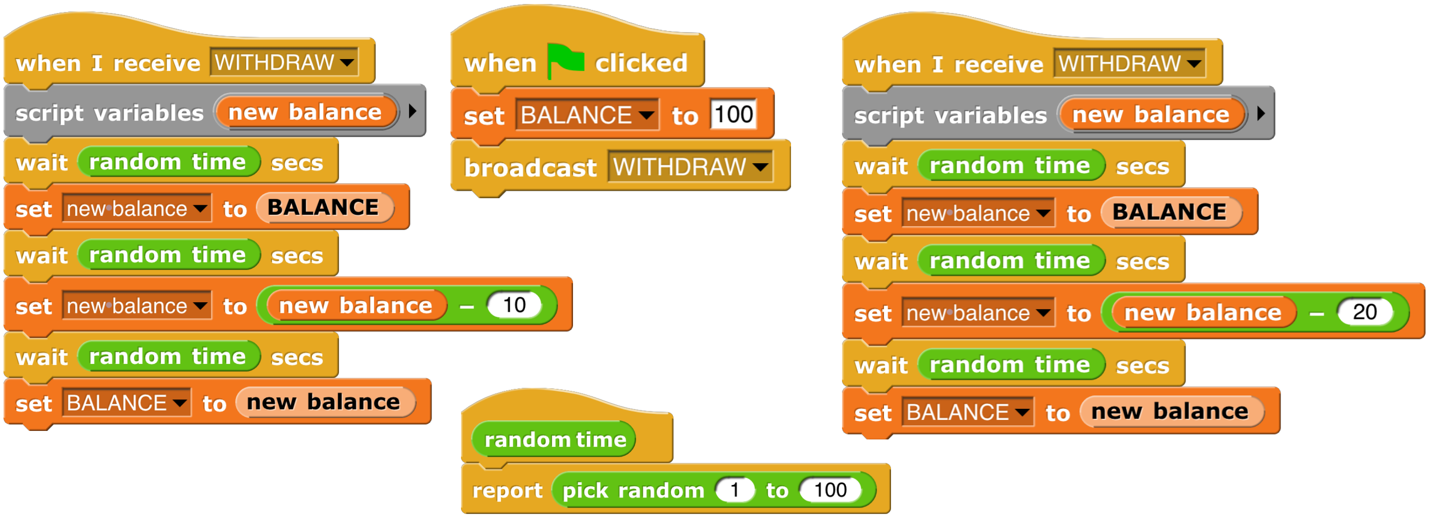
```
mystery (A) (B)
repeat (B)
    set [A] to ((A) × (B))
report (A)
```

○	○	○	○	○	●	○	○	○	○	○
A×B	A^B	B^A	B^2	$A×B^2$	$A×B^B$	$A×B^A$	$A×A^B$	Product of all the numbers from A to B	Error	Infinite Loop

A keeps getting multiplied by B … B times, so the answer is A x B x B x … x B (B times), so that's AxB^B

Question 10: What are possible values of **`BALANCE`** when the program is done? (select ALL that apply, 3 pts)

☐	■	■	■	☐	☐
60	70	80	90	100	110



```
when I receive [WITHDRAW]
script variables (new balance)
wait (random time) secs
set [new balance] to (BALANCE)
wait (random time) secs
set [new balance] to ((new balance) – (10))
wait (random time) secs
set [BALANCE] to (new balance)
```

```
when green flag clicked
set [BALANCE] to (100)
broadcast [WITHDRAW]
```

```
when I receive [WITHDRAW]
script variables (new balance)
wait (random time) secs
set [new balance] to (BALANCE)
wait (random time) secs
set [new balance] to ((new balance) – (20))
wait (random time) secs
set [BALANCE] to (new balance)
```

```
random time
report (pick random (1) to (100))
```

If one of these finishes completely before the other, BALANCE will be 70. But the problem happens when both load the old version of BALANCE=100 (race condition!) and then both change new balance at the same time to their internal number of 80 and 90, then both write the value of BALANCE over each other. So depending who goes LAST, the value is either 90 (the left one) or 80 (the right one), SID: ______________

Question 11: *Will the person with the highest-numbered SID please stand up?* (9 pts)

Here are 3 algorithms to find the student with the highest-numbered Student ID (SID)…note, SIDs are unique. For all problems, assume the number of students is 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 – “Meet Everyone” algorithm

1. Everyone walks around the room and compare SIDs with everyone else. Whenever two students meet, the one whose SID is smaller takes a coin labeled “not the largest” from a pile and pockets it.
2. After everyone has met everyone else and compared SIDs, anyone with a coin in their pocket sits.
3. The person remaining standing has the largest SID.

Algorithm II – “Down the Line” algorithm

1. Everyone lines up, and the first person is designated as the “largest-so-far”.
2. The “largest-so-far” person goes down the line, comparing their SID to that of each new person, 1-on-1.
3. Whenever a new person’s SID is larger than the “largest-so-far”, that new person *replaces* the “largest-so-far” person and continues going down the line, doing step 2.
4. The last person to be “largest-so-far” has the largest SID.

Algorithm III – “Tournament” algorithm

1. Everyone stands up
2. All standing people pair up and compares SIDs. (If there’s an odd number of people, the “odd person out” just stands around idle for that round.)
3. Whomever has a smaller SID sits down.
4. Step 2-3 continues until there is one person standing; that person has the largest SID.

a) In the WORST case, what’s the number of *comparisons* (NOT running time)? (select ONE per row)

Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential
Meet Everyone	○	○	○	●	○	○
Down the Line	○	○	●	○	○	○
Tournament	○	○	●	○	○	○

Meet Everyone involves N * (N – 1) / 2 [for every N people, they meet N-1 other people, but we don’t count A meeting B and B meeting A twice so we divide by 2]. As N grows big, it’s basically N*N comparisons.
Down the Line involves exactly N-1 comparisons (since it’s the same number if the first one is the largest so far and doesn’t change, so the number of comparisons is just the number of other people (N-1.
Tournament is actually the same, since there are N-1 “losers” and every comparison generates one loser.

b) If all SID comparisons must occur in a SINGLE (small) ROOM, and only 2 people could fit in that room, how much clock time (NOT running time) would each algorithm take in the BEST case? (select ONE per row)

Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential
Meet Everyone	○	○	○	●	○	○
Down the Line	○	○	●	○	○	○
Tournament	○	○	●	○	○	○

The BEST case is actually the same as the problem above, since only one comparison can happen at a time.

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 BEST case? (select ONE per row)

Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential
Meet Everyone	○	○	●	○	○	○

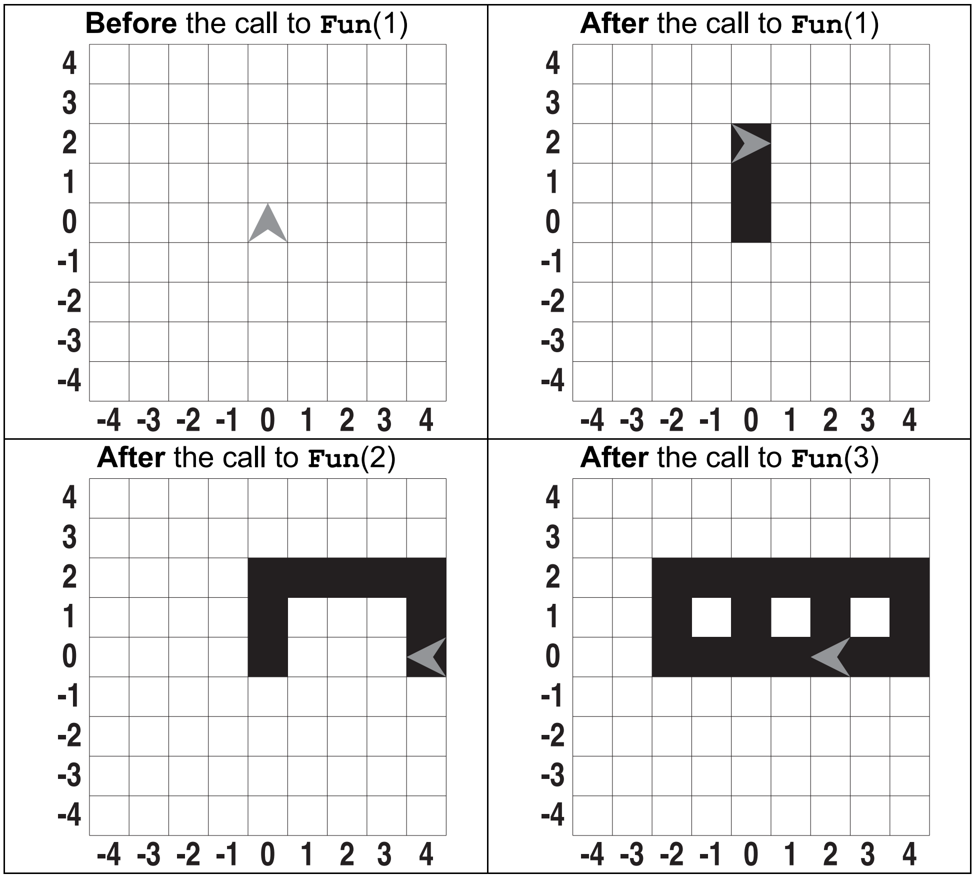
Down the Line	○	○	●	○	○	○
Tournament	○	●	○	○	○	○

Meet everyone hugely benefits from the parallelism, since all people can compare at the same time. If every step, N people are meeting N other people, the whole thing can be done in N time units. Similarly with Tournament, in that it benefits from the parallelism, but there are diminishing performance returns because fewer and fewer people are still around for the next round. Since we cut it in half every time, it's logarithmic. Down the line doesn't benefit from this at all since the algorithm stipulates only one comparison happening at one time.

Question 12: *We put the Fun in Functional Programming…* (15 = 4+4+4+3 pts)

Consider the following code below on the right. 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.

a) Your job is to shade in (completely!) *all* the pixels that will be colored in after calls to **`Fun`** with n set to **2** and **3**; draw sprite at the end in some way that helps you (for this question, we'll only look at the pixels when we're grading). *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.*

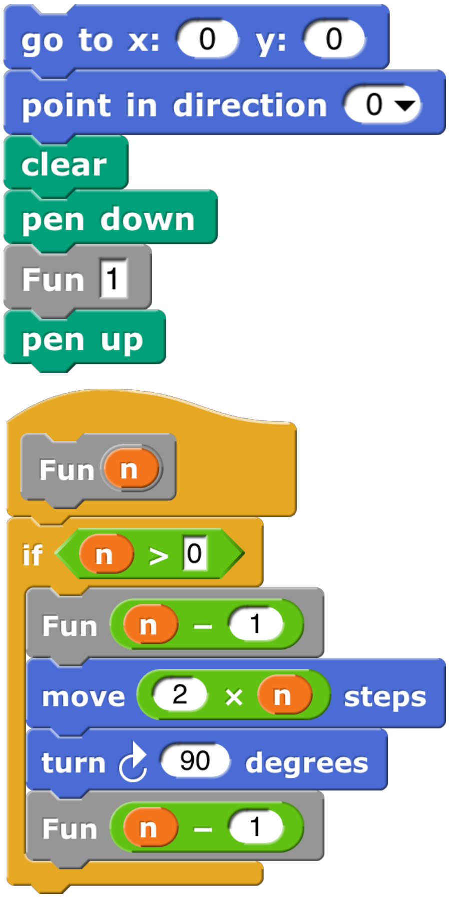


Before the call to **`Fun`**(1)

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

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

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



```
go to x: 0 y: 0
point in direction 0
clear
pen down
Fun 1
pen up

Fun n
if n > 0
    Fun n − 1
    move 2 × n steps
    turn ↻ 90 degrees
    Fun n − 1
```

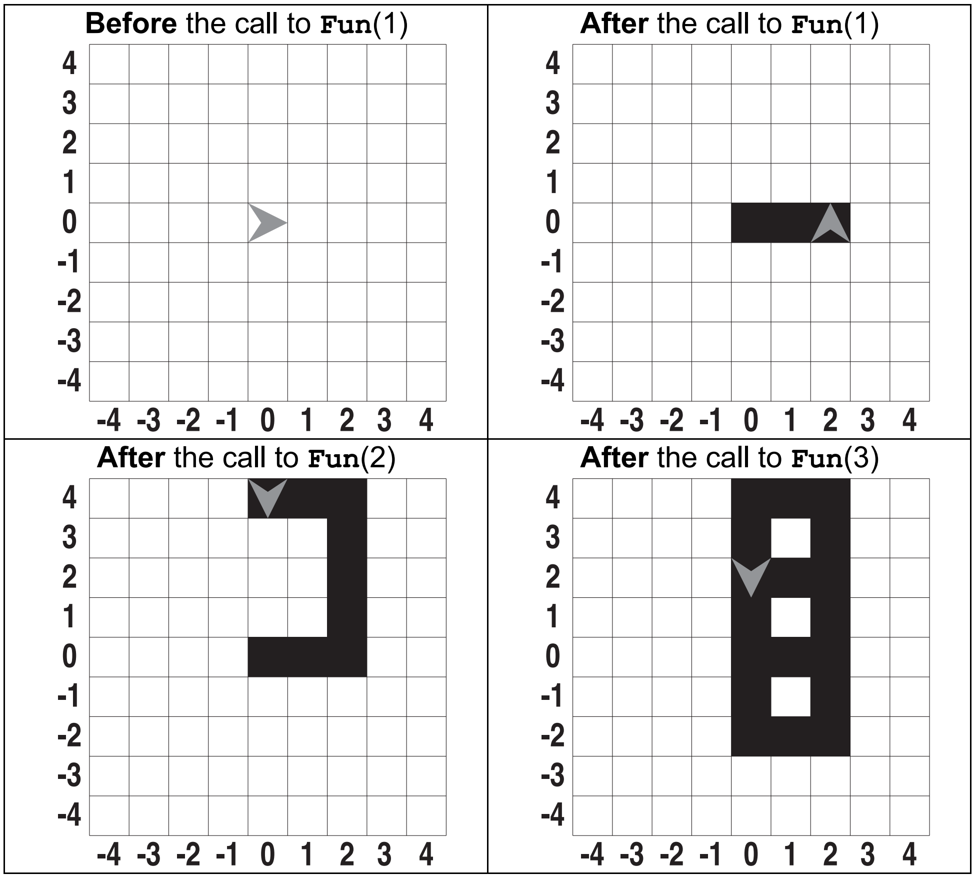
a) What direction is the sprite facing at the end of the call to **Fun**(3)? (select ONE)

● ←
○ ↑
○ →
○ ↓

b) What direction is the sprite facing at the end of the call to **Fun**(100)? (select ONE)

● ←
○ ↑
○ →
○ ↓

The alternate version had the diagram flipped over its xy-axis, since we started the sprite facing right and had it rotate left with every turn, not right.

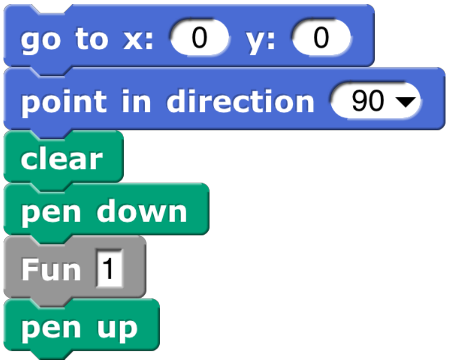


Before the call to **Fun**(1)

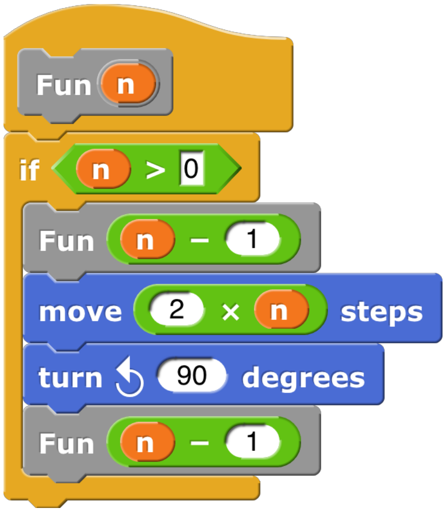
After the call to **Fun**(1)

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

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



```
go to x: 0 y: 0
point in direction 90
clear
pen down
Fun 1
pen up
```



```
Fun n
if n > 0
    Fun n − 1
    move 2 × n steps
    turn ↺ 90 degrees
    Fun n − 1
```

a) What direction is the sprite facing at the end of the call to **Fun**(3)? (select ONE)

○ ←
○↑
○ →
● ↓

b) What direction is the sprite facing at the end of the call to **Fun**(100)? (select ONE)

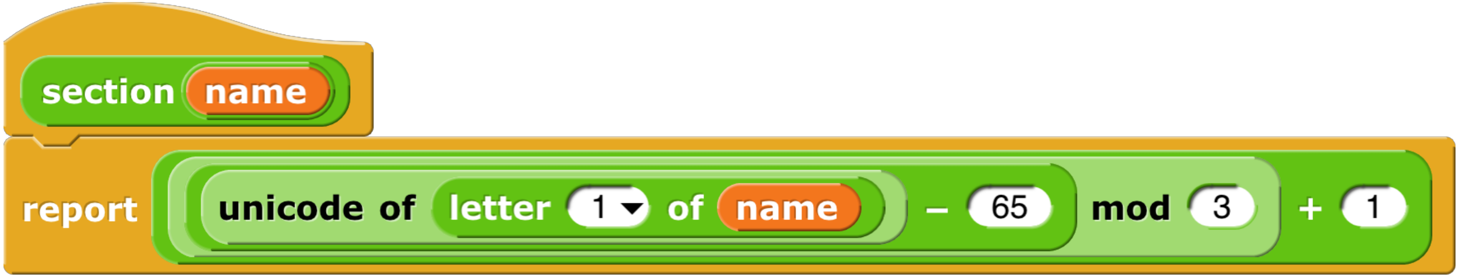
○ ←
○↑
○ →
● ↓

Question 13: *Trust me on the sunscreen…* (6 pts)

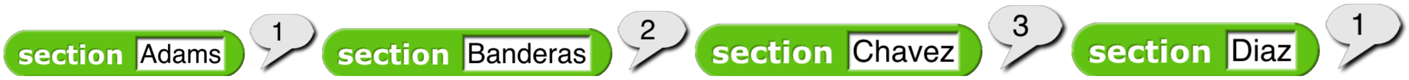
There are three seating areas at graduation, and they assign people based on the first letter of their last name as follows: “A”s in section 1, “B”s in 2, “C”s in 3, “D”s in 1, etc.

Section 1	Section 2	Section 3
Adams Diaz etc.	Banderas Estrada etc.	Chavez Fox etc.

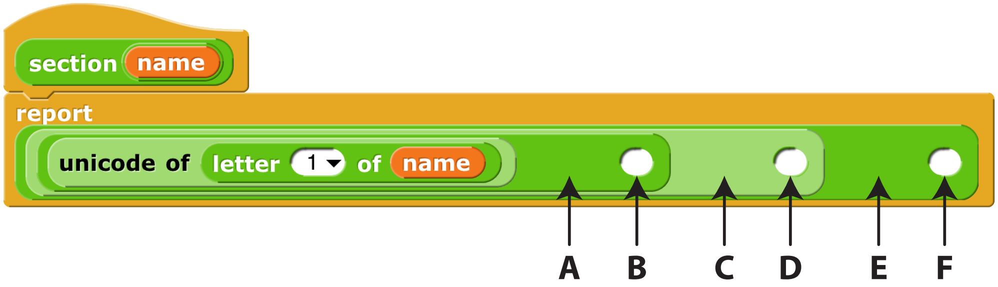
Complete the block that finds the section based on the last name. Hint: The unicode of “A” is 65.
(select ONE per row)



```
section (name)
report ((((unicode of (letter (1) of (name))) − (65)) mod (3)) + (1))
```



```
section Adams     → 1
section Banderas  → 2
section Chavez    → 3
section Diaz      → 1
```

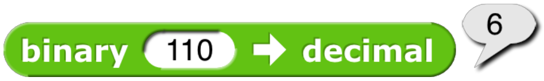


```
section (name)
report unicode of (letter (1) of (name)) ( ) ( ) ( )
```

	+	–	×	÷	mod	and	or	0	1	2	3	4	63	64	65	66	67
A	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
B	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○
C	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○
D	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○
E	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
F	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○

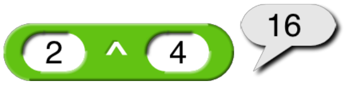
Question 14: *Mommy, he was counting in Binary on his fingers and he gave me the `0b00100`…* (15 pts)

Write a block converting Binary to decimal:

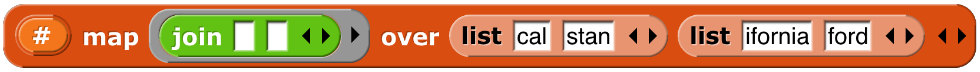

binary (110) → decimal
6

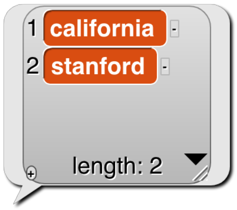
$1*2^2 + 1*2^1 + 0*2^0=4+2=6$

Here is a helper block to compute a base to a power, e.g., $2^4 = 16$:

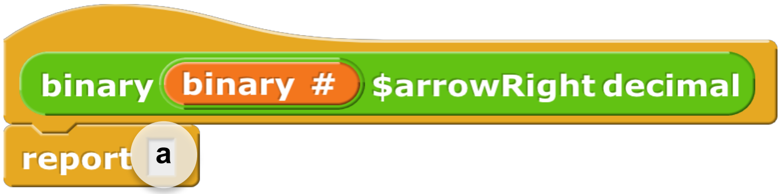

(2) ^ (4)
16

Here's a reminder how the map with two lists works:


(#) map (join [ ] [ ]) over (list cal stan) (list ifornia ford)


1 california
2 stanford
length: 2

A couple of notes. First, your solution should use each of these blocks *exactly once*, so there should be one circle filled in for every column and row (but, multiple blanks are ok). Also, the `numbers-from( )to( )` block works if the first argument is smaller or bigger than the second argument. So `numbers-from(3)to(1)` would return a list with the first (topmost) element 3, the second element 2, and the last (bottommost) element 1.

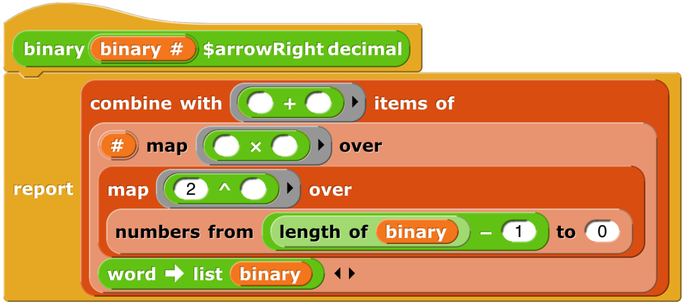

binary (binary #) $arrowRight decimal
report (a)

…into these slots

Put these blocks below…

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
map (b) over (c)	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○
(#) map (d) over (e) (f)	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
combine with (g) items of (h)	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○
numbers from (i) to (j)	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○
(k) × (l)	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○
(m) + (n)	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○
(2) ^ (o)	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○
word → list (binary)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○
(length of (binary)) − (1)	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○
0	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○
…blank…	○	○	○	○	○	○	○	○	○	○	●	●	●	●	●

As a sanity-check, write your solution out below (so you don't get points off for bubbling things wrong!)


```
binary (binary #) $arrowRight decimal
report
  combine with (( ) + ( )) items of
    (#) map (( ) × ( )) over
      map ((2) ^ ( )) over
        numbers from ((length of (binary)) − (1)) to (0)
      word → list (binary)
```

Question 15: *Berkeley `python`'s flying circus [this is a 2-part question]…* (18 pts = 9 * 2pts)

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

```
>>> "".join(["berkeley"[int(i)] for i in "31415"])
```

○`"bbberrrrkeeeee"` ○`"bbrrrkkkk"` ○`"rbkbe"` ●`"keeel"` ○ Error ○ None of these

It uses the indices of 31415 to draw out `k`, `e`, `e`, `e`, `l` which are smushed together with `"".join()`
Remember 0-indexing!

```
def swap_values(A,B):
      A = B
      B = A

def swap_elts12(w):
      w[1] = w[2]
      w[2] = w[1]
      return w

>>> you = 20 ## your age, that is
>>> dan = 50 ## dan's age, that is
>>> swap_values(you, dan)
>>> [you, dan]
```

●`[20,50]` ○`[50,20]` ○`[20,20]` ○`[50,50]` ○ Error ○ None of these

`A` and `B` are local variables (input parameters) – changes to them directly don't change the called values.

```
>>> swap_elts12([5,6,7,8])
```

○`[5,7,6,8]` ○`[6,5,7,8]` ● `[5,7,7,8]` ○`[5,6,6,8]` ○ Error ○ None of these

`w` is a local variables (input parameters) – changes to it directly wouldn't change the called values. However, just as in Snap!, if w is a list you can change the elements of w within the procedure. But this doesn't swap the values, it does it incorrectly, first clobbering the value of the second element (6) with the third (7). The second line of `swap_elts12` does nothing, since there's already two 7s in the second and third location.

```
>>> swap_elts12("bear")
```

○`"baer"` ○`"ebar"` ○`"baar"` ○`"beer"` ● Error ○ None of these

Strings are immutable!

```
def mystery(arg):
      D = {}
      for a in arg:
            for b in a:
                  if a in D:
                        D[a] += 1
                  else:
                        D[a] = 1
      return D
```

```
>>> D = mystery(["cal", "california", "a"])
>>> D["a"]
```

○0 ●1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○ Error ○ None of these

This goes through every word (a) in the list(arg). For every letter (b) in each word (a), it sees if it's seen the word (a) before. If so, it increments the count of it, otherwise it initializes it to 0. So when it's all done, D is {'cal': 3, 'a': 1, 'california': 10} …which is like having a quick lookup of the length of the word! Asking for D["a"] is 1, since the length of "a" is 1.

```
>>> E = mystery("california")
>>> E["a"]
```

○0 ○1 ●2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○ Error ○ None of these

This goes through every letter (a) in the string (arg). A letter is itself a string! So for every letter (b) in each letter (a) – this inner loop happens once for every letter, it sees if it's seen the letter (a) before. If so, it increments the count of it, otherwise it initializes it to 0. So when it's all done, E is {'l': 1, 'a': 2, 'o': 1, 'c': 1, 'i': 2, 'f': 1, 'r': 1, 'n': 1} …which is like having a quick lookup of the letter count of each word! Asking for E["a"] is 2, since there were two "a"s in california.

```
def stutter(word):
       ### cal → ccal
       return word[0]+word
```

```
def reverse(word):
       ### cal → lac
       return word[::-1]
```

```
def duplicate(word):
       ### cal → calcal
       return word+word
```

```
def compose(f,g):
       ### Same as Snap's compose block
       return lambda x: f(g(x))
```

```
### reduce is just like Snap!'s combine: reverse(stutter(duplicate(word)))
frankenstein = reduce(compose, [reverse, stutter, duplicate])

>>> frankenstein("cal")
"laclacc" ### same as Snap!

L = […some combo of duplicate, stutter, reverse (possibly 0 or many of each)…]
>>> frankenstein_bride = reduce(compose, L)
```

```
>>> duplicate(duplicate(2))
```

○"2222" ○"44" ○"8" ○"6" ○2222 ○44 ●8 ○6 ○ Error ○ None of these

+ is overloaded. When the arguments are numbers, it adds them. So the inner duplicate returns 4, and the outer one returns 8.

```
>>> duplicate("duplicate(2)")
```

○"2222" ○"44" ○"8" ○"6" ○2222 ○44 ○8 ○6 ○ Error ● None of these

+ is overloaded. When the arguments are strings, it concatenates them (like join Snap!). So the inner duplicate returns duplicate(2)duplicate(2), and the outer one returns duplicate(2)duplicate(2)duplicate(2)duplicate(2), which is none of these options.

What are possible return values of frankenstein_bride("cal")? (select ALL that apply)

■"call" □"caal" □"ccallcal" □"calac" □"lacal" □"acl" □ None of these

Ok, so there's a huge chunk of "stutter"s (duplicate first letter), "duplicate"s (duplicate) and "reverse"s (reverse). If the input is "cal", what could the output be?

- Can we do "call"? Sure, if it's reduce(compose, [reverse, stutter, reverse])
- Can we do "caal"? Nope, since we can't expose just the a to duplicate it.
- Can we do "ccallcal"? Nope, since we can't have any part of the inner word (here "ll") not also be somewhere in the outer word. We can get ccall, but can't have the original at the end without having the other parts duplicated too. So we can get ccallccall but can't get ccallcal. Hard to explain…
- Can we do "calac"? Nope, since we can't reverse just a part of the word.
- Can we do "lacal"? Nope, since we can't reverse just a part of the word.
- Can we do "acl"? Nope, since we can't rotate the a out of the middle.