

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

Your Name (first last)

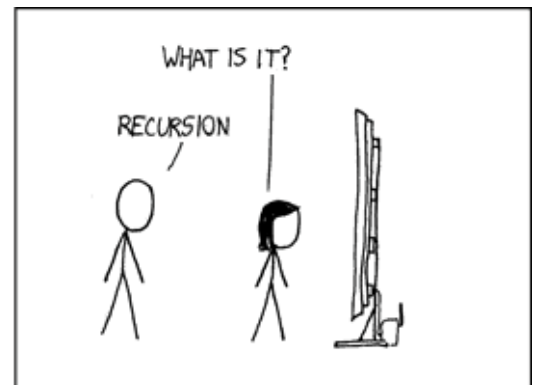
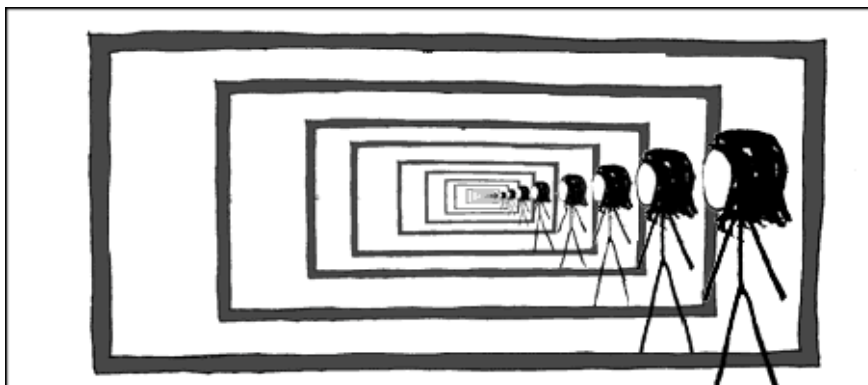
SID

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: 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)!
- ☐ *The fridge/freezer cooling control interface was an example of good design*: two independent, intuitive knobs.
- ☐ The iPhone was *one of the rare devices that didn't require a prototype, a testament to Steve Jobs' genius!*
- ☐ *User interface code is typically 75% of all the code* in “real” programs.
- ☐ 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.
- ☐ Faster computers *don't give us any more resolution of our hurricane models, they just render them faster.*
- ☐ Scientists are using Machine Learning detection (like finding a cat in an image) with hurricane simulations.
- ☐ We use abstraction in CS10, but *scientists don't with the Community Climate Code*, since details matter!
- ☐ 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.
- ☐ Finding a polynomial-time solution to just one NP-Complete problem means $P \neq NP$!
- ☐ A program exists that can tell if another program, run on some input, will eventually stop.
- ☐ The greedy approach to solving the “knapsack problem” will always produce the optimal answer, as we saw.
- ☐ 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.
- ☐ Simulated Neural Networks used to be all the rage, but limited results meant the field has abandoned it.
- ☐ AI systems are now able to generate (not just recognize) images that never existed before.
- ☐ One of the benefits of using AI systems in law enforcement is that they don't contain racial/gender biases.
- ☐ None of these.

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

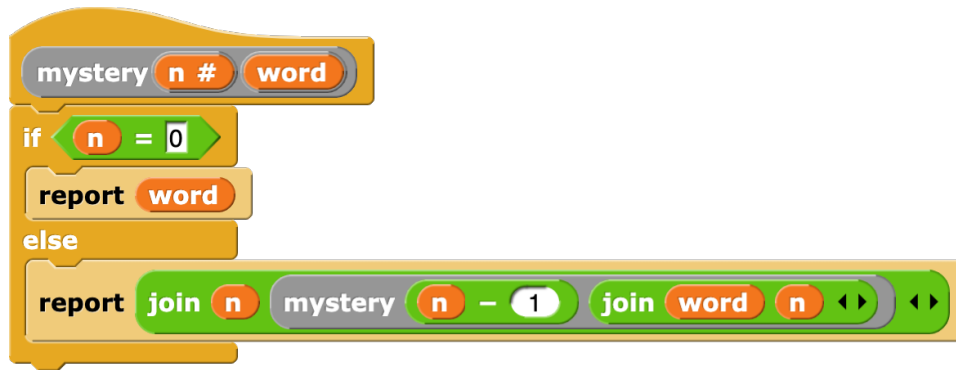
- ☐ A professional game developer at Electronic Arts who advised to take CS classes outside of the CS dept.
- ☐ A professional software engineer at Pixar who advised you not to avoid challenge.
- ☐ A professional backend engineer at Apple who advised you to delete your social media.
- ☐ A professional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

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

<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
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

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

SID: _____



- a) What does **mystery 1 CAL** return?
If it is an error, write "error". If it never returns, write "loops".

- b) What does **mystery 2 CAL** return?
If it is an error, write "error". If it never returns, write "loops".

- 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".

--	--	--	--	--

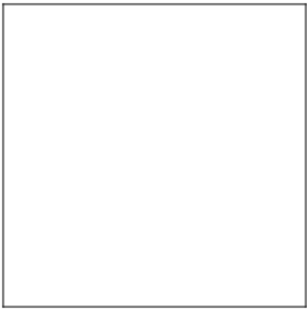
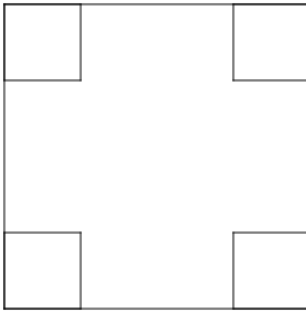
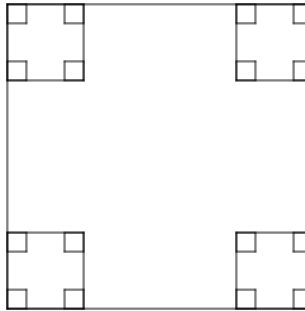
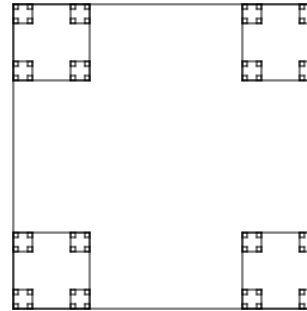

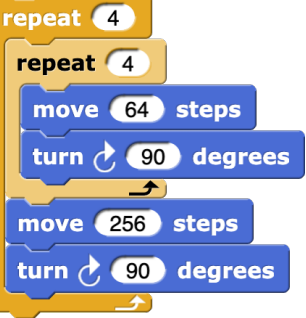
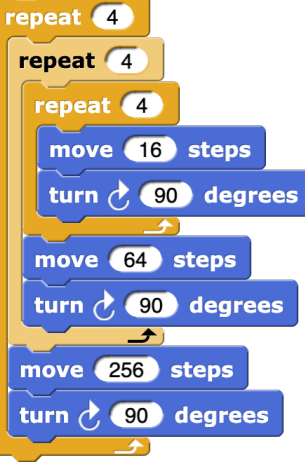
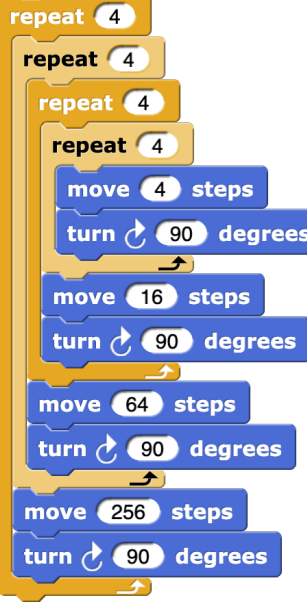
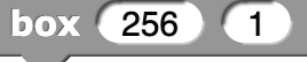



...

--	--	--	--	--

- 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.

Question 8: Box within a box within a box... (8 pts)

SID: _____

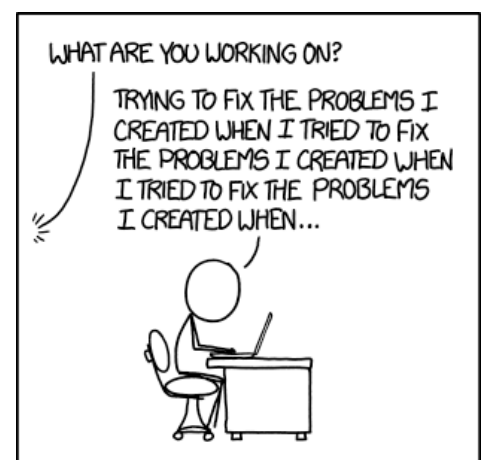
N = 1	N = 2	N = 3	N = 4
			
 <p>In "text your friend" format, this would be:</p> <pre>repeat (4) --> move (256) steps --> turn right (90)</pre>	 <p>In "text your friend" format, this would be:</p> <pre>repeat (4) --> repeat (4) --> move (64) steps --> turn right (90) --> move (256) steps --> turn right (90)</pre>	 <p>Hopefully you see what the "text your friend" pattern would be...</p>	
			

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...

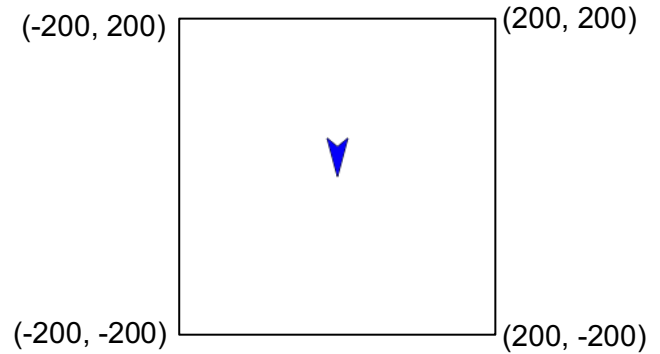
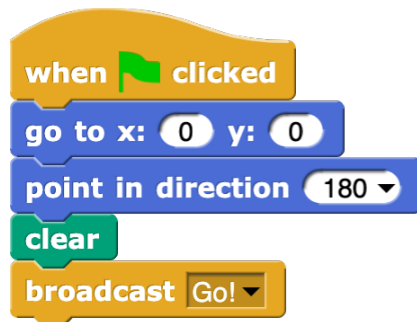
```
box(steps, N)

----> if _____
----> ----> _____
----> ----> ----> _____
----> ----> ----> _____
----> ----> ----> _____
```



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

SID: _____



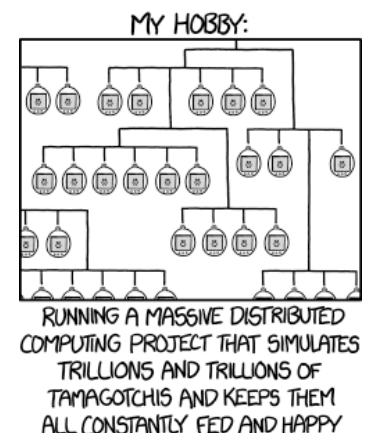
a) *How many different images can be drawn when we click the green flag?*

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?



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).

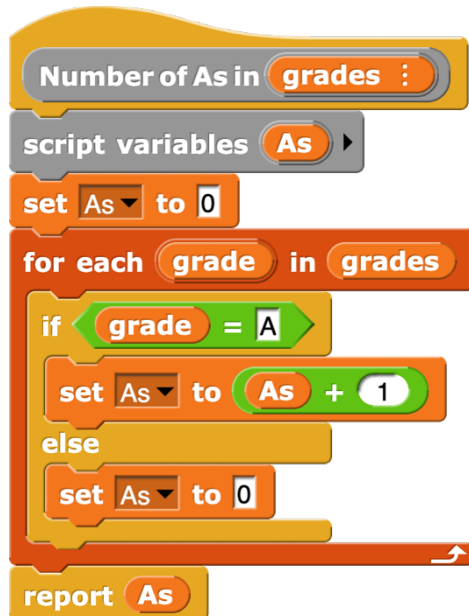
For example, if the cards were (top-to-bottom) A B C D, the table shows what would happen...

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		2							
---	--	---	--	--	--	--	--	--	--

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



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”

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

SID: _____

Algorithm I – “Musical Chairs” algorithm

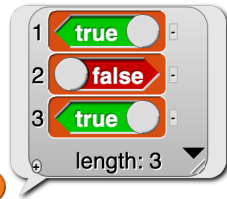
- Algorithm II – “Two Circles” algorithm

-
- Diagram illustrating a transformation of a circular arrangement of six nodes. The initial arrangement (left) consists of nodes labeled A1, B2, C3, D4, E5, and F6. An arrow points to the transformed arrangement (right), which consists of nodes labeled B6, C1, D2, E3, F4, and A5. This represents a clockwise rotation of one position for each node.

Algorithm	Yes	No
<i>Musical Chairs</i>	<input type="radio"/>	<input type="radio"/>
<i>Two Circles</i>	<input type="radio"/>	<input type="radio"/>
<i>One at a Time</i>	<input type="radio"/>	<input type="radio"/>

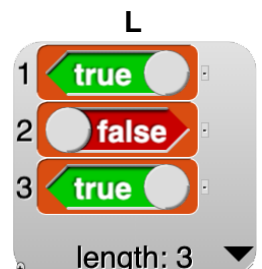
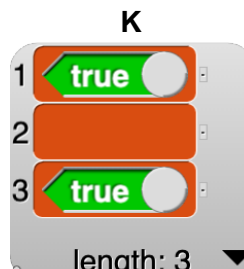
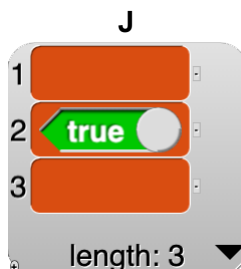
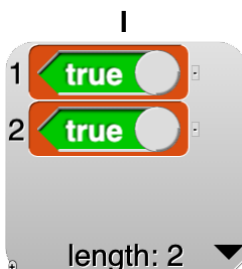
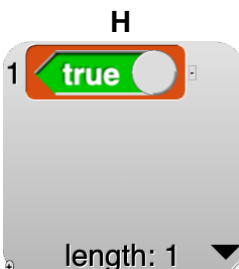
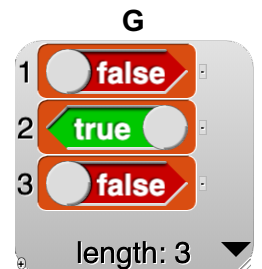
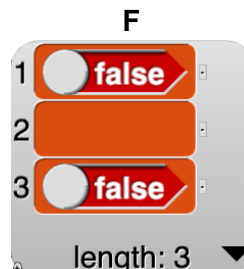
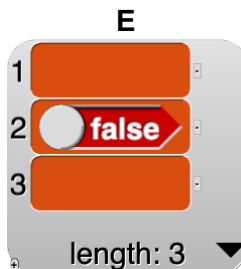
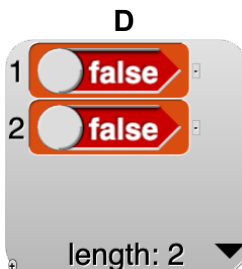
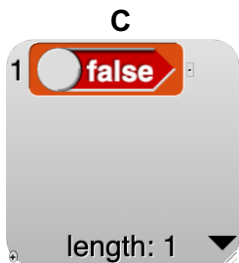
Algorithm	Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential
<i>Musical Chairs</i>	○	○	○	○	○	○
<i>Two Circles</i>	○	○	○	○	○	○
<i>One at a Time</i>	○	○	○	○	○	○

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



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	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
keep items not from DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
map not not over DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
keep items not not from DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
map not over map not over DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
map not over keep items not from DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
keep items not from map not over DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
keep items not from keep items not from DATA	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>



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

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

☐ [4] ☐ [51] ☐ ["4"] ☐ ["51"] ☐ 4 ☐ 51 ☐ "4" ☐ "51" ☐ "31" ☐ "501" ☐ Error

```
>>> 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])
```

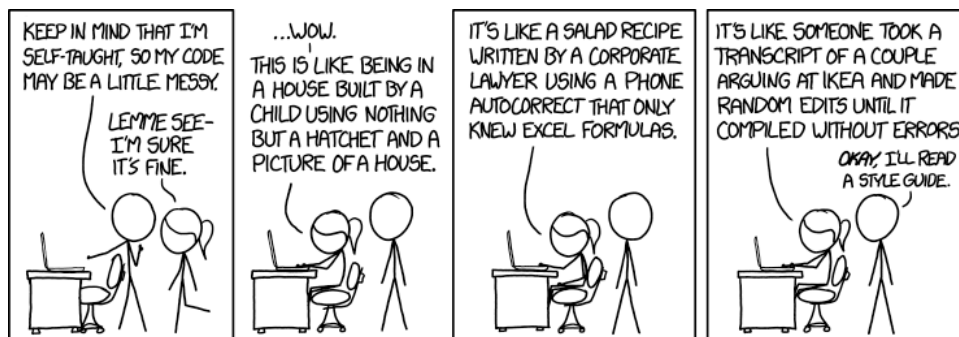
```
>>> print(frankenstein(_____)) ### What input causes 4141 to be printed?
4141
```

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 _____
>>> def any_matches(CalSIDs, StanfordSIDs):
    return _____crossmap(helper, CalSIDs, StanfordSIDs)
```



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 -460° F (absolute 0) and the warmest it'll ever be is 212° 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:

-460°	40°	60°	80°	212°
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 45°, it'd say “Cool”. If it were 99°, 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:

        key_low, key_high = low_high_range    ### simultaneous assignment since
                                                ### low_high_range is a tuple

        if _____

            return _____
```

- 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!
```

- 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 _____

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